Received 15 October 2023, accepted 29 October 2023, date of publication 8 November 2023,
date of current version 20 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3330973

RESEARCH ARTICLE

Overhead Based Cluster Scheduling of Mixed Criticality Systems on Multicore Platform

AMJAD ALI[1], ASAD MASOOD KHATTAK[2], (Member, IEEE), SHAHID IQBAL[1], OMAR ALFANDI[2], (Member, IEEE), BASHIR HAYAT[3], MUHAMMAD HAMEED SIDDIQI[4], AND ADIL KHAN[5], (Member, IEEE)

[1]Department of Computer and Software Technology, University of Swat, Swat, Khyber Pakhtunkhwa 19120, Pakistan
[2]College of Technological Innovation, Zayed University, Abu Dhabi, United Arab Emirates
[3]Center of Excellence in IT, Institute of Management Sciences, Peshawar 25000, Pakistan
[4]Department of Computer Science, Jouf University, Sakaka, Al Jowf 72388, Saudi Arabia
[5]School of Computer Science, University of Hull, HU6 7RX Hull, U.K.

Corresponding author: Asad Masood Khattak (asad.khattak@zu.ac.ae)

ABSTRACT The cluster-based technique is gaining focus for scheduling tasks of mixed-criticality (MC) real-time multicore systems. In this technique, the cores of the MC system are distributed in groups known as clusters. When all cores are distributed in clusters, the tasks are partitioned into clusters, which are scheduled on the cores within each cluster using a global approach. In this study, a cluster-based technique is adopted for scheduling tasks of real-time mixed-criticality systems (MCS). The Decreasing Criticality Decreasing Utilization with the worst-fit (DCDU-WF) technique is used for partitioning of tasks to clusters, whereas a novel mixed-criticality cluster-based boundary fair (MC-Bfair) scheduling approach is used for scheduling tasks on cores within clusters. The MC-Bfair scheduling algorithm reduces the number context switches and migration of tasks, which minimizes the overhead of mixed-criticality tasks. The migration and context switch overhead time is added at the time of each migration and context switch respectively for a task. In low critical mode, the low mode context switch and migration overhead time is added to task execution time, while the high mode overhead time of migration and context switch is added to the execution time of a task in high critical mode. The results obtained from experiments show the better schedulablity performance of proposed cluster-based technique as compared to cluster-based fixed priority (CB-FP), MC-EKG-VD-1, global and partitioned scheduling techniques e.g., for target utilization U=0.6, the proposed technique schedule 66.7% task sets while MC-EKG-VD-1, CB-FP, partitioned and global techniques schedule 50%, 33.3%, 16.7% and 0% task sets respectively.

INDEX TERMS Mixed-criticality systems, real-time systems, cluster-based approach, mixed-criticality boundary fair, context switches, tasks migration.

I. INTRODUCTION

The integration of multiple functionalities i.e., high critical (safety-critical) and low critical functionalities (mission critical) on a common executing platform is a recent trend in real-time systems and is commonly employed on different platforms such as ARINAC [1] for aerospace, and AUTOSAR [2] for automotive industries. For integrating these different multiple functionalities on a common execution platform, the idea of mixed-criticality (MC) was adopted by such platforms. The high critical functionalities have very high importance while the importance of low critical functionalities is low. Such a real-time system having different functionalities is known as mixed-criticality system (MCS). Baruah et.al, noted that task execution time bounds tend to be larger and more conservative as confidence requirements increase. For example the largest execution time observed during tests of normal operating mode scenarios can be specified as the WCET at a low level of assurance; the largest execution time observed during more exhaustive

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Ching Ying.

"code-coverage" tests are more appropriate as the WCET at a higher level of assurance [3]. The high critical functionalities need certification to ensure their correctness. For certification, the certification authorities (CA) make certain assumptions e.g line of code for estimating the WCET (worst-case execution Time) of these high critical functionalities. The WCET estimated by CA is too large than the WCET specified by designer of the system i.e., the WCET acquired through experiments. The certification authorities only concern with the correction of functionalities having high criticality while the designer of the system requires the correctness of both functionalities.

The evolution of embedded systems from single core to multicore platforms is in trending from the recent past and receiving more attention. There are three basic scheduling techniques for scheduling tasks of real-time multicore systems i.e., partitioned, global, and cluster-based approaches. In partitioned technique, each task is allocated to a particular core and the core executes those tasks only which are allocated to it. Tasks can be scheduled on each core using the EDF or RM scheduling algorithms. Partitioning tasks and assigning each partition to one core has the advantages of using the safety verification formula to make sure no task overruns occurs, and task migration between cores is not possible hence execution overhead is also reduced [4]. The problem in partitioned scheduling approach is the allocation of tasks to cores i.e., a task is not allocated to any core when the utilization of a task is less than the total amount of unused utilization of all cores but there is not a single core with unused capacity greater or equal to the utilization of that task. In this case, the task set is not schedulable, which reduces the schedulable workload utilization. Similarly, when the mode is switched from low (LO) to high (HI) mode on any processor, and if the execution time of tasks become higher than the unused space of processor, the MC task set is also not schedulable. Because task migration to another processor is prohibited which have enough capacity for executing the tasks, this causes a reduction of schedulable workload utilization. In semi-partitioned approach, few tasks are splitted into sub tasks, which can utilize the unused capacity of processors to increase the utilization of processors [5]. But this approach increases the number of preemptions and preemption overhead of tasks, and decrease the schedulable utilization of a task set.

In contrast, the global technique consists of a single shared queue containing all tasks to be executed on multicore platform. A task having higher priority is allocated from the global queue to an idle core for execution. The global approach allows migration of tasks among the cores during runtime i.e. a task can execute on any core, which overcomes the limitation of tasks allocation of the partitioned scheduling technique [6]. But this tasks migrations among cores can lead to high run-time overhead in the global scheduling approach.

Moreover, neither partitioned technique nor global technique take over each other, because some task sets are schedulable by partitioned technique but not schedulable by global technique, and vice versa. The number of context switches in both techniques and the number of migrations in global technique causes high overhead, which can't be neglected. Recently, Cluster based scheduling technique is gaining focus for scheduling the tasks of multicore real-time systems. Cluster-based technique overcomes the tasks allocation problem of partitioned technique and high migration overhead of global technique. In cluster-based technique, all cores are divided in clusters and the tasks of system are allocated to these clusters. The cluster-based approach affectively utilizes the unused capacity on all cores i.e., a task is allocated to a cluster when the utilization of a task does not exceed the total amount of unused utilization of the cluster. When the tasks are assigned to clusters, different global scheduling algorithms are used within clusters to schedule the task sets. The cluster-based technique can convert into partitioned technique when every cluster has only one core and can also change to global technique when all the cores are putted in one cluster. Clustering reduces the number of migrations as compared to global approach which leads to reduce the overhead and also overcome the task allocation problem in partitioned technique. Calandrino et.al presented a new hybrid technique for real-time multicore systems named H-EDF, to minimize the problems of G-EDF and P-EDF scheduling approaches. The authors divides cores in various clusters which share a cache and allocate tasks to these clusters. The tasks are scheduled in clusters by a global scheduling algorithm named preemptive global EDF [7].

In this research work, the overhead caused by tasks migration and context switches is reduced in cluster-based technique for multicore mixed-criticality systems. We used an efficient task allocation technique for tasks allocation to clusters. After tasks allocation to the clusters, the tasks are scheduled on the cores within the cluster using a novel global scheduling technique. This research work is one of the initial research works that reduces the overhead caused by context switches and migrations of tasks in cluster-based MC systems. This research work performs better as compared to cluster-based fixed priority (CB-FP) [8], MC-EKG-VD-1 [9] global and partitioned scheduling approaches.

II. RELATED WORK

Initially, Vestal [10] used the notion of mixed-criticality (MC) for scheduling tasks on the unicore platform, but now multicore or multiprocessor platform is gaining focus to adopt the idea of MC scheduling. For multicore MC real-time systems, the partitioned scheduling approach is initially used for scheduling MC tasks. For partitioned scheduling, Kelly et al. discussed different techniques for task ordering and partitioning of tasks among cores. The authors used Decreasing Criticality and Decreasing Utilization techniques for tasks ordering and partitioning of tasks among the cores. They discussed different tasks partitioning heuristic approaches i.e., first-fit, best-fit, and worst-fit. In first-fit, the order of cores is fixed and a task is given to the first core on which it fits, otherwise it is allocated to the next core on which it fits, and so on. In best-fit, the task is allocated to that core having minimum unused capacity among cores. In worst-fit, each task is assigned to that core having maximum unused capacity among the cores. The authors also used the fixed-priority RM and Audslay's approach for tasks scheduling on each core [11]. Later, Ekberg andYi extended EY (Ekberg and Yi) [12] virtual deadline MC unicore scheduling algorithm, and proposed a new algorithm named Mixed-criticality Partitioning with Virtual Deadline (MPVD) for scheduling the MC task set of multicore systems. The MPVD algorithm first allocates high-criticality tasks through worst-fit and then allocates low critical tasks using first-fit approaches [13]. Nagalakshmi and Gomathi proposed a partitioned-based scheduling approach named C-PEDF [14] that groups the MC tasks in clusters. Each cluster contains one executive task i.e. high critical task, and a group of member tasks i.e. low critical tasks. These clusters are then partitioned among cores, which schedule these clusters of tasks using partitioned approach. It should be noted that if the execution budget of tasks exists in two clusters, then these clusters should be allocated to one core to avoid the concurrent execution of the task. As a task may allocated to more than one clusters, therefore the overhead of context switches is increases [14].

After the partitioned approach, the global approach is used for scheduling tasks of MC multicore real-time systems. Initially, Pathan applied fixed-priority response time analysis for the global scheduling of MC sporadic tasks on multicore [15]. Andersson et al. considered utilization bounds for periodic task sets with implicit deadlines. They showed that the maximum utilization bound for any global fixed job priority algorithm is (m+1)/2 on m cores [16]. Baruah derived a sufficient schedulability test for global EDF scheduling of sporadic task sets with constrained deadlines [17]. Li and Baruah [6] extend a uni-core scheduling algorithm EDF-VD (EDF with Virtual Deadline) [18] to multicore and proposed a novel global algorithm named GLOBAL, by applying fpEDF [19] for MC tasks scheduling. EDF-VD is a unicore MC systems scheduling algorithm and fpEDF is a scheduling approach for traditional real-time multicore systems. Lee et al. proposed a fluid model based scheduling algorithm for multiprocessor MC system MC-Fluid. MC-Fluid algorithm executes each tasks in proportion to its criticality-dependent rate. They also propose an exact schedulability condition for MC-Fluid and an optimal assignment algorithm for criticality-dependent execution rates. The authors showed that MC-Fluid has a speedup factor of $(1+\sqrt{5})/2\,(\sim 1.618)$, which is best known in multiprocessor MC scheduling [20].

In the domain of real-time scheduling for heterogeneous multicore platforms, various heuristic cluster-based and semi-partitioned scheduling approaches have been proposed to optimize energy efficiency, temperature management, and task allocation. Sharma et.al introduced a heuristic approach called RT-SEAT for hybrid scheduling approach. The proposed RT-SEAT scheduler operates across four distinct layers. It begins by segmenting the timeline into intervals in the outermost layer. In the subsequent layer, the scheduler handles task-to-core assignments and generates a provisional task schedule for each core. Transitioning to the third layer, the scheduler reorganizes the sequence of task execution on individual cores to effectively manage core temperatures. Finally, in the last layer, it incorporates DVFS to ensure the scheduler's energy awareness. In the following sections, a more detailed exploration of the scheduler's functioning will be presented [21].

Moulik et.al introduced an efficient and low overhead cluster-oriented scheduling technique referred to as SEAMERS. This approach focuses on the allocation of tasks in a heterogeneous multicore environment while minimizing unnecessary computational burdens. SEAMERS implements dynamic voltage and frequency scaling (DVFS) on a per-core basis, enabling optimized task scheduling at the core level [22]. Sharma et.al presented a cluster based heuristic scheduling strategy named CETAS which stands for A Cluster based Energy and Temperature Efficient Real-time Scheduler for heterogeneous platforms, which performs energy as well as temperature aware task scheduling on heterogeneous multicore platforms. This approach efficiently schedule a set of real-time periodic tasks on a DVFS-enabled heterogeneous platform with a focus on energy and temperature considerations [23]. Moulik et.al introduce a heuristic Cluster based Energy Aware Scheduler for Real-Time Heterogeneous Systems referred to as CEAT, aimed at energy-aware scheduling of a set of real-time periodic tasks on a heterogeneous multicore platform with DVFS capabilities. This approach involves three sequential phases: Deadline Segmentation, Allocation of Tasks to Cores, and scheduling that Prioritizes Energy Efficiency [24].

Sharma et.al develop a heuristic scheduling approach named FATS-2TC, addressing the simultaneous control of energy and peak temperature levels. This is achieved through the standby-sparing mechanism on systems with two types of cores, such as the big. LITTLE architecture, enhancing resilience against transient faults [25]. Moulik et.al presents an innovative semi-partitioned heuristic scheduler known as RESET, which stands for "A Real-time Scheduler for Energy and Temperature Aware Heterogeneous Multi-core Systems." This novel scheduler is designed to achieve optimizing resource utilization by intelligently allocating tasks, while concurrently tackling the challenges of reducing dynamic energy consumption and effectively managing core temperatures. By striking this balance, RESET offers a promising approach to enhance the overall performance and efficiency of heterogeneous multi-core systems [26]. Sharma et.al presents a heuristic technique, named ETA-HP, for energy and temperature efficient scheduling of a set of real-time periodic tasks on a DVFS empowered heterogeneous multicore system. The proposed strategy operates in four stages, namely Deadline Partitioning, Task-to-Core Allocation, Temperature-Aware Scheduling, and Energy-Aware Scheduling [27].

Recently, a cluster-based technique has been studied for MC tasks scheduling on multicore systems. In this technique, all the cores are distributed in clusters and the MC tasks are assigned to clusters. After assigning tasks to clusters, a global approach is used for the tasks scheduling within clusters. For the cluster-based scheduling approach, Ali and Kim [8] presented a scheduling algorithm for MC tasks scheduling on multicore systems. The authors partitioned MC tasks among clusters through the worst-fit heuristics approach. In low mode, the authors used small sizes of clusters (sub-clusters) while in a high mode they used larger sizes of clusters for tasks allocation. For task schedulability analysis, the authors used a fixed-priority response time analysis based on Audsley's approach [28] in each cluster and sub-cluster. Burns and Davis provided an extensive literature about scheduling algorithms proposed for the mixed-criticality task sets on multicore platform including cluster based approach [29].

For minimizing the overhead in real time systems, Zhang et al. proposed a novel algorithm named Least Switch and Laxity First to minimize the switching among tasks on unicore systems [30]. To decrease the overhead in real-time systems, Levin et al. used a scheduling algorithm named Deadline Partitioning Fair (DP-fair) [31] which reduced the run-time overhead of tasks. Zhu et al. discussed a novel algorithm for scheduling of tasks named Boundary fair (Bfair) on real-time multicore systems. The Bfair approach makes scheduling decisions for tasks only at the period boundaries, which essentially minimizes the points of scheduling. Furthermore, by computing the execution time of tasks between period boundaries, the Bfair schedule minimizes migrations and context switches of tasks, which reduces the scheduling overhead of real-time systems [32].

III. SYSTEM MODEL

This section presents the framework for the adopted cluster-based scheduling technique. In the cluster-based scheduling technique, the cores are divided into groups known as clusters. After creating clusters, the MC tasks are assigned to clusters by applying DCDU-WF tasks partitioning technique [1]. The MC tasks are scheduled by a novel MC-Bfair approach on cores within clusters. Initially, the tasks are scheduled in low mode i.e each task is executed up to its low mode execution requirement i.e., C_i(LO), but if a high critical task needs further execution after completing C_i(LO), the system is changed to high mode. The low tasks of the MC system are discarded after the mode switch and only each high task is scheduled up to their high mode execution requirement C_i(HI) in high mode.

A. TASK MODEL

The periodic task set of a mixed-criticality system is known as workload, which is represented by τ. In low mode, both LO and HI critical tasks release a job sequence, but in high mode, only HI tasks release a job sequence. A mixed-criticality periodic task τ_i is characterized by 6 parameters i.e., $\tau_i = (P_i, C_i^{Xi}, X_i, CS.A_i^{xi}, M.A_i^{xi})$, where,

- P_i is used for the period of task. It is supposed that $P_i = D_i$ (task's deadline).
- C_i^{Xi} is WCET of a task at criticality level X_i and $C_i^{Xi} = \{C_i(\text{LO}), C_i(\text{HI}\}$. The C_i(LO) and C_i(HI) represent the low mode and high mode WCET of MC task respectively.
- X_i is used to show the task's criticality level, where LO is used for low critical and HI is used for high critical task.
- $CS.A_i^{xi}$ is used for context switch overhead time of a task at criticality level X_i.
- $M.A_i^{xi}$ represents the migration overhead time of a task at criticality level X_i.

The utilization in low and high mode of MC task τ_i is denoted by $\text{U}_\text{i}^\text{LO}$ and $\text{U}_\text{i}^\text{HI}$ respectively and are derived as $\text{U}_\text{i}^\text{LO} = C_i(\text{LO})/P_i$ and $\text{U}_\text{i}^\text{HI} = C_i(\text{HI})/P_i$. The low mode utilization of all MC tasks is represented by U^LM, while in high mode U^HM is used to represent the total utilization of HI tasks and can be calculated as.

$$U^{LM} = \sum\nolimits_{\tau_i \in \tau} U_i^{LO} \tag{1}$$

$$U^{HM} = \sum\nolimits_{\tau_i \in \tau} U_i^{HI} \tag{2}$$

Equation 1 shows the summation of the utilization of the entire task set of the MC system in low mode calculated through low worst-case execution time, while equation 2 shows the summation of the utilization of HI tasks in high mode calculated through high worst-case execution time of HI tasks.

B. CLUSTERS

The cores of a real-time MC system are divided into groups known as clusters. Each cluster is denoted by C having two parameters C(W_C, N_C), where W_C is used for the workload of the cluster while N_C is used for the number of cores within a cluster. The tasks that are assigned to clusters are executed up to C_i (LO) in low mode, while each task is executed up to C_i (HI) in high mode within each cluster.

C. CLUSTER-BASED SCHEDULING FRAMEWORK

A cluster-based technique is adopted for the scheduling of task sets of multicore MC systems, in which the overhead caused due to context switches and migrations of task is reduced. The MC tasks are allocated to clusters by DCDU-WF [8] approach. After tasks partitioning, a novel MC-Bfair scheduling algorithm is used for the scheduling of MC tasks on cores within clusters. The general idea of the cluster-based scheduling framework is given in figure 1 and figure 2 for low and high modes respectively. The system consists of 4 cores on which the task set of table 1 is scheduled. The cores are equally partitioned into cluster C_1 and cluster C_2, each having 2 cores. The MC tasks of the system are partitioned into cluster C_1 and cluster C_2, each having 4 tasks i.e., $\text{C}_1 = \tau_2$, τ_4, τ_6 and τ_8 and $\text{C}_2 = \tau_1$, τ_3, τ_5 and τ_7.

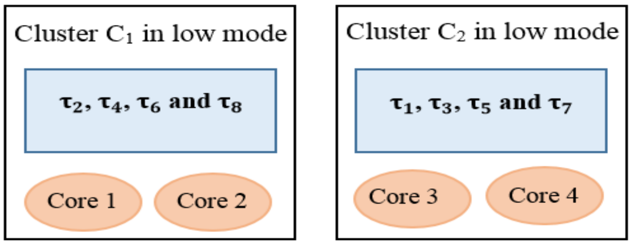


FIGURE 1. Framework of cluster-based approach in low mode.

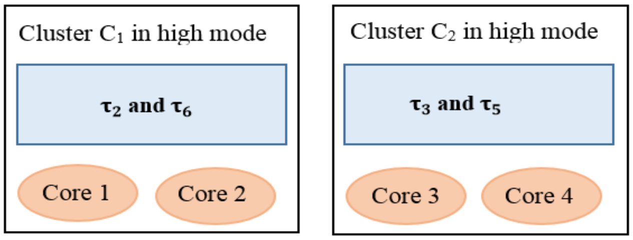


FIGURE 2. Framework of cluster-based approach in high mode.

TABLE 1. MC task set.

τ_i	P_i	$C_i(LO)$	$C_i(HI)$	X_i	U_i^{LO}	U_i^{HI}
τ_1	4	2	0	LO	0.5	0
τ_2	4	2	3	HI	0.5	0.75
τ_3	12	4	7	HI	0.33	0.58
τ_4	12	3	0	LO	0.25	0
τ_5	24	10	12	HI	0.42	0.5
τ_6	24	10	11	HI	0.42	0.46
τ_7	24	3	0	LO	0.13	0
τ_8	12	2	0	LO	0.17	0

IV. RESEARCH MOTIVATION

Global and partitioned scheduling approaches are the two scheduling approaches for multicore mixed-criticality systems. In this research work, a novel MC-Bfair scheduling technique is used for the overhead caused by context switches and migrations of tasks during execution. In global scheduling, both task migration and context switches can occur, while in the partitioned approach, migration of tasks is not allowed, but context switches of tasks occur. A cluster-based MC-Bfair scheduling approach is used for scheduling MC tasks to overcome the problems of partitioned and global approaches. This research work is an extension of the previous cluster-based scheduling approach [18]. The cluster-based approach also dominates the partitioned and global approaches but it lacks the overhead amount for the scheduling of MC tasks.

For low and high modes, the MC-Bfair algorithm is applied for scheduling of tasks on cores. This algorithm minimizes the context switches and migrations of MC tasks which reduces the scheduling overhead, as the Bfair algorithm minimizes the overhead of tasks in traditional real-time systems. The MC-Bfair constructs a periodic schedule for the task set of MC systems as in Bfair algorithm [32]. This approach allocates C_i^{LO} execution time for all tasks in low mode, and for high mode it allocates C_i^{HI} execution time for each HI tasks in the interval [(k − 1)·P_i, k·P_i) for all k ∈ {1, 2, 3,...}. The schedule for a given task set is only considered from 0 to LCM of tasks periods due to its periodic property. B = {$b_0,\ldots, b_f$} is used for expressing the period boundaries of tasks, where b_0 is used for the starting point and is equal to zero, while b_f is used for the final time unit which is equal to LCM. The time units between two consecutive period boundaries b_k and b_{k+1}, is represented by interval $[b_k, b_{k+1})$.

For each period boundary, the total execution time of each task is calculated. The low mode utilization U_i^{LO} of a task is calculated as c_i^{LO}/p_i and the high mode utilization U_i^{HI} is calculated as c_i^{HI}/p_i. The sum of utilizations of all tasks of a system in low mode is defined by $U^{LM} = \sum_{i=1}^{n} U_i^{LO}$ and for high mode the sum of utilizations of high critical tasks is defined by $U^{HM} = \sum_{i=1}^{n} U_i^{HI}$. The remaining work RW_i^{K+1} at period boundary b_k of task τ_i is define as the difference of $b_k.U_i$ and the allocated time units of task τ_i before period boundary b_k. A schedule will be boundary fair at any period boundary, if the remaining work of a task is smaller than one. The mandatory time m_i^{K+1} of an MC task can be calculated as $m_i^{K+1} = \max\{0, \lfloor RW_i^K + (b_{k+1} - b_k) \cdot U_i \rfloor\}$. m_i^{K+1} which is the integer part of the summation of the remaining work at b_k and the work to be done during $[b_k, b_{k+1})$. The corresponding decimal part is defined as pending work and can be calculated as $PW_i^{K+1} = RW_i^K + (b_{k+1} - b_k) \cdot U_i - m_i^{K+1}$. Now, we can calculate the total execution time of each MC task at any period boundary. The total execution time an MC task at each period boundary is calculated as $TE_i^{K+1} = m_i^{K+1} + (PW_i^{K+1} - RW_i^{K+1})$.

A. PARTITIONED APPROACH FOR SCHEDULING OF MIXED-CRITICALITY TASKS

Mixed-criticality tasks are partitioned among cores in the partitioned scheduling approach and are only schedules on the cores to which the tasks are assigned. The migration of tasks among cores is not allowed. Initially, LO and HI tasks are executed in low mode up to $C_i(LO)$ on a core. If the mode is switched to high mode, LO tasks are dropped and only HI tasks are executed up to $C_i(HI)$ on a core. The partitioned approach has the limitation of the partitioning of tasks among cores i.e., if the utilization of a task is larger than the remaining free space on each core, then such task can't be allocated to any core. In this scenario, the MC tasks are not scheduled, which decreases the schedulable utilization of the MC workload. Similarly, when the mode is switched from low to high at any core, and if the execution time of a HI task becomes larger than the remaining free space of the core, then the task set is not schedulable, because the task cannot migrate to a core which has enough capacity to schedule this task.

Example 1: For the task set shown in table 1, consider a system having 4 cores. The tasks τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7 and τ_8 are allocated to 4 cores by the DCDU-WF tasks allocation approach. Through this approach, firstly all HI tasks are assigned to cores, and a HI task of a higher utilization is assigned to that core having the largest unused space. After partitioning all HI tasks, LO tasks are partitioned among cores in the above manner. The MC tasks that are allocated to cores are shown in Figures 3 and figure 4 for both low and high modes respectively.

The MC-Bfair scheduling approach schedule all tasks on each core in low mode including core 3 as shown in figure 5, but when the overhead time caused by the context switch is added to the execution time of each task, then the third, fourth and sixth jobs of τ_1 on core 3 missed the deadlines at 12, 16 and 24 period boundaries respectively as shown in Figure 6. The time of context switch overhead of an MC task is calculated as $CS.A_i^{Xi} = C_i^{Xi} * 5/100$. Table 2 shows the time of the context switch added to the execution time of tasks when the context switches occur. The given task set is not scheduled by partitioned technique.

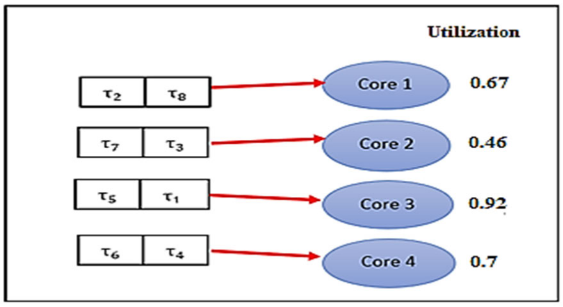


FIGURE 3. Tasks allocation using DCDU-WF for low mode.

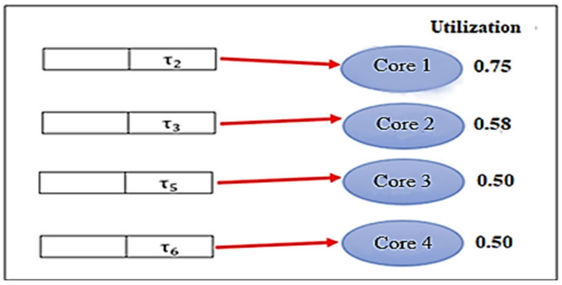


FIGURE 4. Tasks allocation using DCDU-WF for high mode.

TABLE 2. MC tasks for partitioned approach.

τ_i	P_i	$C_i(LO)$	$C_i(HI)$	X_i	U_i^{LO}	U_i^{HI}	$CS.A_i^{LO}$	$CS.A_i^{HI}$
τ_1	4	2	0	LO	0.50	0	0.1	0
τ_2	4	2	3	HI	0.50	0.75	0.1	0.15
τ_3	12	4	7	HI	0.33	0.58	0.2	0.35
τ_4	12	3	0	LO	0.25	0	0.15	0
τ_5	24	10	12	HI	0.42	0.50	0.5	0.6
τ_6	24	10	11	HI	0.42	0.46	0.5	0.55
τ_7	24	3	0	LO	0.13	0	0.15	0
τ_8	12	2	0	LO	0.17	0	0.1	0

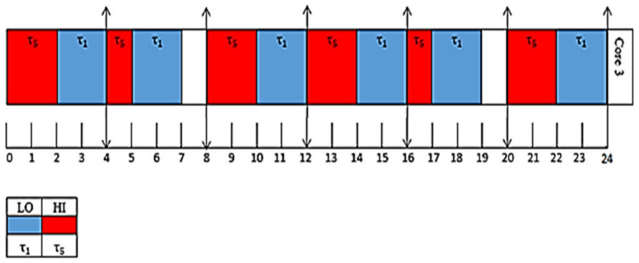


FIGURE 5. Tasks scheduling on core 3.

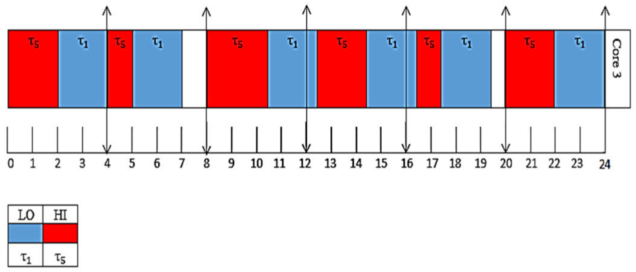


FIGURE 6. Tasks scheduling on core 3 with overhead time.

B. GLOBAL APPROACH FOR MC TASKS SCHEDULING

Global scheduling consists of a global single-ready queue in which all MC tasks are stored. In both low and high modes, tasks are allocated to an idle core for execution. The MC tasks can migrate among cores during run-time, which causes high overhead. Initially, the global approach schedule both LO and HI tasks in low mode. Global scheduling technique overcomes the problem of partitioned scheduling i.e., allocation of tasks to cores, but migrations of tasks may lead to high overhead. The tasks for global approach in low and high mode is shown in figure 7 and figure 8.

Consider the example given in table 1, the MC task set is scheduled in low mode on 4 cores from a single ready queue by using the MC-Bfair algorithm as shown in figure 9. The total utilization of workload in low mode $U^{LM} = 2.72$. The time of context switch overhead of an MC task is calculated as $CS.A_i^{Xi} = C_i^{Xi} * 10/100$, and migration overhead is calculated as $M.A_i^{Xi} = C_i^{Xi} * 20/100$, as given in table 3. When the overhead time caused by the migration ad context switch is added to the execution time of each task, which increases the total execution time of tasks. In low mode, the tasks are not schedulable between period boundaries b_4 and b_5 by adding the additional time of overhead. Table 4 shows the execution time of tasks with overhead time. The 16-time units are available between b_4 and b_5 for the 4 cores,

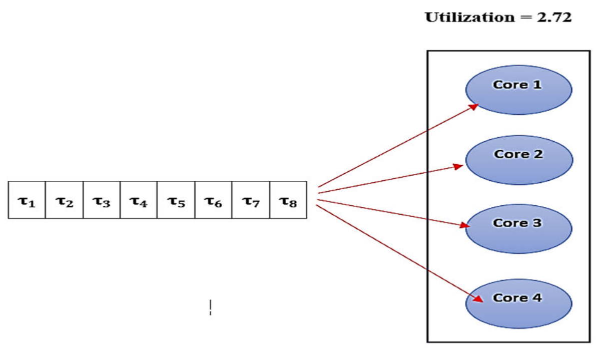


FIGURE 7. Tasks for global scheduling in LO mode.

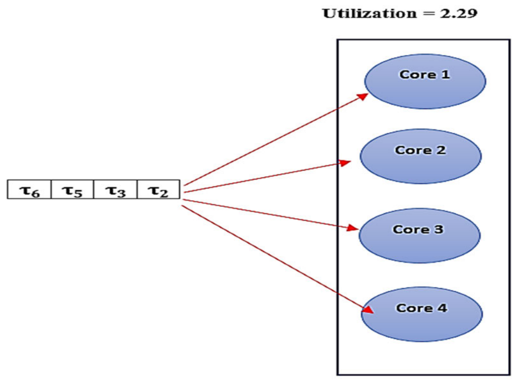


FIGURE 8. Tasks for global scheduling in HI mode.

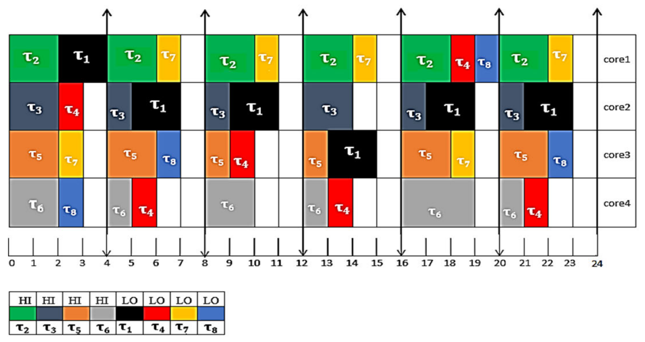


FIGURE 9. Global scheduling of tasks.

while the required execution time units are 18.1 as shown in table 4. Therefore, the MC workload is not scheduled at period boundary 20 in low mode.

C. CLUSTER-BASED APPROACH FOR SCHEDULING OF MC TASKS

In cluster-based technique, cores of systems are divided in clusters and tasks are allocated to the clusters. The cluster-based approach affectively utilizes the unused capacity on all cores i.e., a task is allocated to a cluster when the utilization of a task does not exceed the total amount of unused utilization of the cluster. When the tasks are assigned to clusters, different global scheduling algorithms are used within clusters for scheduling the task sets. The cluster-based

TABLE 3. MC task set for global scheduling.

τ_i	P_i	$C_i(LO)$	$C_i(HI)$	X_i	$CS.A_i^{LO}$	$CS.A_i^{HI}$	$M.A_i^{LO}$	$M.A_i^{HI}$
τ_1	4	2	0	LO	0.2	0	0.4	0
τ_2	4	2	3	HI	0.2	0.3	0.4	0.6
τ_3	12	4	7	HI	0.4	0.7	0.8	1.4
τ_4	12	3	0	LO	0.3	0	0.6	0
τ_5	24	10	12	HI	1	1.2	2	2.4
τ_6	24	10	11	HI	1	1.1	2	2.2
τ_7	24	3	0	LO	0.3	0	0.6	0
τ_8	12	2	0	LO	0.2	0	0.4	0

TABLE 4. The Execution time of each tasks with overhead in global scheduling.

b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
Time	**0**	**4**	**8**	**12**	**16**	**20**	**24**
TE_1^K	0	2	2.4	2	2.4	2.4	2
TE_2^K	0	2	2	2	2	2	2
TE_3^K	0	2	1.4	1.4	2	1.4	1.4
TE_4^K	0	1	1.9	1.9	1.6	1.9	1.9
TE_5^K	0	2	3.0	2.0	2.0	2	3.3
TE_6^K	0	2	2	3	2	4	2
TE_7^K	0	1	0	0	1	1.9	0
TE_8^K	0	1	2.2	0	0	2.2	2.2
Total	*0*	*13*	*14.9*	*12.3*	*13*	*18.1*	*14.8*

technique can change into partitioned technique if each cluster has only one core and can also change to global technique when all the cores are put together in one cluster.

Considering example 1, tasks τ_2, τ_4, τ_6 and τ_8 are allocated to cluster C_1 while tasks τ_1, τ_3, τ_5 and τ_7 are allocated to cluster C_2 in low mode using the DCDU-WF [8] approach, as shown in figure 10. The utilizations of cluster C_1 and cluster C_2 in low mode are higher than utilization in high mode because the utilizations of all HI and LO tasks is added in low mode while only the utilizations of all HI critical tasks is added in high mode. Figure 11 shows the HI critical tasks of cluster C_1 and cluster C_2 in high mode. The task set of Table 1 is scheduled in both low and high modes by the cluster-based technique using a novel MC-Bfair scheduling algorithm.

V. ALLOCATION OF MC TASKS TO CLUSTERS

The tasks of MC system are distributed to clusters by the DCDU-WF [8] tasks allocation technique. In the DCDU technique, MC tasks are arranged by criticality level and utilization i.e., all high tasks are arranged first and then low

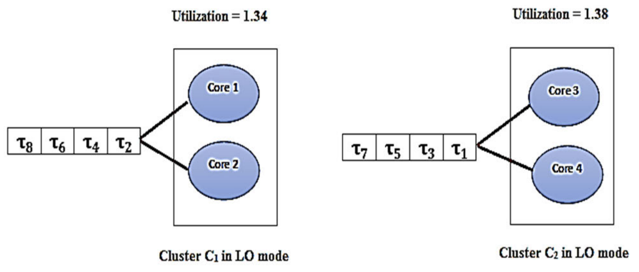


FIGURE 10. **Allocation of tasks to clusters in low mode.**

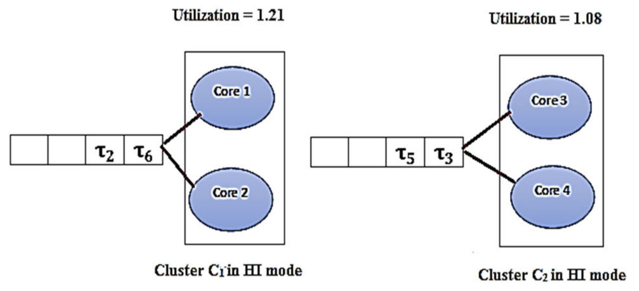


FIGURE 11. **Allocation of tasks to clusters in high mode.**

critical tasks by decreasing utilization. As each high critical tasks has two execution times C_i^{LO} and C_i^{HI}, therefore, it has two utilizations U_i^{LO} and U_i^{HI} for the low mode and high mode respectively, whereas a LO tasks have onlyU_i^{LO} utilization for low mode. The high critical tasks are arranged through U_i^{HI} utilization while low critical tasks are arranged through U_i^{LO} utilization.

After the arrangement of all MC tasks by DCDU technique, the worst-fit heuristic is used for partitioning MC tasks into clusters. This approach assigns an MC task to a cluster having maximum unused space i.e., high remaining utilization. The HI tasks are partitioned among clusters to ensure the efficient scheduling of HI tasks in high mode on the cores up to their C$_i$(HI) execution time. The aim of the first allocation of high-critical tasks to clusters is to ensure the schedulability of high-critical tasks after mode change. After assigning all HI tasks to clusters, LO tasks of the system are partitioned among clusters through the same approach. The LO task with high utilization is assigned first to a cluster with high remaining capacity and so on. The allocation of MC tasks is discussed in the following example 2.

Example 2: Consider the task set shown in Table 1.

The DCDU approach ordered the tasks as τ_2, τ_3, τ_5, τ_6, τ_1, τ_4, τ_7 and τ_8. The low mode utilization of the given task set is $U^{LM} = 2.72$ and the high mode utilization is $U^{HM} = 2.29$. As we know that the number of cores must be greater than the total utilizations of tasks i.e., U^{LM} and U^{HM} for the low mode and high mode respectively. The cores are divided into two clusters, each having two cores. Initially, the high mode utilization of cluster C$_1$ and C$_2$ i.e., UC_1^{HM} and UC_2^{HM} respectively for high tasks are zero. Therefore, the task τ_2 is assigned to cluster C$_1$ and task τ_3 is assigned to cluster C$_2$. The high mode utilization of cluster C$_1$ becomes greater than C$_2$ i.e., $(UC_1^{HM} = U_2^{HI} = 0.75) > (UC_2^{HM} = U_3^{HI} = 0.58)$, so the next HI task with largest utilization i.e., τ_5 is assigned to cluster C$_2$ and the utilization of cluster C$_2$ becomes $UC_2^{HM} = U_3^{HI} + U_5^{HI} = 1.08$ in high mode. The allocation of task τ_5 to cluster C$_2$ minimizes the remaining unused space of cluster C$_2$ than C$_1$, so the task τ_6 is assigned to cluster C$_1$ and the utilization of cluster C$_1$ becomes $UC_1^{HM} = U_2^{HI} + U_6^{HI} = 1.21$ in high mode. The HI tasks of the system are distributed among both clusters C$_1$ and C$_2$ with DCDU-WF. Now, the LO tasks are allocated to clusters by using the same technique. For partitioning of LO tasks among clusters, cluster C$_2$ has the maximum unused space than C$_1$ in low mode i.e., $(UC_2^{LM} = U_3^{LO} + U_5^{LO} = 0.75) < (UC_1^{LM} = U_2^{LO} + U_6^{LO} = 0.92)$, therefore task τ_1 is assigned to C$_2$, because τ_1 have largest utilization among all low critical tasks. After assigning τ_1 to cluster C$_2$, the remaining unused space of C$_1$ becomes larger than C$_2$ i.e., $UC_2^{LM} = U_3^{LO} + U_5^{LO} + U_1^{LO} = 1.25$. Therefore, the next low critical task τ_4 is assigned to cluster C$_1$. The utilization of cluster C$_1$ becomes $UC_1^{LM} = U_2^{LO} + U_6^{LO} + U_4^{LO} = 1.16$, and as the utilization of C$_1$ is still smaller than C$_2$, so task τ_8 is assigned to cluster C$_1$. Now, the remaining unused space of cluster C$_2$ becomes larger than cluster C$_1$ i.e., $UC_1^{LM} = U_2^{LO} + U_6^{LO} + U_4^{LO} + U_8^{LO} = 1.34$, so, the last task τ_7 is assigned to cluster C$_2$. The low mode utilization of cluster C$_2$ becomes, $UC_2^{LM} = U_3^{LO} + U_1^{LO} + U_5^{LO} + U_7^{LO} = 1.38$. The tasks that are assigned to both clusters C$_1$ and C$_2$ are shown in figure 12.

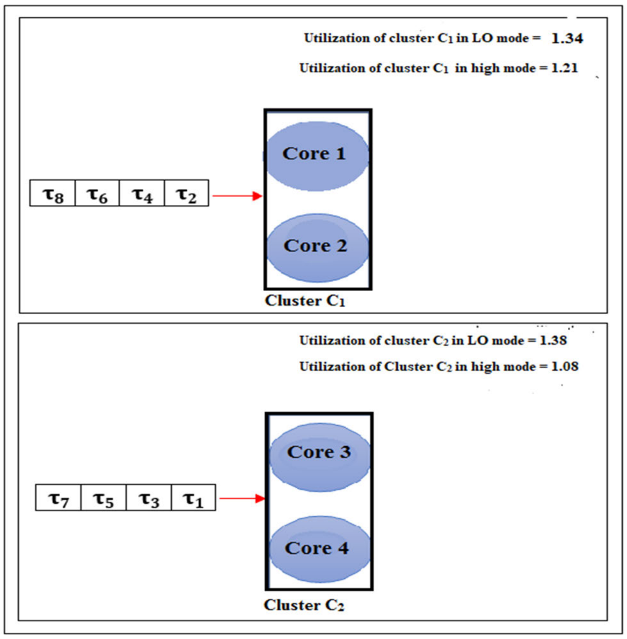

Utilization of cluster C1 in LO mode = 1.34
Utilization of cluster C1 in high mode = 1.21
τ8 τ6 τ4 τ2
Core 1
Core 2
Cluster C1
Utilization of cluster C2 in LO mode = 1.38
Utilization of Cluster C2 in high mode = 1.08
τ7 τ5 τ3 τ1
Core 3
Core 4
Cluster C2

FIGURE 12. **Allocation of tasks to clusters C_1 and C_2.**

VI. CLUSTER-BASED SCHEDULING OF MC SYSTEM

For low and high modes, the MC-Bfair algorithm is applied for globally scheduling of tasks on cores of cluster C$_1$ and cluster C$_2$. The MC-BFair algorithm, first computes the utilizations of all low and high criticality tasks in low criticality mode (U^{LM}). For high criticality mode, it determines the utilizations of only high criticality tasks (U^{HM}) in the mixed criticality system. For initial schedulability test, if the value of U^{LM} or U^{HM} is larger than the total number of cores in the system, then the mixed criticality task set is not schedulable. A schedulability test is applied for each cluster capacity (number of its cores) in high mode. If the utilization of high criticality tasks in a cluster is less than the cluster capacity, the task set is schedulable using MC-BFair scheduling algorithm. Otherwise the task set is not schedulable. Another schedulability test is applied for each cluster capacity in low mode. If the utilization of both low and high criticality tasks in a cluster is less than the cluster capacity, the task set is schedulable using MC-BFair scheduling algorithm. Otherwise the task set is not schedulable. The MC-BFair scheduling algorithm is shown in figure 13.

Algorithm 1 Mixed-Criticality Boundary Fair Scheduling (MC-Bfair)

```
1.   $U^{LM} = \sum_{T_i \epsilon W} \left\{ \frac{C_i(LO)}{P_i} \right\}$
2.   $U^{HM} = \sum_{T_i \epsilon W} \left\{ \frac{C_i(HI)}{P_i} \right\}$
     /* Schedulablity test for each cluster $N_c$ capacity in low mode*/
3.       for $k = 1$ to $n_c$ do
4.       if $UC_K^{LM} < C_K$ size Then
5.       call algorithm2(tasks scheduling in low mode)
6.       else
7.           return false
     /* Schedulablity test for each cluster $N_c$ capacity in high mode*/
8.       for $k = 1$ to $n_c$ do
9.       if $UC_K^{HM} < C_K$ size Then
10.      call algorithm3(tasks scheduling in high mode)
11.      else
12.      return false
     /* Schedulablity test for each cluster $N_c$ capacity in mode change*/
13.      for $k = 1$ to $n_c$ do
14.      if $UC_K^{HM} < C_K$ size Then
15.      call algorithm4(tasks scheduling in mode change)
16.      else
17.      return false
18.      else
19.      return false
```

FIGURE 13. Algorithm for mixed-criticality boundary fair scheduling.

The schedulability test is applied during mode change. If the utilization of high criticality task for mode change is less than the cluster capacity, the mixed criticality task set is schedulable using MC-BFair scheduling algorithm. Otherwise, the mixed-criticality task set is schedulable on multicore platform using cluster-based approach.

A. TOTAL EXECUTION TIME CALCULATION IN LOW MODE

For low mode, the low mode utilization $U_i^{LO} = c_i^{LO}/p_i$ is used in each equation for finding the total execution time of tasks in each period boundary. As explained earlier, the remaining work (RW_i^{K+1}), mandatory work (m_i^{K+1}), and pending work (PW_i^{K+1}) are calculated for each MC task to calculate the total execution time units (TE_i^{K+1}) of each task. The calculated total execution time TE_i^{K+1} of tasks is allocated to cores by the DCDU-WF approach for scheduling on cores at each period boundary. The tasks that are allocated to cluster C_1 are shown in table 5 and the tasks of cluster C_2 is given in table 6. The overhead time of context switch and migration of each task is calculated as, $CS.A_i^{Xi} = C_i^{Xi} * 5/100$ and $M.A_i^{Xi} = C_i^{Xi} * 10/100$ respectively. The values of RW_i^{K+1}, m_i^{K+1}, PW_i^{K+1} and TE_i^{K+1} of each task in low mode at each period boundary is shown in table 7 and table 8 for clusters C_1 and C_2 respectively. The schedule generated from table 7 and table 8 is shown in figure 14 and figure 15.

TABLE 5. Cluster C_1 Task.

τ_i	P_i	C_i^{LO}	C_i^{HI}	X_i	U_i^{LO}	U_i^{HI}	$CS.A_i^{LO}$	$CS.A_i^{HI}$	$M.A_i^{LO}$	$M.A_i^{HI}$
τ_2	4	2	3	HI	0.50	0.75	0.1	0.15	0.2	0.3
τ_4	12	3	0	LO	0.25	0	0.15	0	0.3	0
τ_6	24	10	12	HI	0.42	0.46	0.5	0.6	1	1.2
τ_8	12	2	0	LO	0.17	0	0.1	0	0.2	0

TABLE 6. Cluster C_2 tasks.

τ_i	P_i	C_i^{LO}	C_i^{HI}	X_i	U_i^{LO}	U_i^{HI}	$CS.A_i^{LO}$	$CS.A_i^{HI}$	$M.A_i^{LO}$	$M.A_i^{HI}$
τ_1	4	2	0	LO	0.50	0	0.1	0	0.2	0
τ_3	12	4	7	HI	0.33	0.58	0.2	0.35	0.4	0.7
τ_5	24	10	12	HI	0.42	0.5	0.5	0.6	1	1.2
τ_7	24	3	0	LO	0.13	0	0.15	0	0.3	0

The time of overhead is added to the execution time of each task. The time of migration overhead is added when the task migrates from one core to another during run-time scheduling and the time of context switch is added when context switch occurs. In low mode, the overhead time of migration and context switch is calculated $MA_i^{LO} = c_i^{LO} * 10/100$ and $CSA_i^{LO} = c_i^{LO} * 5/100$ respectively. The execution time with overhead time units of each task at each period boundary

TABLE 7. Calculation of execution time for tasks of cluster C_1 in low mode.

Time	0	4	8	12	16	20	24
b_k	b_0	b_2	b_3	b_4	b_5	b_6	b_7
RW_2^K	0	0	0	0	0	0	0
RW_4^K	0	0	0	0	0	0	0
RW_6^K	0	- 1/3	1/3	0	- 1/3	- 2/3	0
RW_8^K	0	- 1/3	1/3	0	- 1/3	- 2/3	0
m_2^K	*	2	2	2	2	2	2
m_4^K	*	1	1	1	1	1	1
m_6^K	*	1	1	2	1	1	1
m_8^K	*	0	0	1	0	0	0
PW_2^K	*	0	0	0	0	0	0
PW_4^K	*	0	0	0	0	0	0
PW_6^K	*	2/3	1/3	0	2/3	1/3	0
PW_8^K	*	2/3	1/3	0	2/3	1/3	-0
TE_2^K	*	2	2	2	2	2	2
TE_4^K	*	1	1	1	1	1	1
TE_6^K	*	2	1	2	2	2	1
TE_8^K	*	1	0	1	1	1	0

TABLE 8. Calculation of execution time for tasks of cluster C_2 in low mode.

Time	0	4	8	12	16	20	24
b_k	b_0	b_2	b_3	b_4	b_5	b_6	b_7
RW_1^K	0	0	0	0	0	0	0
RW_3^K	0	- 1/3	- 2/3	0	- 1/3	1/3	0
RW_5^K	0	- 1/3	1/3	0	- 1/3	- 2/3	0
RW_7^K	0	- ½	0	- ½	0	- ½	0
m_1^K	*	2	2	2	2	2	2
m_3^K	*	1	1	1	1	1	2
m_5^K	*	1	1	2	1	1	1
m_7^K	*	0	0	0	0	0	0
PW_1^K	*	0	0	0	0	0	0
PW_3^K	*	2/3	1/3	0	2/3	1/3	0
PW_5^K	*	2/3	1/3	0	2/3	1/3	0
PW_7^K	*	½	0	½	0	½	0
TE_1^K	*	2	2	2	2	2	2
TE_3^K	*	2	2	1	2	1	2
TE_5^K	*	2	1	2	2	2	1
TE_7^K	*	1	0	1	0	1	0

is given in table 9 and table 10 for cluster C_1 and cluster C_2 respectively. The generated schedule form table 9 and table 10 with overhead amount for cluster C_1 and cluster C_2

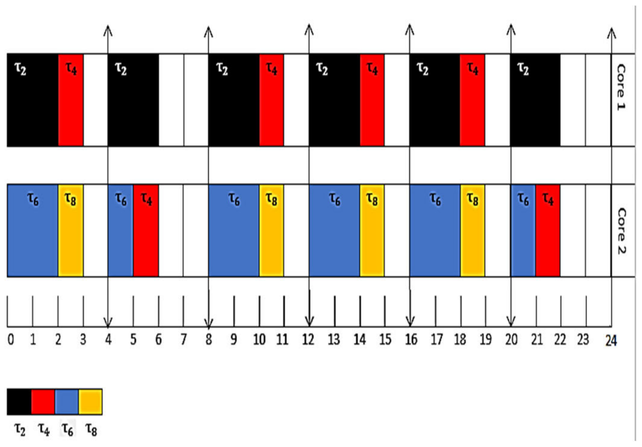


FIGURE 14. LO mode scheduling of cluster C_1.

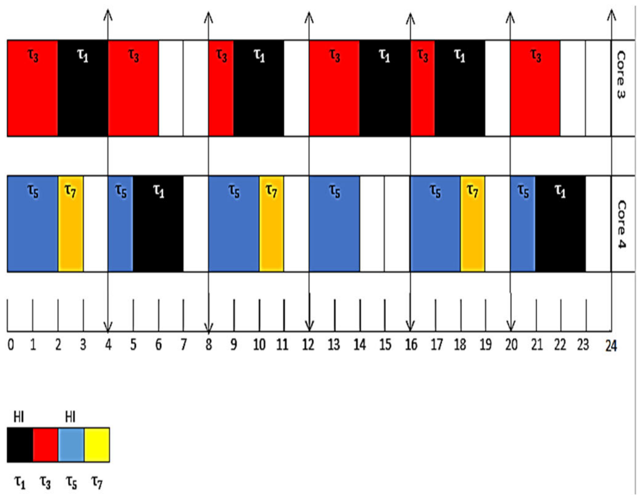


FIGURE 15. LO mode scheduling of cluster C_2.

TABLE 9. The LO mode execution time with overhead time for tasks of cluster C_1.

b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
Time	0	4	8	12	16	20	24
TE_2^K	0	2	2	2	2	2	2
TE_4^K	0	1	1.5	1.5	1	1.2	1.5
TE_6^K	0	2	1.5	2.5	2.5	2.5	1.5
TE_8^K	0	1	0	1.1	1.1	1.1	0
Total	0	6	5	7.1	6.6	6.8	5

is shown in figure 16 and figure 17, respectively. The novel MC-Bfair approach scheduled the task set given in table 1 successfully for low mode with overhead time in both cluster C_1 and cluster $C_{2.}$ The MC-BFair scheduling algorithm for low mode is shown in figure 18.

B. TOTAL EXECUTION TIME CALCULATION IN HIGH MODE

For high mode, the high mode utilization $U_i^{HI} = c_i^{HI}/\, p_i$ of high critical tasks is used in each equation for calculating the

TABLE 10. The LO mode execution time with overhead time for tasks of cluster C_2.

b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
Time	0	4	8	12	16	20	24
TE_1^K	0	2	2.2	2.2	2	2	2.2
TE_3^K	0	2	2.3	1.3	2	1.3	2.3
TE_5^K	0	2	1.5	2.5	2.5	2.5	1.5
TE_7^K	0	1	0	1.2	0	1.2	0
Total	0	7	6	7.1	6.5	6.9	6

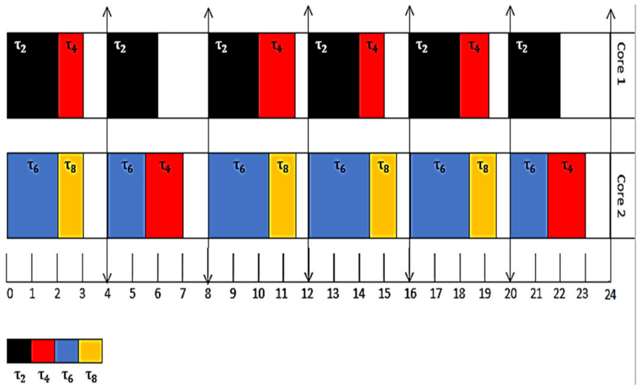


FIGURE 16. Cluster C_1 scheduling with overhead for low mode.

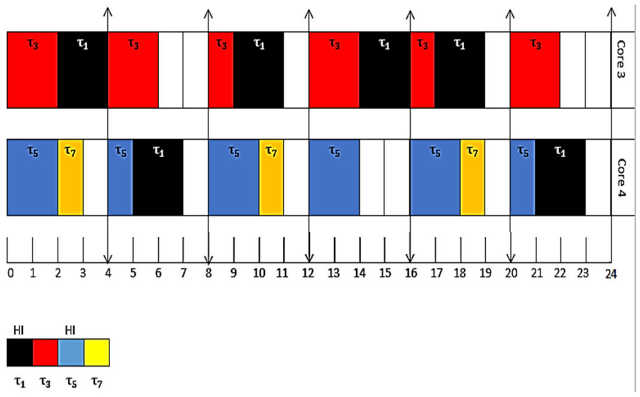


FIGURE 17. Cluster C_2 scheduling with overhead for low mode.

total execution time of each task in each period boundary. As explained earlier, the remaining work (RW_i^{K+1}), mandatory work (m_i^{K+1}), and pending work (PW_i^{K+1}) are calculated for each task to calculate the total execution time units (TE_i^{K+1}) of each task. When TE_i^{K+1} is calculated for each high task, the tasks are allocated to cores by the DCDU-WF approach for scheduling at each period boundary. The tasks that are assigned to cluster C_1 are shown in table 5 and the task of cluster C_2 is given in table 6. The overhead time of context switch and migration of each task is calculated by $CS.A_i^{Xi} = C_i^{Xi}*5/100$ and $M.A_i^{Xi} = C_i^{Xi}*10/100$ respectively. The values of RW_i^{K+1}, m_i^{K+1}, PW_i^{K+1} and TE_i^{K+1} of each task in high mode at each period boundary is shown in table 11 and table 12 for cluster C_1 and cluster C_2 respectively. The

Algorithm 2 Tasks Scheduling in Low Mode

```
1.    for each task $\tau_i \epsilon W$ do /* W is workload of tasks*/
  /* Calculate total execution time $TE_i^{K+1}$ of each task in interval $[b_k, b_{k+1})$ */
2.        $TE_i^{K+1}$=$m_i^{K+1}$+($PW_i^{K+1}$-$RW_i^{K+1}$)
3.        for ($\tau_1$.......$\tau_n$) do
4.        $RW_i^{K+1} = b_k.U_i^{LO}$ (allocated time units of task $\tau_i$ before period boundary $b_k$)
5.        $m_i^{K+1}$= max{0, $\lfloor RW_i^K + (b_{k+1} - b_k) \cdot U_i^{LO} \rfloor$}. $m_i^{K+1}$
6.        $PW_i^{K+1}$= $RW_i^K$+$(b_{k+1} - b_k) \cdot U_i^{LO} - m_i^{K+1}$.
7.        GenerateSchedule($b_k$, $b_{k+1}$);
8.        if Migration Occurs then
  /* Add migration time $M.A_i^{LO}$ to execution time of tasks
9.        $M.A_i^{LO} = C_i^{LO}$ *10/100
10.       if ContextSwitch Occurs then
  /* Add context switch time $CS.A_i^{LO}$ to execution time of tasks
11.       $CS.A_i^{LO} = C_i^{LO}$ *5/100
12.       GenerateSchedulewithoverhead($b_k$, $b_{k+1}$);
13. else
14.  return false
```

FIGURE 18. Algorithm for tasks scheduling in low mode.

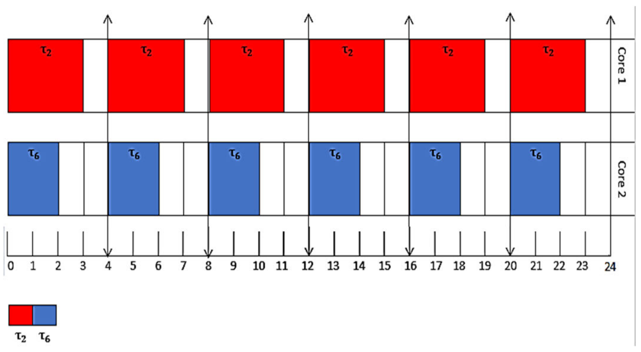


FIGURE 19. Cluster C_1 scheduling in high mode.

generated schedule form table 11 and table 12 cluster C_1 and cluster C_2 is shown in figure 19 and figure 20, respectively.

For high mode, the time of overhead is added to the task execution time. The time of migration overhead is added at each period boundary when the task migrates from one core to another during run-time scheduling and the time of context switch is added when context switch occurs. In high mode, the overhead time of migration and context switch is calculated as $MA_i^{HI} = c_i^{HI}*10/100$ and $CSA_i^{HI} = c_i^{HI}*5/100$ respectively. The overhead time added to the execution time

TABLE 11. Calculation of execution time for tasks of cluster C_1 in HI mode.

Time	0	4	8	12	16	20	24
b_k	b_0	b_2	b_3	b_4	b_5	b_6	b_7
RW_2^K	0	0	0	0	0	0	0
RW_4^K	0	0	0	0	0	0	0
RW_6^K	0	0	0	0	0	0	0
RW_8^K	0	0	0	0	0	0	0
m_2^K	*	3	3	3	3	3	3
m_4^K	*	0	0	0	0	0	0
m_6^K	*	2	2	2	2	2	2
m_8^K	*	0	0	0	0	0	0
PW_2^K	*	0	0	0	0	0	0
PW_4^K	*	0	0	0	0	0	0
PW_6^K	*	0	0	0	0	0	0
PW_8^K	*	0	0	0	0	0	0
TE_2^K	*	3	3	3	3	3	3
TE_4^K	*	0	0	0	0	0	0
TE_6^K	*	2	2	2	2	2	2
TE_8^K	*	0	0	0	0	0	0

TABLE 12. Calculation of execution time for tasks of cluster C_2 in HI mode.

Time	0	4	8	12	16	20	24
b_k	b_0	b_2	b_3	b_4	b_5	b_6	b_7
RW_1^K	0	0	0	0	0	0	0
RW_3^K	0	- 2/3	2/3	0	- 2/3	2/3	0
RW_5^K	0	0	0	0	0	0	0
RW_7^K	0	0	0	0	0	0	0
m_1^K	*	0	0	0	0	0	0
m_3^K	*	2	1	3	2	1	3
m_5^K	*	2	2	2	2	2	2
m_7^K	*	0	0	0	0	0	0
PW_1^K	*	0	0	0	0	0	0
PW_3^K	*	1/3	2/3	0	1/3	2/3	0
PW_5^K	*	0	0	0	0	0	0
PW_7^K	*	0	0	0	0	0	0
TE_1^K	*	0	0	0	0	0	0
TE_3^K	*	3	1	3	3	1	3
TE_5^K	*	2	2	2	2	2	2
TE_7^K	*	0	0	0	0	0	0

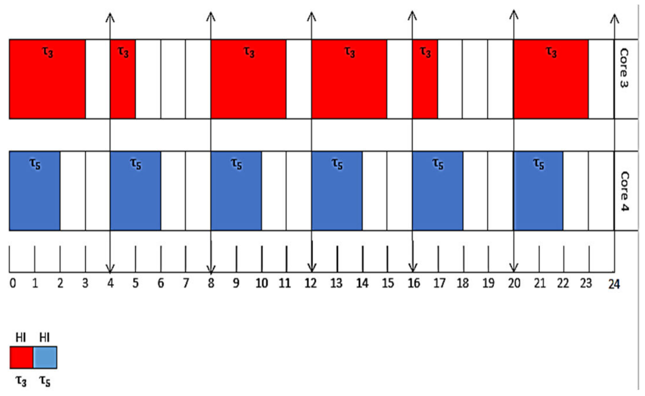


FIGURE 20. Cluster C_2 scheduling in high mode.

TABLE 13. The HI mode execution time with overhead time for tasks of cluster C_1.

b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
Time	0	4	8	12	16	20	24
TE_2^K	0	3	3	3	3	3	3
TE_4^K	0	0	0	0	0	0	0
TE_6^K	0	2	2.6	2.6	2.6	2.6	2.6
TE_8^K	0	0	0	0	0	0	0
Total	0	5	5.6	5.6	5.6	5.6	5.6

TABLE 14. The HI mode execution time with overhead time for tasks of cluster C_2.

b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
Time	0	4	8	12	16	20	24
TE_1^K	0	0	0	0	0	0	0
TE_3^K	0	3	1.4	3.4	3	1.4	3.4
TE_5^K	0	2	2.6	2.6	2.6	2.6	2.6
TE_7^K	0	0	0	0	0	0	0
Total	0	5	4	6	5.6	4	6

of tasks is given in table 13 table 14 for cluster C_1 and cluster C_2 respectively. The schedule generated from table 13 and table 14 is shown in figure 21 and figure 22. The novel MC-Bfair approach scheduled the task set given in table 1 successfully for high mode with overhead time in both cluster C_1 and cluster C_2. The MC-BFair scheduling algorithm for high mode is shown in figure 23.

C. TOTAL EXECUTION TIME CALCULATION IN MODE CHANGE

For the mode switch from low critical to high critical, the task set given in table 1 is schedulable for all mode switch times s_t of high critical tasks. The mode switch can occur in cluster C_1 by HI task t_2 on 2, 6, 10, 14, 18 and 22 time units while task t_6 can cause mode switch on 21 time unit. Similarly the mode switch can occur in cluster C_2 by HI task t_3 on 9 and 22 time

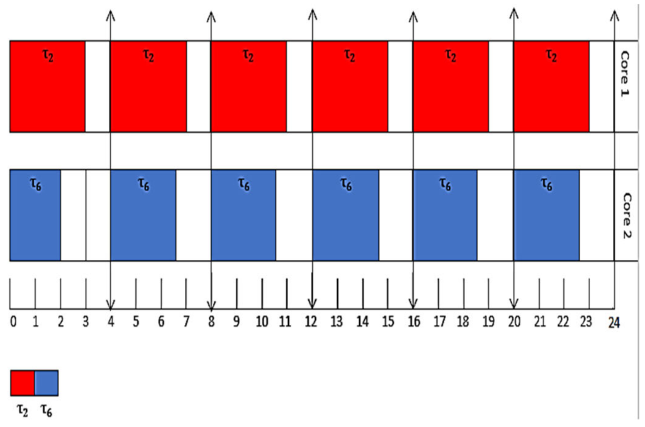


FIGURE 21. Cluster C_1 scheduling with overhead for high mode.

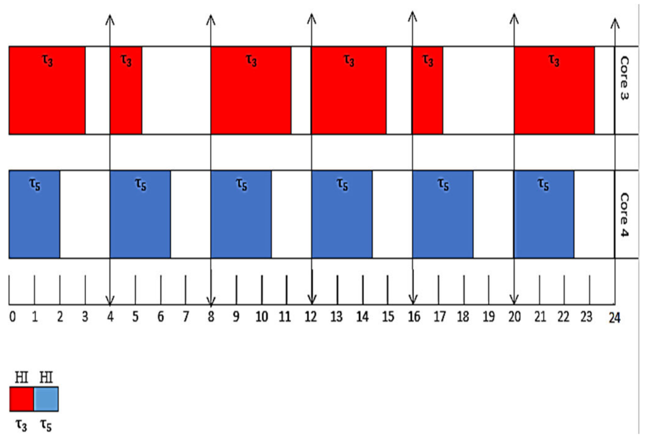


FIGURE 22. Cluster C_2 scheduling with overhead for high mode.

units, while t_5 can cause mode switch on 21 time unit. These switch times are the possible mode switch time units of high critical tasks of cluster C_1 and cluster C_2. If a high critical task causes mode switch at any time of mode switch time of cluster C_1 and cluster C_2, the values of remaining work RW_i^{K+1}, mandatory time unit sm_i^{K+1}, pending work PW_i^{K+1} and total execution time of high tasks is also calculated. For calculating RW_i^{K+1}, m_i^{K+1} and PW_i^{K+1}, the high mode utilization $U_i^{HI} = c_i^{HI}/p_i$ is used for a HI task at the time of mode change and after mode change. For all tasks before mode switch, the low mode utilization $U_i^{LO} = c_i^{LO}/p_i$ is used for calculating RW_i^{K+1}, m_i^{K+1} and PW_i^{K+1} in each period boundary. If the deadline of a high critical task exists in a period boundary after mode change, the total execution time TE_i^{K+1} of HI tasks is calculated as follows,

$$TE_i^{K+1} = \left\lceil m_i^{K+1} + (PW_i^{K+1} - RW_i^{K+1}) \right\rceil + (c_i^{HI} - \sum\nolimits_{bk_o \to bk_n} TE_i^{LO})$$

where, bk_o is that period boundary in which these tasks start their execution and bk_n is the period boundary before the mode switch. While, $\sum_{bk_o \to bk_n} TE_i^{LO}$ is the sum of the total execution time of high critical tasks in the low mode in period boundaries bk_o to bk_n. The above equation is also used for calculating total execution time TE_i^{K+1} of an HI task in period boundaries after the mode switch which contain the deadline of a high critical tasks. When a task τ_2 causes a mode change at time 14, the tasks τ_4 and τ_8 are discarded and only τ_2 and τ_6 are scheduled in high mode. The execution time of tasks for cluster C_1 is calculated in table 15 while the schedule generated from table 15 of cluster C_1 for mode switch time at time unit 14 is shown in figure 24.

Algorithm 3 Tasks Scheduling in High Mode

```
1.      for each task $\tau_i \epsilon W$ do /* W is workload of tasks*/
     /* Calculate $TE_i^{K+1}$ of each task in interval $[b_k, b_{k+1})$ */
2.        $TE_i^{K+1}$=$m_i^{K+1}$+($PW_i^{K+1}$-$RW_i^{K+1}$)
3.        for ($\tau_1$.......$\tau_n$) do
4.          $RW_i^{K+1}$ = $b_k.U_i^{HI}$ - (allocated time units of task $\tau_i$ before period boundary $b_k$)
5.          $m_i^{K+1}$= max{0, $\lfloor RW_i^K + (b_{k+1} - b_k) \cdot U_i^{HI} \rfloor$}. $m_i^{K+1}$
6.          $PW_i^{K+1}$= $RW_i^K + (b_{k+1} - b_k) \cdot U_i^{HI} - m_i^{K+1}$.
7.        GenerateSchedule($b_k$, $b_{k+1}$);
8.        if Migration Occurs then
     /* Add migration time $M.A_i^{HI}$ to execution time of tasks
9.          $M.A_i^{HI}$ = $C_i^{HI}$ *10/100
10.       if ContextSwitch Occurs then
     /* Add context switch time $CS.A_i^{HI}$ to execution time of tasks
11.         $CS.A_i^{HI}$ = $C_i^{HI}$ *5/100
12.       GenerateSchedulewithoverhead($b_k$, $b_{k+1}$);
13.   else
14.   return false
```

FIGURE 23. Algorithm for tasks scheduling in high mode.

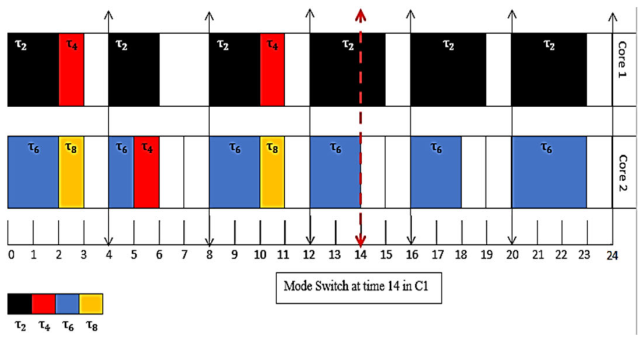


FIGURE 24. Cluster C_1 scheduling for mode change.

The time of migration overhead is added when the task migrates among cores during scheduling and the time of context switch is added when context occurs. The overhead time added to the execution time of each task is given in table 16 for cluster C_1. The generated schedule from table 16 is for the cluster C_1 during mode change is shown in figure 25.

TABLE 15. Calculation of execution time for tasks of cluster C_1 for mode change.

Time	0	4	8	12	16	20	24
b_k	b_0	b_2	b_3	b_4	b_5	b_6	b_7
RW_2^K	0	0	0	0	0	0	0
RW_4^K	0	0	0	0	0	0	0
RW_6^K	0	- 1/3	1/3	0	1/3	1/6	0
RW_8^K	0	0	0	0	0	0	0
m_2^K	*	2	2	2	3	3	3
m_4^K	*	1	1	1	0	0	0
m_6^K	*	1	1	2	1	2	2
m_8^K	*	1	1	1	0	0	0
PW_2^K	*	0	0	0	0	0	0
PW_4^K	*	0	0	0	0	0	0
PW_6^K	*	2/3	1/3	0	5/6	1/6	0
PW_8^K	*	0	0	0	0	0	0
TE_2^K	*	2	2	2	3	3	3
TE_4^K	*	1	1	1	0	0	0
TE_6^K	*	2	1	2	2	2	3
TE_8^K	*	1	1	1	0	0	0

TABLE 16. The mode change execution time with overhead time units for tasks of cluster C_1.

b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
Time	0	4	8	12	16	20	24
TE_2^K	0	2	2	2	3	3	3
TE_4^K	0	1	1.45	1.45	0	0	0
TE_6^K	0	2	1.5	2.5	2.6	2.6	3.6
TE_8^K	0	1	0	1.1	0	0	0
Total	0	5	5	7.1	5.6	5.6	6.6

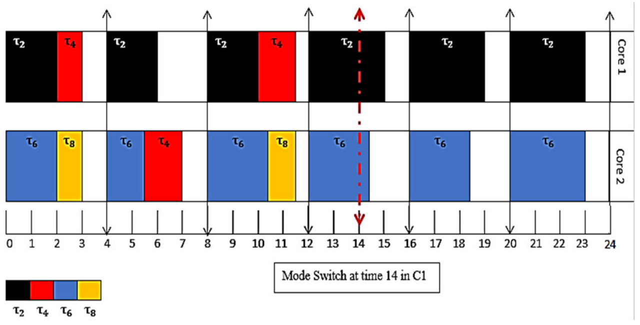


FIGURE 25. Cluster C_1 scheduling with overhead for mode change.

Figure 22 shows the mode switch occurred at time unit = 9 in cluster C_2 by a HI task τ_5. After mode change, tasks τ_1 and τ_7 are discarded from further execution and only HI tasks τ_3 and τ_5 are scheduled in high mode. The total execution time of tasks is calculated in table 17. The schedule generated from table 17 for the mode switch is shown in figure 26.

TABLE 17. Calculation of execution time for tasks of cluster C_2 for mode change.

Time	0	4	8	12	16	20	24
b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
RW_1^K	0	0	0	0	0	0	0
RW_3^K	0	- 1/3	- 2/3	0	1/3	- 1/3	0
RW_5^K	0	- 1/3	1/3	0	0	0	0
RW_7^K	0	- ½	0	0	0	0	0
m_1^K	*	2	2	0	0	0	0
m_3^K	*	1	1	1	2	2	2
m_5^K	*	1	1	2	2	2	2
m_7^K	*	0	0	0	0	0	0
PW_1^K	*	0	0	0	0	0	0
PW_3^K	*	2/3	1/3	0	1/3	2/3	0
PW_5^K	*	2/3	1/3	1/3	0	0	0
PW_7^K	*	½	0	0	0	0	0
TE_1^K	*	2	2	0	0	0	0
TE_3^K	*	2	2	3	2	3	2
TE_5^K	*	2	1	3	2	2	2
TE_7^K	*	1	0	0	0	0	0

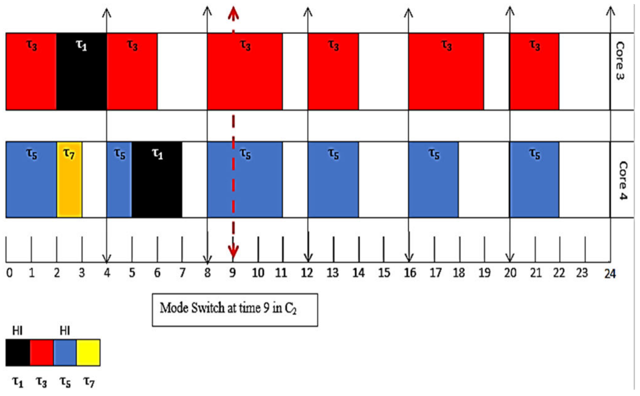


FIGURE 26. Cluster C_2 scheduling for mode change.

The time of overhead added to the tasks execution time task for mode change in cluster C_2 is given in table 18. The time of overhead of migration and context switches of a task τ_i in high mode is calculated as $MA_i^{HI} = c_i^{HI} * 10/100$ and $CSA_i^{HI} = c_i^{HI} * 5/100$ for migration and context switch respectively and is shown in table 18. For mode change, the schedule obtained from table 18 is shown in figure 27. For both cluster C_1 and cluster C_2, the tasks given in table 1 is scheduled by the novel MC-Bfair scheduling algorithm during mode change. The The MC-BFair scheduling algorithm for mode change is shown in figure 28.

VII. EXPERIMENTAL EVALUATION

The efficiency of the cluster-based scheduling technique is shown by experimental evaluation. The experiments are evaluated on mixed-criticality task sets of MC systems.

TABLE 18. The mode change execution time with overhead time units for tasks of cluster C_2.

b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6
Time	0	4	8	12	16	20	24
TE_1^K	0	2	2.2	0	0	0	0
TE_3^K	0	2	2.2	3.35	2	3.35	2.35
TE_5^K	0	2	1.5	3.6	2.6	2.6	2.6
TE_7^K	0	1	0	0	0	0	0
Total	0	7	5.9	6.95	4.6	5.95	4.95

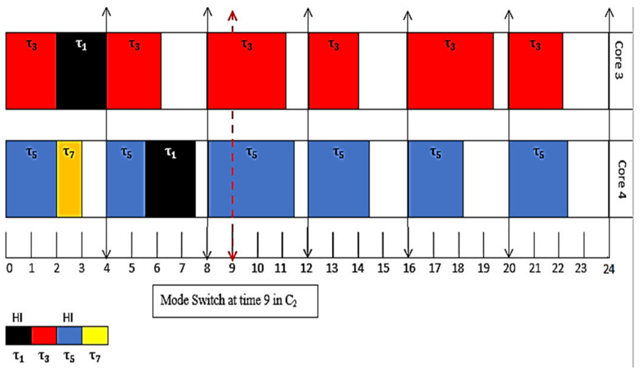


FIGURE 27. Cluster C_2 scheduling with overhead for mode change.

We compared the proposed approach with other multicore scheduling approaches i.e., CB-FP [18], MC-EKG-VD-1 [9], partitioned and global approaches to show the effectiveness of our approach. The same time of context switch and migration overhead of MC-Bfair approach is considered for CB-FP and MC-EKG-VD-1 approach. The results obtained from experiments show the better performance of cluster-based technique as compared to CB-FP, MC-EKG-VD-1, partitioned and global approaches.

A. GENERATION OF MC TASK SETS

For experimental evaluation, the MC task sets are randomly generated, which are controlled by four parameters i.e., P^{HI}, R^{HI}, $C_i(LO)^{max}$ and P^{max}. Where, P^{HI} represents the probability of HI tasks in a task set, the parameter R^{HI} is used to represent the ratio between LO and HI WCET of high critical tasks in a task set, $C_i(LO)^{max}$ denotes the maximum low mode WCET of MC tasks and P^{max} represents the maximum period of MC tasks. A task set is created through the following parameters,.

- Using P^{HI}, if X_i = HI, the task is a HI task otherwise X_i = LO for LO critical task
- C_i(LO) is generated within range [1, $C_i(LO)^{max}$], where $C_i(LO)^{max} < P^{max}$.
- C_i(HI) of a task is generated by C_i(HI) = C_i(LO). R^{HI}.
- The task's period P_i is generated within [1, P^{max}] range.

Algorithm 4 Tasks Scheduling in Mode Change

```
1.  for each task $\tau_i \epsilon W$ do /* W is workload of tasks*/
/* Calculate total execution time $TE_i^{k+1}$ of each task in interval $[b_k, b_{k+1})$ */
2.    if $\tau_i \rightarrow$ carry _in /* carry in are those high critical tasks execution
starts in low mode but still need further execution in high mode*/
3.    $TE_i^{K+1} = \lceil m_i^{K+1} + (PW_i^{K+1} - RW_i^{K+1}) \rceil + (c_i^{HI} - \sum_{bk_o \rightarrow bk_n} TE_i^{LO})$
4.  else
5.    $TE_i^{K+1} = m_i^{K+1} + (PW_i^{K+1} - RW_i^{K+1})$
6.    for $(\tau_1 \ldots\ldots \tau_n)$ do
7.    $RW_i^{K+1} = b_k . U_i^{HI}$ - (allocated time units of task $\tau_i$ before period
boundary $b_k$)
8.    $m_i^{K+1}$ = max{0, $\lfloor RW_i^K + (b_{k+1} - b_k) \cdot U_i^{HI} \rfloor$}. $m_i^{K+1}$
9.    $PW_i^{K+1} = RW_i^K + (b_{k+1} - b_k) \cdot U_i^{HI} - m_i^{K+1}$.
10.     GenerateSchedule($b_k$, $b_{k+1}$);
11.     if Migration Occurs then
 /* Add migration time $M.A_i^{HI}$ to execution time of tasks
12.       $M.A_i^{HI} = C_i^{HI}$ *10/100
13.      if ContextSwitch Occurs then
   /* Add context switch time $CS.A_i^{HI}$ to execution time of tasks
14.        $CS.A_i^{HI} = C_i^{HI}$ *5/100
15.        GenerateSchedulewithoverhead($b_k$, $b_{k+1}$);
16.        else
17.        return false
```

FIGURE 28. Algorithm for tasks scheduling in mode change.

The MC task set is randomly generated by means of target utilization U = U^{LM}. The generation of an MC task set with exact target utilization is difficult, so the task set is generated between U^{min} and U^{max}. Whereas U^{min} = U – 0.005, and U^{max} = U + 0.005. The generated task set is discarded if the utilization in low or high mode is bigger than the number of cores of the MC real-time system.

B. RESULT ANALYSIS

The experiments are carried out for MC task sets using different parameters i.e., $R^{HI} \in \{1, 2, 3, 4, 5\}$, $P^{HI} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, $P^{max} = \{60\}$ and n $\in \{9, 10, 12, 15, 18\}$. Where, $R^{HI} = C_i$(HI)/C_i(LO), is used for the ratio between HI and LO WCET. The P^{HI} Shows percentage of HI tasks in a task set and n represents the number of tasks in a task set. For result analysis, at least 300 MC task sets are used at each point of figure 29, figure 30, figure 31, and figure 32. The results obtained from the cluster-based technique are compared with CB-FP, MC-EKG-VD-1, global and partitioned techniques.

Under target utilization U = {0.25, 0.375, 0.5, 0.55, 0.6, 0.65, 0.70, 0.75}, figure 29 displays the percentage of schedulable MC task sets for four cores using the cluster-based, CB-FP, partitioned and global approaches. As shown in figure 29, the cluster-based technique scheduled 100%

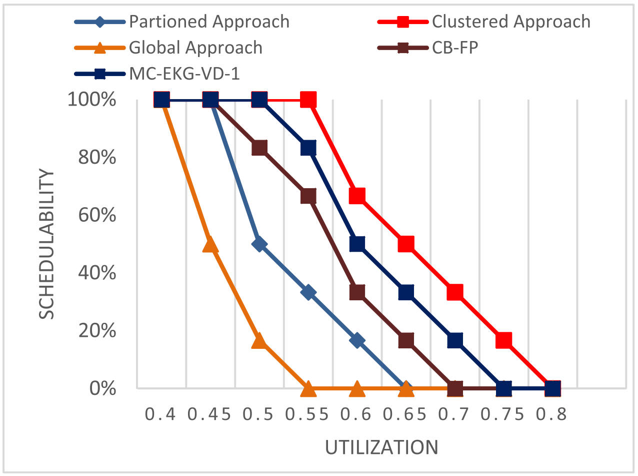


FIGURE 29. Utilization vs schedulability for m = 4 and n = 10.

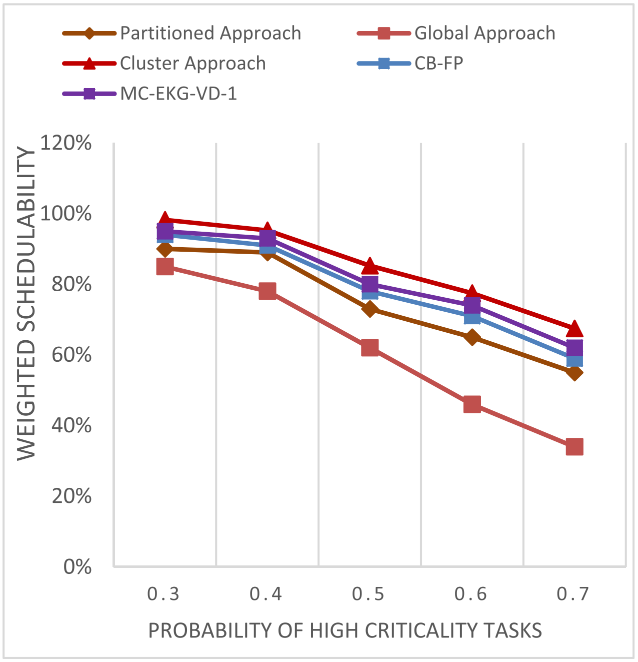


FIGURE 30. High criticality probability vs weighted schedulability for m = 4 and n = 10.

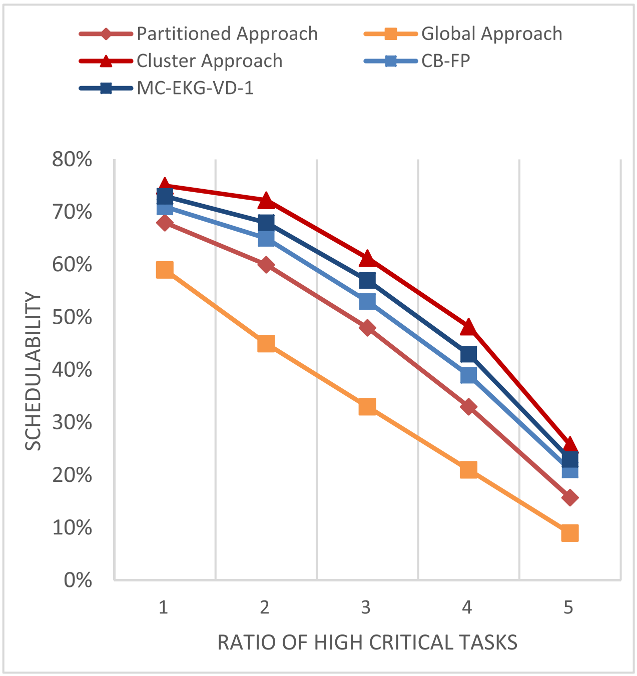


FIGURE 31. LO and HI WCET ratios vs weighted schedulability for m = 4 and n = 10.

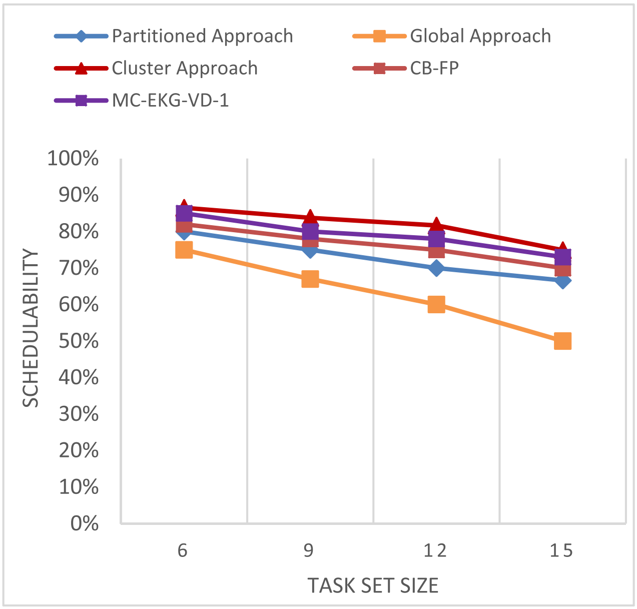


FIGURE 32. Task set size vs weighted schedulability for m = 4.

task sets up to U = 0.55, while the partitioned, and CB-FP techniques scheduled 100% of generated task sets up to U = 0.45 and the global technique scheduled 100% of the generated task sets up to U = 0.4. The percentage of schedulable task sets are decreased onwards by increasing the target utilization, but still, the performance of our proposed cluster-based technique is higher than the other techniques up to U = 0.79. At target utilization U = 0.50, the cluster-based and MC-EKG-VD-1 techniques schedule 100% MC task sets while CB-FP technique schedule 83.3%, partitioned technique schedule 50% and global technique schedule 16.7% task sets respectively. Similarly, for U = 0.6 the proposed technique schedule 66.7% task sets while MC-EKG-VD-1, CB-FP, partitioned and global techniques schedule 50%, 33.3%, 16.7% and 0% task sets respectively. Figure 29 also shows, if the task sets utilization is increased, the schedulability of task sets has shown a decrease for all techniques.

In a task set, the impact of the probability of high critical tasks on schedulability is shown in figure 30. The probability of high critical tasks for target utilizations gives a 3-diamesional plot. To reduce 3-diamensional plots to 2-diamensionals plots, we used weighted schedulability measures [33]. The weighted acceptance ratio under the given target utilization is obtained as $\sum_{Ui\epsilon u} Ui.\ \mathrm{A}(\mathrm{U_i})/\sum_{Ui\epsilon u} Ui$, where A(Ui) represents the acceptance ratio for target utilization U_i. For probability, at least 15 $A(U_i)$ are computed for different utilizations U_i. Figure 30 shows that if the percentage of HI critical tasks increases, the percentage of schedulable task sets decreases due to an increase in the utilization of MC workload. As stated previously, the proposed cluster-based technique performs better than the other techniques using the parameter P^{HI}.

The impact of the ratio between $C_i(HI)$ and $C_i(LO)$ execution times on schedulability is shown in figure 31. For parameter R^{HI}, at least 15 acceptance ratios are computed for different utilizations U_i. Figure 31 shows that if the ratio between $C_i(HI)$ and $C_i(LO)$ increases, the task set schedulability decreases, because if the R^{HI} between $C_i(HI)$ and $C_i(LO)$ increases, it multiplies the utilization of a given task set, which decreases the task sets schedulability. The proposed technique performs better than MC-EKG-VD-1 [9], CB-FP [18], global and partitioned techniques in terms of high utilization of task sets due to increasing value of parameter R^{HI}.

The impact of task set size on schedulability is shown in figure 32. For each variable parameter n, at least 15 $A(U_i)$ with different target utilization are computed. For n = 6, 9, 12 and 15, the proposed cluster-based technique has higher weighted schedulability as compared to the other three techniques i.e., CB-FP, partitioned and global techniques. The schedulability of our cluster-based approach is 84% for n = 9, while the MC-EKG-VD-1, CB-FP, partitioned technique and global technique have shown 80%, 78%, 75% and 67% weighted schedulability respectively. As compared to CB-FP, global and partitioned scheduling techniques, the weighted schedulability of our cluster-based technique is higher for n = 6, 9, 12, and 15.

VIII. CONCLUSION AND FUTURE WORK

In this research work, a cluster-based technique is used for scheduling the task set of multicore MC systems. In this study, the overhead time of task migration and context switches is reduced. The cores are divided into clusters and MC tasks are distributed in these clusters by means of the DCDU-WF heuristic approach [8]. A novel mixed-criticality cluster-based boundary fair (MC-Bfair) scheduling algorithm is used for the scheduling of tasks on cores within clusters. The boundary fair algorithm minimizes the migrations and context switches. The system initially executes the tasks in low mode. The MC tasks are executed up to $C_i(LO)$ time units, but when a HI task is executed upto $C_i(LO)$ execution time in any cluster and still needs further execution i.e $C_i(HI)-C_i(LO)$, the system is changed to high mode. In high mode, all LO tasks are dropped and HI tasks are only executed up to $C_i(HI)$ through MC-Bfair algorithm in each cluster. This approach has reduced the overhead caused by context switches and migrations of tasks. For each migration of a task or context switch, the overhead of migration and context switch is added to the execution time of the MC task. In low critical mode, the low mode context switch and migration overhead time are added to the task's execution time while in high critical mode, the high mode context switch and migration overhead time is added to the execution time of the task for each context switch and migration respectively. The results obtained from experiments show the better performance of the cluster-based technique as compared to the MC-EKG-VD-1, CB-FP [18], partitioned and global techniques.

As a future work, the cluster-based scheduling of multicore MC systems with overhead reduction can be extended to a system having more criticality levels. It is also needed to examine fault tolerance in cluster-based technique. As a future work we will execute MC tasks on fraction rate of processors like MC-Fluid [20] to check the performance of our cluster-based scheduling approach as compared to partitioned and global scheduling approaches.

APPENDIX
LIST OF EQUATIONS

Equation	Definition
$U_i^{LO} = C_i(LO)/P_i$	Utilization of task in low mode
$U_i^{HI} = C_i(HI)/P_i$	Utilization of task in high mode
$U^{LM} = \sum_{\tau i \in W} U_i^{LO}$	Summation of utilizations of all tasks in low mode
$U^{HM} = \sum_{\tau i \in W} U_i^{HI}$	Summation of utilizations of all high tasks in high mode
UC_k^{LM}	Utilization of cluster in low mode
UC_k^{HM}	Utilization of cluster in high mode
RW_i^{K+1}	The remaining work of task τ_i at boundary b_k as in (Zhu et al., 2011).
$m_i^{K+1} = \max\{0, \lfloor RW_i^K + (b_{k+1} - b_k) \cdot U_i \rfloor\}$	The mandatory time units that have to be assigned to task τ_i to keep its remaining work within one time unit
$PW_i^{K+1} = RW_i^K + (b_{k+1} - b_k) \cdot U_i - m_i^{K+1}$	The pending work is the corresponding decimal part of the summation of remaining work at b_k and the work to be done
$TE_i^{K+1} = m_i^{K+1} + (PW_i^{K+1} - RW_i^{K+1})$.	The total execution time of a task in interval $[b_k, b_{k+1})$
$CS.A_i^{Xi} = C_i^{Xi} *5/100$	Context switch overhead amount at criticality level X_i
$M.A_i^{Xi} = C_i^{Xi} *10/100.$	Migration overhead amount at criticality level X_i
$TE_i^{K+1} = \left\lceil m_i^{K+1} + (PW_i^{K+1} - RW_i^{K+1}) \right\rceil + (c_i^{HI} - \sum_{bk_o \to bk_n} TE_i^{LO})$	Total execution time units of tasks at period boundary in which mode switch is occurred from low to high mode.

LIST OF ACRONYMS

Acronym	Explanation
MC	Mixed Criticality
MCS	Mixed Criticality System
DCDU	Decreasing Criticality Decreasing Utilization
MC-Bfair	Mixed-Criticality Clustered-based Boundary Fair
CA	Certification Authorities
WCET	Worst-Case Execution Time
EDF	Earliest Deadline First
RM	Rate Monotonic
MPVD	Mixed-Criticality Partitioning With Virtual Deadline
EDF-VD	EDF With Virtual Deadline
FpEDF	Fixed-Priority EDF
DP-fair	Deadline Partitioning Fair
Bfair	Boundary Fair
CB-FP	Cluster-Based Fixed Priority

REFERENCES

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

[2] M. Staron, "AUTOSAR (automotive open system architecture)," in *Automotive Software Architectures*. Cham, Switzerland: Springer, 2021, pp. 97–136.

[3] 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.

[4] S. Ramanathan and A. Easwaran, "Utilization difference based partitioned scheduling of mixed-criticality systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 238–243.

[5] M. Naghibzadeh, P. Neamatollahi, R. Ramezani, A. Rezaeian, and T. Dehghani, "Efficient semi-partitioning and rate-monotonic scheduling hard real-time tasks on multi-core systems," in *Proc. 8th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2013, pp. 85–88.

[6] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 166–175.

[7] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger, "A hybrid real-time scheduling approach for large-scale multicore platforms," in *Proc. 19th Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2007, pp. 247–258.

[8] A. Ali and K. H. Kim, "Cluster-based multicore real-time mixed-criticality scheduling," *J. Syst. Archit.*, vol. 79, pp. 45–58, Sep. 2017.

[9] C. Yang, H. Wang, J. Zhang, and L. Zuo, "Semi-partitioned scheduling of mixed-criticality system on multiprocessor platforms," *J. Supercomput.*, vol. 78, no. 5, pp. 6386–6410, Apr. 2022.

[10] 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.

[11] O. R. Kelly, H. Aydin, and B. Zhao, "On partitioned scheduling of fixed-priority mixed-criticality task sets," in *Proc. IEEE 10th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Nov. 2011, pp. 1051–1059.

[12] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Syst.*, vol. 50, no. 1, pp. 48–86, Jan. 2014.

[13] C. Gu, N. Guan, Q. Deng, and W. Yi, "Partitioned mixed-criticality scheduling on multiprocessor platforms," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2014, pp. 1–6.

[14] K. Nagalakshmi and N. Gomathi, "Criticality-cognizant clustering-based task scheduling on multicore processors in the avionics domain," *Int. J. Comput. Intell. Syst.*, vol. 11, no. 1, pp. 219–237, 2018.

[15] R. M. Pathan, "Schedulability analysis of mixed-criticality systems on multiprocessors," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 309–320.

[16] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proc. 22nd IEEE Real-Time Syst. Symp. (RTSS)*, Jul. 2001, pp. 193–202.

[17] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 119–128.

[18] S. K. Baruah, B. Vincenzo, D. Gianlorenzo, M. S. Alberto, S. van der Ster, and S. Leen, "Mixed-criticality scheduling of sporadic task systems," in *Proc. Eur. Symp. Algorithms*. Berlin, Germany: Springer, 2011, pp. 555–566.

[19] S. K. Baruah, "Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors," *IEEE Trans. Comput.*, vol. 53, no. 6, pp. 781–784, Jun. 2004.

[20] J. Lee, K.-M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee, "MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2014, pp. 41–52.

[21] Y. Sharma and S. Moulik, "RT-SEAT: A hybrid approach based real-time scheduler for energy and temperature efficient heterogeneous multicore platforms," *Results Eng.*, vol. 16, Dec. 2022, Art. no. 100708.

[22] S. Moulik, Z. Das, R. Devaraj, and S. Chakraborty, "SEAMERS: A semi-partitioned energy-aware scheduler for heterogeneous multicore real-time systems," *J. Syst. Archit.*, vol. 114, Mar. 2021, Art. no. 101953.

[23] Y. Sharma and S. Moulik, "CETAS: A cluster based energy and temperature efficient real-time scheduler for heterogeneous platforms," in *Proc. 37th ACM/SIGAPP Symp. Appl. Comput.*, Apr. 2022, pp. 501–509.

[24] S. Moulik, Z. Das, and G. Saikia, "CEAT: A cluster based energy aware scheduler for real-time heterogeneous systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2020, pp. 1815–1821.

[25] Y. Sharma and S. Moulik, "FATS-2TC: A fault tolerant real-time scheduler for energy and temperature aware heterogeneous platforms with two types of cores," *Microprocessors Microsystems*, vol. 96, Feb. 2023, Art. no. 104744.

[26] S. Moulik, "RESET: A real-time scheduler for energy and temperature aware heterogeneous multi-core systems," *Integration*, vol. 77, pp. 59–69, Mar. 2021.

[27] Y. Sharma, S. Chakraborty, and S. Moulik, "ETA-HP: An energy and temperature-aware real-time scheduler for heterogeneous platforms," *J. Supercomput.*, vol. 78, no. 8, pp. 1–25, May 2022.

[28] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Dept. Comput. Sci., Univ. York, Heslington, U.K., Tech. Rep. YCS-164, 1991.

[29] A. Burns and R. I. Davis, *Mixed Criticality Systems—A Review*. New York, NY, USA, 2022.

[30] W. Zhang, S. Teng, Z. Zhu, X. Fu, and H. Zhu, "An improved least-laxity-first scheduling algorithm of variable time slice for periodic tasks," in *Proc. 6th IEEE Int. Conf. Cognit. Informat.*, Aug. 2007, pp. 548–553.

[31] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *Proc. 22nd Euromicro Conf. Real-Time Syst.*, Jul. 2010, pp. 3–13.

[32] D. Zhu, X. Qi, D. Mossé, and R. Melhem, "An optimal boundary fair scheduling algorithm for multiprocessor real-time systems," *J. Parallel Distrib. Comput.*, vol. 71, no. 10, pp. 1411–1425, Oct. 2011.

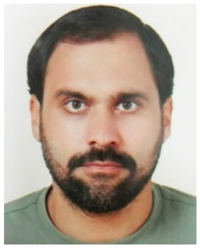
[33] A. Bastoni, B. B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," in *Proc. OSPERT*, no. 10, 2010, pp. 33–44.

AMJAD ALI received the B.Sc. and M.S. degrees in computer science from the University of Peshawar, Pakistan, in 1997 and 2010, respectively, the M.Sc. degree in computer science from Gomal University, Pakistan, 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-tolerance computing.

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 was a Postdoctoral Fellow and an Assistant Professor with the Department of Computer Engineering, Kyung Hee University. He is currently an Associate Professor with the College of Technological Innovation, Zayed University, Abu Dhabi, United Arab Emirates, that he joined in August 2014. He is 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 journal and conference papers in highly reputed venues. He has delivered keynote speeches, invited talks, guest lectures, and short courses in many universities. He serving as a reviewer, a program committee member, and the guest editor of many conferences and journals. He and his team have secured several national and international awards in different competitions.

SHAHID IQBAL received the B.S. and M.S. degrees in computer science from the University of Swat, Swat, Pakistan, in 2016 and 2022, respectively. His research interests include real-time systems, power-aware computing, and fault-tolerance computing.

OMAR ALFANDI (Member, IEEE) received the M.Sc. degree in telecommunication engineering from the Technical University of Kaiserslautern, Germany, in 2005, and the Ph.D. degree in computer engineering from the Georg-August University of Göttingen, Germany, in 2009. From 2009 to 2011, he enjoyed a Postdoctoral Fellowship with the Telematics Research Group and he founded the Research and Education Sensor Laboratory, where he is currently a Laboratory Advisor. Before that, he carried his Ph.D. research as part of the Industry, Academia, and Research Centers collaboration with the European Union (EU) Project. In 2015, he was appointed as the Assistant Dean of the Abu Dhabi Campus. He was the Package Leader of EU DAIDALOS II, 6th Framework Project. He is an Associate Professor with the College of Technological Innovation, Zayed University. He is the Co-Founder and the Co-Director of the Sensors and Mobile Applications Research and Education (SMART) Laboratory, CTI. He published numerous articles on Authentication Framework for 4G Communication Systems, *Future Internet*, Trust and Reputation Systems in mobile ad hoc, and Sensor Networks. His current research interests include the Internet of Things (IoT), security in next generation networks, smart technologies, security engineering, and mobile and wireless communications.



BASHIR HAYAT received the M.S. degree in computer science from the 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-si, South Korea, in 2020. He is currently a Faculty Member with the Institute of Management Sciences, Peshawar, Pakistan, that he joined in 2011.

MUHAMMAD HAMEED SIDDIQI received the Bachelor of Computer Science degree (Hons.) from the Islamia College Peshawar (Chartered University), Peshawar, Khyber Pakhtunkhwa, Pakistan, in 2007, and the master's and Ph.D. degrees from the Ubiquitous Computing (UC) Laboratory, Department of Computer Engineering, Kyung Hee University, Suwon-si, South Korea, in 2012 and 2016, respectively. He was a Graduate Assistant with Universiti Teknologi PETRONAS, Malaysia, from 2008 to 2009. He was also a Postdoctoral Research Scientist with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon-si, from March 2016 to August 2016. He was an Assistant Professor with the Department Computer Science, Jouf University, from September 2016 to October 2020. He has been an Associate Professor with the Department of Computer Science, Jouf University, Sakaka, Saudi Arabia, since November 2020. He published more than 85 research articles in highly reputable international journals and conferences. His research interests include image processing, pattern recognition, machine intelligence, activity recognition, and facial expression recognition. He is also a reviewer of different journals and conferences.

ADIL KHAN (Member, IEEE) is currently a seasoned Professor and a Prolific Researcher in machine learning. With a robust background in machine learning, deep learning, and representation learning, he is passionately dedicated to both pedagogy and innovative research in the realm of artificial intelligence. His research journey started in South Korea, in 2006, where he concentrated on human activity recognition through wearable sensors. His groundbreaking discoveries were published in reputable journals and employed by leading technology firms for their healthcare applications. Over his career, he has undertaken more than ten research projects, obtaining substantial funding, and has published in excess of 90 research articles. He has supervised more than two dozen Ph.D. and M.S. students to completion. His expertise and experience are not limited to a single geographic location. His academic career has spanned across various prestigious universities in South Korea, Denmark, Russia, United Arab Emirates, Switzerland, and U.K. These diverse collaborations and experiences have enriched his knowledge and cultural understanding, augmenting his holistic approach toward research and education.