IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 March 2024, accepted 25 March 2024, date of publication 2 April 2024, date of current version 10 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3384232

RESEARCH ARTICLE

Mixed Criticality Reward-Based Systems Using Resource Reservation

AMJAD ALI[1], **SHAH ZEB**[1], **MADALLAH ALRUWAILI**[2], **(Member, IEEE)**,
ASAD MASOOD KHATTAK[3], **(Member, IEEE)**, **BASHIR HAYAT**[4], **AND KI-IL KIM**[5]
[1]Department of Computer and Software Technology, University of Swat, Swat, Khyber Pakhtunkhwa 19120, Pakistan
[2]College of Computer and Information Sciences, Jouf University, Sakaka, Aljouf 2014, Saudi Arabia
[3]College of Technological Innovation, Zayed University, Dubai, United Arab Emirates
[4]Centre of Excellence in IT, Institute of Management Sciences, Peshawar, Khyber Pakhtunkhwa 25000, Pakistan
[5]Department of Computer Science and Engineering, Chungnam National University, Daejeon 34134, Republic of Korea

Corresponding author: Madallah Alruwaili (madallah@ju.edu.sa)

This work was supported by the Deputyship for Research & Innovation, Ministry of Education, Saudi Arabia, under Project 223202.

ABSTRACT Real-time systems mostly interact with the external world and each input operation must meet predetermined deadlines to be useful. However, in many real-time applications, a partial result is also acceptable. We developed a reward-based mixed criticality system based on the resource reservation approach to address the problem of ensuring the effective execution of low- and high-criticality tasks in both low- and high modes, even under heavy workloads. Using dedicated servers with pessimistic resource allocation for each high criticality task ensured their execution in both modes unaffected by low criticality tasks. The surplus resources are reclaimed and assigned to low critical tasks' server by utilizing a greedy reclamation of unused bandwidth (GRUB) algorithm. Three strategies were suggested for server allocation to low criticality tasks: a dedicated server for all low criticality tasks, a single server for each low criticality task, and two servers (mandatory and optional) for each low criticality task. Our analysis revealed efficiency of the first approach by achieving 100% schedulability at a 1.1 target utilization, scheduling 20% and 50% more task sets than the second and third approaches, respectively. Moreover, the effectiveness of the proposed approach over existing imprecise mixed criticality approaches were demonstrated through comprehensive experimentation.

INDEX TERMS Greedy reclamation, imprecise computation, imprecise mixed criticality (IMC) systems, mixed criticality systems, real-time systems, resource reservation reward-based scheduling, servers.

I. INTRODUCTION

Real-time systems include a wide range of systems, from basic control loops in embedded systems to complex networks of distributed systems. The specifications, standards, and applications of these systems cover a wide range of areas such as industrial automation, aviation, automobile, smartphones, and medical systems. Modern real-time embedded systems frequently interact outside the environment; consequently, their output must be generated in real time, i.e., within a certain amount of time or it is rejected and considered a failure. Therefore, both logical and temporal correctness are key factors for defining the correctness of the systems. Being logically correct implies generating an accurate result, which is typically required in general computing systems. However, real-time system scheduling imposes additional emphasis on temporal correctness, which simply refers to generating accurate results within a given period. Real-time systems are reactive systems that must produce relevant outputs for specific inputs under predetermined time constraints. For example, the airbag in a vehicle must be deployed in a fractions of a second, neither too soon nor too late. Safety-critical applications are primarily based on real-time systems wherein a controller malfunction can result in severe casualties, thereby endangering life. Fundamentally, there are two categories of real-time constraints: hard and soft

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.

constraints. In hard real-time systems, missing a deadline can result in catastrophic consequences, including fatalities and loss of resources. Examples include vehicle airbag systems or high-speed rail signaling systems. Unlike hard real-time systems, deadlines missed in soft real-time systems such as surveillance systems or the global positioning system (GPS) does not cause catastrophic consequences; however, it affects the service quality of the system.

Many applications employing real-time systems now exhibit another crucial feature known as criticality level, which is attributed to the emergence of safety critical systems with various functionalities such as safety critical, mission critical, and low critical. The assurance level required to prevent a safety-critical system component from failing is referred to as criticality and there can be up to five levels of criticality. A safety-critical system with two or more levels of functionalities is commonly known as a mixed criticality system (MCS). A good example of an MCS with multiple levels of functionalities is a modern vehicle. The modern vehicle performs safety-critical functions with a higher level of criticality, such as in airbag systems or antilock braking systems, and mission critical functions with a lower level of criticality, such as in radio or air-conditioning systems. Because the operational safety of the entire system relies heavily on safety-critical functionalities, their failure may have disastrous effects, such as loss of assets or human life. Mission-critical functionalities have no effect on the safety of the system because of their lower criticality level; however, their failure degrades the overall service quality of the system. A safety-critical system with different functionalities of various criticality levels operating in a single and reliable computing environment is referred to as an MCS.

The traditional safety-critical real-time systems had to be tested and certified in their entirety to show that they were safe to use. On the other hand, the design of an MCS is subjected to certification at different criticality levels by the certification authority (CA) for their successful deployment. The concern of CA is only the temporal correctness of the safety-critical part of an MCS under very pessimistic assumptions (e.g. worst-case execution times (WCETs) calculated from static analysis of a code), while the responsibility of system designer is to ensure that the entire system operates correctly by guaranteeing the correctness under less pessimistic assumptions (e.g. WCETs from the measurements of a code). Therefore, for the guaranteed execution of high critical tasks in any scenario, CAs consider highly pessimistic cases that are rare and improbable to really occur. Consequently, exaggerated WCETs are considered for these highly critical tasks, leading to wastage of resources. Vestal [1] suggested designating a high critical task with multiple WCETs to manage this exaggeration. Owing to the optimistic WCETs of high critical tasks, all high- and low critical tasks can be successfully scheduled, and hence, they can better utilize the hardware resources. This operating mode of an MCS is known as the low criticality mode. The system mode is changed to a high criticality mode when, in a rare scenario, the execution of a highly critical task cannot be completed according to its optimistic WCET. In the high critical mode, certified pessimistic assumptions are utilized for scheduling high critical tasks, and this affects the schedulability of low critical tasks, and therefore, they are dropped in this mode.

The problem of scheduling real-time tasks in overload conditions refers to those critical situations in which the computational demand requested by the task set exceeds the time available on the processor. The primary risk is potential deadline misses for critical tasks, impacting overall system behavior. In real-world scenarios, system overloads can arise due to factors like simultaneous arrival of tasks, malfunctioning of input devices, unexpected environmental variations, and operating system exceptions. For example, the systems that are designed with inefficient approach might function well under regular circumstances but collapse during peak load situations when demands exceed resources. Even well-designed systems can face load spikes due to multiple simultaneous events.

The motivation behind the studied work is to address the challenges posed by overloaded scenarios in real-time MCSs. In realistic MCS model [2], the execution of low criticality tasks in both low and high modes holds significant importance. These systems play a vital role in safety-critical domains such as aerospace, defense, and medical devices, where timely and dependable task execution is crucial. The schedulability of MCSs with heavy workloads is a challenging research problem in the field of real-time systems. Ensuring the schedulability of high criticality tasks while guaranteeing partial or complete schedulability of low criticality tasks in the high mode is difficult because heavy workloads can lead to deadline misses and system failures. The significance of this research lies in the development of an effective MCS scheduling strategy that ensures the successful execution of high critical tasks while maintaining system stability and quality by ensuring the execution of low criticality tasks in both low and high modes. By achieving this, the proposed approach contributes to enhancing the reliability, safety, and performance of MCSs under demanding workloads, thus advancing the field of real-time systems and critical applications.

To address the issue of the schedulability of low critical tasks in both modes, several scheduling strategies were proposed by Burns and Baruah [3]. To enable low criticality tasks to make progress after the system shifts to high mode, they suggested retaining the schedulability of these tasks without affecting the schedulability of high criticality tasks. In the context of a fixed priority system, this can be achieved by: (i) adjusting the priority of these tasks to be lower than any high criticality task, (ii) reducing the execution time requirements of low criticality tasks to ensure their execution in the spare capacity available in high mode, or

(iii) extending the period of low criticality tasks to achieve a similar outcome. The proposed approach in our research work draws inspiration from the second approach. However, in the proposed approach, by associating a reward with the execution of low criticality tasks, their reduced WCET can be extended to enhance their output quality, leading to an improvement in overall system performance.

In this research work, we adopted a resource reservation technique with reward-based scheduling. The resource reservation technique is a class of techniques that can ensure the schedulability of high criticality tasks in MCSs during periods of heavy workload. These techniques work by reserving system resources such as CPU time, memory, or bandwidth for high criticality tasks, while still allowing low criticality tasks to execute when resources are available. The main idea of the proposed approach is providing high criticality tasks with sufficient resources to ensure their execution in both modes, even under heavy system workloads. Reward-based scheduling is adopted for scheduling low critical tasks. The basic concept of this technique is to compromise precision for timeliness when resources are insufficient for achieving the WCET guarantees. It divides each low critical task into a pair of logical parts: the mandatory part produces an output with the minimum acceptable quality, and the execution of the optional part enhances the quality of the produced output when there is sufficient computing capacity; otherwise, it is discarded.

The performance of the proposed approach is evaluated through a comprehensive comparison with other state-of-the-art [3], [4], [5] scheduling algorithms for the IMC model. The results demonstrate that the proposed technique outperforms these approaches in terms of heavy workloads. The proposed technique effectively ensures the schedulability of high criticality tasks without being affected by the behavior of low critical tasks. Three different strategies are provided to schedule low critical tasks in both modes. The advantages of the proposed system include improved schedulability, i.e., reserving sufficient system resources for high criticality tasks, even in heavy workload situations; providing strong guarantees for WCETs and response times of high criticality tasks; and ensuring a predictable and reliable system performance. Further, the proposed technique enables efficient resource sharing by allowing low criticality tasks to use unused resources, which increases overall system efficiency while providing strong guarantees for highly critical tasks.

The main contributions of this study can be summarized as follows:

- We proposed a modified version of the earliest deadline first (EDF) approach with a tuned deadline EDF-TD for high servers that schedule high critical tasks.
- We proposed a resource reservation technique in the reward-based scheduling paradigm that effectively schedules high criticality tasks independently of the behavior of low criticality tasks. Furthermore, low criticality tasks are also provided some service in the high mode instead of being discarded. The pseudocode for the proposed technique is also provided.
- We performed extensive experiments to evaluate the effectiveness of the proposed technique over existing IMC strategies.

To the best of our knowledge, this is an initial attempt in MCSs to address a resource reservation based approach within the paradigm of reward-based scheduling in a single-core environment. Furthermore, in the context of the multicore partitioned approach, where tasks are allocated to different cores, the scheduling behavior of each core closely emulates that of a single-core system, subject to the exclusion of various sources of indeterminism within the multicore paradigm. This is due to the fact that the multicore partitioned approach lacks task migration between cores. Hence, the advent of multicore processors can also benefit from the applicability of single-core algorithms. Moreover, this research work focuses on periodic task model that are commonly encountered in various real-world applications, including industrial automation, automotive systems, and multimedia processing, where tasks often need to be executed at regular intervals. For example, in a chemical plant temperatures, pressure and other attribute are measured periodically and all information is passed to the controller. By focusing on periodic task models, we aim to provide a targeted analysis of the challenges and solutions related to this specific scenario. However, the proposed system holds the potential to incorporate sporadic task models.

The remainder of the paper is organized as follows: Section II discusses the relevant literature. Section III describes the proposed system model by defining a task model and discussing the behavior of the system. Section IV describes the proposed technique and defines server specifications along with the pseudocode of the proposed technique. The motivation to support this approach is also discussed. Section V outlines the necessary conditions required for the schedulability of the proposed technique. Section VI discusses the schedulability analysis of the proposed technique in both low and high modes. Section VII analyzes the results obtained from the experimental evaluation. Section VIII concludes the study, and Section IX highlights the future work of the proposed technique.

II. RELATED WORK

The domain of real-time systems on a uniprocessor, wherein schedulability is the primary concern, has been widely studied. Most studies investigated overload systems in which it is difficult to satisfy deadlines for all tasks, and the execution or abortion of tasks is determined by the system. In such systems, the primary goal is increasing the performance factor. To address this problem, imprecise computation with mandatory and optional semantics for a task were initially introduced. The primary focus of this approach was to guarantee the execution of mandatory parts, while mitigating the overall system error. The time required to execute an optional part indicates the accuracy of the system;

a complete execution indicates zero error, whereas abortions indicate the highest possible error. Another strategy of a similar nature is described in a framework called increased rewards with increased service (IRIS), wherein there is no separation of a task into mandatory and optional parts, and the task can run for as long as the scheduler permits with no upper bound. Therefore, an increased reward is associated with increased services. Aydin et al. [6] developed an optimal reward-based scheduling algorithm for periodic tasks where the reward function was associated with the execution of the optional part of a task beyond its mandatory part. Based on this approach, a schedule is considered feasible if each mandatory part of all tasks is completed by the deadline. If the accrued reward is equal to the optional portion of all tasks, the schedule is considered precise; that is, all mandatory and optional parts can be scheduled successfully to provide the maximum reward function. Another strategy involves defining distinct quality-of-service (QoS) parameters for various tasks. Based on the specific QoS measures, the system selects certain QoS levels for various applications depending on the available resources for maximizing the overall system utility.

An emerging practice in the design of safety-critical real-time systems is the integration of different applications with different criticality levels into a shared computing platform known as an MCS, which was first introduced by Vestal [1] in 2007. An MCS is executed in two modes: the low critical mode and the high critical mode. All low- and high-criticality tasks are successfully scheduled in low modes for reduced WCETs for highly critical tasks. The system switches to highly critical mode when a highly critical task signals for more computations than their reduced WCETs. Therefore, to guarantee the schedulability of high criticality tasks with their pessimistic WCETs, all low criticality tasks are discarded in the high critical mode. Several earlier scheduling strategies, including fixed priority and dynamic priority scheduling algorithms were expanded to accommodate the behavior of MCS.

The MCS schedulability problem has recently attracted considerable research attention. Burns and Davis [2] offer a comprehensive overview of MCSs designed to handle tasks with varying levels of criticality, from safety-critical to non-critical, on a single computing platform. They discuss various aspects of MCSs, such as their challenges, design principles, scheduling techniques, and fault tolerance mechanisms, aiming to provide a valuable resource for researchers and practitioners in the field. Most MCS research is based on Vestal's [1] mixed criticality model. However, this model significantly affects the low criticality task services, and therefore, the vestal mixed criticality model is criticized by certain system designers [3], [7]. Therefore, operating in the high mode while serving low critical tasks is very challenging. To address this problem, several new MC scheduling strategies have been developed. For example, adjusting the priorities of certain low criticality tasks [8] or reducing the execution frequency of low criticality tasks through a period extension [7] for ensuring schedulability. However, some applications may choose a lower-quality output to avoid missing deadline instead of achieving a delayed result with precise quality. Consequently, extending the period of low criticality tasks is ineffective for such applications. Various techniques have been investigated to address this problem, including the reduction of a few or all low critical tasks' executions times [3], [4] by using an imprecise mixed criticality (IMC) model [5], [9], or by efficient controlling the budget [10].

The first paper of Baruah and Vestal [11] on MCS with EDF Scheduling was published in 2008. A slack-based mixed criticality technique, criticality-based EDF (CBEDF), was developed by Park and Kim [12] for EDF scheduled tasks. They attempt to delay the execution of highly critical tasks as far as possible by using a series of online and offline analyses. Delaying high criticality tasks helped acquire free time slots for scheduling low critical tasks. Su et al. [13] used the elastic task model, which makes use of available capacity by altering the task periods. By defining the maximum period, they suggested a reduced level of service for each low criticality task. Lipari and Buttazzo [14] utilized a reservation model to examine EDF in MCSs wherein sufficient budget is reserved for the pessimistic and certified WCETs of high criticality tasks. Some low criticality tasks can be executed in the slack spared by highly critical tasks using an efficient reclamation mechanism because the high criticality tasks are executed in the low mode with optimistic WCETs. Further, these high criticality tasks are executed as early as possible by tuning their deadlines to maximize the reclaimed capacity. Our proposed system is loosely based on this approach .

The IMC model is used by Burns and Baruah [3], and it helps reduce the execution budget for low criticality tasks when the system switches to the high criticality mode. They extended the adaptive mixed criticality (AMC) approach to test the schedulability of an IMC task set under fixed-priority scheduling. Via comparison, the schedulability under EDF with virtual deadlines (EDF-VD) outperforms AMC. Furthermore, Baruah et al. [4] analyzed the schedulability of an IMC model under MC-fluid scheduling. However, in practical scenarios, it suffers from a substantial scheduling overhead caused by the frequent context switching, significantly affecting the scheduling performance of the system. The initial attempt to analyze the schedulability and performance of the IMC model under EDF-VD scheduling was reported by Liu et al. [5]. This approach is similar to that of the earlier mixed criticality model. However, unlike the earlier model, this approach does not discard low criticality tasks after switching to the high criticality mode. Instead, they are scheduled at reduced execution times. If a highly critical task is executed because of its low WCET without signaling completion, then the high criticality mode is activated. After the switch time instant, high criticality tasks are scheduled using their high criticality WCET whereas low criticality tasks are scheduled using a reduced execution time. However, this approach suffers from limitations such as poor performance in heavy workloads and no guarantee of executing low criticality tasks beyond its high WCET in the high mode if resources are available. As previously mentioned, the IMC model allows controlled performance reduction for low criticality tasks in high mode due to shorter WCET.

The following papers demonstrates recent trends in the IMC framework. The study conducted by Jiang et al. [9] proposed HIART-MCS introducing a novel hardware processor enabling task approximation along with an intermediate system mode for running less critical tasks with reduced precision. They developed a theoretical model and schedulability analysis to ensure system timing and optimize mode switching. HIART-MCS was the first practical framework for imprecise MCSs. Jiang et al. [15] presented a novel IMC framework that mitigated computation errors caused by imprecise computation, achieved real-time performance near that of a conventional MCS, enhanced system-level throughput, and provided flexibility for run-time configuration. Zhang et al. [16] studied the energy minimization problem of non-preemptive dynamic priority scheduling and developed schedulability tests for non-preemptive earliest deadline first with virtual deadline (NP-EDFVD) for IMC tasks. They introduced the uniform single-speed (USS) algorithm based on the schedulability tests of NP-EDFVD. The USS algorithm aimed to reduce energy consumption while safeguarding IMC correctness and achieved an average energy savings of 25.62%. In a study conducted by Zhang et al. [17], they explored the partitioned scheduling of an IMC taskset on a uniform multiprocessor platform by employing EDF-VD as the uniprocessor task scheduling algorithm. They primarily aimed to address the optimization problem of identifying a feasible task-to-processor assignment while optimizing the processor speed in low mode in order to minimize the system's average energy consumption in low mode. Zhang [18] introduces an energy-aware mixed criticality real-time scheduling approach for the IMC task model. Their work addresses the energy minimization problem by using dynamic voltage and frequency scaling (DVFS). The proposed energy-aware IMC scheduling algorithm (EA-IMC) efficiently schedules tasks with the energy-efficient speed of S^{LO} in low mode and the maximum processor speed S^{max} in high mode. Experimental results showed an average 24.55% energy reduction. Zhang et al. [19] introduced a criticality-aware EDF (CA-EDF) scheduling algorithm designed to enhance schedulability by delaying low criticality task execution. Their work established schedulability conditions for CA-EDF using the Demand Bound Function. Experimental results demonstrated about 13.10% improvement in the schedulability ratio. Zhang and Chen [20] introduced the energy-efficient allocating algorithm (EEAA), a partitioning algorithm driven by genetic algorithms. They investigated a range of parameter combinations to optimize task-to-processor assignments and minimize energy consumption. Their experiments demonstrated a substantial energy savings of 12.56% achieved by EEAA. Another noteworthy trend is the adoption of precise mixed criticality scheduling [21] where all tasks receive full execution budget. Inspired by MC-Fluid and using dynamic voltage and frequency scaling (DVFS), it minimizes processor speed under normal conditions and restores full speed during task overruns (mode switch). The precise MC scheduling is extended to constrained-deadline tasks using demand-based schedulability analysis, allowing flexible virtual-deadline settings [22].

Besides uniprocessor systems, significant research has also been conducted in MCSs in a multicore environment. Research related to QoS in the multicore paradigm was extensively studied under MCSs. Chai et al. [23] provided a comprehensive survey of the recent research on MCSs. Their work covers various aspects of MCSs, including the system architecture, scheduling algorithms, resource allocation, fault tolerance, and QoS provisioning. Further, they discuss the challenges and opportunities in the design and implementation of MCSs, and they provide insights into future research directions. Another study by Pathan [24] proposes a federated-scheduling algorithm for parallel mixed criticality tasks on multiple processors that improves the schedulability and QoS of the system. The proposed algorithm employs efficient task partitioning technique to maximize system utilization and reduce the interference between tasks of different criticality levels. Roy et al. [25] proposed a scheduling algorithm called SLAQA, which considers the quality level of tasks in a task graph and the heterogeneity of the distributed system to optimize scheduling. SLAQA assigns high-quality tasks to powerful processing nodes and low-quality tasks to less powerful nodes, minimizing the execution time while meeting quality requirements, which makes it useful for high-performance applications such as video processing and multimedia.

The motivation behind the proposed approach is to address the challenges posed by overloaded scenarios in real-time MCSs. As heavy workloads might result in missed deadlines or even system breakdowns, it is challenging even in low mode to guarantee the schedulability of high criticality tasks along with the partial or complete execution of low criticality tasks. The significance of this research work lies in the development of an effective MCS scheduling strategy that ensures the successful execution of high critical tasks while maintaining system stability and quality by ensuring the execution of low criticality tasks in both low and high modes even in heavy workloads. In this research work, we adopted a resource reservation technique with reward-based scheduling framework. To the best of our knowledge, this is an initial attempt in MCSs to address heavy workload scenarios by utilizing a resource reservation based approach within the paradigm of reward-based scheduling in a single core environment.

III. SYSTEM MODEL

We utilized a resource reservation-based approach for scheduling high criticality tasks independent from the behavior of low criticality tasks. Low criticality tasks in the systems are characterized as soft real-time tasks that allow imprecise computation. Therefore, reward-based scheduling framework was used to schedule them. We adopted a uniprocessor architecture to schedule reward-based mixed criticality workload.

A. TASK MODEL

In our system, we considered n independent periodic and preemptive tasks set $T = (t_1, t_2, t_3, ..t_n)$ scheduled in a uniprocessor system. A task in the system is represented by t_i which refers to i^{th} task in a tasks set T. However, a specific job of task is represented by t_{ij} which refers to j^{th} job of the i^{th} task. MCS tasks are scheduled in the low criticality mode at first, and then, they are switched to the high criticality mode when a highly critical task does not signal completion within its low-mode WCET. Each task t_i in the MCS task set T is specified by a 4-tupled parameter $t_i = (P_i, C_i^{LO}, C_i^{HI}, L)$ as

- $P_i \in R^+$ is the task's period, defined as the shortest interval between successive jobs. (The task's deadline D_i and period P_i are considered equal, and $D_i = P_i$).
- $C_i^{LO} \in R^+$ represents the optimistic WCET of a task.
- $C_i^{HI} \in R^+$ represents the pessimistic WCET of a task.
- $L \in (LO, HI)$ represents the criticality level, with LO and HI indicating low criticality and high criticality of a task, respectively.

Reward-based scheduling is only concerned with the execution of low criticality tasks in the system which is composed of the mandatory part execution e_i^m, and the optional part execution e_i^o. Execution of the mandatory part delivers the minimum acceptable output, whereas the optional part enhances the quality of the output. A reward function $R_i(x_{ij})$ is used to measure the quality of a system, where x_{ij} denotes the optional part of the j^{th} job of the i^{th} task that is being executed by the CPU. The reward function $R_i(x_{ij})$ of a task t_i is given by (1).

$$R_i(x_{ij}) = w_i \cdot f_i(x_{ij}) \tag{1}$$

where x_{ij} is the amount of CPU time allocated to the execution of optional part e_i^o and w denotes the weight of a low critical task which allows us to distinguish between the importance of optional parts of different low criticality tasks. The function f_i is considered to be a continuously differentiable, non-decreasing, and linear function over non-negative real numbers [6] which means that the benefit to the overall system increases uniformly during the execution of optional parts. The overall cumulative reward R_i^{cum} obtained during a hyper period H for a low critical task t_i is determined using (2).

$$R_i^{cum} = \sum_{j=1}^{H/P_i} R_i(x_{ij}) \tag{2}$$

Consider two low criticality tasks with $P_1 = 6$, $e_1^m = 2$, $e_1^o = 2$, $P_2 = 12$, $e_2^m = 3$, $e_2^o = 3$, and one high criticality task with $P_3 = 8$, $C_3^{LO} = 2$, $C_3^{HI} = 4$. Moreover, lets

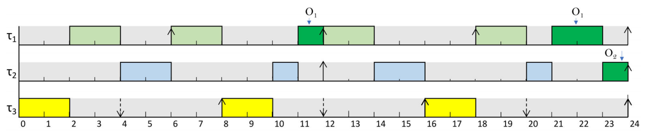


FIGURE 1. **Schedule of a given task set.**

assume the weight w_1 of task t_1 to be 5 and the weight w_2 of task t_2 to be 3 which highlights the importance of t_1. The schedule produced by the EDF algorithm of the given task set is shown in Fig. 1.

As evident from Fig. 1, task t_3 which is classified as a high critical task, consistently achieves successful execution in each invocation. Conversely, tasks t_1 and t_2, categorized as low critical tasks, face challenges in securing complete execution within the available resources. However, the execution of low critical tasks is ensured by adopting the reward-based scheduling framework. In this context, the successful execution of low critical tasks pertains to the successful execution of their mandatory parts, while the execution of their optional parts contribute to enhancing the output quality. Fig. 1 illustrates this scenario, showcasing the accomplished execution of mandatory parts of both tasks t_1 and t_2 during each invocation, coupled with the additional benefit of executing their optional parts as denoted by O_i, resulting in a reward. The optional part of task t_1 receives no CPU time in its initial invocation due to the absence of an available time window. However, during the first invocation of task t_2 and the second invocation of task t_1, a one-unit time window becomes available, allowing execution of the optional part of task t_1 due to its higher weight. Likewise, during the fourth invocation of task t_1 and the second invocation of task t_2, a three-unit time window is available, facilitating the complete execution of the optional part of task t_1 due to its higher weight. Subsequently, execution of one unit of the optional part of task t_2 is also carried out. The reward accrued from the given task set can be calculated by considering the CPU time consumed by optional parts of jobs for low criticality tasks. Let the linear reward functions of t_1 and t_2 be given by $f_1(t_1) = 5 \cdot x_{1j}$ and $f_2(t_2) = 3 \cdot x_{2j}$. Fig. 1 reveals that task t_1 is invoked 4 times within a 24-unit hyperperiod, as task t_1 has a period of 6 units. In its initial invocation, the optional part of task t_1 does not receive any execution (i.e., $e_i^o = 0$ units), and hence $5 * 0$. Subsequently, during the second, third, and fourth invocations, the optional part of task t_1 receives 1 unit, 0 units, and 2 units of execution, respectively. Therefore, the total reward for a low criticality task t_1 over a hyperperiod of 24 is given by $\sum_{j=1}^{4} R_1(x_{1j})$ which is calculated as $(5*0)+(5*1)+(5*0)+(5*2) = 15$. Similarly, the total reward for the low criticality task t_2 over the hyperperiod will be $\sum_{j=1}^{2} R_2(x_{2j})$ which elaborates into $(3*0) + (3*1) = 3$. This yields the total cumulative reward for the low critical tasks as $15 + 3 = 18$.

In order to more clearly see the reward accrued for a task during a hyperperiod, we compute the reward in percentage

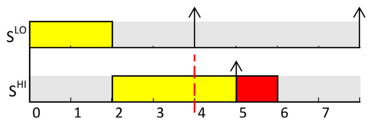


FIGURE 2. Schedulability of tasks using the EDF algorithm.

(RP_i) using (3), whereas the total reward in percentage (TRP) accrued for all tasks in a particular task set can be calculated using (4).

$$RP_i = \frac{\sum_{i=1}^{H/P_i} x_i}{e_i^o \cdot H} \cdot 100 \tag{3}$$

$$TRP = \frac{RP_i + RP_{i+1} +, \cdots + RP_n}{n} \cdot 100 \tag{4}$$

We implemented a server-based technique in our system where a task or group of tasks is assigned to a server. To schedule these servers efficiently, we used a novel approach to the EDF algorithm, i.e., EDF with tuned deadline (EDF-TD) to schedule these servers. While EDF-VD offers benefits in terms of handling varying execution times, its complexity and overhead makes it less suitable for the specific requirements of the proposed system because the proposed system seeks an approach with optimized efficiency and lower complexity. In the proposed system, we are only interested in tuning the deadline of high servers. Consequently, the additional complexity and increased computational overhead of EDF-VD for the calculation of virtual deadlines have an impact on the overall system performance. Therefore, to effectively tune the deadline of a high server, we utilize the equation in (5) to tune D_t as

$$D_t = \frac{C_i^{LO}}{C_i^{HI}/P_i} \tag{5}$$

The EDF-TD scheduler is used because the bandwidth assigned to a high critical server is highly pessimistic. Therefore, when scheduled close to the deadline, it consumes more bandwidth than required. Therefore, we used a modified EDF in which the deadline of a high server was tuned to schedule jobs of high criticality tasks as early as possible. Consider two tasks with $t_1 = (4, 2, 0, LO)$ and $t_2 = (5, 2, 3.5, HI)$ and their schedulability with EDF algorithm is demonstrated in the Fig. 2. Due to a later deadline, task t_2 is scheduled close to the deadline which leaves only a one unit window. This available window is not feasible if the system switches to high mode at time 4 thereby highlighting the significance of an early deadline.

The early execution of the high server also spares some unused bandwidth that is reclaimed and utilized for the execution of a low server. In other algorithms, it is difficult to achieve such bandwidth reclamation. For example, in the least laxity first algorithm, the tasks are scheduled according to their laxity, which is the difference between the deadline and the time remaining to complete the task. Tasks with higher minimum laxity were assigned higher priorities and executed first; this refers to the possibility that a task with a shorter execution time and a later deadline may be scheduled before tasks with longer execution times and earlier deadlines. Therefore, it does not provide an efficient schedulability of tasks. In the maximum urgency first (MUF) algorithm, priorities are assigned to tasks based on the maximum urgency of tasks. The maximum urgency is the ratio of the remaining time to the deadline. The major drawback of MUF is that it is a nonpreemptive algorithm. The modified maximum urgency first (MMUF) algorithm is a modification of the MUF algorithm. The MMUF algorithm extends MUF by adding preemption threshold to prevent the starvation of lower priority tasks. The preemption threshold specifies the minimum urgency level that a task must reach before it can preempt the task currently being executed. However, after each instant of execution, the urgency level of each task is recalculated, and the task with the highest urgency level is selected for execution by the scheduler. This dynamic nature of MMUF introduces some overhead because the priorities need to be recalculated frequently, which affects system performance.

B. SYSTEM BEHAVIOR

The system schedules all tasks of MCS in a uniprocessor environment. The system, also known as a dual-criticality system, has two operational modes that are recognized as the low criticality and high criticality modes. The system contains two types of tasks based on their criticality levels: low and high critical tasks. Each task behaves differently in each mode. All highly critical tasks are executed with their optimistic WCETs, or C_i^{LO}, when the system initiates task execution in the low mode. In the improbable case, the system is alerted to mode transition using a high critical task when it fails to complete execution because of its low WCET C_i^{LO}. Upon changing the mode, all high critical tasks C_i^{LO} are switched to C_i^{HI} and continue to execute until they are completed successfully. Afterwards, the system operates in the high critical mode.

By implementing a resource reservation technique in a reward-based scheduling framework, our main objective was ensuring the effective schedulability of low critical tasks in overloaded settings without compromising the execution of highly critical tasks in either mode. To schedule highly critical tasks in low- and high-critical modes, a separate server was assigned to each highly critical task with a bandwidth corresponding to the high-mode utilization of all highly critical tasks. This indicates that highly critical tasks are independent of the behavior of low criticality tasks because they have sufficient bandwidth to schedule C_i^{LO} and C_i^{HI} at low and high-critical modes, respectively. Excess Bandwidth from high critical task servers are allocated to a server or servers for scheduling low criticality tasks. It follows that the the allocated bandwidth will be sufficient for the server to schedule all low criticality tasks if their bandwidth is greater than or equal to the utilization of low critical tasks they are serving. However, if the server bandwidth is less than the utilization of low critical tasks, then it is difficult to schedule all low critical tasks in the server with a given bandwidth.

TABLE 1. Reward-based mixed criticality task set.

T	C_i^{LO} Mandatory	C_i^{LO} Optional	C_i^{HI}	P_i	L	U_i^{LO}	U_i^{HI}
t_1	2	2	–	6	LO	0.67	0.67
t_2	3	–	4	8	HI	0.38	0.5

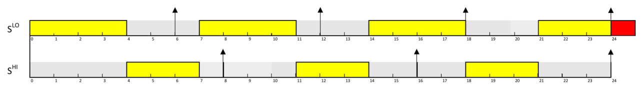


FIGURE 3. Schedule of the task set without reward-based scheduling.

In the low mode, highly critical tasks rarely take longer than their C_i^{LO}; the server of the low critical tasks uses this extra bandwidth from high server to efficiently schedule low critical tasks at least for their mandatory execution. In addition, each low critical task additionally executes beyond its mandatory part to enhance the quality of imprecise outcomes when the recovered bandwidth is sufficient. The schedulability of highly critical tasks is not disturbed by mode transition or by being in a high mode because the server for low critical tasks executes only in the bandwidth spared by the server of the high critical tasks. However, the schedulability of low critical tasks is also affected owing to the pessimistic C_i^{HI} of a highly critical task in the highly critical mode. In contrast to many earlier techniques that fully discard all low critical tasks in the high mode, our proposed system attempts to execute as many low critical tasks as possible within the the available bandwidth.

C. RESEARCH MOTIVATION

The feasibility problem for a periodic workload of a reward-based MCS can be solved by simultaneously satisfying two key factors: guaranteed schedulability of highly critical tasks with C_i^{LO} and low critical tasks with e_i^m with an additional effort to enhance the system performance by executing some or complete e_i^o.

Our proposed system efficiently satisfies these conditions, particularly in overloaded situations. As an example, consider the periodic task set $T = (t_1, t_2)$ listed in Table 1.

The task set summarized in Table 1 makes it apparent that the total utilization of task set is greater than 1 i.e., $\sum_{i=1}^{n} \frac{C_i^{LO}}{P_i} = 1.04$, therefore, it cannot be scheduled by using any fixed or dynamic priority scheduling algorithm, as depicted in Fig. 3.

The above task set can be scheduled with an acceptable result and some additional reward, as shown in Fig. 4, using a reward-based scheduling approach, which trades off the precision of low critical tasks with their timeliness. By applying (4), a total cumulative reward of 87.5% is produced for low critical tasks over a hyperperiod of 24.

IV. PROPOSED MIXED CRITICALITY REWARD-BASED SCHEDULING

In an insufficient number of resources for ensuring worst-case outcomes for soft real-time tasks, the reward-based

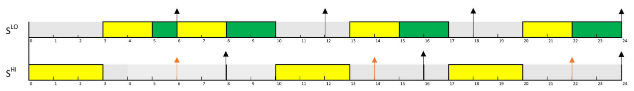


FIGURE 4. Schedule of the task set with reward-based scheduling.

scheduling framework compromises accuracy in the favor of timeliness. The feasibility of such a workload can be satisfied by fulfilling only the following conditions for a task between its arrival time and deadline: Each low critical task either completely executes or receives at least mandatory execution; each highly critical task receives sufficient execution such that in low mode, its C_i^{LO} is executed successfully and when the system switches to the high mode, its C_i^{HI} execution is ensured.

A. SERVER SPECIFICATION

In this study, a server-based technique is proposed for scheduling tasks. In this approach, a task or a group of tasks is assigned to the server S_i. We used two types of servers, low server S_i^{LO} and high server S_i^{HI} for the schedulability of low- and high-criticality tasks, respectively. Specifications of the low server $S_i^{LO} = (Q_i^{LO}, T_i)$, where Q_i^{LO} denotes the budget of the server used to schedule low critical tasks, and T_i denotes the time period of the server after which the server replenishes its budget. Likewise, the specifications of a high server $S_i^{HI} = (Q_i^{LO}, Q_i^{HI}, T_i)$, where Q_i^{LO} denotes the low budget of the server used to schedule the high- critical tasks in the low mode, whereas Q_i^{HI} denotes the high budget of the server used to schedule highly critical tasks in the high mode where T_i denotes the period of the server after which the server replenishes its budget. The tuple $(Q_i^{LO}, Q_i^{HI}, T_i)$ specifies the parameters for S_i^{HI}, in which $Q_i^{LO} = C_i^{LO}$, $Q_i^{HI} = C_i^{HI}$, $T_i = P_i$, whereas low sever S_i^{LO} may be described by only two components, (Q_i^{LO}, T_i), where $Q_i^{LO} = B^{LO} \cdot T_i$ and $T_i = P_i$ and B^{LO} refers to the bandwidth of the low server. These servers act as a dedicated CPU for schedulabilty of these tasks. The proposed server-based technique involves dividing the available resource of a uniprocessor into distinct servers, each with allocated resources and capacity similar to a dedicated CPU. The criticality levels of tasks determine the server they are assigned to, and tasks are executed within their allocated resources during predefined time slots. This approach ensures the effective execution of tasks while ensuring critical tasks receive the required resources in any scenario. Each server is assigned a portion of the bandwidth, such that the total bandwidth of all these servers is equal to 1.

In our proposed system, each highly critical task is assigned a separate server S_i^{HI}. The bandwidth of this high server B_i^{HI} corresponds to the high-mode utilization of the highly critical task. The allocation of such a high fraction of resources to S_i^{HI} ensures the schedulability of highly critical tasks at both low and high modes.

$$B_i^{HI} = U_{t_i^{HI}}^{HI} \tag{6}$$

The notation t_i^{HI} in (5) represents the i^{th} high critical task, while U^{HI} signifies the high mode utilization. As a whole, it refers to the high mode utilization of i^{th} high critical task. Equation (6) indicates adequate bandwidth allocation to a high server, and as a result, the schedulability of a highly critical task is ensured in both modes and becomes independent of the behaviors of low criticality tasks. Now, to schedule low critical tasks, the leftover bandwidth (B^{LO}) from the high servers is allocated to the low servers S_i^{LO}, as shown in (7).

$$B^{LO} = 1 - \sum_{t_i \in T} B_i^{HI} \tag{7}$$

The server can be in any of the following states: The server is in the *inactive state* if no tasks are available for execution at time t. When a task arrives at time t, the server switches to an *active state*. When a certain task begins execution and consumes the capacity of a server by using (8), the server shifts from an *active state* to a *running state*. Equation (8) shows the consumption rate of a server's budget at a certain time.

$$dQ_i = -U^{act} dt \tag{8}$$

The bandwidth of the currently active servers is shown by the variable U^{act}. At time t, if all servers are active, then the value of U^{act} will be 1, meaning that the server capacity is used as the unit rate. If some of the servers are active, the capacity is consumed at a slower rate (i.e., $U^{act} < 1$), allowing for the recovery of the free bandwidth of the system. The server returns to an active state from a running state when it is preempted by another server with an earlier deadline. Once all the server tasks have been successfully executed, it shifts to an inactive state.

From (7), we can conclude that all low critical tasks in the system can be scheduled successfully if utilization of low critical tasks is less than or equal to B^{LO}. However, all low critical tasks cannot be scheduled successfully if utilization of low critical tasks exceeds B^{LO}. To overcome this problem, the proposed technique schedules the mandatory execution of all low critical tasks. Furthermore, it ensures the maximum possible execution of the optional parts of low critical tasks for the maximum reward. As discussed previously, the execution of a highly critical tasks beyond C_i^{LO} are rare. Therefore, we reclaimed the excess bandwidth from a high server using an algorithm called greedy reclamation of the unused bandwidth (GRUB) [26]. The reclaimed bandwidth is then allocated to the low servers. By using variable U^{act}, the reclaiming mechanism of the GRUB algorithm is used. This variable determines the current CPU utilization rate by a task in execution. When a highly critical job within a high server executes for e_{ij} units and $e_{ij} < C_i^{HI}$, then the recovered bandwidth can be calculated using (9).

$$U^{rec} = \frac{C_i^{HI} - e_{ij}}{P_i} \tag{9}$$

As mentioned previously, each high critical task is assigned a separate server. For assigning low criticality tasks to low server, three different strategies are proposed in this research. In the first strategy, all low critical tasks are allocated to a single server. In the second strategy, each low criticality task is allocated to a single dedicated server, whose cumulative bandwidth should not exceed maximum B^{LO}. In the last strategy, all low critical tasks are assigned to two separate servers. Mandatory parts of these low critical tasks are scheduled on one server, and their optional parts, on another server. The sum of the bandwidths of both servers must not exceed B^{LO}. Consequently, the effectiveness of these strategies are evaluated and compared.

B. REWARD-BASED RESOURCE RESERVATION ALGORITHM

The following novel reward-based algorithm (Reward-Based Resource Reservation (RBRR) uses the discussed server mechanism to schedule tasks of an MCS in a reward-based scheduling environment.

In Algorithm 1, line 1 initializes the active servers to zero. Line 2 demonstrates criticality mode of the system. In lines 3 and 4, all active tasks are assigned to the servers using the function given in lines 8–16. Highly critical tasks were assigned to dedicated high servers with a tuned deadline, whereas low critical tasks are assigned to a low server. The bandwidth of these servers is summed and the variable U^{act} is updated accordingly. Line 6 schedules all active servers using EDF-TD. In line 7, the server with earliest deadline is shifted to the running state and its tasks begin to execute using the function in lines 17–30. In line 18, if a server contains more than one task, then these tasks are scheduled based on their deadlines. In line 20, a job with an earlier deadline is executed. Line 21 shows the consumption rate of the capacity of the server. If a highly critical task does not indicate the completion and the capacity of the server is fully consumed, then the server's tuned deadline is shifted to the actual deadline and the criticality mode of the system is switched to the high mode in lines 22–27. However, in lines 28–29, when the capacity of a low server is exhausted, its deadline is postponed, its capacity is replenished, and the function is returned.

After analyzing the time complexities of the main loops in the pseudocode, we obtain an approximate upper bound on the overall time complexity. The first loop *While activeTasks* $=$ *true* iterates over all tasks in the task set, and the second loop *While* $U^{act} \neq 0$ executes until all tasks have been scheduled and executed. The time complexity of these loops depends on the number of tasks and the amount of time it takes to execute operations within loops.

Assuming that the EDF algorithm used to schedule tasks has a time complexity of $O(nlogn)$, where n represents the number of tasks and that the execution of a job with the earliest deadline takes constant time, the time complexity of the second loop can be approximated using $O(nlogn)$.

The time complexity of the first loop depends on operations performed within the loop. Assuming that the operations within the loop take constant time, the time complexity can be approximated using $O(n)$. Therefore, the overall

Algorithm 1 Reward-Based Resource Reservation Algorithm (RBRR)

```
Input  : A reward-based mixed criticality task set $T$
Output: Schedulability of the given task set $T$
1  $U^{act} \leftarrow 0$
2  $L \leftarrow LO$
3  while $activeTasks = true$ do
4      $Assign\_Server(t_{ij})$
5  while $U^{act} \neq 0$ do
6      $Schedule\ Active\ Servers\ using\ EDF - TD$
7      $Server\_Execution()$
   /* Function for Assigning Server */
8  Function $Assign\_Server$ ($t_{ij}$):
9      if $t_{ij} \in T_i^{HI}$ then
10         $S_i^{HI} = active$
11         $d_i^{HI} = floor(C_i^{LO}/(C_i^{HI}/P))$
12         $U^{act} = U^{act} + B_i^{HI}$
13     else
14         $S_i^{LO} = active$
15         $U^{act} = U^{act} + B^{LO}$
16     return
   /* Function for Server Execution */
17 Function $Server\_Execution$ ():
18     $Schedule\ tasks\ inside\ server\ using\ EDF$
19     while $server\_capacity \neq 0\ or\ tasks\ available$ do
20         $Execute\ job\ with\ earliest\ deadline$
21         $dQ_i = -U^{act}dt$
22         if $server\_capacity = 0$ then
23             if $t_{ij} \in T_i^{HI}$ then
24                 if $L = LO\ and\ d_i \neq P$ then
25                     $d_i = P$
26                     $L = HI$
27             else
28                 $D_i = d_i + P$
29                 $Q_i\ is\ replenished$
30     return
```

time complexity of a given pseudocode is approximated as $O(nlogn)$.

V. SCHEDULABILITY TESTS

We employed a series of schedulability tests to ascertain the schedulability of tasks in different modes and on various servers. These tests assess whether a given task set can be executed within the specified constraints, ensuring that critical tasks meet their deadlines while effectively utilizing the available system resources.

A. SCHEDULABILITY TEST IN LOW MODE

The schedulability of all tasks executed in low mode is assessed in this test. The utilization of all high critical tasks t_i^{HI} in low mode $U_{t_i^{HI}}^{LO}$ and the utilization of the mandatory parts e_i^m of all low critical tasks is calculated and the following condition is tested:

$$\sum_{i=1}^{n}(U_{t_i^{HI}}^{LO} + \frac{e_i^m}{P}) <= 1$$

If this condition holds for all tasks executed in low mode, the task set is considered schedulable in low mode.

Proof: A necessary and sufficient schedulability condition for an EDF system [27] is that the total utilization of the task set must be less than or equal to 1, $\sum_{i=1}^{n}(U_i) <= 1$. The feasibility of the proposed system in low mode depends on the successful execution of all high critical tasks t_i^{HI} as well as at least the mandatory parts e_i^m of all low critical tasks. Therefore, the total utilization of t_i^{HI} and e_i^m are equal to $\sum_{i=1}^{n}(U_{t_i^{HI}}^{LO})$ and $\sum_{i=1}^{n}(e_i^m/P)$, respectively. By combining, we get the schedulability condition of the proposed system in low mode.

B. SCHEDULABILITY TEST IN HIGH MODE

In this test, we focus on the high critical tasks executed in high mode. The total bandwidth B^{HI} used by these high critical tasks in high mode is calculated and the following condition is tested:

$$B^{HI} <= 1$$

If this condition holds for all high critical tasks executed in high mode, the task set is considered schedulable in high mode.

Proof: The feasibility of the proposed system in high mode depends on the successful execution of all high critical tasks t_i^{HI} for their high mode WCET C_i^{HI}. From (6), the bandwidth allocated to all t_i^{HI} is equal to $\sum_{i=1}^{n}(U_{t_i^{HI}}^{LO})$. Therefore, the schedulability condition in high mode holds if the available bandwidth is less than or equal to 1.

C. SCHEDULABILITY TEST OF HIGH SERVER

For each high critical task t_i^{HI} executed on a high server, their utilizations in low mode $U_{t_i^{HI}}^{LO}$ and high mode $U_{t_i^{HI}}^{HI}$ are calculated and the following conditions are tested:

$$\sum_{i=1}^{n} U_{t_i^{HI}}^{LO} <= B^{HI}$$

$$\sum_{i=1}^{n} U_{t_i^{HI}}^{HI} <= B^{HI}$$

If these conditions are satisfied for all high critical tasks executed on high servers, the high server is considered schedulable.

D. SCHEDULABILITY TEST OF LOW SERVER

For a low server, the utilization of the mandatory parts e_i^m of all low critical tasks is calculated. This calculated value is then compared with the available bandwidth B^{LO} of low

TABLE 2. Reward-based mixed criticality task set.

T	C_i^{LO} Mandatory	C_i^{LO} Optional	C_i^{HI}	P_i	L	U_i^{LO}	U_i^{HI}
t_1	2	2	–	6	LO	0.67	0.67
t_2	3	3	–	12	LO	0.5	0.5
t_3	2	–	6	8	HI	0.25	0.75

server along with the reclaimed bandwidth U^{rec} as follows:

$$\sum_{i=1}^{n} \frac{e_i^m}{P} <= B^{LO} + U^{rec}$$

If this condition holds for a low server, it is considered schedulable.

Proof: As the B^{HI} is equal to the total utilization of high critical tasks in high mode, the B^{HI} is not fully utilized by high critical tasks in low mode and the remaining bandwidth is allocated to low server. Therefore, the low server is deemed schedulable if the total utilization of low critical tasks assigned to it is less than or equal to its allocated bandwidth B^{HI} along with the additional bandwidth reclaimed U^{rec}.

VI. SCHEDULING ANALYSIS

We analyzed the schedulability of our proposed system in both low and high modes for a reward-based mixed criticality workload for three alternate strategies. We employed a task set of three tasks denoted by t_1, t_2, and t_3, where t_1 and t_2 represent low critical tasks and t_3 represents a highly critical task. Table 2 presents the parameters of the given task set.

A. SCHEDULABILITY IN THE LOW MODE

As mentioned previously, we used the following three strategies to schedule a given task set in the low mode: a single server with bandwidth B^{LO} is assigned to all low critical tasks, a single dedicated server with a cumulative bandwidth less than or equal to B^{LO} is assigned to each low critical task, and each low critical task is assigned to two servers i.e., a mandatory server and an optional server, both having a total bandwidth less than or equal to B^{LO}. High critical task t_3^{HI} is assigned to a dedicated server with $B^{HI} = 0.5$, and therefore, all low critical tasks are scheduled in the remaining available bandwidth of 0.5. Note that the color schemes to depict the analysis of task set schedulability employed in Figures 5 to 10 are as follows: light green indicates the mandatory part of task t_1, sky blue represents the mandatory part of task t_2, yellow represents task t_3, and dark green is utilized for both tasks t_1 and t_2 to denote their optional parts. However, the red color represents the amount of execution time missed by a task.

1) SINGLE SERVER FOR ALL LOW CRITICAL TASKS

In this strategy, we use two servers with $B^{LO} = 0.5$ and $B^{HI} = 0.5$ because a single server is allocated to all low critical tasks. The given task set does not appear schedulable because utilization of low criticality tasks in low mode $U^{LO} > B^{LO}$. Budget Q_i and period T_i for S_i^{LO} and S_i^{HI} will

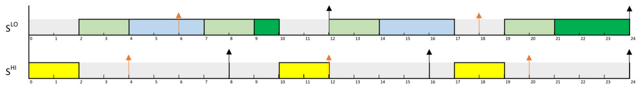


FIGURE 5. Schedulability with a single server for all low critical tasks.

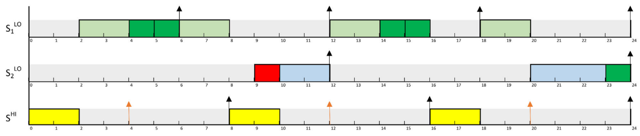


FIGURE 6. Schedulability with dedicated servers for all low critical tasks.

be (6, 12) and (2, 4, 8), respectively. The proposed schedule is shown in Figure 5.

Because of the availability of all tasks at $t = 0$, both servers are in an active state. The capacity of server S_i^{LO} is 6 and that of server S_i^{HI} is 2. Server S_i^{HI} begins executing t_3^{HI} because of an earlier deadline. Because all servers are active, $U^{act} = S_i^{LO} + S_i^{HI} = 1$, which is the servers' capacity consumed per unit of rate.

At $t = 2$, t_3 is executed completely, and low critical tasks inside a low server begin to execute at a unit rate because $d_i^{HI} = 4$ and S_i^{HI} is active until $t = 4$. Server S_i^{HI} switches to the inactive state at $t = 4$ and its bandwidth B^{HI} is subtracted from U^{act}. The server S_i^{LO} capacity is consumed at a rate of 0.5.

At the current rate, the server's capacity is consumed at a 0.5 rate while executing tasks for one unit of time using the GRUB algorithm's [26] reclaiming mechanism. Although $U^{LO} > B^{LO}$, the reclaimed bandwidth is what is required to satisfy the schedulability of low critical tasks within the server S_i^{LO}. Thus, all jobs were effectively scheduled by both servers, and it obtained a total reward of 29.2% for the low critical tasks.

2) SINGLE SERVER FOR EACH LOW CRITICAL TASKS

In this strategy, the system contains three servers, S_1^{LO}, S_2^{LO}, and S_i^{HI}, with bandwidths $B_1^{LO} = 0.25$, $B_2^{LO} = 0.25$, and $B^{HI} = 0.5$, respectively. This is because each low critical task is assigned a separate server. $S_1^{LO} = (1.5, 12)$, $S_2^{LO} = (3, 12)$, $S_i^{HI} = (2, 4, 8)$, shows the parameters of the given servers. The proposed algorithm's schedule is shown in Fig. 6.

When the S_2^{LO} budget is completely used, at $t = 10$, it is recharged at $q_2 = 3$ and calculates its deadline as $d_2 = 24$. Subsequently, S_i^{HI} preempts S_2^{LO} to schedule t_3, and when t_3 is executed at $t = 12$, the second job of t_1 misses its deadline. Therefore, the schedule above cannot satisfy the feasibility constraints for a given task set.

3) SEPARATE SERVERS FOR MANDATORY AND OPTIONAL PARTS

A low critical task comprises two parts, and therefore, it is assigned to two servers, S_M^{LO} and S_O^{LO}, to execute its mandatory part e_i^m and its optional part e_i^o, respectively. When

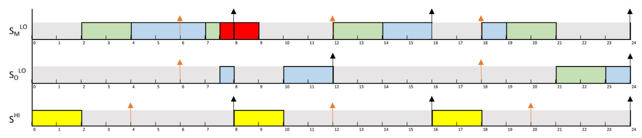


FIGURE 7. Separate servers for the mandatory and optional parts of low critical tasks.

mandatory part e_i^m is completely executed by server S_M^{LO}, only then the server S_O^{LO} is activated to execute optional part e_i^o. Therefore, in this strategy, there are three servers in the system S_i^{HI}, S_M^{LO}, and S_O^{LO} with bandwidths $B^{HI} = 0.5$, $B_M^{LO} = 0.25$, and $B_{LO}^{LO} = 0.25$, respectively. $S_M^{LO} = (2, 8)$, $S_O^{LO} = (3, 12)$, and $S_i^{HI} = (2, 4, 8)$ are the parameters for the given servers. The proposed algorithm's schedule is shown in Fig. 7.

At time 0, only S_M^{LO} and S^{HI} servers are active and the capacity utilization rate of servers is 0.75. Task t_3 begins execution within the S^{HI} server due to the server's earliest deadline and finishes execution at time 2. The mandatory part of task t_1 inside S_M^{LO} starts executing at 0.75 rate because server S^{HI} will still be active until its deadline. At time 4, mandatory part of task t_1 finishes due to which the server S_O^{LO} becomes active and server S^{HI} switches to inactive state, thus changing the capacity utilization rate to 0.5. At time 5, the budget of server S_M^{LO} is exhausted. Consequently, the server S_M^{LO} replenishes its budget and recalculates its deadline to 16. However, due to the earlier deadline, server S_O^{LO} preempts server S_M^{LO} to execute the optional part of task t_1. At time 6, server S_O^{LO} switches to inactive state as there are no pending optional parts to execute. Consequently, server S_M^{LO} remains the only active server within the system, continuing the execution of task t_2 at a rate of 0.25 and finishing mandatory part of task t_2 at time 8, subsequently triggering the activation of server S_O^{LO}. At time 8, a new job of task t_3 is released and the server S^{HI} becomes active. With all servers concurrently active in the system, task t_3 begins execution at a unit rate. Task t_3 completes the execution at $t = 10$, but server S_i^{HI} will remain active until $t = 12$ and therefore, the e_i^m of t_2 is executed at a unit rate inside S_O^{LO} because of its earliest deadline. When the server S_O^{LO} reaches its deadline at $t = 12$, it stops executing e_2^o, recharges its budget $q_1 = 3$, calculates its next deadline $d = 24$, and shifts to an inactive state. At this point, the e_i^m of t_2 is not completed by its deadline and the given schedule is not feasible.

4) SCHEDULABILITY IN HIGH MODE

Schedulability of the task set in the high mode is similar to that in the low mode with only two differences: a high-budget Q_i^{HI} is used by server S_i^{HI}; the actual deadlines of highly critical tasks will be used for server S_i^{HI} instead of shorter deadlines. Fig. 8, Fig. 9, and Fig. 10 show the schedules provided by all three strategies in the high mode.

Fig. 8, Fig. 9, and Fig. 10 show that the proposed system can successfully deliver complete or partial services to low critical tasks while guaranteeing the schedulability of the highly critical tasks in the high mode. We observe that the third strategy, depicted in Fig. 10, performs worse than the other two strategies, as shown in Fig. 8 and Fig. 9.

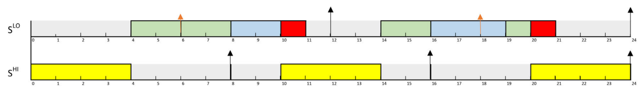


FIGURE 8. Single server for all low critical tasks in the high mode.

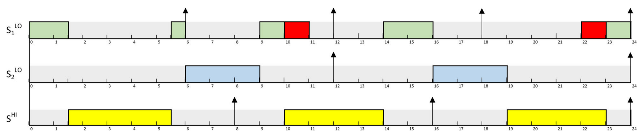


FIGURE 9. Dedicated servers for each low critical task in the high mode.

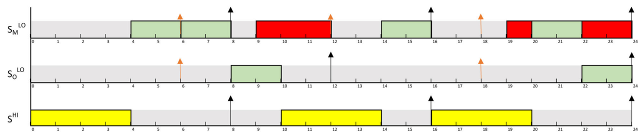


FIGURE 10. Separate servers for the mandatory and optional parts of low critical tasks in the high mode.

VII. EXPERIMENTS AND RESULTS

Reward-based mixed criticality task sets were used as the workload to analyze the effectiveness of the proposed technique. For a given utilization, the effectiveness was studied by comparing all three proposed strategies discussed earlier with studies considering the IMC model, AMC approach [3], MC-Fluid approach [4], utilization-based test [5], and demand-bound function test [5] with respect to schedulability performance. In the experiment, SS, DS, MOS, UTIL, DBF, AMC, and MCF denote the single-server strategy, dedicated server strategy, mandatory and optional server strategies, utilization-based test [5], demand-bound function test [5], Adaptive Mixed Criticality approach [3], and MC-fluid approach [4], respectively. The experimental results show that the SS strategy outperforms all other approaches.

A. TASK SET GENERATION

Task sets of reward-based mixed criticality workloads were produced randomly for the experimental evaluation, and the utilization of the initial task being produced was set to zero. B^{HI}, R^{HI}, R^{LO}, and P^{Max} parameters are adjusted to create a random task set. The parameter B^{HI} specifies the bandwidth assigned to the high servers. The ratio of C_i^{LO} to C_i^{HI} for highly critical tasks is represented as R^{HI}. Similarly, the ratio of e_i^m to e_i^o for low critical tasks is represented by R^{LO}. The maximum task period is denoted as P^{Max}. A dedicated server for high critical tasks captures the probability parameter of a generated task in the system to be a highly critical task because it provides sufficient bandwidth to each highly

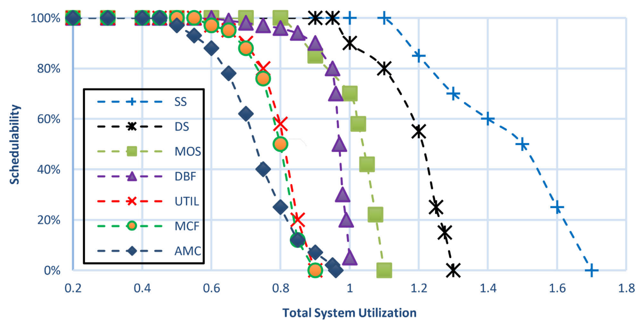


FIGURE 11. Schedulability analysis of system with $B^{HI} = 0.5$.

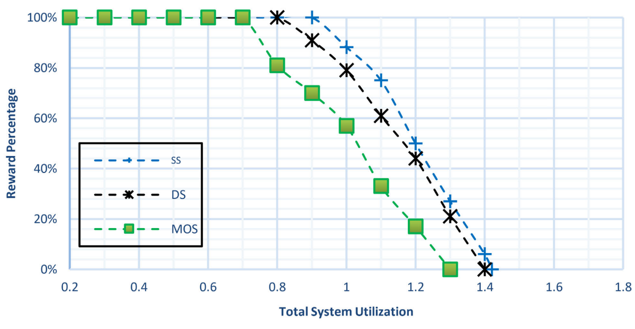


FIGURE 12. Reward percentage of system with $B^{HI} = 0.5$.

critical task to guarantee its schedulability in both modes. The criticality level, computation time, and period of a mixed criticality task characterize the random generation. If $L = LO$, the task being created is a low critical task; if $L = HI$, it is a high critical task. The C_i^{LO} of a low critical task consists of two parts: e_i^m and e_i^o, where $e_i^m = e_i^o \cdot R^{LO}$. Similarly, the computation time for high critical tasks is $C_i^{HI} = C_i^{LO} \cdot R^{HI}$. R^{LO} and R^{HI} are fixed multipliers, ranging from 1 to 1.5. For each mixed criticality task t_i, period P is determined through a uniform random distribution with a range of $1ms$ to P^{Max}.

B. RESULT ANALYSIS

The mixed criticality task set size n ranges from 2 to 6, R^{LO} and R^{HI} range from 1 to 1.5, B^{HI} ranges from 0.1 to 0.9, and P^{Max} is $30ms$. The settings were evaluated experimentally using various simulation parameters. The n parameter represents the number of tasks in the task set, R^{LO} represents e_i^o/e_i^m, R^{HI} represents C_i^{HI}/C_i^{LO}, and for a high critical task, B^{HI} represents their servers' total bandwidth. For the experimental evaluation, there were at least 100 task sets at each point in Fig. 11 and Fig. 12, and at least 1000 task sets for each of the points in Fig. 13 and Fig. 14. The results of the experiments conducted on the behavior of low critical tasks in a given task set using a single server, multiple dedicated servers, and mandatory and optional servers were compared with other IMC approaches.

Percentage of task sets that can be scheduled using all these methods are shown in Fig. 11, with $B^{HI} = 0.5$, $R^{HI} = 1.5$, and target utilizations ranging from 0.1 to 1.8. Target utilizations beyond 1 indicate overloaded scenarios. The SS, DS, and MOS strategies had 100% schedulability for 1.1, 0.9, and 0.8 target utilizations, respectively. Similarly, the DBF, UTIL, MCF, and AMC strategies had 100% schedulability with 0.6, 0.55, 0.55, and 0.45 target utilizations, respectively. The schedulability percentage for all strategies decreases gradually; however, the SS strategy still outperforms all other strategies showing higher schedulabilty. Moreover, the performances of the AMC, MCF, UTIL, and DBF strategies can can be observed only within the maximum utilization of 1. This is because of the lack of schedulability of task sets in overloaded scenarios. The significance of the proposed approach is as follows: observed at a target utilization $U = 1.1$, where the SS strategy schedules 100% of the task sets and DS and MOS strategies schedules 80% and 50% of the task sets whereas DBF, UTIL, MCF, and AMC strategies had zero schedulability. Therefore, the approaches are evaluated, and the results indicate that the SS strategy schedules 20% more task sets than that using the DS strategy, 50% more tasks than that using the MOS strategy, and 100% more tasks than that using the DBF, UTIL, MCF, and AMC strategies for $U = 1.1$. Moreover, when $U = 1.2$, the schedulability of the SS strategy decreases to 84%, whereas those of DS and MOS decrease to 56% and 24%, respectively. The SS strategy has a comparatively 28% and 60% higher schedulability than that of the DS and MOS strategies, respectively. Further, the schedulability of all strategies for mixed criticality task sets decreases with an increase in target utilization, which highlights the effects of increasing utilization on the overall system performance.

Unlike other approaches, the schedulability of the proposed approach accrues a reward, as shown in Fig. 12. The SS, DS, and and MOS strategies yield rewards of 100% for $U = 0.9$, $U = 0.6$, and $U = 0.4$, respectively. The 100% of the reward percentage of a strategy for a given target utilization reveals that the complete execution of all low critical tasks (mandatory parts, as well as optional parts). In terms of accrued rewards, the SS strategy outperforms the other two strategies. The rewards gained by the DS and MOS strategies is 76% and 52%, respectively, for $U = 0.9$. This indicates that the rewards gained by using the SS strategy are 24% and 48% higher than those of the DS and MOS strategies.

Fig. 13 illustrates the influence of the imprecise mixed criticality task sets on the schedulability of all IMC approaches using the weighted acceptance ratio and varying B^{HI} settings. Fig. 13 reveals that the SS strategy outperforms all other strategies with varying parameters of B^{HI}. However, the DS and MOS strategy perform better than the DBF, UTIL, MCF, and AMC strategies, and after $B^{HI} = 0.5$, their difference in performance with DBF becomes negligible. Moreover, the UTIL and MCF strategies exhibited similar trends with different B^{HI} values, whereas the performance of AMC up to $B^{HI} = 0.5$ is considered the worst of all strategies; however, after $B^{HI} = 0.5$, the difference is minimal.

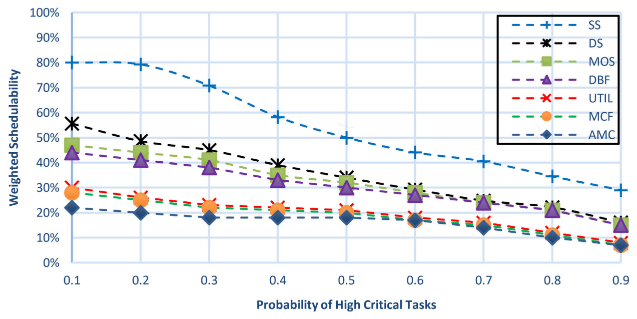


FIGURE 13. High server bandwidth B^{HI} vs. weighted schedulability.

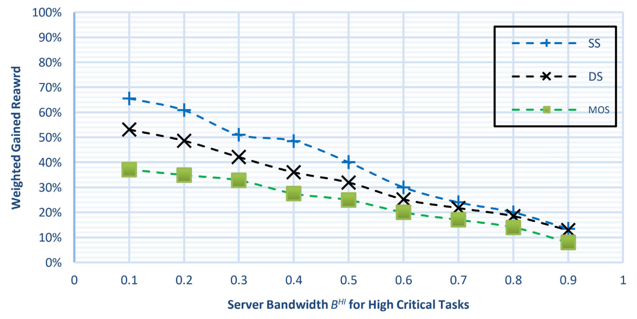


FIGURE 14. High server bandwidth B^{HI} versus weighted gain reward.

The weighted ratio of the accrued reward when scheduling reward-based mixed criticality task sets for various parameters of B^{HI} are shown in Fig. 14. We observe that the SS strategy outperforms the other two in terms of providing services to low critical tasks with different values of B^{HI}. However, when $B^{HI} = 0.6$, the difference in the performance of SS and DS strategies is minimal, whereas the MOS strategy performs the worst of all three proposed strategies.

A 3D graph was converted into a 2D graph by utilizing the weighted schedulability measure [28]. Target utilization set u and the weighted schedulability ratio $Sw(u)$ is evaluated as using (10), where $S(U_i)$ represents the schedulability ratio for target utilization U_i.

$$Sw(u) = \sum_{U_i \in u} U_i \cdot S(U_i) / \sum_{U_i \in u} U_i \tag{10}$$

The weighted schedulability ratio of the gained rewards $Rw(u)$ can be evaluated using (11), where $R(U_i)$ represents the reward ratio for target utilization U_i.

$$Rw(u) = \sum_{U_i \in u} U_i \cdot R(U_i) / \sum_{U_i \in u} U_i \tag{11}$$

Fig. 13 illustrates that weighted schedulability decreases with an increase in B^{HI}. A higher value of B^{HI} indicates more highly critical tasks in the task set than low critical tasks. A higher value of B^{HI} increases the workload utilization to grow, which in turn affects the schedulability and accrued rewards. For higher workload utilization, the SS strategy performed better compared with all other strategies. The dominance of the SS strategy compared with the other strategies in terms of the weighted schedulability ratio and the weight obtained reward ratio is shown in Fig. 13 and Fig. 14, respectively.

In the proposed approach, scalability has been a central consideration during the design and development phases. The comprehensive evaluations conducted, including simulations and analysis, assess the performance of the proposed approach as the scale of the system increases. The results indicate that the resource reservation and reward-based scheduling approach holds promise for real-time systems like autonomous vehicles, medical implants, industrial control systems, and advanced avionics, where high critical tasks such as collision avoidance, pacemakers, flight control, etc. are guaranteed execution while low critical tasks such as in-flight entertainment or cabin temperature control can adapt to available resources or even delayed if resources are limited. These considerations are aimed at accommodating the growing demands and complexities that may be encountered in practical deployments. The proposed approach has the potential to maintain a balance between efficiency and scalability to meet the requirements of real-world applications.

VIII. CONCLUSION

We used a reward-based scheduling framework for real-time MCS in a uniprocessor paradigm and addressed the scheduling problems for low- and high-criticality tasks in both modes under overload conditions. We proposed a resource reservation-based system in which the schedulability of the low critical tasks has no effect on the schedulability of highly critical tasks in both low and high modes. The proposed RRBS algorithm reflects the concepts of dividing the resource CPU into server partitions and each task being scheduled inside the servers with a specified budget and period. Using the server mechanism, we assigned the dedicated server to each highly critical task for guaranteed schedulability in both modes. For scheduling low critical tasks, we applied and analyzed three strategies in the available bandwidth, along with the reclaimed bandwidth spared by the servers of highly critical tasks caused by their optimistic WCET rather than the certified WCET. In the first strategy, all low critical tasks were assigned to a single reservation server. In the second strategy, a single dedicated reservation server was allocated to each low criticality task. In the third strategy, each low criticality task was executed on two different servers, i.e., the mandatory and optional servers. The proposed technique was evaluated using extensive experimentations with other IMC approaches. The experimental evaluation indicated that the proposed technique outperformed other IMC approaches. The proposed technique successfully achieved the schedulability of low critical tasks with no impact on the highly critical tasks' schedulability, even in overloaded scenarios. However, the schedulability of the low critical tasks are affected by certified WCETs of high critical tasks in the high mode. Therefore, the proposed system attempts to schedule the maximum number

TABLE 3. List of equations.

Equation	Meaning
$R_i(x_{ij}) = w_i \cdot f_i(x_{ij})$	A reward function of an optional part of the j^{th} job of the i^{th} low critical task along with its weight/importance denoted as w_i
$R_i^{cum} = \sum_{j=1}^{H/P_i} R_i(x_{ij})$	A cumulative reward function of a low critical task t_i^{LO} during a hyper period H
$RP_i = \frac{\sum_{i=1}^{H/P_i} x_i}{e_i^o \cdot H} \cdot 100$	The reward accrued for a low critical task t_i^{LO} denoted in terms of percentage
$TRP = \frac{RP_i + RP_{i+1} +, \cdots + RP_n}{n} \cdot 100$	The total reward accrued for all low critical tasks denoted in percentage
$D_t = \frac{C_i^{LO}}{C_i^{HI}/P_i}$	Tuning the deadline of a high server
$B_i^{HI} = U_{t_i^{HI}}^{HI}$	Bandwidth allocation to a high server
$B^{LO} = 1 - \sum_{t_i \in T} B_i^{HI}$	Bandwidth allocation to a low server
$dQ_i = -U^{act} dt$	The consumption rate of a server's budget at a certain time
$U^{rec} = \frac{C_i^{HI} - e_{ij}}{P_i}$	Calculation of the spare bandwidth recovered from a high critical task t_i^{HI}
$Sw(u) = \sum_{U_i \in u} U_i \cdot S(U_i) / \sum_{U_i \in u} U_i$	Calculation of the weighted schedulability ratio for the target utilization set u
$Rw(u) = \sum_{U_i \in u} U_i \cdot R(U_i) / \sum_{U_i \in u} U_i$	Calculation of the weighted schedulability ratio of the gained rewards for the target utilization set u

TABLE 4. List of acronyms.

Acronym	Meaning
T	Task set
t_i	i^{th} task
t_{ij}	j^{th} job of i^{th} task
R^+	Non-negative real numbers
P_i	Period of an i^{th} task
D_i	Deadline of an i^{th} task
C_i^{LO}	Low mode WCET of i^{th} task
C_i^{HI}	High mode WCET of i^{th} task
e_i^m	Mandatory execution time of i^{th} low criticality task
e_i^o	Optional execution time of i^{th} low criticality task
H	Hyperperiod: LCM of periods of all the tasks
D^t	Tuned deadline of a high server S^{HI}
B^{HI}	Cumulative bandwidth of high servers S^{HI}
B^{LO}	Cumulative bandwidth of low servers S^{HI}
U^{rec}	Reclaimed bandwidth from high server S^{HI}
U^{act}	The cumulative bandwidth of the active servers in the system
U^{LO}	Utilization of tasks in low mode
U^{HI}	Utilization of tasks in high mode

of low criticality tasks within the available bandwidth, unlike earlier approaches that completely discard low critical tasks.

IX. FUTURE WORK

The adoption of more than two criticality levels for mixed criticality tasks is a recent trend in real-time mixed criticality systems. Another important tendency is to go beyond the uniprocessor paradigm and utilize multiprocessor architectures for mixed criticality systems. We aim to investigate a system with more than two criticality levels in a multicore environment for our future work. We used a periodic task model in the proposed method in which tasks are executed within a specified window and repeated after the specified period. We are also looking forward to extend the proposed method to an aperiodic task model, or to a combination of periodic and aperiodic tasks model. For real-time mixed criticality systems, the proposed method can be applied to fault-tolerant and power-aware systems.

APPENDIX

See Tables 3 and 4.

REFERENCES

[1] S. Vestal, ''Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,'' in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 239–243.

[2] A. Burns and R. I. Davis, ''Mixed criticality systems—A review,'' Version 13, Tech. Rep., 2021.

[3] A. Burns and S. K. Baruah, ''Towards a more practical model for mixed criticality systems,'' in *Proc. 1st Workshop Mixed Criticality Syst. (WMC)*, 2013, pp. 1–6.

[4] 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. (ECRTS)*, Jul. 2016, pp. 131–138.

[5] D. Liu, N. Guan, J. Spasic, G. Chen, S. Liu, T. Stefanov, and W. Yi, ''Scheduling analysis of imprecise mixed-criticality real-time tasks,'' *IEEE Trans. Comput.*, vol. 67, no. 7, pp. 975–991, Jul. 2018.

[6] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, ''Optimal reward-based scheduling for periodic real-time tasks,'' *IEEE Trans. Comput.*, vol. 50, no. 2, pp. 111–130, Feb. 2001.

[7] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, ''Service adaptions for mixed-criticality systems,'' in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 125–130.

[8] S. Iacovelli and R. Kirner, ''A lazy bailout approach for dual-criticality systems on uniprocessor platforms,'' *Designs*, vol. 3, no. 1, p. 10, Feb. 2019.

[9] Z. Jiang, X. Dai, and N. Audsley, ''HIART-MCS: High resilience and approximated computing architecture for imprecise mixed-criticality systems,'' in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2021, pp. 290–303.

[10] X. Gu and A. Easwaran, ''Dynamic budget management and budget reclamation for mixed-criticality systems,'' *Real-Time Syst.*, vol. 55, no. 3, pp. 552–597, Jul. 2019.

[11] S. Baruah and S. Vestal, ''Schedulability analysis of sporadic tasks with multiple criticality specifications,'' in *Proc. Euromicro Conf. Real-Time Syst.*, Jul. 2008, pp. 147–155.

[12] T. Park and S. Kim, ''Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems,'' in *Proc. 9th ACM Int. Conf. Embedded Softw. (EMSOFT)*, Oct. 2011, pp. 253–262.

[13] H. Su, D. Zhu, and D. Mossé, ''Scheduling algorithms for elastic mixed-criticality tasks in multicore systems,'' in *Proc. IEEE 19th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2013, pp. 352–357.

[14] G. Lipari and G. Buttazzo, ''Resource reservation for mixed criticality systems,'' in *Proc. Workshop Real-Time Syst. Past, Present, Future*, 2013, pp. 60–74.

[15] Z. Jiang, X. Dai, A. Burns, N. Audsley, Z. Gu, and I. Gray, ''A high-resilience imprecise computing architecture for mixed-criticality systems,'' *IEEE Trans. Comput.*, vol. 72, no. 1, pp. 29–42, Jan. 2023.

[16] Y.-W. Zhang, ''Energy efficient non-preemptive scheduling of imprecise mixed-criticality real-time tasks,'' *Sustain. Computing: Informat. Syst.*, vol. 37, Jan. 2023, Art. no. 100840.

[17] Y.-W. Zhang, R.-K. Chen, and Z. Gu, ''Energy-aware partitioned scheduling of imprecise mixed-criticality systems,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 11, pp. 3733–3742, Nov. 2023.

[18] Y.-W. Zhang, ''DVFS-based energy-aware scheduling of imprecise mixed-criticality real-time tasks,'' *J. Syst. Archit.*, vol. 137, Apr. 2023, Art. no. 102849.

[19] Y.-W. Zhang, J.-P. Ma, H. Zheng, and Z. Gu, ''Criticality-aware EDF scheduling for constrained-deadline imprecise mixed-criticality systems,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 2, pp. 480–491, Feb. 2024.

[20] Y.-W. Zhang and R.-K. Chen, ''Energy-efficient scheduling of imprecise mixed-criticality real-time tasks based on genetic algorithm,'' *J. Syst. Archit.*, vol. 143, Oct. 2023, Art. no. 102980.

[21] A. Bhuiyan, S. Sruti, Z. Guo, and K. Yang, ''Precise scheduling of mixed-criticality tasks by varying processor speed,'' in *Proc. 27th Int. Conf. Real-Time Netw. Syst.*, Nov. 2019, pp. 123–132.

[22] T. She, Z. Guo, and K. Yang, ''Scheduling constrained-deadline tasks in precise mixed-criticality systems on a varying-speed processor,'' in *Proc. 30th Int. Conf. Real-Time Netw. Syst.*, Jun. 2022, pp. 94–102.

[23] H. Chai, G. Zhang, J. Sun, A. Vajdi, J. Hua, and J. Zhou, ''A review of recent techniques in mixed-criticality systems,'' *J. Circuits, Syst. Comput.*, vol. 28, no. 7, Jun. 2019, Art. no. 1930007.

[24] R. M. Pathan, ''Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors,'' in *Proc. Leibniz Int. Informat. (LIPIcs)*, vol. 106, 2018, pp. 1–22.

[25] S. K. Roy, R. Devaraj, A. Sarkar, and D. Senapati, ''SLAQA: Quality-level aware scheduling of task graphs on heterogeneous distributed systems,'' *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–31, Sep. 2021.

[26] G. Lipari and S. Baruah, ''Greedy reclamation of unused bandwidth in constant-bandwidth servers,'' in *Proc. 12th Euromicro Conf. Real-Time Syst.*, 2000, pp. 193–200.

[27] C. Liu and J. Layland, ''Scheduling algorithms for multiprogramming in a hard real-time environment,'' *J. ACM*, vol. 20, no. 1, pp. 46–51, Jan. 1973.

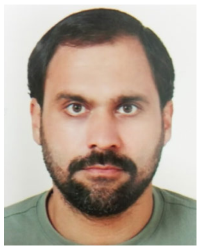
[28] A. Bastoni, B. Brandenburg, and J. Anderson, ''Cache-related preemption and migration delays: Empirical approximation and impact on schedulability,'' in *Proc. OSPERT*, vol. 10, 2010, pp. 33–44.

AMJAD ALI received the B.Sc. degree in computer science from the University of Peshawar, Pakistan, in 1997, the M.S. degree in computer science from Gomal University, Pakistan, in 2000, the M.S. degree in computer science from the University of Peshawar, in 2010, and the Ph.D. degree from the Real-Time Systems Laboratory, Gyeongsang National University, South Korea, in 2016. He was a Lecturer with the University of Peshawar, from 2001 to 2011. Since 2012, he has been an Assistant Professor with the Department of Computer and Software Technology, University of Swat. His research interests include real-time systems, power-aware computing, and fault-tolerant computing.

SHAH ZEB received the B.S. and M.S. degrees in computer science from the University of Swat, Pakistan, in 2017 and 2023, respectively. His research interests include real-time systems, imprecise computation, power awareness, and fault tolerance.

MADALLAH ALRUWAILI (Member, IEEE) received the bachelor's degree (Hons.) from Jouf University, Saudi Arabia, in 2005, the M.S. degree from the University of Science, Malaysia, in 2009, and the Ph.D. degree from Southern Illinois University, Carbondale, IL, USA, in 2015. His Ph.D. dissertation titled ''Enhancement and Restoration of Dust Images.'' He is currently an Associate Professor in computer engineering and networks with Jouf University. His research interests include healthcare, data analysis, image processing, image quality analysis, pattern recognition, computer vision, and biomedical imaging.

ASAD MASOOD KHATTAK (Member, IEEE) received the M.S. degree in information technology from the National University of Sciences and Technology (NUST), Islamabad, Pakistan, in 2008, and the Ph.D. degree in computer engineering from Kyung Hee University, South Korea, in 2012. He is currently an Associate Professor with the College of Technological Innovation, Zayed University, Abu Dhabi, United Arab Emirates, in August 2014. He was a Postdoctoral Fellow with the Department of Computer Engineering, Kyung Hee University, South Korea, and later joined the College of Technological Innovation as an Assistant Professor. He is currently leading three research projects, collaborating in four research projects, and has successfully completed five research projects in the fields of data curation, context-aware computing, the IoT, and secure computing. He has authored/coauthored more than 120 journals and conference papers in highly reputed venues. He is serving as a reviewer, a program committee member, and a guest editor for many conferences and journals. He has delivered keynote speeches, invited talks, guest lectures, and has delivered short courses in many universities. He and his team have secured several national and international awards in different competitions.



BASHIR HAYAT received the M.S. degree in computer science from Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST), Islamabad, Pakistan, in 2010, and the Ph.D. degree in informatics from Gyeongsang National University, Jinju, South Korea, in 2020. He is currently a Faculty Member of the Institute of Management Sciences Peshawar, Pakistan, in 2011.

KI-IL KIM received the M.S. and Ph.D. degrees in computer science from Chungnam National University, Daejeon, South Korea, in 2002 and 2005, respectively. He is currently with the Department of Computer Science and Engineering, Chungnam National University. He has been with the Department of Informatics, Gyeongsang National University, since 2006. His research interests include machine learning for networks, wireless/mobile networks, fog computing, MANET, QoS for wireless, multicast, and sensor networks.