Time-triggered Scheduling of Mixed-Criticality Systems

Lalatendu Behera
and
Purandar Bhaduri

Department of Computer Science & Engineering
Indian Institute of Technology Guwahati, India
Guwahati - 781039
Assam, India
March, 2017

Contents

Abstract

Real-time and embedded systems are moving from the traditional design paradigm to integration of multiple functionalities onto a single computing platform. Some of the functionalities are safety-critical and subject to certification. The rest of the functionalities are non-safety critical and don't need to be certified. Designing efficient scheduling algorithms which can be used to meet the certification requirement is challenging. Our research considers the time-triggered approach to scheduling of mixed-criticality jobs with two criticality levels. The first proposed algorithm for the time-triggered approach is based on the OCBP scheduling algorithm which finds a fixed-priority order of jobs. Based on this priority order, the existing algorithm constructs two scheduling tables S_{LO}^{oc} andS_{HI}^{oc}. The scheduler uses these tables to find a scheduling strategy. Another time-triggered algorithm called MCEDF was proposed as an improvement over the OCBP-based algorithm. Here we propose an algorithm which directly constructs two scheduling tables without using a priority order. Furthermore, we show that our algorithm schedules a strict superset of instances which can be scheduled by the OCBP-based algorithm as well as by MCEDF. We show that our algorithm outperforms both the OCBP-based algorithm and MCEDF in terms of the number of instances scheduled in a randomly generated set of instances. We generalize our algorithm for jobs with m criticality levels. Subsequently, we extend our algorithm to find scheduling tables for periodic and dependent jobs. Finally, we show that our algorithm is also applicable to mixed-criticality synchronous programs upon uniprocessor platforms and schedules a bigger set of instances than the existing algorithm.

1 Introduction

Now-a-days, there is a rapid increase in the use of real-time and embedded systems in day to day life. A real-time system is required to produce not only the correct result but it should produce it within the stipulated time. The growing demand of real-time systems leads to the complexity in the design of such systems. The applications of real-time systems are in the field of defense and space systems, networked multimedia systems, embedded automotive systems and avionics. Generally a real-time system used to be based on a single criticality level. But the current trend is towards systems with multiple functionalities of different criticality levels. In such a system, some functionalities are more critical than others. For example, unmanned aerial vehicles (UAV's) must fly safely and then capture images. Some of the functionalities are subject to mandatory certification requirements by *statutory organizations*. It is extremely difficult to come up with procedures that will allow for the cost effective certification of such mixed-criticality systems. There are many organizations who have mixed-criticality architecture requirements (MCAR) program for streamlining the certification process. In recent times, some of the software standards like AUTOSAR and ARINC in the automotive and avionics domain confront mixed-criticality issues.

Mixed-criticality Systems: A *mixed-criticality real-time system* (MCRTS) [1, 2, 3, 4, 5, 6, 7] is one that has two or more distinct levels of criticality, such as, safety-critical, mission-critical, non-critical, etc. Typical names of the criticality levels used in industries are ASIL (Automotive Safety and Integrity Levels) and SIL (Safety Integrity Level), etc.

We introduce the mixed-criticality scheduling problem with an example [1] from the domain of unmanned aerial vehicles (UAV's). The functionalities of UAV's may be classified into two categories, i.e., *mission-critical* and *flight-critical*.

- **Mission-critical** functionalities include capturing images from the ground and transmitting those to the base station, etc.
- **Flight-critical** functionalities include safe operation while performing the mission.

Here it is mandatory that the flight-critical functionality must be certified to be correct because if this functionality fails then it will be catastrophic. There are different certification authorities (CAs) for different functionalities. The CAs for flight-critical jobs tend to be very conservative. These authorities are not concerned with the mission-critical functionalities. During the certification process, the CAs focus mainly on the run-time behavior of the systems. The analytical tools, techniques and methodologies used by the CAs estimate more pessimistic results than the system

designers. The mission-critical functionalities are validated by the system designers. System designers are interested in both flight-critical and mission-critical functionalities but are not as rigorous as compared to the CAs with respect to the notion of correctness. For example, computation of the exact worst-case execution time (WCET) of a non-trivial piece of code is extremely difficult due to the complex architecture of today's systems. So, a safe upper bound on the actual WCET requires great effort. A CA may estimate the WCET of the piece of code to be far higher (pessimistic) while the system designer may choose a little lower estimate. This leads to two different WCET estimates, i.e., one by the CA which is very pessimistic and the other one by the system designer which is probably lower. The gaps between the CAs and the system designers are more likely to increase in future as pointed out in [8]. It is unlikely that a system would realize the higher WCET estimate for the piece of code. As a result, most of the resources which are provided to run the piece of code go unused if the pessimistic estimates are adhered to.

Example 1: Consider the system in the table below with two jobs: J_1 is a flight-critical (HI-criticality) job and J_2 is a mission-critical (LO-criticality) job. Since job J_1 has higher criticality than job J_2, its WCET estimate of 4 by the CAs is more than that of 2 by the system designers.

Job	Arrival	Deadline	Criticality	LO-criticality WCET	HI-criticality WCET
J_1	0	4	HI	2	4
J_2	0	2	LO	2	2

So J_1 needs certification whereas J_2 doesn't, being a LO-criticality job. □

The challenge in scheduling such mixed-criticality systems is to find a single scheduling policy so that the requirements of both the system designers and the CAs are met. In Example 1 this means that when both the jobs complete their executions by their LO-criticality WCETs, they must both be scheduled correctly. On the other hand, when the execution time of the HI-criticality job exceeds its LO-criticality WCET, then it is only this job which needs to meet its deadline to satisfy the CAs. In this report we focus on time-triggered scheduling of mixed-criticality jobs proposed by Baruah and Fohler [9] and present an algorithm that can schedule a superset of instances than can be scheduled by their algorithm as well as by MCEDF [10].

Time-triggered Scheduling: The scheduling activities in a time-triggered paradigm of real-time scheduling are activated by the progression of time. In such an approach a complete schedule for the entire duration is calculated before run-time. Generally, this pre-calculated schedule is kept in a table format. The scheduler takes the scheduling decisions according to this pre-calculated scheduling table. It is not possible to modify the scheduling table at run-time. Various types of time-triggered scheduling paradigm have been proposed, for example:

- **Slot shifting:** In this paradigm, the pre-computed scheduling table is partially calculated. Some of the additional scheduling decisions are made depending on the occurrence of run-time events.
- **Mode change** This is the paradigm adopted in this report. In this paradigm, there are various pre-computed scheduling tables. The occurrence of certain run-time events trigger a transition from one scheduling table to another. The transitions are pre-computed as well and it is also taken care that the ongoing activities are not interrupted.

Mixed-criticality Synchronous Reactive Systems: The time-triggered mode change paradigm is used in [11] to find time triggered schedules for mixed-criticality synchronous reactive (SR) programs upon uniprocessor platforms. A *synchronous reactive model* [12] is a discrete system where signals are absent at all times except at ticks of a global clock. The synchronous reactive model is widely used in the design and implementation of real-time systems. The behavioral aspects of reactive systems are specified using an assumption called the synchrony hypothesis [13]. The behavior of a system is an infinite series of steps, i.e., the system reads its input at each logical time instant t and computes its output based on the current state and the inputs received and the same process continue at time $(t + 1)$ and so on. These models are very easy to formally verify. There are many tools available in the market for

this purpose. But the main aim of these tools is to validate the design and not their implementations. But it is important to verify these implementations with respect to conservative assumptions about execution times.

In this report, we propose an algorithm which constructs time-triggered scheduling tables for mixed-criticality instances. Then we show the dominance of the proposed algorithm over existing algorithms. The proposed algorithm is generalized for m criticality levels, with $m \geq 2$ and extended for periodic and dependent jobs. Finally, we focus on how to implement mixed-criticality synchronous programs upon a uniprocessor platform using the time-triggered paradigm with efficient use of the resources and compare our method with the existing OCBP-based algorithm.

The rest of the report is organized as follows: Section 2 describes the system model and presents definitions and related work on mixed-criticality real-time systems and time-triggered scheduling. In Section 3, we propose a new algorithm which constructs two tables to find a time-triggered schedule for a dual-criticality MC instance. In Section 4, we extend our algorithm to construct m tables which can find a time-triggered schedule for m criticality MC systems. Sections 6 and 5 discuss the scheduling of mixed-criticality periodic and dependent jobs respectively. Section 7 describes our algorithm for the time-triggered scheduling of mixed-criticality synchronous programs. Section 8 includes experimental results based on a large number of randomly generated mixed-criticality instances. Section 9 concludes the report.

2 Problem Definition

The mixed-criticality model used in this section is based on at most two levels of criticality, LO and HI. A job is characterized by a 5-tuple of parameters: $j_i = (a_i, d_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}))$, where

- $a_i \in \mathbb{N}$ denotes the *arrival time*.
- $d_i \in \mathbb{N}^+$ denotes the *absolute deadline*.
- $\chi_i \in \{LO, HI\}$ denotes the *criticality* level.
- $C_i(\text{LO}) \in \mathbb{N}^+$ denotes the LO-criticality *worst-case execution time*.
- $C_i(\text{HI}) \in \mathbb{N}^+$ denotes the HI-criticality *worst-case execution time*.

We assume that the system is *preemptive* and $C_i(\text{LO}) \leq C_i(\text{HI})$ for $1 \leq i \leq n$. Note that in this report, we consider arbitrary arrival times of jobs.

An ***instance*** of mixed-criticality (MC) [1] job set can be defined as a finite collection of MC jobs, i.e., $I = \{j_1, j_2, ..., j_n\}$. The job j_i in the instance I is available for execution at time a_i and should finish its execution before d_i. The job j_i must execute for c_i amount of time which is the actual execution time between a_i and d_i, but this can be known only at the time of execution. The collection of actual execution time (c_i) of the jobs in an instance I at run time is called a ***scenario***. The scenarios in our model can be of two types, i.e., *LO-criticality scenarios* and *HI-criticality scenarios*. When each job j_i in instance I executes c_i units of time and signals completion before its $C_i(\text{LO})$ execution time, it is called a LO-criticality scenario. If any job j_i in instance I executes c_i units of time and doesn't signal its completion after it completes the $C_i(\text{LO})$ execution time, then this is called a HI-criticality scenario.

Each mixed-criticality instance needs to be scheduled by a scheduling strategy where both kinds of scenarios (LO and HI) can be scheduled. If we have prior knowledge about the scenario, then the scheduling strategy is known as a *clairvoyant scheduling strategy*. If we don't have prior knowledge about the scenario, then the scheduling strategy is called an *online scheduling strategy*. We can check that the instance given in Example 1 is clairvoyantly schedulable but doesn't have an online scheduling strategy. Here we assume that if any job continues its execution without signaling its completion at $C_i(\text{LO})$ then no LO-criticality jobs are required to complete by their deadlines. Now, we define the notion of MC-schedulability.

Definition 1: An instance I is MC-schedulable if it admits a correct online scheduling policy.

Here we focus on the **time-triggered schedules** [9] of MC instances. We will construct two tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ for a given instance I for use at run time. The length of the tables is the length of the interval $[\min_{j_i \in I}\{a_i\}, \max_{j_i \in I}\{d_i\}]$. The rules to use the tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ at run time, (i.e., the *scheduler*) are as follows:

- The criticality level indicator Γ is initialized to LO.
- While (Γ = LO), at each time instant t the job available at time t in the table $\mathcal{S}_{\text{LO}}$ will execute.
- If a job executes for more than its LO-criticality WCET without signaling completion, then Γ is changed to HI.
- While (Γ = HI), at each time instant t the job available at time t in the table $\mathcal{S}_{\text{HI}}$ will execute.

Definition 2: A dual-criticality MC instance I is said to be **time-triggered schedulable** [9] if it is possible to construct the two schedules $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ for I, such that the run-time scheduler algorithm described above schedules I in a correct manner.

2.1 Related Work

Vestal [6] introduced the notion of mixed-criticality real-time systems (MCRTS) by using an extension of the standard fixed priority (FP) real-time scheduling theory. The paper showed that both the deadline monotonic and rate monotonic algorithms are not optimal for MCRTS. Baruah and Vestal [7] generalized the problem by using a sporadic task model with fixed job-priority and dynamic priority scheduling algorithms. They showed that the earliest deadline first (EDF) algorithm [14] doesn't outperform fixed-priority schemes in the presence of criticality levels. They also showed that some of the feasible systems are not schedulable by EDF.

Burns and Baruah [5] proposed three schedulability algorithm based on the response time analysis of the task set. They proved that the proposed algorithms dominate the existing fixed-priority algorithms for traditional real-time systems.

Baruah et. al. [1] proved MC-schedulability is NP-hard in the strong sense. They also proved the problem to be in NP if the number of criticality levels is bounded by a fixed constant. They have shown that the general case where the criticality is part of the input belongs to the class PSPACE. They showed that the MC-schedulability problem with the same deadline for all the jobs is an easier problem.

Baruah *et al* [1] proposed a priority-based scheduling technique known as OCBP (Own Criticality Based Priority scheduling) for mixed-criticality jobs. The OCBP algorithm chooses a job j_i and assigns it the lowest priority if there is at least $C_i(\chi_i)$ time units available between its arrival time and its deadline when every other job j_k is executed with higher priority than j_i for $C_k(\chi_i)$ time units.

Baruah and Fohler [9] introduced a technique to schedule MC jobs using the *time-triggered* framework. Their objective was to ensure that adequate resources are reserved for each application to be able to guarantee the timing requirements. They used the OCBP algorithm to assign priorities to the jobs. Using this priority, they constructed two tables $\text{S}^{\text{oc}}_{\text{LO}}$ and $\text{S}^{\text{oc}}_{\text{HI}}$ which are used by the dispatch algorithm [9] to schedule the jobs. We show in Section 3 that our algorithm can schedule a strict superset of instances schedulable by the OCBP-based algorithm. In Section 8 we quantify the number of instances scheduled by the two algorithms on a set of randomly generated instances and show that our algorithm has better performance.

Socci et al [10], [15] proposed a fixed priority scheduling approach called MCEDF for mixed-criticality jobs. In this paper, they construct two priority tables, i.e., PT_{LO} and PT_{HI}. The scheduling of jobs starts with the table PT_{LO}, while the table PT_{HI} is used after a mode change occurs. In Section 8 we quantify the number of instances scheduled by MCEDF and our algorithm and show that the latter performs better.

In [16] Theis et al present a backtracking based iterative deepening algorithm for the generation of the scheduling tables. We were not able to compare this algorithm with ours because of the absence of implementation details.

Baruah [11], [17] proposed a schedule-generation algorithm for mixed-criticality synchronous programs upon uniprocessor platforms. He proved that proposed algorithm for single-rate synchronous programs is optimal. He

then proved that an efficient and optimal schedule generation problem for multi-rate synchronous program is NP-hard in the strong sense. He also proposed a schedule generation algorithm based on OCBP for multi-rate synchronous programs. In Section 7, we show that our algorithm can schedule a strict superset of instances of this OCBP-based algorithm.

2.2 Our Work

We know that OCBP and MCEDF are unable to schedule all the MC-instances that are MC-schedulable. In this report, we present an algorithm which can schedule not only the instances which are schedulable by the OCBP-based algorithm [9] and MCEDF algorithm [10] but additional ones as well. Then we generalize the algorithm for the m criticality case. Subsequently we extend the algorithm to construct the scheduling tables for periodic and dependent jobs.

Example 2: Consider the MC instance of 6 jobs given in Table 1.

Job	Arrival time	Deadline	Criticality	C_i(LO)	C_i(HI)
j_1	0	14	HI	1	8
j_2	0	3	LO	1	1
j_3	0	8	LO	2	2
j_4	0	8	LO	2	2
j_5	8	13	HI	2	3
j_6	0	12	HI	2	3

Table 1: Instance for Example 2

The above MC instance is not OCBP schedulable because we will not be able to assign a priority order as shown below.

- If j_1 is assigned the lowest priority, then j_2, j_3, j_4 and j_6 could consume 8 units of time (i.e., C_2(HI) + C_3(HI) + C_4(HI) + C_6(LO)) over [0, 8) as j_1 is a HI-criticality job. In the interval $[8, 11)$, j_5 execute its C_5(HI) units of execution, thus leaving no time for j_1 to execute its C_1(HI) before its deadline.
- If j_2 is assigned the lowest priority, then j_1, j_3, j_4 and $j_6$could consume 7 units of time (i.e., C_1(LO) + C_3(LO) + C_4(LO) + C_4(LO)) over [0, 7). This leaves no time for j_2 to execute its C_2(LO) time to finish by its deadline.
- If j_3 is assigned the lowest priority, then j_1, j_2, j_4 and j_6 could consume 6 units of time (i.e., C_1(LO) + C_2(LO) + C_4(LO) + C_6(LO)) over [0, 6). Job j_5 execute its C_5(LO) units of execution over [8, 11), thus leaving two units of space over [6, 8) for j_3 to execute its C_3(LO) units of execution before its deadline. So, j_3 can be assigned the lowest priority.
- Similarly, job j_4 can also be assigned as the lowest priority jobs among $\{j_1, j_2, j_4, j_5, j_6\}$ after removing job j_3.

Next, we remove the job j_4 and consider $\{j_1, j_2, j_5, j_6\}$ and try to assign the next lowest priority.

- If j_1 is assigned the lowest priority, then j_2 and j_6 could consume 4 units of time (i.e., C_2(HI) + C_6(HI)) over [0, 4) and j_5 could consume 3 units of C_2(HI) execution time over [8, 11), thus leaving 7 units of time for j_1 to execute its C_1(HI) units of execution before its deadline which is not possible.
- If j_2 is assigned the lowest priority, then j_1 and j_6 could consume 3 units of time (i.e., C_1(LO) + C_6(LO)) over [0, 3), thus leaving no time for j_2 to execute its C_2(LO) units of execution before its deadline which is not possible.

- If j_5 is assigned the lowest priority, then j_1, j_2 and j_6 could consume 12 units of time (i.e., C_1(HI) + C_2(HI) + C_6(HI)) over [0, 12), thus leaving 1 unit of time for j_5 to execute its C_5(HI) units of execution before its deadline which is not possible.

- If j_6 is assigned the lowest priority, then j_1, j_2 and j_5 could consume 12 units of time (i.e., C_1(HI) + C_2(HI) + C_5(HI)) over [0, 12), thus leaving no time for j_6 to execute its C_6(HI) units of execution before its deadline which is not possible.

Since, no other job can be assigned the lowest priority, we declare the MC instance is not OCBP-schedulable. So due to the unavailability of an OCBP order, we cannot construct a time-triggered schedule. □

Now we try to schedule the same instance with the MCEDF algorithm [10], [15]. We find the two priority tables PT_{LO} and PT_{HI} and check the schedulability. According to the MCEDF algorithm, if the instance is schedulable in the LO scenario, then it generates a priority tree. The nodes of the priority tree are sorted using topological sort [18]. The table PT_{LO} is constructed from the order generated by the topological sort. The table PT_{HI} is nothing but a simple EDF order of HI-criticality jobs. The algorithm checks for each possible HI scenario failure. If it doesn't get any HI scenario failure, then the algorithm declares success, otherwise it declares failure.

The EDF order of the above instance given in Table 1 is $(2, 3, 4, 6, 5, 1)$. The MCEDF algorithm generates the priority tree shown in Fig. 1. The instance is schedulable in LO scenario. The instance has one busy interval, i.e., $[0, 10]$. This means the lowest priority job of this interval will be the root of the priority tree. In this busy interval, j_{LO}^{Late} is job j_4 and j_{HI}^{Late} is job j_1. Clearly, j_1 is chosen to be the lowest priority job as the deadline of j_4 is less than 10. Next, j_1 is removed which splits the busy interval into two, i.e., $[0, 7]$ and $[8, 10]$. Job j_5 is the single job in the busy interval $[8, 10]$. So it can be assigned as one of the children of the root, i.e., job j_5 is removed from the interval. In the busy interval $[0, 7]$, j_{LO}^{Late} is job j_4 and j_{HI}^{Late} is job j_6. Here the MCEDF algorithm chooses job j_4 as one of the lowest priority job as its deadline is greater than 7. After removal of j_4, the busy interval splits into two intervals, i.e., $[0, 3]$ and $[5, 7]$. Now the priority tree generation steps are trivial. The resulting priority tree is given in Fig. 1.

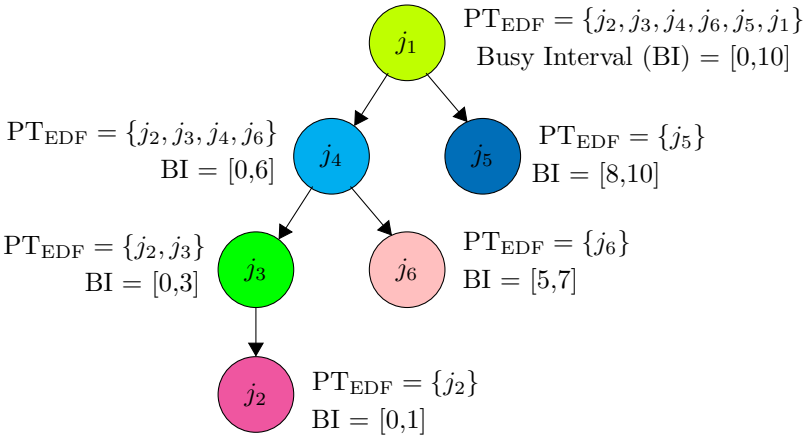

Figure 1: Priority tree of the instance given in Table 1

Now the MCEDF algorithm uses topological sort to find a priority order of the instance which in this case could be chosen to be $\{j_2, j_3, j_6, j_4, j_5, j_1\}$. The table PT_{LO} according to the priority order is given in Fig. 2.

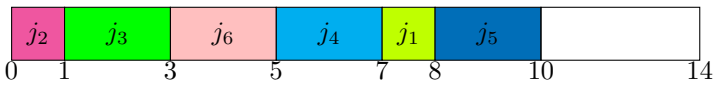

Figure 2: Table PT_{LO} of the instance given in Table 1

Then the MCEDF algorithm checks all possible HI-criticality scenarios for a deadline miss. When the job j_6 at time instant 5 doesn't signal its completion, there must be sufficient time for $1, 3$ and 8 units of execution for jobs j_6, j_5 and j_1 respectively before time instant 14. But, we have only 9 units of time left to complete these 12 units of execution. So, MCEDF cannot schedule the given instance.

We propose an algorithm which can construct a time-triggered schedule for this instance and is an improvement over OCBP in terms of the set of instances that can be scheduled. We show through experiments that the number of instances schedulable by our algorithm exceeds those schedulable by OCBP and MCEDF by a significant amount on randomly generated instances. We describe the algorithm in detail in the next section. The two scheduling tables generated by our algorithm for the instance in Table 1 are shown in Fig. 3. □

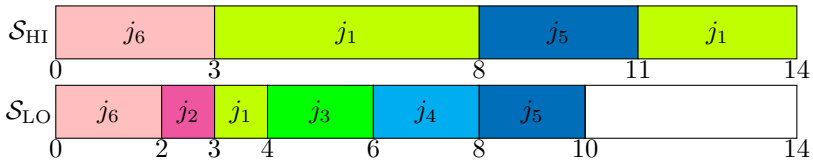

Figure 3: Tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ constructed by our algorithm for the instance given in Table 1

3 The Proposed Algorithm

From Section 2.2, it is clear that both the MCEDF and OCBP algorithms fail to schedule some instances due to a fixed priority assignment to the jobs. Both these algorithms construct the scheduling tables from the priority order of the jobs. That means, if the algorithms don't find a priority order then they will not be able to construct the scheduling tables. We propose an algorithm which can directly construct the scheduling tables without using priorities. We also focus on scheduling more number of instances than the OCBP and MCEDF algorithms. The main insight behind our algorithm is as follows.

- We want to find a time-triggered schedule not based on a priority order.
- We want to find the exact time to run a job in a scheduling table by merging two tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ containing jobs of the two different criticality levels.
- The LO-criticality execution time of HI-criticality jobs must be completed at a time instant t such that there is sufficient time to complete the remaining execution before its deadline.
- We want to construct the table $\mathcal{S}_{\text{LO}}$ by filling the vacant time slots of $\mathcal{T}_{\text{HI}}$ by the available jobs of $\mathcal{T}_{\text{LO}}$ at those time slots.

3.1 The Algorithm

In this section, we propose an algorithm which can schedule more instances than the OCBP-based algorithm. This algorithm has a pseudo-polynomial time complexity. The proposed algorithm constructs two tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ for the given MC instance, if possible. Our intention is to find $\mathcal{S}_{\text{LO}}$ and then construct $\mathcal{S}_{\text{HI}}$ keeping the same starting time for all the jobs as in $\mathcal{S}_{\text{LO}}$.
We define D_{max} which is the maximum deadline of the MC instance I.

$$D_{max} = max\{d_i\} \tag{1}$$

We construct $\mathcal{S}_{\text{LO}}$ from two temporary tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. Algorithm 1 and 2 describe the construction processes of $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. The length of the two temporary tables $\mathcal{T}_{\text{HI}}$ and $\mathcal{T}_{\text{LO}}$ is the same as the length of $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$.

Algorithm 1 Construct_$\mathcal{T}_{\text{LO}}(I)$

Notation:
$I = \{j_1, j_2, ..., j_n\}$, where
$j_i =< a_i,\ d_i,\ \chi_i,\ C_i(\text{LO}),\ C_i(\text{HI}) >$.
Input : I
Output : $\mathcal{T}_{\text{LO}}$
Assume earliest arrival time is 0.

1: Find the maximum deadline (D_{max}) of the jobs;
2: Prepare a temporary table $\mathcal{T}_{\text{LO}}$ of maximum length D_{max};
3: Let Ψ be the set of LO-criticality jobs of instance I;
4: Let O be the EDF order of the jobs of Ψ on the time-line using $C_i(\text{LO})$ units of execution for job j_i ;
5: **if** (any job cannot be scheduled) **then**
6: Declare failure;
7: **end if**
8: Starting from the rightmost job segment of the EDF order of Ψ, move each segment of a job j_i as close to its deadline as possible in $\mathcal{T}_{\text{LO}}$.

Algorithm 1 constructs the temporary table $\mathcal{T}_{\text{LO}}$. This algorithm chooses the LO-criticality jobs from the instance I and orders them in EDF order [14]. Then, all the job segments of the EDF schedule are moved as close to their deadline as possible so that no job misses its deadline in $\mathcal{T}_{\text{LO}}$. For example, we have an EDF order of three jobs as in Fig. 4 whose arrival times are $0, 2, 5$, execution times $6, 4, 2$ and deadlines $16, 11, 10$, respectively. The up and down arrows in the figure refer to the release and completion times respectively. Then, starting from the right end of the schedule, we shift each job segment as close to its deadline as possible so that no job misses its deadline. Here we move the rightmost job segment, i.e., j_1's segment as close to its deadline, i.e., from [8,12] to [12,16]. Then we move the next job segment of j_2 from [7,8] to [10,11]. Then the job segment of j_3 is moved right from [5,7] to [8,10] as the deadline of j_3 is 10. Then the job segment of j_2 is moved right from [2,5] to [5,8]. Finally, j_1's segment in the interval [0,2] is moved as close to its deadline as possible. Since at this stage there is an empty space at [11,12], j_1's segment in the interval [0,2] is distributed over [4,5] and [11,12]. The resulting table $\mathcal{T}_{\text{LO}}$ is given in Fig. 5. Note that, if the arrival times of the jobs are not the same, then the jobs may execute in more than one segment, in general. If the arrival times of all the jobs are the same then, the jobs will execute in one segment.

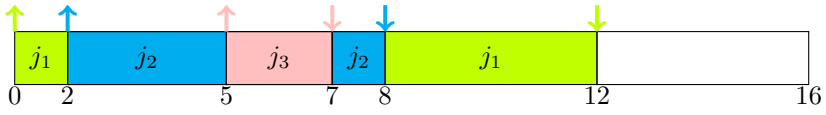

Figure 4: EDF order of three jobs. Up arrows indicate arrival and down arrows indicate completion times

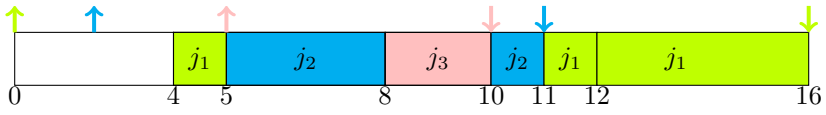

Figure 5: After the shifting of jobs

Algorithm 2 constructs the temporary table $\mathcal{T}_{\text{HI}}$. This algorithm chooses the HI-criticality jobs from the instance I and orders them in EDF order. Then, all the job segments of the EDF schedule are moved as close to their deadline as possible so that no job misses its deadline in $\mathcal{T}_{\text{HI}}$. Then, out of the total allocation so far, the algorithm allocates $C_i(\text{LO})$ units of execution of job j_i in $\mathcal{T}_{\text{HI}}$ from the beginning of its slot and leaves the rest of the execution time of j_i

Algorithm 2 Construct_$\mathcal{T}_{\text{HI}}(I)$

Notation:
$I = \{j_1, j_2, ..., j_n\}$, where
$j_i =< a_i,\ d_i,\ \chi_i,\ C_i(\text{LO}),\ C_i(\text{HI}) >$.
Input : I
Output : $\mathcal{T}_{\text{HI}}$
Assume earliest arrival time is 0.

1: Find the maximum deadline (D_{max}) of the jobs;
2: Prepare a temporary table $\mathcal{T}_{\text{HI}}$ of maximum length D_{max};
3: Let Ψ be the set of HI-critical jobs of instance I;
4: Let O be the EDF order of the jobs of Ψ on the time-line using $C_i(\text{HI})$ units of execution for job j_i ;
5: **if** (any job cannot be scheduled) **then**
6: Declare failure;
7: **end if**
8: Starting from the rightmost job segment of the EDF order of Ψ, move each segment of a job j_i as close to its deadline as possible in $\mathcal{T}_{\text{HI}}$.
9: **for** $i := 1$ to m **do**
10: Allocate $C_i(\text{LO})$ units of execution to job j_i from its starting time in $\mathcal{T}_{\text{HI}}$ and leave the rest unallocated;
11: **end for**

unallocated in $\mathcal{T}_{\text{HI}}$. Suppose, there is an instance I which contains three HI-criticality jobs j_1, j_2 and j_3 with arrival times $0, 2, 5$, execution times $(2, 6), (2, 4), (2, 2)$ and deadlines $16, 11, 10$, respectively. This instance is arranged in EDF order and then each job segment is shifted as close to its deadline as possible. The resulting allocation is given at the top of Fig. 6, which happens to be the same as in the earlier example for $\mathcal{T}_{\text{LO}}$. Then algorithm 2 allocates $C_i(\text{LO})$ units of execution and leaves $(C_i(\text{HI}) - C_i(\text{LO}))$ units of execution unallocated. The resulting allocation is shown at the bottom of Fig. 6.

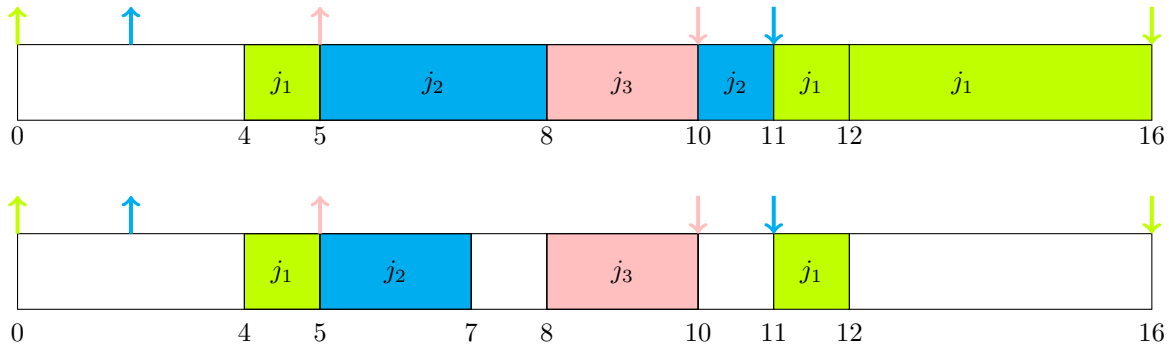

Figure 6: Allocating $C_i(\text{LO})$ units of execution only

Now, we use Algorithm 3 to construct the table $\mathcal{S}_{\text{LO}}$ from $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. The algorithm starts the construction of $\mathcal{S}_{\text{LO}}$ from time 0 and checks the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ simultaneously. There are four possibilities while merging the two temporary tables to construct $\mathcal{S}_{\text{LO}}$.
At time slot t, one of the following situations can occur.

1. Both $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are empty.
2. Both $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are not empty.
3. $\mathcal{T}_{\text{LO}}$ is empty and $\mathcal{T}_{\text{HI}}$ is not empty.

Algorithm 3 TT_Merge($I, \mathcal{T}_{\text{LO}}, \mathcal{T}_{\text{HI}}$)

Notation:

$I = \{j_1, j_2, ..., j_n\}$.

$j_i =< a_i,\ d_i,\ \chi_i,\ C_i(\text{LO}),\ C_i(\text{HI}) >$.

Input : $I, \mathcal{T}_{\text{LO}}, \mathcal{T}_{\text{HI}}$

Output : Tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$

Construction of $\mathcal{S}_{\text{LO}}$.
Find the maximum deadline (D_{max}) of the jobs;
The maximum length of tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ are both D_{max};
$t := 0$;
while ($t \leq D_{max}$) **do**
  if ($\mathcal{T}_{\text{LO}}[t] = NULL$ & $\mathcal{T}_{\text{HI}}[t] = NULL$) **then**
    Search the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ simultaneously from the beginning to find the first available job at time t;
    Let k be the first occurrence of a job j_i in $\mathcal{T}_{\text{LO}}$ or $\mathcal{T}_{\text{HI}}$;
    if (Both LO-criticality & HI-criticality job are found) **then**
      $\mathcal{S}_{\text{LO}}[t] := \mathcal{T}_{\text{LO}}[k]$;
      $\mathcal{T}_{\text{LO}}[k] := NULL$;
    else if (LO-criticality job is found) **then**
      $\mathcal{S}_{\text{LO}}[t] := \mathcal{T}_{\text{LO}}[k]$;
      $\mathcal{T}_{\text{LO}}[k] := NULL$;
    else if (HI-criticality job is found) **then**
      $\mathcal{S}_{\text{LO}}[t] := \mathcal{T}_{\text{HI}}[k]$;
      $\mathcal{T}_{\text{HI}}[k] := NULL$;
    else if (NO job is found) **then**
      $\mathcal{S}_{\text{LO}}[t] := NULL$
      $t := t + 1$;
    end if
  else if ($\mathcal{T}_{\text{LO}}[t] = NULL$ & $\mathcal{T}_{\text{HI}}[t]$!= $NULL$) **then**
    $\mathcal{S}_{\text{LO}}[t] := \mathcal{T}_{\text{HI}}[t]$;
    $\mathcal{T}_{\text{HI}}[t] := NULL$;
    $t := t + 1$;
  else if ($\mathcal{T}_{\text{LO}}[t]$!= $NULL$ & $\mathcal{T}_{\text{HI}}[t] = NULL$) **then**
    $\mathcal{S}_{\text{LO}}[t] := \mathcal{T}_{\text{LO}}[t]$;
    $\mathcal{T}_{\text{LO}}[t] := NULL$;
    $t := t + 1$;
  else if ($\mathcal{T}_{\text{LO}}[t]$!= $NULL$ & $\mathcal{T}_{\text{HI}}[t]$!= $NULL$) **then**
    Declare failure;
  end if
end while
This is the table $\mathcal{S}_{\text{LO}}$;

Construction of $\mathcal{S}_{\text{HI}}$
Copy all the jobs from table $\mathcal{S}_{\text{LO}}$ to table $\mathcal{S}_{\text{HI}}$;
Scan the table $\mathcal{S}_{\text{HI}}$ from left to right:
for each HI-criticality job j_i, allocate an additional $C_i(\text{HI}) - C_i(\text{LO})$ time units immediately after the rightmost segment of job j_i, recursively pushing all the overlapping HI-criticality job segments in $\mathcal{S}_{\text{HI}}$ (except those whose allocation time is same as in $\mathcal{T}_{\text{HI}}$) to the right and overwriting any LO-criticality jobs in the process.

---

4. $\mathcal{T}_{\text{LO}}$ is not empty and $\mathcal{T}_{\text{HI}}$ is empty.

If situation 1 occurs, then the algorithm will allocate the nearest ready job to the right at time slot t where a LO-criticality job gets higher priority over a HI-criticality job. In this case, the place of the ready job in $\mathcal{T}_{\text{LO}}$ or $\mathcal{T}_{\text{HI}}$ is marked as empty. In case of situation 2, the algorithm declares failure to schedule. In situation 3, the algorithm allocates the HI-criticality job from $\mathcal{T}_{\text{HI}}$, whereas in situation 4, the algorithm allocates the LO-criticality job from $\mathcal{T}_{\text{LO}}$. Once an instant of a job is allocated in $\mathcal{S}_{\text{LO}}$, the place where it was scheduled in $\mathcal{T}_{\text{LO}}$ or $\mathcal{T}_{\text{HI}}$ is emptied.

We then construct the table $\mathcal{S}_{\text{HI}}$ from $\mathcal{S}_{\text{LO}}$. We first copy the jobs of table $\mathcal{S}_{\text{LO}}$ to $\mathcal{S}_{\text{HI}}$. Then the HI-criticality jobs are allocated $C_i(\text{HI}) - C_i(\text{LO})$ units of HI-criticality execution time after their $C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{HI}}$. These additional time units are allocated by pushing all overlapping HI-criticality jobs in $\mathcal{S}_{\text{HI}}$ to the right and overwriting any LO-criticality job in the process. An exception to this is when the allocation time of an overlapping HI-criticality job is the same in both the tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{T}_{\text{HI}}$, in which case the additional time units are allocated after this job. A LO-criticality job j_k present in table $\mathcal{S}_{\text{LO}}$ will not appear in table $\mathcal{S}_{\text{HI}}$ if and only if the additional $C_i(\text{HI}) - C_i(\text{LO})$ time units of allocation of any HI-criticality job overlaps with the allocation of j_k in table $\mathcal{S}_{\text{LO}}$.

3.2 Intuition behind the Algorithm

In the following subsections, we show that our algorithm dominates both the existing mixed-criticality time-triggered scheduling algorithms by being able to schedule a larger subset of instances. Here we briefly explain the working of our algorithm, contrasting it with the existing algorithms.

The OCBP algorithm fails to find a priority order for instance I, if it is unable to choose a lowest priority job from I. For example, a HI-criticality job j_i is assigned the lowest priority if all other jobs can finish their HI-criticality execution times before their deadlines and still leave sufficient time for j_i to finish its execution. But this is too strong a requirement, since in a HI-criticality scenario the LO-criticality jobs need not meet their deadlines. Since it is not possible for OCBP to check the worst-case starting and completion time of each job separately at each criticality level, it fails to assign priorities in some cases. We construct an algorithm which doesn't depend on any priority, while finding a time-triggered schedule. We construct two separate schedules for the two different criticality levels. We merge the two tables to find a LO-criticality schedule and then find the HI-criticality schedule using this LO-criticality schedule.

The core idea behind our algorithm is to allocate jobs at each instant of the time-triggered schedule without depending on any priority such that both the scenarios (HI-criticality and LO-criticality) can be successfully scheduled. To this end, we find the worst-case starting and completion times of each job of the same criticality for the LO-criticality scenario separately in the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. Algorithms 1 and 2 find the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ by shifting the job segments of the EDF order of jobs as close to their deadlines as possible considering $C_i(\text{LO})$ and $C_i(\text{HI})$ units of executions, respectively. Then Algorithm 2 keeps $C_i(\text{LO})$ units of execution for each HI-criticality job in $\mathcal{T}_{\text{HI}}$ and empties the rest of the slots. From table $\mathcal{T}_{\text{HI}}$, we know the worst-case completion time of a LO-criticality execution of a HI-criticality job. These two tables are identical to the OCBP order for the jobs of the same criticality, which we prove later in Lemma 4 and 5. Algorithm 3 merges the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ to construct the table $\mathcal{S}_{\text{LO}}$, where all the tables have the same schedule length, i.e., D_{max}. Algorithm 3 keeps the jobs of table $\mathcal{T}_{\text{HI}}$ at their assigned slots and fills the empty places of this table with the jobs of the table $\mathcal{T}_{\text{LO}}$. This guarantees the timely execution of HI-criticality jobs in both the scenarios which is not always possible in the case of the OCBP-based and MCEDF algorithms. Since jobs of the table $\mathcal{T}_{\text{LO}}$ fill the empty spaces of the table $\mathcal{T}_{\text{HI}}$, we prefer a LO-criticality job to be allocated at time t, if both the tables are empty at time t. If a LO-criticality job is not available at t and a HI-criticality job is available, then that HI-criticality job segment is chosen to be allocated at t.

We illustrate the operation of this algorithm by an example.

Example 3: Consider the MC instance given in Table 2.
Let us first find the two temporary tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ in which the LO-criticality and HI-criticality jobs are allocated

Job	Arrival time	Deadline	Criticality	C_i(LO)	C_i(HI)
j_1	1	8	HI	1	2
j_2	1	6	HI	1	2
j_3	2	4	HI	1	2
j_4	0	4	LO	1	1
j_5	0	4	LO	2	2

Table 2: Instance for Example 3

respectively.

- $D_{max} = 8$.
- The maximum length of $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ is 8.
- According to Algorithm 1, we choose the LO-criticality jobs and allocate them in $\mathcal{T}_{\text{LO}}$ in EDF order. Then, each segment of the jobs in EDF order are shifted as close to their deadlines as possible according to their C_i(LO) units of execution. So j_4 is allocated in the interval [1,2] and j_5 is allocated in the interval [2,4]. The resulting table $\mathcal{T}_{\text{LO}}$ is given in Fig. 7.

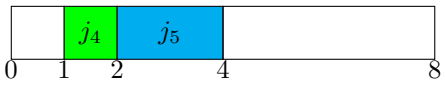

Figure 7: Temporary table $\mathcal{T}_{\text{LO}}$

- According to Algorithm 2, we choose the HI-criticality jobs to allocate them in $\mathcal{T}_{\text{HI}}$ in EDF order. Then, each segment of the jobs in EDF order are shifted as close to their deadlines as possible according to their C_i(HI) units of execution. So j_3 is allocated in the interval [2,4], j_2 is allocated in the interval [4,6] and j_1 is allocated in the interval [6,8]. The resulting table $\mathcal{T}_{\text{HI}}$ is given in Fig. 8.

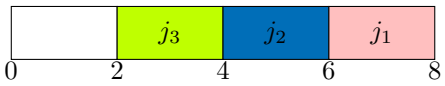

Figure 8: Intermediate temporary table $\mathcal{T}_{\text{HI}}$

- Then, we allocate C_i(LO) units of execution of j_i and leave the (C_i(HI)$-C_i$(LO)) units of execution unallocated. Here j_3 has been allocated its C_i(LO) units of execution time in the interval [2,3]. So we empty the occurrence of j_3 in the interval [3,4]. We repeat the same process for both j_2 and j_1. After this modification of $\mathcal{T}_{\text{HI}}$, the resulting table $\mathcal{T}_{\text{HI}}$ is given in Fig. 9.

- Finally, we construct $\mathcal{S}_{\text{LO}}$ from these two temporary tables.

We construct the table $\mathcal{S}_{\text{LO}}$ according to Algorithm 3.

- We start from time $t = 0$.
- At $t = 0$, both $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are empty. So we allocate the LO-criticality job from $\mathcal{T}_{\text{LO}}$ which is ready at $t = 0$, i.e., j_4. We empty the interval [1,2] in $\mathcal{T}_{\text{LO}}$ from where the first occurrence of j_4 is found.

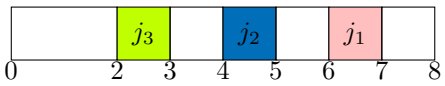

Figure 9: Temporary table $\mathcal{T}_{\text{HI}}$

- At $t = 1$, both $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are empty. So we allocate the LO-criticality job from $\mathcal{T}_{\text{LO}}$ which is ready at $t = 1$, i.e., j_5. We empty the interval [2,3] in $\mathcal{T}_{\text{LO}}$ from where the first occurrence of j_5 is found.
- At $t = 2$, $\mathcal{T}_{\text{LO}}$ is empty and $\mathcal{T}_{\text{HI}}$ contains j_3. So we allocate j_3 from $\mathcal{T}_{\text{HI}}$ and empty the interval [2,3] of $\mathcal{T}_{\text{HI}}$.
- At $t = 3$, $\mathcal{T}_{\text{LO}}$ contains j_5 and $\mathcal{T}_{\text{HI}}$ is empty. So we allocate j_5 from $\mathcal{T}_{\text{LO}}$ and empty the interval [3,4] of $\mathcal{T}_{\text{LO}}$.
- At $t = 4$, $\mathcal{T}_{\text{LO}}$ is empty and $\mathcal{T}_{\text{HI}}$ contains j_2. So we allocate j_2 from $\mathcal{T}_{\text{HI}}$ and empty the interval [4,5] of $\mathcal{T}_{\text{HI}}$.
- At $t = 5$, both $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are empty. So we allocate a ready LO-criticality job from $\mathcal{T}_{\text{LO}}$. But, no LO-criticality jobs are there to be allocated. So we allocate the remaining jobs of $\mathcal{T}_{\text{HI}}$.
- The resulting table $\mathcal{S}_{\text{LO}}$ is given in Fig. 10.

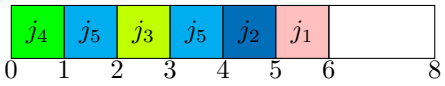

Figure 10: Table $\mathcal{S}_{\text{LO}}$

Now, we construct the table $\mathcal{S}_{\text{HI}}$ from $\mathcal{S}_{\text{LO}}$ using the steps shown in Fig. 11.

- We copy the table $\mathcal{S}_{\text{LO}}$ to table $\mathcal{S}_{\text{HI}}$.
- For the first HI-criticality job j_3, $C_3(\text{HI}) - C_3(\text{LO})$ units of execution time are allocated in the interval $[3, 4]$. In this process, we overwrite job j_5 which was present in the interval $[3, 4]$. This is shown in the top table of Fig. 11.
- Then $C_2(\text{LO})$ units of execution time of j_2 are allocated in the interval $[4, 5]$ followed by $C_2(\text{HI}) - C_2(\text{LO})$ units of execution time in the interval $[5, 6]$. In this process, we push job j_1 to its right, i.e., to the interval $[6, 7]$ from $[5, 6]$. Finally, j_1 is allocated in the interval [6,8]. This is shown in the middle table of Fig. 11.
- The resulting table $\mathcal{S}_{\text{HI}}$ is given in the table at the bottom of Fig. 11.

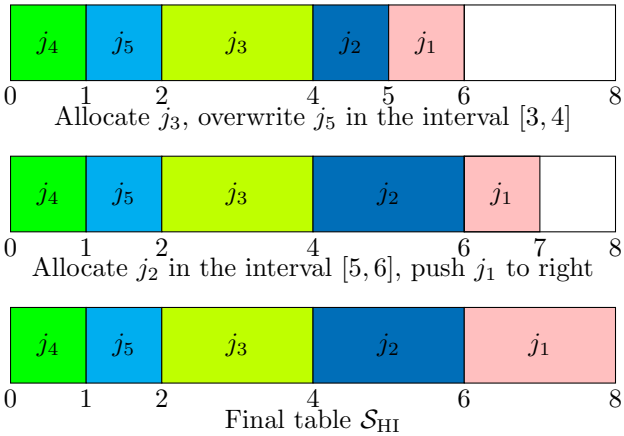

Figure 11: Construction of table $\mathcal{S}_{\text{HI}}$

□

Now we present an example to show the tables constructed by different existing algorithm and our algorithm for the same instance.

Example 4: The point of this example is to show how the tables constructed by our algorithm differ from the ones constructed by the OCBP-based algorithm, when both the algorithms are successful. Consider the MC instance given in Table 3. Fig. 12 shows the tables constructed using the OCBP-based algorithm. The MCEDF algorithm

Job	Arrival time	Deadline	Criticality	C_i(LO)	C_i(HI)
j_1	0	2	LO	1	1
j_2	0	7	HI	2	3
j_3	2	10	LO	4	4
j_4	5	10	HI	2	5

Table 3: Instance for Example 4

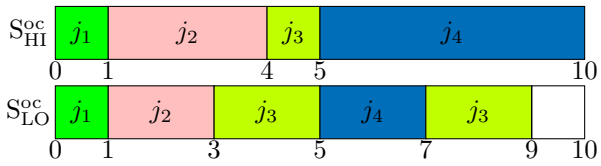

Figure 12: Scheduling tables according to OCBP-based algorithm

computes the same priority order as OCBP. So it constructs the same tables as OCBP. Fig. 13 shows the scheduling tables according to our algorithm.

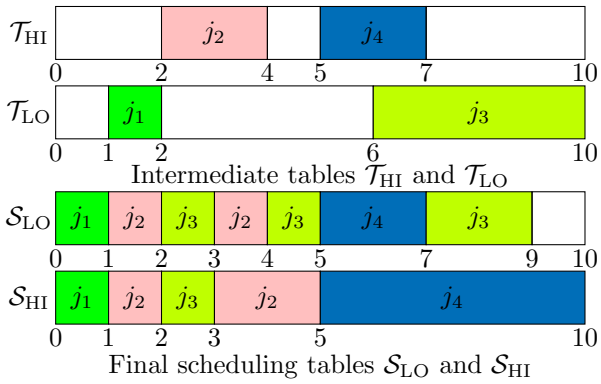

Figure 13: Scheduling tables according to our algorithm

3.3 Correctness Proof

For correctness, we have to show that if our algorithm finds the two scheduling tables $\mathcal{S}_{LO}$ and $\mathcal{S}_{HI}$, then these two tables will give a correct scheduling strategy. We start with the proof of some properties of the schedule.

Observation 1: The table $\mathcal{T}_{HI}$ shows the latest possible allocation of the initial (LO-criticality) segment of a HI-criticality job that can still meet its deadline in a schedule. To see this, recall that the table $\mathcal{T}_{HI}$ is constructed from the EDF order of the HI-criticality jobs. Each job segment in the EDF order is pushed as close to its deadline as possible. Then the initial C_i(LO) time units of each job are kept and the rest are unallocated. By the construction, no job segment can be pushed further to the right and still meet its deadline.

Remark: We know that the table $\mathcal{S}_{\text{LO}}$ allocates each HI-criticality job on or before its allocation in $\mathcal{T}_{\text{HI}}$. Then no job can be pushed to the right in the table $\mathcal{S}_{\text{HI}}$ after its allocation in $\mathcal{T}_{\text{HI}}$ as it will miss its deadline. This follows from Observation 1.

Lemma 1: If Algorithm 3 doesn't declare failure, then each job j_i receives $C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{LO}}$ and each HI-criticality job j_k receives $C_k(\text{HI})$ units of execution in $\mathcal{S}_{\text{HI}}$ by its deadline.

Proof. First, we show that any job j_i receives $C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{LO}}$. We construct $\mathcal{S}_{\text{LO}}$ from the temporary tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. Each job j_i can be scheduled in $\mathcal{S}_{\text{LO}}$ on or before its scheduled time in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. If our algorithm finds the table $\mathcal{S}_{\text{LO}}$ then each job must receive $C_i(\text{LO})$ units of execution.

Next we show that any HI-criticality job j_k receives $C_k(\text{HI})$ units of execution in $\mathcal{S}_{\text{HI}}$. We start constructing $\mathcal{S}_{\text{HI}}$ by copying the jobs in $\mathcal{S}_{\text{LO}}$. But according to our algorithm, the HI-criticality jobs are allocated their remaining $C_k(\text{HI}) - C_k(\text{LO})$ units of allocation in $\mathcal{S}_{\text{HI}}$ after they complete their $C_k(\text{LO})$ units of allocation in $\mathcal{S}_{\text{HI}}$ by pushing recursively all the following HI-criticality job segments to the right except those whose allocation is the same as in table $\mathcal{T}_{\text{HI}}$. This means we can push a job segment to the right in $\mathcal{S}_{\text{HI}}$ only if it is allocated before its allocation in $\mathcal{T}_{\text{HI}}$ and moreover, no job is pushed beyond its allocation in $\mathcal{T}_{\text{HI}}$, because if the construction of $\mathcal{T}_{\text{HI}}$ doesn't declare failure then it allocates enough time for the execution of all the HI-criticality jobs. In this case, all the jobs can get sufficient time to schedule their $C_k(\text{HI}) - C_k(\text{LO})$ units of execution as they are allocated on or before the allocation in table $\mathcal{T}_{\text{HI}}$. This is clear from the remark following Observation 1. If a HI-criticality job j_h cannot be pushed to the right then it will get its remaining $C_h(\text{HI}) - C_h(\text{LO})$ units of execution time in table $\mathcal{S}_{\text{HI}}$ by a similar reasoning as above. □

Lemma 2: At any time t, if a job j_i is present in $\mathcal{S}_{\text{HI}}$ but not in $\mathcal{S}_{\text{LO}}$, then the job j_i has finished its LO-criticality execution before time t in $\mathcal{S}_{\text{LO}}$.

Proof. We use the same order of jobs in $\mathcal{S}_{\text{LO}}$ to construct $\mathcal{S}_{\text{HI}}$. We know the HI-criticality jobs are allocated their $C_i(\text{HI}) - C_i(\text{LO})$ units of execution after the allocation of $C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{HI}}$. In $\mathcal{S}_{\text{HI}}$, the HI-criticality jobs are preferred over the LO-criticality jobs, i.e., a HI-criticality job is chosen to be allocated in table $\mathcal{S}_{\text{HI}}$ if a LO-criticality job is found in $\mathcal{S}_{\text{LO}}$ while allocating $C_i(\text{HI}) - C_i(\text{LO})$ units of execution in table $\mathcal{S}_{\text{HI}}$. This means each of the job segments present in table $\mathcal{S}_{\text{HI}}$ is either at the same position in $\mathcal{S}_{\text{LO}}$ or to the right of it. When a job j_i is present in $\mathcal{S}_{\text{HI}}$ and not in $\mathcal{S}_{\text{LO}}$ at time t, it means this has already completed its LO-criticality execution in $\mathcal{S}_{\text{LO}}$. □

Lemma 3: At any time t, when a mode change occurs, each HI-criticality job still has $C_i(\text{HI}) - c_i$ units of execution in $\mathcal{S}_{\text{HI}}$ after time t to complete its execution, where c_i is the execution time already completed by job j_i before time t in $\mathcal{S}_{\text{LO}}$.

Proof. Suppose a mode change occurs at time t. This means all the HI-criticality jobs scheduled before time t have either signaled their completion or the current HI-criticality job is the first one to complete its $C_i(\text{LO})$ units of execution without signaling its completion. We know that all the HI-criticality jobs are allocated their $C_i(\text{HI}) - C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{HI}}$ after the completion of their $C_i(\text{LO})$ units of execution in both $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$. If a job j_i has already executed its $C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{LO}}$, then it requires $C_i(\text{HI}) - C_i(\text{LO})$ units of time to be completed in $\mathcal{S}_{\text{HI}}$. When job j_i initiates the mode change, this is the first job which doesn't signal its completion after completing its $C_i(\text{LO})$ units of execution. Before time t, the scheduler uses the table $\mathcal{S}_{\text{LO}}$ to schedule the jobs, while subsequently the scheduler uses table $\mathcal{S}_{\text{HI}}$ due to the mode change. If a job j_i has already executed its c_i units of execution in $\mathcal{S}_{\text{LO}}$, then it requires $C_i(\text{HI}) - c_i$ units of time to be completed in $\mathcal{S}_{\text{HI}}$ its execution. We know that the tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ have same order and according to lemma 1 and 2, each job will get sufficient time to complete its $C_i(\text{HI})$ units of execution. Hence, each HI-criticality job will get $C_i(\text{HI}) - c_i$ units of time in $\mathcal{S}_{\text{HI}}$ to complete its execution after the mode change at time t. □

Theorem 1: If the scheduler dispatches the jobs according to $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$, then it will be a correct scheduling strategy.

Proof. For LO-criticality scenarios, all jobs can be correctly scheduled by the table $\mathcal{S}_{\text{LO}}$ as proved in Lemma 1. Now, we need to prove that in a HI-criticality scenario, all the HI-criticality jobs can be correctly scheduled by the table $\mathcal{S}_{\text{HI}}$. In Lemma 1, we have already proved that all the HI-criticality jobs get sufficient units of time in $\mathcal{S}_{\text{HI}}$ to complete their execution. In Lemma 3, we have proved that when the mode change occurs at time t, all the HI-criticality jobs can be scheduled without missing their deadline. So from Lemma 1 and Lemma 3, it is clear that if the scheduler uses the tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ to dispatch the jobs then it will be a correct scheduling strategy. □

3.4 Dominance over OCBP-based Algorithm

We know that the algorithm proposed by Baruah and Fohler [9] is based on the OCBP order [1] and constructs the tables $\text{S}^{\text{oc}}_{\text{LO}}$ and $\text{S}^{\text{oc}}_{\text{HI}}$ based on this order. We show that if the OCBP-based algorithm constructs the tables $\text{S}^{\text{oc}}_{\text{LO}}$ and $\text{S}^{\text{oc}}_{\text{HI}}$ for an instance then our algorithm will also construct the two tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ for the same instance.

Notation: We use $\text{S}^{\text{oc}}_{\text{LO}}$ and $\text{S}^{\text{oc}}_{\text{HI}}$ for the tables constructed by the OCBP-based algorithm and $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ for the tables constructed by our algorithm. Further, we use $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ for the two temporary tables in our algorithm.

Lemma 4: If OCBP chooses a latest deadline job as the lowest priority job at each stage, then the OCBP priority order of jobs of the same criticality is the same as that assigned by EDF.

Proof. See observation 1 of Lemma 2 from Park and Kim [19] for the proof of this lemma. □

Lemma 5: If OCBP finds a priority order for an instance I, then there exists an OCBP priority order for I in which all jobs of the same criticality are in EDF order.

Proof. Suppose there exists an OCBP priority order for instance I. Let j_i and j_k be two jobs of the same criticality, where j_i is assigned higher priority than j_k by OCBP and $d_i \geq d_k$. OCBP assigns lower priority to j_k because all other jobs including j_i finish their $C(\chi_k)$ units of execution and there is sufficient time in the interval $[a_k, d_k]$ for j_k to finish its $C(\chi_k)$ units of execution. If we swap the priority levels of j_i and j_k, then j_k certainly meets its deadline and even though the execution segments of j_i are shifted to the right, its deadline d_i is not violated, since $d_i \geq d_k$. So we can exchange their priority which means there exists a priority order for I in which all jobs of the same criticality are in EDF order. □

Without loss of generality, by Lemma 5 all the jobs in the table $\text{S}^{\text{oc}}_{\text{LO}}$ constructed by the OCBP-based algorithm of the same criticality are in EDF order.

Lemma 6: If OCBP finds a priority order for an instance I, then Algorithms 1 and 2 can construct the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ and these are obtained from the OCBP order by moving the job segments to the right starting from the right end of the schedule for the LO-criticality and HI-criticality jobs respectively.

Proof. Follows from Lemma 4 and 5. □

Theorem 2: If an instance I is schedulable by the OCBP-based scheduling algorithm, then it is also schedulable by our algorithm.

Proof. OCBP generates a priority order for an instance I. Then the OCBP-based algorithm finds the tables $\text{S}^{\text{oc}}_{\text{LO}}$ and $\text{S}^{\text{oc}}_{\text{HI}}$ for the instance I using this priority order. We need to show that if there exists tables $\text{S}^{\text{oc}}_{\text{LO}}$ and $\text{S}^{\text{oc}}_{\text{HI}}$ constructed by the OCBP-based algorithm, then our algorithm will not encounter a situation where at time slot t $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are non-empty, for any t.

We know that $C_i(\text{LO})$ units of execution are allocated to each job j_i for constructing the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. Each job in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ is allocated as close to its deadline as possible. That means no job can execute after its allocation time in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ without affecting the schedule of any other job and still meet its deadline. Algorithm 3 declares failure if it finds a non-empty slot at any time t in both the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. This means the two jobs

which are found in the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ respectively cannot be scheduled with all other remaining jobs from this point, because all the jobs to the right have already been moved as far to the right as possible.

Suppose there is an OCBP priority order of the jobs of instance I and the LO-criticality table $\text{S}_{\text{LO}}^{\text{oc}}$ follows this priority order.

Let j_l and j_h be two jobs in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ respectively found at time t during the construction of table $\mathcal{S}_{\text{LO}}$ by our algorithm, which means all job segments in the interval $[0, t-1]$ from $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ have already been assigned in $\mathcal{S}_{\text{LO}}$. But, we know that OCBP has assigned priorities to these jobs j_l and j_h. There are two cases to consider.

In the first case, assume j_l is assigned lower priority than j_h by OCBP. Let a_l be the arrival time of job j_l and t_l and $t_l{}'$ be the starting and completion times of j_l in $\mathcal{T}_{\text{LO}}$ computed by Algorithm 1. Since job j_l can be scheduled only on or after the arrival time a_l, we need to show that the job segment of j_l found at time t cannot be scheduled in the interval $[a_l, t-1]$ by the OCBP-based algorithm. We know that Algorithm 3 can allocate a job in table $\mathcal{S}_{\text{LO}}$ on or before its allocation in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. But Algorithm 3 has not allocated the job segments found in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ at time t in the interval $[a_l, t-1]$ of the table $\mathcal{S}_{\text{LO}}$ and by Lemma 6 this is due to the presence of equal or higher priority job segments of the OCBP priority order in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. We know that all the jobs in $\mathcal{T}_{\text{LO}}$ in the interval $[a_l, t]$ and the jobs in $\mathcal{T}_{\text{HI}}$ including job j_h in the interval $[a_l, t]$ are of priority greater or equal to that of j_l according to OCBP since, by moving job segments to the right starting from the OCBP schedule the jobs to the left of j_l are of priority greater than or equal to that of j_l. This means the jobs in the interval $[a_l, t-1]$ of table $\mathcal{S}_{\text{LO}}$ are either equal or higher priority jobs than j_l according to OCBP. So both the algorithms, the OCBP-based one and ours, allocate higher or equal priority jobs (or, job segments according to Algorithm 3) before time t. Then it is clear that after the jobs of higher priority than j_l finish their $C(\text{LO})$ units of execution by time t, there will not be sufficient time for j_l to finish its $C_l(\text{LO})$ units of execution in the interval $[a_l, t_l{}']$ in the OCBP schedule. This is because at time t, the OCBP-based algorithm has already allocated all ready jobs with higher or equal priority than j_l (according to OCBP) in the interval $[a_l, t]$ with no vacant slot for further allocation of j_l's segment found at time slot t, which is the case for Algorithm 3 as well. A similar statement holds for j_h. Therefore j_h and j_l cannot be simultaneously scheduled to meet their deadlines in the remaining time, according to the OCBP-based algorithm.

In the second case, assume j_h is assigned lower priority than j_l by OCBP. Let a_h be the arrival time of job j_h and let the starting and completion times of the LO-criticality execution of j_h be t_h and $t_h{}'$ respectively, and the completion time of the HI-criticality execution be t_e. As in the previous case, all the jobs in $\mathcal{T}_{\text{HI}}$ in the interval $[a_h, t]$ and the jobs in $\mathcal{T}_{\text{LO}}$, including job j_l, in the interval $[a_h, t]$ are of priority (according to OCBP) greater than or equal to that of j_h. OCBP considers $C(\text{HI})$ units of execution time to assign a priority to a HI-criticality job. As seen above, it is clear that after the jobs of higher priority than j_h finish their $C(\text{LO})$ units of execution by time t, there will not be sufficient time for j_h to finish its $C_h(\text{LO})$ units of execution in the interval $[a_h, t_h{}']$ according to OCBP. We know that $C(\text{LO}) \leq C(\text{HI})$. If job j_h doesn't get sufficient time to execute its $C_h(\text{LO})$ units of execution in the interval $[a_h, t_h{}']$, then it will not get sufficient time to execute its $C_h(\text{HI})$ units of execution in the interval $[a_h, t_e]$ either.

From the above two cases, it is clear that OCBP cannot assign priorities to job j_l and j_h, which is a contradiction. This means if there exists an OCBP priority order for instance I, then our algorithm will not encounter a situation where both the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are non-empty at any time t for the instance I.

Note that we need to consider only the LO-criticality scenarios in the proof since Lemma 3 implies that if $\mathcal{S}_{\text{LO}}$ can be constructed, then so can $\mathcal{S}_{\text{HI}}$. □

3.5 Dominance over MCEDF Algorithm

Now we show the dominance of our algorithm over the MCEDF algorithm [10].

Lemma 7: If MCEDF finds a priority order for an instance I, then there exists an MCEDF priority order for I in which all jobs of the same criticality are in EDF order.

Proof. This can be derived directly from the priority assignment to the jobs by the MCEDF algorithm.

□

Theorem 3: If an instance I is schedulable by the MCEDF algorithm, then it is also schedulable by our algorithm.

Proof. The MCEDF algorithm generates a priority order for an instance I. This priority order is used to find the table $\mathrm{PT_{LO}}$. We need to show that if there exists a table $\mathrm{PT_{LO}}$ and the $anyHIscenarioFailure()$ subroutine in Algorithm MCEDF on page 95 of [10] doesn't fail, then our algorithm will not encounter a situation where $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ are non-empty at any time slot t.

We know that $C_i(\mathrm{LO})$ units of execution are allocated to each job j_i for constructing the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$. Each job in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ is allocated as close to its deadline as possible. That means no job can execute after its allocation time in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ without affecting the schedule of any other job and still meet its deadline. Algorithm 3 declares failure if it finds a non-empty slot at any time t in both the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$. This means the two jobs which are found in the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ respectively cannot be scheduled with all other remaining jobs from this point, because all the jobs to the right have already been moved as far to the right as possible.

By Lemma 7, without loss of generality, the MCEDF order is the same as the EDF orders for jobs of the same criticality. So the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ are obtained from the MCEDF order by moving the job segments to the right starting from the right end of the schedule for the LO-criticality and HI-criticality jobs respectively.

Suppose there is an MCEDF priority order of the jobs of instance I and a table $\mathrm{PT_{LO}}$ according to this priority order and suppose the $anyHIscenarioFailure()$ subroutine doesn't fail.

Let j_l and j_h be two jobs in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ respectively found at time t during the construction of table $\mathcal{S}_{\mathrm{LO}}$ by our algorithm, which means all the job segments in the interval $[0, t-1]$ from $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ have already been assigned in $\mathcal{S}_{\mathrm{LO}}$. But, we know that MCEDF has assigned priorities to these jobs j_l and j_h. Now there are two cases.

In the first case, assume j_l is assigned lower priority than j_h by MCEDF. Let a_l be the arrival time of job j_l and t_l and $t_l{}'$ be the starting and completion times of j_l in $\mathcal{T}_{\mathrm{LO}}$ computed by Algorithm 1. Since job j_l can be scheduled only on or after the arrival time a_l, we need to show that the job segment of j_l found at time t cannot be scheduled in the interval $[a_l, t-1]$ by the MCEDF algorithm. We know that Algorithm 3 can allocate a job in table $\mathcal{S}_{\mathrm{LO}}$ on or before its allocation in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$. But Algorithm 3 has not allocated the job segments found in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ at time t in the interval $[a_l, t-1]$ of the table $\mathcal{S}_{\mathrm{LO}}$, and by Lemma 7, this is due to the presence of equal or higher priority job segments of the MCEDF priority order in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$. We know that all the jobs in $\mathcal{T}_{\mathrm{LO}}$ in the interval $[a_l, t]$ and the jobs in $\mathcal{T}_{\mathrm{HI}}$ including job j_h in the interval $[a_l, t]$ are of priority greater or equal to that of j_l according to the MCEDF algorithm since, by moving job segments to the right starting from the EDF schedule, the jobs to the left of j_l are of priority greater than or equal to that of j_l. This means the jobs in the interval $[a_l, t-1]$ of table $\mathcal{S}_{\mathrm{LO}}$ are either equal or higher priority jobs than j_l according to MCEDF. So both the algorithms, MCEDF and ours, allocate higher or equal priority jobs (or, job segments according to Algorithm 3) before time t. Then it is clear that after the jobs of higher priority than j_l finish their $C(LO)$ units of execution, there will not be sufficient time for j_l to finish its $C_l(\mathrm{LO})$ units of execution in the interval $[a_l, t_l{}']$ in the MCEDF schedule. This is because at time t, the MCEDF algorithm has already allocated all ready jobs with higher or equal priority than j_l (according to MCEDF) in the interval $[a_l, t]$ with no vacant slot for further allocation of j_l's segment found at time slot t, which is the case for Algorithm 3 as well. A similar statement holds for j_h. Therefore j_h and j_l cannot be simultaneously scheduled to meet their deadlines in the remaining time, according to MCEDF. In the second case, assume j_h is assigned lower priority than j_l by MCEDF. Let a_h be the arrival time of job j_h and let the starting and completion times of the LO-criticality execution of j_h be t_h and $t_h{}'$ respectively, and the completion time of the HI-criticality execution be t_e. As in the previous case, all the jobs in $\mathcal{T}_{\mathrm{HI}}$ in the interval $[a_h, t]$ and the jobs in $\mathcal{T}_{\mathrm{LO}}$, including job j_l, in the interval $[a_h, t]$ are of priority (according to MCEDF) greater or equal to that of j_h. MCEDF considers $C(\mathrm{HI})$ units of execution time to assign a priority to a HI-criticality job. As seen above, it is clear that after the jobs of higher priority than j_h finish their $C(\mathrm{LO})$ units of execution, there will not be sufficient time for j_h to finish its $C_h(\mathrm{LO})$ units of execution in the interval $[a_h, t_h{}']$ according to MCEDF. We know that $C(\mathrm{LO}) \leq C(\mathrm{HI})$. If job j_h doesn't get sufficient time to execute its $C_h(\mathrm{LO})$ units of execution in the interval $[a_h, t_h{}']$, then it will not get sufficient time to execute its $C_h(\mathrm{HI})$ units of execution in the interval $[a_h, t_e]$ either.

From the above two cases, it is clear that MCEDF may assign priorities to job j_l and j_h but the $anyHIscenarioFailure()$ subroutine will fail, which is a contradiction. This means if the MCEDF algorithm finds a schedule for instance I, then our algorithm will not encounter a situation where both the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are non-empty at any time t for the instance I.

Note that we need to consider only the LO-criticality scenarios in the proof since Lemma 3 implies that if $\mathcal{S}_{\text{LO}}$ can be constructed, then so can $\mathcal{S}_{\text{HI}}$. □

4 Extension for m criticality levels

The algorithm discussed in Section 3 constructs two scheduling tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ for the dual-criticality instances which can be used by the scheduler to dispatch the jobs. Now we extend our algorithm for instances with m criticality levels. Here we need to create m different tables for m criticality levels which can be used by the scheduler to dispatch the jobs.

4.1 Model

A job is characterized by a 5-tuple of parameters: $j_i = (a_i, d_i, \chi_i, \{C_i(1), C_i(2), \ldots, C_i(m)\})$, where

- $a_i \in \mathbb{N}$ denotes the *arrival time*.
- $d_i \in \mathbb{N}^+$ denotes the *absolute deadline*.
- $\chi_i \in \mathbb{N}^+$ denotes the *criticality* level.
- $\{C_i(1), C_i(2), \ldots, C_i(m)\}$ denotes the *worst-case execution time* at each criticality level.

We assume that $C_i(k)$ is monotonically increasing with increasing k, i.e., $\forall i : C_i(1) \leq C_i(2) \leq \ldots \leq C_i(m)$, where $1 \leq i \leq n$.

Definition 3: An m *criticality* MC instance I is said to be *time-triggered schedulable* if it is possible to construct m tables $\mathcal{S}_1$, $\mathcal{S}_2$, …, $\mathcal{S}_m$ such that the scheduler can schedule any non-erroneous scenario of instance I.

The following scheduler algorithm is used to dispatch the jobs using the m tables at run time.

- Initially $\chi_i = 1$
- The criticality level indicator Γ is initialized to χ_i.
- Repeat
  - While ($\Gamma = \chi_i$), at each time instant t the job available at time t in the table $\mathcal{S}_{\chi_i}$ will execute.
  - If a job executes for more than its χ_i-criticality WCET without signaling completion, then Γ is changed to $\chi_i + 1$.

4.2 Algorithm

Here we need to construct m tables to find a time triggered schedule. Each table is of length D_{max} as in Equation 1.

Algorithm 4 constructs m temporary tables $\mathcal{T}_1$, $\mathcal{T}_2$, …, $\mathcal{T}_m$. For each table $\mathcal{T}_{\chi_i}$ where $\chi_i \in \{1, 2, \ldots, m\}$, Algorithm 4 chooses jobs with χ_i-criticality level and orders them in EDF order. Then, all the job segments of the EDF order are moved as close to their deadline as possible so that no job misses its deadline in $\mathcal{T}_{\chi_i}$. Then out of the total allocation so far, the algorithm allocates $C_i(1)$ units of execution of j_i in $\mathcal{T}_{\chi_i}$ from the beginning of its slot and leaves the rest of the execution time of j_i unallocated in $\mathcal{T}_{\chi_i}$. This is similar to the dual criticality case.

Now, we use Algorithm 5 to construct the table $\mathcal{S}_1$ from tables $\mathcal{T}_1$, $\mathcal{T}_2$, …, $\mathcal{T}_m$. The algorithm starts the construction of $\mathcal{S}_1$ from time 0 and checks all m tables simultaneously. There will be three situations while merging these tables to construct $\mathcal{S}_1$. At time slot t, one of the following can occur:

Algorithm 4 Construct_TT_m-crit_$\mathcal{T}_{\chi_i}(I)$

Notation:
$I = \{j_1, j_2, \ldots, j_n\}$, where
$j_i =< a_i, d_i, \chi_i, \{C_i(1), C_i(2), \ldots, C_i(m)\} >$.
Input : I
Output : $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m$
Assume earliest arrival time is 0.

```
1: Find the maximum deadline (D_max) of the jobs;
2: for χ_i := 1 to m do
3:    Prepare a temporary table T_χi of maximum length D_max;
4:    Let Ψ be the set of χ_i-criticality jobs of instance I;
5:    Let O be the EDF order of the jobs of Ψ on the time-line using C_i(χ_i) units of execution for job j_i ;
6:    if (any job cannot be scheduled) then
7:       Declare failure;
8:    end if
9:    Starting from the rightmost job segment of the EDF order of Ψ, move each segment of a job j_i as close to its
      deadline as possible in T_χi.
10:   for k := 1 to |Ψ| do
11:      Allocate C_k(1) units of execution to job j_k from its starting time in T_χi and leave the rest unallocated;
12:   end for
13: end for
```

1. All m tables are empty.
2. Two or more tables from the m tables are not empty.
3. Exactly one table from the m tables is not empty.

If situation 1 occurs, then the algorithm will allocate the nearest ready job to the right at time slot t where a lower criticality job gets higher priority than a higher criticality job. After the allocation of the job j_i in $\mathcal{S}_1$, that instant of j_i in $\mathcal{T}_{\chi_i}$ is marked empty. In case of situation 2, the algorithm declares failure to schedule. In situation 3, the algorithm allocates the first available job from the table which is non-empty at time t in $\mathcal{S}_1$.

Then we construct the table $\mathcal{S}_2$ from $\mathcal{S}_1$. We first copy the jobs of table $\mathcal{S}_1$ to table $\mathcal{S}_2$. Then all the jobs whose criticality are greater than 1 need to be allocated $C_i(2) - C_i(1)$ units of execution time immediately after their $C_i(1)$ units of execution in $\mathcal{S}_2$. These additional time units is allocated by pushing all overlapping jobs whose criticality is greater than or equal to 2 to the right and overwriting any job with criticality 1 in the process. If the allocation time of a job whose criticality is 2 or more which needs to be pushed is same in both the tables $\mathcal{S}_2$ and $\mathcal{T}_2$, then the additional time units are allocated after this job.

Similarly, we construct the table $\mathcal{S}_{\chi_i}$ from $\mathcal{S}_{\chi_i-1}$. We first copy the jobs of table $\mathcal{S}_{\chi_i-1}$ to table $\mathcal{S}_{\chi_i}$. Then the χ_i-criticality jobs are allocated $C_i(\chi_i) - C_i(\chi_i - 1)$ units of χ_i-criticality execution time immediately after their $C_i(\chi_i - 1)$ units of execution in $\mathcal{S}_{\chi_i}$. These additional time units is allocated by pushing all overlapping jobs whose criticality is greater than or equal to χ_i to the right and overwriting any job with criticality less than or equal to $(\chi_i - 1)$ in the process. If the allocation time of a χ_i-criticality job which needs to be pushed is same in both the tables $\mathcal{S}_{\chi_i}$ and $\mathcal{T}_{\chi_i}$, then the additional time units are allocated after this job.

Algorithm 5 TT_Merge_m-crit(I, $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_m$)

Notation:

$I = \{j_1, j_2, \ldots, j_n\}$.

$j_i =< a_i, d_i, \chi_i, \{C_i(1), C_i(2), \ldots, C_i(m)\} >$.

Input : I, $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_m$

Output : Tables $\mathcal{S}_1$, $\mathcal{S}_2$, ..., $\mathcal{S}_m$

Construction of $\mathcal{S}_1$.
Find the maximum deadline (D_{max}) of the jobs;
The maximum length of tables $\mathcal{S}_1$, $\mathcal{S}_2$,..., $\mathcal{S}_m$ are D_{max} each;
$t := 0$;
while ($t \leq D_{max}$) **do**
  if ($|\{\chi_i | T_{\chi_i}[t] \neq NULL\}| = 0$) **then**
    Search the tables $\mathcal{T}_{\chi_i}$ simultaneously from the beginning to find the first available job at time t;
    Let k be the first occurrence, if any, of such a job j_i in $\mathcal{T}_{\chi_i}$;
    if (more than one job is found) **then**
      LC := the lowest criticality such that a job j_i is found in $\mathcal{T}_{LC}$;
      $\mathcal{S}_1[t] := \mathcal{T}_{LC}[k]$;
      $\mathcal{T}_{LC}[k] := NULL$;
    else if (only job of χ_i-criticality level is found) **then**
      $\mathcal{S}_1[t] := \mathcal{T}_{\chi_i}[k]$;
      $\mathcal{T}_{\chi_i}[k] := NULL$;
    else if (no job is found) **then**
      $\mathcal{S}_1[t] := NULL$
      $t := t + 1$;
    end if
  else if ($|\{\chi_i | T_{\chi_i}[t] \neq NULL\}| = 1$) **then**
    $\mathcal{S}_1[t] := \mathcal{T}_{\chi_i}[t]$;
    $\mathcal{T}_{\chi_i}[t] := NULL$;
    $t := t + 1$;
  else if ($|\{\chi_i | T_{\chi_i}[t] \neq NULL\}| > 1$) **then**
    Declare failure;
  end if
end while
This is the table $\mathcal{S}_1$;

Construction of $\mathcal{S}_{\chi_i}$ where $2 \leq \chi_i \leq m$
for $\chi_i := 2$ to m **do**
  Copy all the jobs from table $\mathcal{S}_{\chi_i - 1}$ to table $\mathcal{S}_{\chi_i}$;
  Scan the table $\mathcal{S}_{\chi_i}$ from left to right:
  for each χ_i-criticality job j_l, allocate an additional $C_l(\chi_i) - C_l(\chi_i - 1)$ time units after the rightmost segment of job j_l, recursively pushing all the overlapping job segments with criticality greater or equal to χ_i-criticality in $\mathcal{S}_{\chi_i}$ (except those whose allocation time is same as in $\mathcal{T}_{\chi_i}$) to the right and overwriting any jobs with criticality $(\chi_i - 1)$-criticality or lesser in the process.
end for

4.3 Correctness Proof

Theorem 4: If the scheduler dispatches the jobs according to tables $\mathcal{S}_1$, $\mathcal{S}_2$, ..., $\mathcal{S}_m$, then it will be a correct scheduling strategy.

Proof. We prove the theorem by strong induction.
Let $S(i)$ be the statement "If the scheduler dispatches the jobs according to tables $\mathcal{S}_1$, $\mathcal{S}_2$, ..., $\mathcal{S}_i$, then it will be a correct scheduling strategy up to criticality level i."
BASE STEP ($i = 2$): Since $i = 2$ is a dual criticality instance for which the correctness has already been proved in the previous section, $S(2)$ is true.
INDUCTIVE STEP: Fix some $i \geq 2$, and assume that for every t satisfying $2 \leq t \leq i$, the statement $S(t)$ is true.
Now we need to show that $S(i+1)$ is true, i.e., if the algorithm finds a correct online scheduling policy up to the i-criticality level using the first i scheduling tables, then there exists an online scheduling policy for $(i+1)$-criticality levels using the first $(i+1)$ tables. We know that $S(i)$ is true which means for the i-criticality level, the scheduler dispatches the job according to the first i tables which is a correct online scheduling strategy for i-criticality levels. Algorithm 5 starts constructing the table $\mathcal{S}_{(i+1)}$ from the table $\mathcal{S}_i$ keeping the same order of the jobs. According to Algorithm 5, after $C_l(i)$ units of execution for each job j_l of $\chi_{(i+1)}$-criticality level is allocated, the remaining $\{C_l(i+1) - C_l(i)\}$ units of execution has been allocated to them immediately after the rightmost job segment in the table $\mathcal{S}_{(i+1)}$ while following the job order of table $\mathcal{S}_i$. So, each job j_l of $\chi_{(i+1)}$-criticality gets sufficient time to execute their $C_l(i+1)$ units of execution in $\mathcal{S}_{(i+1)}$. This proof is similar to the dual criticality case. Hence, we get a correct online scheduling policy. □

5 Extension for dependent jobs

In previous sections we have considered instances with independent jobs. Now we consider the case of dual-criticality instances with dependent jobs. In this section we design algorithms to find two scheduling tables such that if the scheduler discussed in Section 2 dispatches the jobs according to these two tables then it will be a correct online scheduling strategy without violating the dependencies between them. To the best of our knowledge, there is no existing algorithm which can schedule the jobs of an instance I with dependencies, although a similar type of problem is discussed in Baruah [11] based on synchronous programs. First we discuss the case of non-recurrent jobs and then we extend it for recurrent or periodic jobs.

5.1 Model

A job is characterized by a 5-tuple of parameters: $j_i = (a_i, d_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}))$, where

- $a_i \in \mathbb{N}$ denotes the *arrival time*.
- $d_i \in \mathbb{N}^+$ denotes the absolute *deadline*.
- $\chi_i \in \{LO, HI\}$ denotes the *criticality* level.
- $C_i(\text{LO}) \in \mathbb{N}^+$ denotes the LO-criticality *worst-case execution time*.
- $C_i(\text{HI}) \in \mathbb{N}^+$ denotes the HI-criticality *worst-case execution time*.

We assume that $\forall i : C_i(\text{LO}) \leq C_i(\text{HI})$, where $1 \leq i \leq n$ and $\chi_i \in \{LO, HI\}$.

An instance of a mixed-criticality system with dependent jobs can be defined as a *directed acyclic graph* (DAG). An instance I is represented in the form of $I(V, E)$, where V represents the set of jobs $\{j_1, j_2, \ldots, j_n\}$ and E represents the dependencies between the jobs. We also assume that no HI-criticality job can depend on a LO-criticality job. This means, there will be no instance where an outward edge from a LO-criticality job becomes an inward edge to a HI-criticality job.

Definition 4: A dual-criticality MC instance I with job dependencies is said to be **time-triggered schedulable** if it is possible to construct the two schedules $\mathcal{S}_{LO}$ and $\mathcal{S}_{HI}$ for I without violating the dependencies, such that the run-time scheduler algorithm described above schedules I in a correct manner.

5.2 The Algorithm

Here we propose an algorithm which can construct two scheduling tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ for a dual-criticality instance with dependent jobs. If the scheduler discussed in Section 2 dispatches job according to these two tables, then this will be a correct scheduling strategy.

We construct the tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ from two temporary tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. The length of all these tables are D_{max}, i.e., the length of the maximum deadline among all the jobs in the instance.

Algorithm 6 constructs a subgraph Ψ which consists of all the LO-criticality jobs and the edges between them. Then it finds a job j_i with the smallest deadline and no inward edges and allocates its $C_i(\text{LO})$ units of execution in $\mathcal{T}_{\text{LO}}$. After $C_i(\text{LO})$ units of execution of the job is allocated, the job and all its outward edges are removed from Ψ. The process continues until all the jobs in Ψ are scheduled. Then all job segments in $\mathcal{T}_{\text{LO}}$ are shifted as close to their deadlines as possible without violating the dependencies between them so that no job misses their deadline. For an example see Fig. 4 and Fig. 5

Algorithm 6 Construct_Dependency_$\mathcal{T}_{\text{LO}}(I)$

Notation:
$I = \{j_1, j_2, \ldots, j_n\}$, where
$j_i =< a_i, d_i, p_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}) >$.
Input : I
Output : $\mathcal{T}_{\text{LO}}$
Assume earliest arrival time is 0.

1: Find the maximum deadline (D_{max}) of the jobs;
2: Prepare a temporary table $\mathcal{T}_{\text{LO}}$ of maximum length D_{max};
3: Let Ψ be the subgraph of DAG I containing LO-criticality jobs and the edges between them;
4: **repeat**
5: Choose an available job j_i from Ψ with the earliest deadline that doesn't have an inward edge.
6: Allocate j_i's execution time at the next available slot in the temporary table $\mathcal{T}_{\text{LO}}$;
7: **if** (j_i's $C_i(\text{LO})$ units of execution is allocated) **then**
8: delete j_i and its outward edges from Ψ;
9: **end if**
10: **if** (job j_i misses its deadline) **then**
11: Declare failure and exit;
12: **end if**
13: **until** (all the jobs in Ψ are allocated)
14: Let O be the final order of the jobs of Ψ on the time-line of $\mathcal{T}_{\text{LO}}$ using $C_i(\text{LO})$ units of execution for job j_i;
15: Starting from the rightmost job segment of the order O, move each segment of a job j_i as close to its deadline as possible in $\mathcal{T}_{\text{LO}}$ without violating the dependency;

Algorithm 7 constructs a subgraph Ψ which consists of all the HI-criticality jobs and the edges between them. Then it finds a job j_i with the smallest deadline and no inward edges and allocates $C_i(\text{HI})$ units of execution to it in $\mathcal{T}_{\text{HI}}$. After $C_i(\text{HI})$ units of execution of the job is allocated, the job and all its outward edges are removed from Ψ. The process continues until all the jobs in Ψ are scheduled. Then all job segments in $\mathcal{T}_{\text{HI}}$ are shifted as close to their deadlines as possible without violating the dependencies between them so that no job miss their deadline.

Then out of the total allocation so far, it allocates C_i(LO) units of execution of job j_i in $\mathcal{T}_{\text{HI}}$ from the beginning of its slot and leaves the rest of the execution time of j_i unallocated in $\mathcal{T}_{\text{HI}}$, as in Fig. 5 and Fig. 6.

Algorithm 7 Construct_Dependency_$\mathcal{T}_{\text{HI}}(I)$

Notation:
$I = \{j_1, j_2, \ldots, j_n\}$, where
$j_i =< a_i, d_i, p_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}) >$.
Input : I
Output : $\mathcal{T}_{\text{HI}}$
Assume earliest arrival time is 0.

1: Find the maximum deadline (D_{max}) of the jobs;
2: Prepare a temporary table $\mathcal{T}_{\text{HI}}$ of maximum length D_{max};
3: Let Ψ be the subgraph of DAG I containing HI-criticality jobs that the edges between them;
4: **repeat**
5: Choose an available job j_i from Ψ with the earliest deadline and doesn't have an inward edge.
6: Allocate j_i's execution time at the next available slot in the temporary table $\mathcal{T}_{\text{HI}}$;
7: **if** (j_i's C_i(HI) units of execution is allocated) **then**
8: delete j_i and its outward edges from Ψ;
9: **end if**
10: **if** (job j_i misses its deadline) **then**
11: Declare failure and exit;
12: **end if**
13: **until** (all the jobs in Ψ are allocated)
14: Let O be the final order of the jobs of Ψ on the time-line of $\mathcal{T}_{\text{HI}}$ using C_i(HI) units of execution for job j_i ;
15: Starting from the rightmost job segment of the order O, move each segment of a job j_i as close to its deadline as possible in $\mathcal{T}_{\text{HI}}$ without violating the dependency.
16: **for** $i := 1$ to m **do**
17: Allocate C_i(LO) units of execution to job j_i from its starting time in $\mathcal{T}_{\text{HI}}$ and leave the rest unallocated;
18: **end for**

Now, we use Algorithm 8 to construct the table $\mathcal{S}_{\text{LO}}$ from $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ and then construct $\mathcal{S}_{\text{HI}}$ from $\mathcal{S}_{\text{LO}}$. The algorithm starts the construction of $\mathcal{S}_{\text{LO}}$ from time 0 and checks the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ simultaneously at each instant. There are four possibilities while merging the two temporary tables to construct $\mathcal{S}_{\text{LO}}$.
At time t, one of the following situations can occur.

1. Both $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are empty.
2. Both $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are not empty.
3. $\mathcal{T}_{\text{LO}}$ is empty and $\mathcal{T}_{\text{HI}}$ is not empty.
4. $\mathcal{T}_{\text{LO}}$ is not empty and $\mathcal{T}_{\text{HI}}$ is empty.

If situation 1 occurs, then the algorithm will search both the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ to find the first available job in both the table. Then, it allocates one of the available jobs at time t where a LO-criticality job gets higher priority over a HI-criticality job. If a LO-criticality job is chosen to be allocated, then all the predecessor of that job must be finished allocation. Then the place of the ready job in $\mathcal{T}_{\text{LO}}$ or $\mathcal{T}_{\text{HI}}$ is marked as empty. In case of situation 2, the algorithm declares failure to schedule. In situation 3, the algorithm allocates the HI-criticality job from $\mathcal{T}_{\text{HI}}$ whereas in situation 4, it allocates the LO-criticality job from $\mathcal{T}_{\text{LO}}$ if and only if all the predecessor of the job has already

finished allocation. Once an instant of a job is allocated in $\mathcal{S}_{\text{LO}}$, the place where it was scheduled in $\mathcal{T}_{\text{LO}}$ or $\mathcal{T}_{\text{HI}}$ is emptied.

We then construct the table $\mathcal{S}_{\text{HI}}$ from $\mathcal{S}_{\text{LO}}$. We first copy the jobs of table $\mathcal{S}_{\text{LO}}$ to $\mathcal{S}_{\text{HI}}$. Then the HI-criticality jobs are allocated their $C_i(\text{HI}) - C_i(\text{LO})$ units of HI-criticality execution time immediately after their $C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{HI}}$. These additional time units are allocated by recursively pushing all overlapping HI-criticality jobs in $\mathcal{S}_{\text{HI}}$ to the right and overwriting any LO-criticality job in the process. An exception to this is when the allocation time of an overlapping HI-criticality job is the same in both the tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{T}_{\text{HI}}$, in which case the additional time units are allocated after this job without violating the dependency constraints.

We illustrate the algorithm by an example.

Example 5: Consider the instance shown in Fig. 14. This is an instance with five jobs j_1, j_2, j_3, j_4 and j_5 with

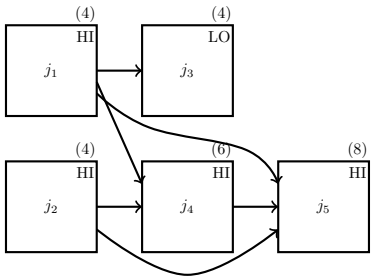

Figure 14: A DAG showing job dependencies. The numbers in parentheses indicates deadline

dependencies between them. The properties of these jobs can be seen from Table 4.

Job	Arrival time	Deadline	Criticality	$C_i(\text{LO})$	$C_i(\text{HI})$
j_1	0	4	HI	1	2
j_2	0	4	HI	1	2
j_3	0	4	LO	1	1
j_4	0	6	HI	1	2
j_5	3	8	HI	1	2

Table 4: Instance for Example 5

We find the two temporary tables as in Example 3. But, here we need to take care of the job dependencies. So, we apply Algorithm 6 and 7 to construct the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ shown in Fig. 15 and 16.

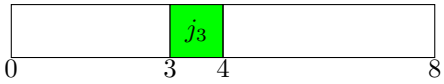

Figure 15: Temporary table $\mathcal{T}_{\text{LO}}$

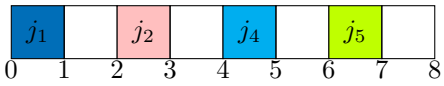

Figure 16: Temporary table $\mathcal{T}_{\text{HI}}$

Then we use Algorithm 8 to find the table $\mathcal{S}_{\text{LO}}$ which is shown in Fig. 17. Finally, we construct the table $\mathcal{S}_{\text{HI}}$ from the table $\mathcal{S}_{\text{LO}}$ which is shown in Fig. 18. □

Algorithm 8 TT_Dependency_Merge_crit($I, \mathcal{T}_{\rm LO}, \mathcal{T}_{\rm HI}$)

$I = \{j_1, j_2, \ldots, j_n\}$.

$j_i =< a_i,\ d_i,\ p_i,\ \chi_i,\ C_i(\text{LO}),\ C_i(\text{HI}) >$.

Input : $I, \mathcal{T}_{\rm LO}, \mathcal{T}_{\rm HI}$

Output : Tables $\mathcal{S}_{\rm LO}$ and $\mathcal{S}_{\rm HI}$

1: **Construction of** $\mathcal{S}_{\rm LO}$.
2: Find the maximum deadline (D_{max}) of the jobs;
3: The maximum length of tables $\mathcal{S}_{\rm HI}$ and $\mathcal{S}_{\rm LO}$ are both D_{max};
4: $t := 0$;
5: **while** ($t \leq L$) **do**
6: **if** ($\mathcal{T}_{\rm LO}[t] = NULL$ & $\mathcal{T}_{\rm HI}[t] = NULL$) **then**
7: Search the tables $\mathcal{T}_{\rm LO}$ and $\mathcal{T}_{\rm HI}$ simultaneously from the beginning to find the first available job at time t;
8: Let k be the first occurrence of a job j_i in $\mathcal{T}_{\rm LO}$ or $\mathcal{T}_{\rm HI}$;
9: **if** (Both LO-criticality & HI-criticality job are found) **then**
10: **if** (Predecessors of $\mathcal{T}_{\rm LO}[k]$ has been allocated its C_i(LO) execution time) **then**
11: $\mathcal{S}_{\rm LO}[t] := \mathcal{T}_{\rm LO}[k]$;
12: $\mathcal{T}_{\rm LO}[k] := NULL$;
13: **else**
14: $\mathcal{S}_{\rm LO}[t] := \mathcal{T}_{\rm HI}[k]$;
15: $\mathcal{T}_{\rm HI}[k] := NULL$;
16: **end if**
17: **else if** (LO-critical job is found) **then**
18: **if** (Predecessors of $\mathcal{T}_{\rm LO}[k]$ has been allocated its C_i(LO) execution time) **then**
19: $\mathcal{S}_{\rm LO}[t] := \mathcal{T}_{\rm LO}[k]$;
20: $\mathcal{T}_{\rm LO}[k] := NULL$;
21: **else**
22: $\mathcal{S}_{\rm LO}[t] := NULL$;
23: $t := t + 1$;
24: **end if**
25: **else if** (HI-criticality job is found) **then**
26: $\mathcal{S}_{\rm LO}[t] := \mathcal{T}_{\rm HI}[k]$;
27: $\mathcal{T}_{\rm HI}[k] := NULL$;
28: **else if** (NO job is found) **then**
29: $\mathcal{S}_{\rm LO}[t] := NULL$
30: $t := t + 1$;
31: **end if**
32: **else if** ($\mathcal{T}_{\rm LO}[t] = NULL$ & $\mathcal{T}_{\rm HI}[t]$!= $NULL$) **then**
33: $\mathcal{S}_{\rm LO}[t] := \mathcal{T}_{\rm HI}[t]$;
34: $\mathcal{T}_{\rm HI}[t] := NULL$;
35: $t := t + 1$;
36: **else if** ($\mathcal{T}_{\rm LO}[t]$!= $NULL$ & $\mathcal{T}_{\rm HI}[t] = NULL$) **then**
37: $\mathcal{S}_{\rm LO}[t] := \mathcal{T}_{\rm LO}[t]$;
38: $\mathcal{T}_{\rm LO}[t] := NULL$;
39: $t := t + 1$;

else if ($\mathcal{T}_{\text{LO}}[t]$!= $NULL$ & $\mathcal{T}_{\text{HI}}[t]$!= $NULL$) **then**
  Declare failure;
end if
end while
This is the table $\mathcal{S}_{\text{LO}}$;

Construction of $\mathcal{S}_{\text{HI}}$
Copy all the jobs from table $\mathcal{S}_{\text{LO}}$ to table $\mathcal{S}_{\text{HI}}$;
Scan the table $\mathcal{S}_{\text{HI}}$ from left to right:
for each HI-criticality job j_i, allocate an additional $C_i(\text{HI}) - C_i(\text{LO})$ time units immediately after the rightmost segment of job j_i, recursively pushing all the overlapping HI-criticality job segments in $\mathcal{S}_{\text{HI}}$ (except those whose allocation time is same as in $\mathcal{T}_{\text{HI}}$) to the right and overwriting any LO-criticality jobs in the process.

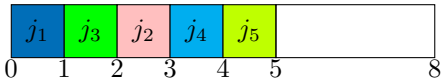

Figure 17: Final table $\mathcal{S}_{\text{LO}}$

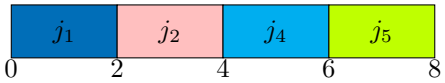

Figure 18: Final table $\mathcal{S}_{\text{HI}}$

5.3 Correctness Proof

We need to show that if our algorithm finds the two tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$, then the scheduler can find an online scheduling strategy using these tables.

Lemma 8: If Algorithm 8 doesn't declare failure, then each job j_i receives $C_i(\text{LO})$ units of execution in $\mathcal{S}_{\text{LO}}$ and each HI-criticality job j_k receives $C_k(\text{HI})$ units of execution in $\mathcal{S}_{\text{HI}}$ without violating the dependency constraints.

Proof. The table $\mathcal{S}_{\text{LO}}$ is constructed from the two temporary tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. Each LO-criticality job j_i can be allocated in $\mathcal{S}_{\text{LO}}$ on or before its scheduled time instant in $\mathcal{T}_{\text{LO}}$ if and only if all of its predecessor jobs have completed their allocation, which doesn't violate the dependencies. We have already assumed that no HI-criticality job depends on any LO-criticality job. We know that each job in $\mathcal{T}_{\text{HI}}$ is allocated according to its dependency constraints. So each HI-criticality job j_i can be allocated in $\mathcal{S}_{\text{LO}}$ on or before its scheduled time instant in $\mathcal{T}_{\text{HI}}$. If our algorithm finds a table $\mathcal{S}_{\text{LO}}$ then each job must receives $C_i(\text{LO})$ units of execution time.

Next we show that any HI-criticality job j_k receives $C_k(\text{HI})$ units of execution in $\mathcal{S}_{\text{HI}}$. We start constructing $\mathcal{S}_{\text{HI}}$ by copying the jobs in $\mathcal{S}_{\text{LO}}$. But according to our algorithm, the HI-criticality jobs are allocated their remaining $C_k(\text{HI}) - C_k(\text{LO})$ units of allocation in $\mathcal{S}_{\text{HI}}$ after they complete their $C_k(\text{LO})$ units of allocation in $\mathcal{S}_{\text{HI}}$ by pushing recursively all the following HI-criticality job segments to the right except those whose allocation is the same as in table $\mathcal{T}_{\text{HI}}$ and without violating the dependency constraints. This means we can push a job segment to the right in $\mathcal{S}_{\text{HI}}$ only if it is allocated before its allocation in $\mathcal{T}_{\text{HI}}$ and, moreover, no job is pushed beyond its allocation in $\mathcal{T}_{\text{HI}}$, because if $\mathcal{T}_{\text{HI}}$ doesn't declare failure then it allocates enough time for the execution of all the HI-criticality jobs without violating the dependency constraints. In this case, all the jobs can get sufficient time to schedule their $C_k(\text{HI}) - C_k(\text{LO})$ units of execution as they are allocated on or before the allocation in table $\mathcal{T}_{\text{HI}}$. This is clear from the remark following Observation 1 which holds for dependent jobs as well. If a HI-criticality job j_h cannot be pushed to the right then it will get its remaining $C_h(\text{HI}) - C_h(\text{LO})$ units of execution time in table $\mathcal{S}_{\text{HI}}$ by a similar reasoning as above. □

Theorem 5: If the scheduler dispatches the jobs according to $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$, then it will be a correct scheduling strategy without violating the dependency constraints.

Proof. Algorithm 6 and 7 take care of all the dependencies between LO-criticality and HI-criticality jobs respectively. We know that Algorithm 8 checks the dependencies of the LO-criticality jobs on HI-criticality jobs before allocating the LO-criticality jobs. We have assumed that no HI-criticality job depends on a LO-criticality job. So the construction of the tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ doesn't violate the dependency constraints of instance I. From Lemma 8, it is clear that each job in $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ receives $C(\text{LO})$ and $C(\text{HI})$ units of execution respectively. The rest of the proof is similar to that of Theorem 1. □

5.4 Generalizing the algorithm for m criticality levels

We know that Algorithm 8 can find two tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ which can be used by the scheduler for correct online scheduling policy. In Section 4, we have already proved that the dual-criticality algorithm can be modified to find m number of tables which can be used by the scheduler to find a correct online scheduling strategy for m criticality levels. Thus, we can say that the algorithm discussed in this section can be extended to find m tables which can be used by the scheduler to find a correct online scheduling strategy.

6 Extension for periodic jobs

Now we extend our algorithm for periodic or recurrent jobs. Here, a job is characterized by a 5-tuple of parameters: $j_i = (a_i, p_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}))$, where

- $a_i \in \mathbb{N}$ denotes the *arrival time*.
- $p_i \in \mathbb{N}^+$ denotes the *period*.
- $\chi_i \in \{LO, HI\}$ denotes the *criticality* level.
- $C_i(\text{LO}) \in \mathbb{N}^+$ denotes the LO-criticality *worst-case execution time*.
- $C_i(\text{HI}) \in \mathbb{N}^+$ denotes the HI-criticality *worst-case execution time*.

We assume that $\forall i : C_i(\text{LO}) \leq C_i(\text{HI})$, where $1 \leq i \leq n$ and $\chi_i \in \{LO, HI\}$. Note that in this report, we also assume that $p_i = d_i$, where d_i is the deadline and $1 \leq i \leq n$.

As we can see that the job model is very much similar to the non-recurrent jobs except the periods which initiate the new instance of the job. The process of constructing a time-triggered schedule for the jobs having the above dual-criticality model will be very similar to the one we have discussed in Section 3. Here we follow the same algorithms as in Section 3 to find the two tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$. These two tables will be used by the scheduler to dispatch the jobs at each instant of time.

All algorithms discussed in Section 3 constructed the tables of length D_{max}. But, in this case, all the tables will have length equals to the lcm or hyper-period L of periods of all the jobs. Here we need to modify Algorithms 1 and 2 only. We find the EDF order of the LO-criticality and HI-criticality jobs up to the hyper-period L in the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ respectively. Then we can use Algorithm 3 to find tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$.

7 Comparison with mixed-criticality synchronous programs

Baruah [11] proposed a technique to schedule mixed-criticality synchronous programs on a uniprocessor system. He showed that the scheduling of mixed-criticality *single-rate* synchronous program is polynomial time solvable whereas the optimal and efficient scheduling of mixed-criticality *multi-rate* synchronous program is NP-hard in the strong sense. He also proved that the schedule generation algorithm which finds a schedule for single-rate synchronous

programs is optimal. Baruah used graphs to represent the reactive blocks and their dependencies. The multi-rate graph of a synchronous program is unrolled to find a *directed acyclic graph (DAG)* in which each invocation of each block within an interval, of length equal to the lcm of the periods, is explicitly represented as a separate node. Each node is then assigned a priority according to the OCBP algorithm. From the above priorities, two tables can be constructed which can be used by the scheduler to dispatch the blocks. We present an algorithm which can construct two tables with which we can schedule a strict superset of OCBP-schedulable mixed-criticality multi-rate programs.

7.1 Model

We follow the same model of synchronous program as suggested in [11]. The model is described as follows.

- The synchronous program is represented as a *directed acyclic graph* $G(V, E)$, where V is the set of vertices and E is the set of edges. The blocks $(B_1, \ldots, B_n)$ of the synchronous program are represented as the vertices of the graphs, i.e., $B_i \in V$. The dependencies between the blocks (B_i, B_j) is represented by the directed edges, i.e., $(B_i, B_j) \in E$.
- Some of the blocks are designated as *output* blocks and *input* blocks; these generate output and input values of the synchronous program. Other blocks are called *internal* blocks.
- Each block is characterized by a 5-tuple of parameters: $B_i = (a_i, p_i, \chi_i, C_i(LO),$ $C_i(HI))$, where
  - $a_i \in \mathbb{N}$ denotes the *arrival time*.
  - $p_i \in \mathbb{N}^+$ denotes the *period*.
  - $\chi_i \in \{LO, HI\}$ denotes the *criticality* level.
  - $C_i(\text{LO}) \in \mathbb{N}^+$ denotes the LO-criticality *worst-case execution time*.
  - $C_i(\text{HI}) \in \mathbb{N}^+$ denotes the HI-criticality *worst-case execution time*.
- We assume that $\forall i : C_i(\text{LO}) \leq C_i(\text{HI})$, where $1 \leq i \leq n$ and $\chi_i \in \{LO, HI\}$. Note that in this report, we also assume that $p_i = d_i$, where d_i is the deadline.
- Each output block can either be a HI-criticality or a LO-criticality block. We assume that a HI-criticality block cannot depend upon a LO-criticality block. This means if a block B_i is a HI-criticality block, then all the preceding blocks of B_i will be HI-criticality blocks.

As discussed earlier, the CAs are interested in the certification of the values of the HI-criticality output blocks only, whereas the system designers want to verify the correctness of all the blocks in a synchronous program.

Example 6: Let us consider an instance I given in Table 5 and its corresponding DAG in Fig. 19. Since we are

Block	Arrival time	Period	Criticality	C_i(LO)	C_i(HI)
B_1	0	14	HI	3	5
B_2	2	14	HI	1	2
B_3	0	7	LO	3	3
B_4	0	14	HI	3	7

Table 5: Instance for Example 6

considering a periodic instance, the instance I is unrolled according to the method given in [11]. The resulting DAG is given in Fig. 20. We try to apply the OCBP algorithm to find the priority from which the tables $\text{S}^{\text{oc}}_{\text{LO}}$ and $\text{S}^{\text{oc}}_{\text{HI}}$ are constructed. As the procedure shown in [11], the blocks $B_3(1)$ is chosen to be assigned the lowest priority block. Since $B_3(1)$ is a LO-criticality block, we need to consider $C(\text{LO})$ units of execution of each block. Now we can see

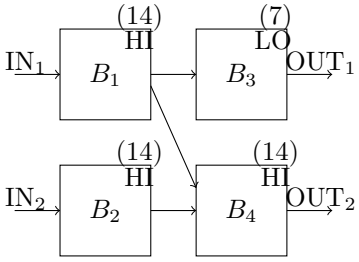

Figure 19: DAG of instance I given in Table 5

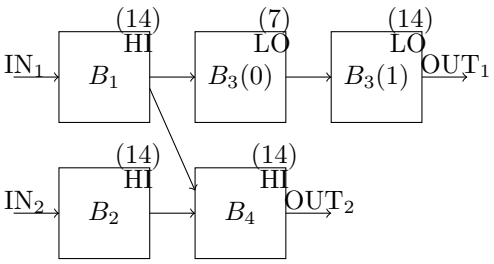

Figure 20: DAG after unroll

that block B_1 can execute over $[0, 3]$, block $B_3(0)$ can execute over $[3, 6]$, block B_2 can execute over $[6, 7]$ and block B_4 can execute over $[7, 10]$. So there is sufficient time for $B_3(1)$ to execute its three units of execution. Thus, block $B_3(1)$ is assigned lowest priority. Now, we can see that no more blocks can be assigned a priority. Since, there is no OCBP priority order, the algorithm discussed in [11] cannot construct the two scheduling tables $\mathrm{S}^{\mathrm{oc}}_{\mathrm{LO}}$ and $\mathrm{S}^{\mathrm{oc}}_{\mathrm{HI}}$.

Now we apply our algorithm 8 on the synchronous program given in Example 6 to construct the two scheduling tables $\mathcal{S}_{\mathrm{LO}}$ and $\mathcal{S}_{\mathrm{HI}}$. We consider the unrolled synchronous program given in Fig. 20 to find the scheduling tables. As we know, we need two temporary tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ to construct the scheduling table $\mathcal{S}_{\mathrm{LO}}$. Then $\mathcal{S}_{\mathrm{HI}}$ will be constructed using $\mathcal{S}_{\mathrm{LO}}$.

First, Algorithm 6 and 7 construct the two temporary tables as shown in Fig. 21. Then Algorithm 8 constructs

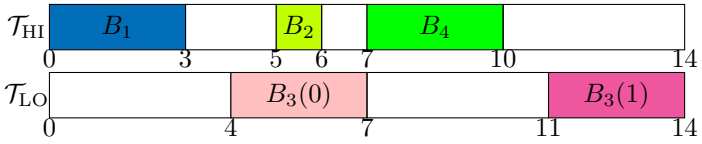

Figure 21: Tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$

the table $\mathcal{S}_{\mathrm{LO}}$ as shown in Fig. 22 from which the table $\mathcal{S}_{\mathrm{HI}}$ is constructed as shown in Fig. 23.

We follow all the Lemmas from Section 3.4 to prove Theorem 6.

Theorem 6: If a mixed-criticality synchronous program is schedulable by the OCBP-based algorithm, then it is also schedulable by our algorithm.

Proof. The OCBP algorithm generates priority orders for the synchronous programs. Then the OCBP-based algorithm finds tables $\mathrm{S}^{\mathrm{oc}}_{\mathrm{LO}}$ and $\mathrm{S}^{\mathrm{oc}}_{\mathrm{HI}}$ for the synchronous programs using this priority order. We need to show that if the OCBP-based algorithm constructs the tables $\mathrm{S}^{\mathrm{oc}}_{\mathrm{LO}}$ and $\mathrm{S}^{\mathrm{oc}}_{\mathrm{HI}}$ for an instance, then our algorithm will not encounter a situation where at time slot t the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ are non-empty, for any t.

We know that $C_i(\mathrm{LO})$ units of execution is allocated to each block B_i for constructing the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$. Each block in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ is allocated as close to its deadline as possible without violating the dependency constraints. That means no block can execute after its allocation time in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ without affecting the schedule

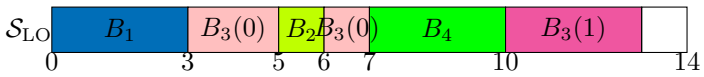

Figure 22: Table $\mathcal{S}_{\text{LO}}$

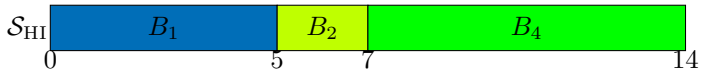

Figure 23: Table $\mathcal{S}_{\text{HI}}$

of any other block and still meet its deadline. Algorithm 8 never allocates a block in $\mathcal{S}_{\text{LO}}$ whose predecessors have not completed its $C(\text{LO})$ units of execution in $\mathcal{S}_{\text{LO}}$. Because, Algorithm 6 and 7 take care of the dependencies between the LO-criticality and HI-criticality blocks respectively and Algorithm 8 takes care the dependencies of a LO-criticality block on HI-criticality block. Algorithm 8 declares failure if it finds a non-empty instant at any time t in both the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. This means the two blocks which are found in the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ respectively cannot be scheduled with all other remaining blocks from this point, because all the blocks to the right have already been moved as far to the right as possible.

Let there be an OCBP priority order of the blocks of synchronous program and a table S_{LO} according to this priority order.

Let B_l and B_h be two blocks in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ respectively found at time t during the construction of $\mathcal{S}_{\text{LO}}$, by our algorithm which means all job segments in the interval $[0, t-1]$ from $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ have already been assigned in $\mathcal{S}_{\text{LO}}$. But, we know that OCBP has assigned priorities to these blocks B_l and B_h. Now there are two cases.

In the first case, assume B_l is assigned lower priority than B_h by OCBP. Let a_l be the arrival time of B_l and the starting and completion times of B_l in $\mathcal{T}_{\text{LO}}$ be t_l and $t_l{}'$ respectively. Since block B_l can be scheduled only on or after the arrival time a_l, we need to show that the block segment of B_l found at time t cannot be scheduled in the interval $[a_l, t-1]$ by the OCBP-based algorithm. We know that Algorithm 8 can allocate a block in table $\mathcal{S}_{\text{LO}}$ on or before its allocation in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ without violating the dependency constraints. But Algorithm 8 has not allocated the block segments found in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ at time t in the interval $[a_l, t-1]$ of the table $\mathcal{S}_{\text{LO}}$, and by Lemma 6, this is due to the presence of equal or higher priority block segments of the OCBP priority order in $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. We know that all the blocks in $\mathcal{T}_{\text{LO}}$ in the interval $[a_l, t]$ and the blocks in $\mathcal{T}_{\text{HI}}$ including block B_h in the interval $[a_l, t]$ are of priority greater or equal to that of B_l according to OCBP since, by moving block segments to the right starting from the OCBP schedule the blocks to the left of B_l are of priority greater or equal to that of B_l. This means the blocks in the interval $[a_l, t-1]$ of table $\mathcal{S}_{\text{LO}}$ are either equal or higher priority blocks than B_l according to OCBP. So both the algorithms, the OCBP-based one and ours, allocate higher or equal priority jobs (or, block segments according to Algorithm 8) before time t. Then it is clear that after the blocks of higher priority than B_l finish their $C(LO)$ units of execution, there will not be sufficient time for B_l to finish its $C_l(\text{LO})$ units of execution in the interval $[a_l, t_l{}']$ in the OCBP schedule. This is because at time t, the OCBP-based algorithm has already allocated all ready blocks with higher or equal priority than B_l (according to OCBP) in the interval $[a_l, t]$ with no vacant slot for further allocation of B_l's segment found at time slot t which is the case for Algorithm 8 as well. A similar statement holds for B_h. Therefore B_h and B_l cannot be simultaneously scheduled to meet their deadlines in the remaining time, according to the OCBP-based algorithm.

In the second case, assume B_h is assigned lower priority than B_l by OCBP. Let a_h be the arrival time of block B_h and let the starting and completion times of the LO-criticality execution of B_h be t_h and $t_h{}'$, respectively and the completion time of the HI-criticality execution be t_e. As in the previous case, all the blocks in $\mathcal{T}_{\text{HI}}$ in the interval $[a_h, t]$ and the blocks in $\mathcal{T}_{\text{LO}}$, including block B_l, in the interval $[a_h, t]$ are of priority (according to OCBP) greater than or equal to that of B_h. OCBP considers $C(\text{HI})$ units of execution time to assign a priority to a HI-criticality block. As seen above, it is clear that after the blocks of higher priority than B_h finish their $C(\text{LO})$ units of execution, there will not be sufficient time for B_h to finish its $C_h(\text{LO})$ units of execution in the interval $[a_h, t_h{}']$ according to

OCBP. We know that $C(\text{LO}) \leq C(\text{HI})$. If block B_h doesn't get sufficient time to execute its $C_h(\text{LO})$ units of execution in the interval $[a_h, t_h']$, then it will not get sufficient time to execute its $C_h(\text{HI})$ units of execution in the interval $[a_h, t_e]$ either.

From the above two cases, it is clear that OCBP cannot assign priorities to job B_l and B_h, which is a contradiction. This means if there exists an OCBP priority order for a synchronous program, then our algorithm will not encounter a situation where both the tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are non-empty at any time t for the same synchronous program.

Note that we need to consider only the LO-criticality scenarios since Lemma 3 implies that if $\mathcal{S}_{\text{LO}}$ can be constructed, then so can $\mathcal{S}_{\text{HI}}$. □

8 Results and Discussion

In this section we present the experiments conducted to evaluate our algorithm for the dual-criticality case. The experiments show the impact of utilization on our algorithm versus the OCBP-based and MCEDF algorithms. The comparison is done over a large number of instances with randomly generated parameters.
The job generation policy may have significant effect on the experiments. The details of the job generation policy are as follows.

- The utilization (u_i) of the jobs of instance I are generated according to the UUniFast algorithm [20].
- We use the exponential distribution proposed by Davis *et al* [21] to generate the deadline (d_i) of the jobs of instance I.
- The $C_i(\text{LO})$ units of execution time of the jobs are calculated as $u_i \times d_i$.
- The $C_i(\text{HI})$ units of execution time of the jobs are calculated as $C_i(\text{HI}) = \text{CF} \times C_i(\text{LO})$ where CF is the criticality factor which varies between 2 and 6 for each HI-criticality job j_i.
- Each instance I contains at least one HI-criticality job and one LO-criticality job.
- For each point on the X-axis, we have plotted the average result of 10 runs.

In the first experiment, we fix the utilization at LO-criticality level of each instance at 0.9 and let the deadline of the jobs vary between 1 and 2000. The number of jobs in each instance is set to 10. The graph in Fig. 24 shows the number of schedulable instances out of different numbers of randomly generated instances.

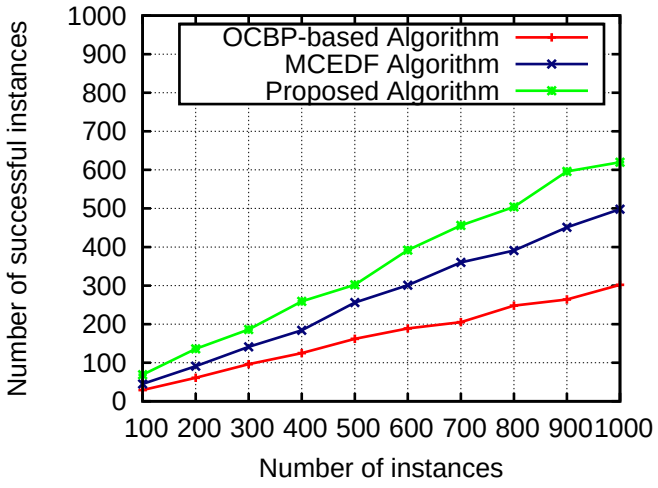

Figure 24: Comparison of number of MC-schedulable instances at an utilization of 0.9

From the graph in Fig. 24, it is clear that our algorithm schedules more instances successfully than both the OCBP-based algorithm and the MCEDF algorithm. As can be seen from Fig. 24, for an utilization of 0.9 about 620 instances out of 1000 instances are successfully scheduled by our algorithm which is two times more than the OCBP-based algorithm and 1.25 times more than the MCEDF algorithm. As the number of instances increases, the success ratio is more or less stable.

The next experiment checks the impact of the utilizations on the schedulable instances. Here the number of jobs in an instance is fixed at 20. The deadlines of the jobs in an instance range between 1 and 2000. The utilizations at LO-criticality level of the instances are varied between 0.1 and 0.9. The graph in Fig. 25 shows the number of schedulable instances from 1000 randomly generated instances.

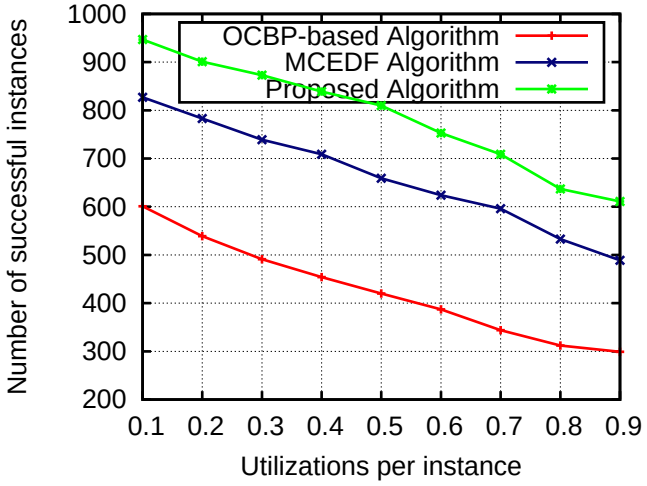

Figure 25: Comparison of number of MC-schedulable instances with different utilizations

From the graph, it is clear that our algorithm constructs more tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$ successfully than the OCBP-based scheduling algorithm. We can see that our algorithm also schedules more instances than MCEDF by a factor of 1.25. Typically our algorithm is successful in scheduling twice the number of instances than the OCBP-based algorithm. We can see that the number of schedulable instances decrease with the increase in the utilization.

We have done another experiment where the number of jobs in an instance varied between 5 and 100. For this experiment, we fix the utilization at LO-criticality level of each instance at 0.9 and let the deadline of the jobs vary between 1 and 2000. We plot the result from 1000 randomly generated instances in Fig. 26.

From the graph in Fig. 26 it is clear that our algorithm successfully schedules significantly more (by a factor of two) instances successfully than the OCBP-based algorithm and also schedules more instances than the MCEDF algorithm.

9 Conclusion

In this report, we proposed a new algorithm for the time-triggered scheduling of mixed-criticality systems. We proved that our algorithm can schedule a bigger set of instances than the previous algorithm based on OCBP. We also show that our algorithm schedules more instances than MCEDF. The experiments show the differences in number of schedulable instances between our algorithm and the OCBP-based algorithm and MCEDF. We have also extended the work for periodic and dependent jobs. Finally, we proved that our algorithm for dependent jobs can be used to schedule the blocks of a synchronous program and for which it schedules a bigger set of instances than the algorithm based on OCBP.

As part of future work we plan to extend this algorithm for multiprocessor systems and investigate resource sharing aspects.

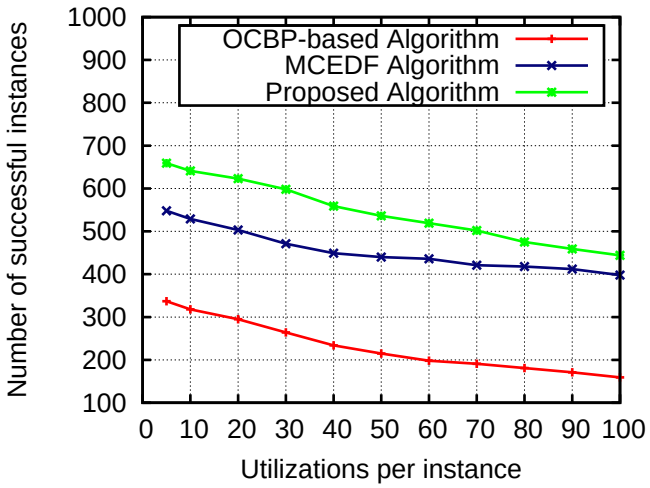

Figure 26: Comparison of number of MC-schedulable instances with different number of jobs per instance

References

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

[2] James Barhorst, Todd Belote, Pam Binns, Jon Hoffman, James Paunicka, Prakash Sarathy, John Scoredos, Peter Stanfill, Douglas Stuart, and Russel Urzi. A research agenda for mixed-criticality systems. In *Cyber-Physical Systems Week*, APR 2009.

[3] Haohan Li and Sanjoy Baruah. Load-based schedulability analysis of certifiable mixed-criticality systems. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 99–108. ACM, 2010.

[4] Alan Burns and Rob Davis. Mixed criticality systems: A review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[5] Alan Burns and Sanjoy Baruah. *Timing Faults and Mixed Criticality Systems*, volume 6875 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011.

[6] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE International Real-Time Systems Symposium, 2007. RTSS 2007.*, pages 239–243. IEEE, 2007.

[7] Sanjoy Baruah and Steve Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Euromicro Conference on Real-Time Systems, 2008. ECRTS'08.*, pages 147–155. IEEE, 2008.

[8] Thomas A Henzinger and Joseph Sifakis. The embedded systems design challenge. In *FM 2006: Formal Methods*, pages 1–15. Springer, 2006.

[9] Sanjoy Baruah and Gerhard Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *32nd IEEE Real-Time Systems Symposium (RTSS)*, pages 3–12. IEEE, 2011.

[10] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Mixed critical earliest deadline first. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 93–102, July 2013.

[11] Sanjoy Baruah. Implementing mixed-criticality synchronous reactive programs upon uniprocessor platforms. *Real-Time Systems*, 50(3):317–341, 2014.

[12] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. Lee & Seshia, 2011.

[13] Albert Benveniste and Gérard Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.

[14] Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[15] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Time-triggered mixed-critical scheduler on single and multi-processor platforms. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, pages 684–687, Aug 2015.

[16] Jens Theis, Gerhard Fohler, and Sanjoy Baruah. Schedule table generation for time-triggered mixed criticality systems. In *Proc. WMC, RTSS*, pages 79–84, 2013.

[17] Sanjoy Baruah. Semantics-preserving implementation of multirate mixed-criticality synchronous programs. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 11–19. ACM, 2012.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[19] Taeju Park and Soontae Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 253–262. ACM, 2011.

[20] Enrico Bini and Giorgio Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[21] Robert I. Davis, Attila Zabos, and Alan Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers,*, 57(9):1261–1276, 2008.