Mixed-Criticality Federated Scheduling for Relaxed-Deadline DAG Tasks

Fei Guan[1,5], Jinkyu Lee[2], Chun Jason Xue[3], Jen-Ming Wu[4] and Nan Guan[5]
[1]Northeast Forestry University, [2]Sungkyunkwan University, [3]Mohamed bin Zayed University of Artificial Intelligence
[4]Hon Hai Research Institute, [5]City University of Hong Kong

2024 IEEE Real-Time Systems Symposium (RTSS) | 979-8-3315-4026-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/RTSS62706.2024.00038

***Abstract*—A mixed-criticality (MC) system is a computational platform shared by tasks with two or more safety-critical levels. An important research topic related to MC systems is designing scheduling algorithms that can satisfy the computation requirements of tasks with different criticality levels. Numerous studies have focused on this topic, but only a few have considered parallel tasks. To address the research gap, we propose a dual-criticality scheduling algorithm based on federated scheduling for parallel tasks with Directed Acyclic Graph (DAG) structures. We particularly focus on the task set in which each task has a deadline longer than its release period. To the best of our knowledge, our work is the first that does not assume the constrained- or implicit-deadline in the MC DAG task model. In addition to simulation experiments, we demonstrate that our algorithm has a capacity augmentation bound of 4, providing a quantitative worst-case performance guarantee for our algorithm.**

I. Introduction

Standards in the automotive or aviation industry such as DO254 and ISO 26262 give definitions for critical levels and associate each level with a distinct level of assurance against failure. Because an increasing number of complex computer systems still suffer from resource constraint situations, enabling functions with two or more criticality levels to share a common hardware platform is sometimes unavoidable. Such a system is usually called a mixed-criticality (MC) system. When designing MC systems, task scheduling is crucial in optimizing computing resources and ensuring the reliability of high-criticality tasks. A thorough review of existing approaches to this problem has been provided by Burns and Davis [1]. It can be seen that most of the studies have been focused on the scheduling of non-parallel tasks. However, as the application scenarios of current MC systems get more complex, the scheduling schemes towards parallel tasks become commonly needed.

A parallel task is composed of multiple sub-tasks that have data dependencies or resource constraints, resulting in a specific internal parallel structure. Generally, the internal structures of parallel tasks can be modeled with Directed Acyclic Graphs (DAGs). It takes additional efforts to form an efficient scheduling algorithm for DAG tasks, as the subtasks and their precedence constraints introduce many uncertainties in the scheduling and need to be analyzed very carefully. For example, Zhao et al. [2] discussed how DAG structures introduce uncertainty to inter-task interference. This also contributes to the often observed high degree of pessimism in the response time analysis of DAG tasks in global scheduling. A few papers have attended to the scheduling of MC DAG tasks [3]–[10], but all of these papers have focused on DAG tasks with implicit- or constrained-deadlines, where a task's deadline is not allowed to be longer than its release period.

Relaxed-deadline. In this paper, we focus on MC DAG tasks with deadlines longer than their periods, referred to as *relaxed-deadline* DAG tasks. Unlike implicit- or constrained-deadline DAG tasks, which can have at most one unfinished job, relaxed-deadline DAG tasks may have multiple simultaneous jobs, complicating the scheduling. Burns and Davis [11] have discussed the practicality of such tasks in real systems, using buffer reading as an example. Another example can be found in the autonomous vehicle industry. Consider an autonomous vehicle's software system with functions like obstacle avoidance (high-criticality), electrical power system control (high-criticality), and path planning (low-criticality), each modeled as a DAG. These tasks involve complex computations and have long deadlines (e.g., the tolerable deadline could be $160ms$ when the vehicle runs at a speed of $20km/h$ [12]), but shorter inter-release times to ensure frequent sensor data sampling (e.g., the sampling frequency of the camera is typically $30Hz$, i.e., period is $30ms$). Besides the practicality of the relaxed-deadline tasks, solving the scheduling problem of such tasks will also be one step closer to solving the scheduling problem of arbitrary-deadline tasks.

Federated scheduling. *Federated scheduling* algorithms are a class of algorithms that can efficiently schedule DAG tasks by assigning several processors exclusively to each task. Guan et al. [13] introduced a federated scheduling algorithm to schedule relaxed-deadline DAG tasks considering the case that there is only one criticality level. Given the situation that multiple jobs generated by a relaxed-deadline task could run concurrently, their algorithm assigns dedicated processors to each job. Comparing with other scheduling algorithms that share processors among all the tasks [14], [15], or limit the sharing between jobs released by the same task [16], [17], their algorithm has shown the advantage in increasing the analytical schedulability. Moreover, because the method presented by Guan et al. [13] avoids interference between jobs, which reduces the difficulty in schedulability analysis, extending it to support the MC scenario is more convenient than adopting other approaches. With the above insight, we

Corresponding author: Nan Guan ⟨nanguan@cityu.edu.hk⟩

2576-3172/24/$31.00 ©2024 IEEE
DOI 10.1109/RTSS62706.2024.00038

develop a federated scheduling algorithm to schedule relaxed-deadline DAG tasks in dual-criticality systems, reusing several results provided by Guan et al. [13].

The contributions of this paper are as follows: 1) We propose a federated scheduling algorithm for scheduling sporadic relaxed-deadline DAG task sets in a dual-criticality system; 2) We provide a quantitative worst-case performance guarantee for the proposed algorithm in terms of the capacity augmentation bounds (whose definition has been given by Li et al. [18] and will be recapitulated in Section V); and 3) We conduct experiments with randomly generated task sets evaluating our algorithm in terms of the acceptance ratio.

The rest of this paper is organized as follows. Section II presents the related literature. Section III describes the task and system model. Section IV presents the algorithm for scheduling high-utilization (utilization > 1) tasks. Section V proves that our proposed algorithm has a capacity augmentation bound of 4 when scheduling high-utilization tasks. Section VI extends our algorithm to support both high- and low-utilization tasks. Section VII presents the results of experiments in terms of acceptance ratios. Section VIII presents some concluding remarks and discusses possible directions for future research.

II. Related Work

A few papers have attended to the scheduling problem of parallel MC tasks [3]–[10], applying a decomposition based algorithm [3], global scheduling algorithms [4], [5] or federated scheduling algorithms [6]–[10] in the designs.

Liu et al. [3] proposed to decompose every parallel task into multiple sequential tasks, each with its own deadline and release period. Then the sequential tasks can be partitioned to respective processors, and scheduled with an uniprocessor MC scheduling algorithm such as OCBP [19]. Their algorithm only supports the synchronous task model, where each parallel task consists of many computation segments, and each segment contains one or more parallel subtasks that synchronize at the end of the segment. Medina et al. [4], [5] proposed to generate offline scheduling tables for each criticality mode by statically applying an existing global scheduling algorithm. Their approach has to know the exact DAG structures of all the tasks before runtime, and the release interval of every task has to be strictly equal. Li et al. [6], [7] proposed a federated scheduling algorithm that separates high- and low-criticality tasks by permitting each of them exclusive access to a certain number of processors. During runtime monitoring, high-criticality tasks will receive additional processors while low-criticality tasks will be discarded upon an overrun of high-criticality tasks. Also within the scheme of federated scheduling, Pathan [8] increased processor utilization by considering the impact of low-utilization tasks in processor allocation; Agrawal and Baruah [9] maximized the number of idle processors when there is no overrun of high-criticality tasks observed; and Yang et al. [10] lowered the resource waste by allowing some amount of processor sharing between tasks.

The aforementioned studies primarily address implicit- or constrained-deadline DAG tasks, where deadlines do not exceed release periods. Studies on scheduling of relaxed-deadline DAG tasks [13]–[17], [20]–[24] have been limited to task sets where all tasks share the same criticality level. To the best of our knowledge, our paper is the first study in scheduling relaxed-deadline DAG tasks within the context of MC systems.

III. Task and System Model

We consider a task set τ with N independent sporadic parallel tasks $\tau = \{\tau_1, \tau_2, ..., \tau_N\}$ with two criticality levels scheduled on a platform with M ($M \geq 2$) homogeneous processors where each processor has a speed equal to one. Each task τ_i is represented as a DAG, denoted by $G_i = (E_i, V_i)$. V_i is a set of vertices, and each vertex $v \in V_i$ models the workload of some sequential instruction blocks. E_i is a set of directed edges. An edge $(v, u) \in E_i$ represents the dependency between two vertices, where vertex v is a predecessor of vertex u and vertex u is a successor of vertex v. A vertex without predecessors is called a *source*, and a vertex without successors is called a *sink*. We assume that each task has exactly one source and one sink. This assumption will not weaken the generalization of this model because any DAG task with multiple sources or sinks can be easily converted into a single-source/sink DAG task by adding a dummy source/sink with zero Worst-Case Execution Time (WCET). A *complete path* of a task is a sequence of vertices that begins with the source vertex and contains, in succession, a successor until it reaches the sink vertex. The *path length* is the aggregate WCET of its vertices. The *longest path* of τ_i is the complete path with the longest length.

Each task $\tau_i \in \tau$ is characterized by the tuple $\{\chi_i, C_i^L, C_i^H, L_i^L, L_i^H, T_i, D_i\}$. $\chi_i \in \{LO, HI\}$ is the criticality level of τ_i, representing τ_i is a low-criticality (LO) task or a high-criticality (HI) task. C_i^L and C_i^H are two estimates for the total WCET of all the vertices of τ_i. Similarly, L_i^L and L_i^H are two estimates for the length of the longest path. For each HI task, consider C_i^H and L_i^H to be more conservative estimates than C_i^L and L_i^L, satisfying the conditions $0 < C_i^L \leq C_i^H$ and $0 < L_i^L \leq L_i^H$. For each LO task, C_i^L and L_i^L apply when the task executes ($C_i^L > 0$, $L_i^L > 0$). C_i^H and L_i^H apply when the task is discarded and not consuming resources ($C_i^L = 0$, $L_i^L = 0$). D_i and T_i are the relative deadline and the minimum inter-arrival time of τ_i, respectively. These two values indicate that τ_i will repeatedly release a job that inherits all the characters of τ_i after a time interval that is no less than T_i, and each job must finish execution after time D_i. We focus on relaxed-deadline tasks, where $D_i > T_i$ holds for every $\tau_i \in \tau$. Note that while we assume all tasks in the task set have parallel structures, our design does not require prior knowledge of the specific details of any task's structure.

Some additional symbols will be also used throughout the remainder of this paper. We use τ^{LO} and τ^{HI} to denote two subsets of τ, grouping all the LO tasks and HI tasks, respectively. It follows that $\tau^{LO} \cup \tau^{HI} = \tau$. U_i^L and U_i^H denote the utilizations of task τ_i, and are equal to C_i^L/T_i and C_i^H/T_i, respectively. A task is a *high-utilization* task if either U_i^L or U_i^H is larger than 1; the task is a *low-utilization* task,

TABLE I: Summary of Major Notations

Notation	Description
T_i	Minimum inter-arrival time of τ_i
D_i	Relative deadline of τ_i
L_i^L, L_i^H	Two estimates for the length of τ_i's longest path
C_i^L, C_i^H	Two estimates for the total WCET of τ_i
$D_i^{'}$	Virtual deadline of τ_i
M_i^L	No. of processors assigned to a job of τ_i in the typical state
M_i^{H1}	No. of processors assigned to a carry-over job of τ_i
M_i^{H2}	No. of processors assigned to a non-carry-over job of τ_i
S_i^L	No. of processors assigned to τ_i in the typical state
S_i^H	No. of processors assigned to τ_i in the critical state
R_i^{H1}	Worst-case response time of any carry-over job released by τ_i
t_s	System mode transition time
t_r, t_f	Release and finish times of a job J
X, Y	Key path execution and non-execution durations of J within $[t_r, \min\{t_s, t_f\}]$
$X^{'}$, $Y^{'}$	Key path execution and non-execution durations of J within $[\max\{t_s, t_r\}, t_f]$
Ψ_i	Pairs of $\{S_i^L, S_i^H\}$ returned by Algorithm 1
$\Psi_{i,j}$	j_{th} item in Ψ_i
Γ	$\{\Psi_i \mid i \in \{1, ..., \lvert\tau_{Hu}^{HI}\rvert\}\}$
$Z_i(d)$	Minimum cost sum to the multiple-choice knapsack problem defined on the first i sets in Γ and with restricted capacity d

otherwise. Because any low-utilization task can be easily converted to a sequential implicit-deadline task and scheduled by any existing MC scheduler designed for such a task model, this paper focuses on the scheduling of high-utilization tasks. We use τ_{Hu}^{LO}, τ_{Hu}^{HI}, τ_{Lu}^{LO} and τ_{Lu}^{HI} to denote four disjunct subsets of τ_i. Specifically, $\tau_{Hu}^{LO} = \{\tau_i \in \tau \mid \chi_i = LO \wedge U_i^L > 1\}$, $\tau_{Hu}^{HI} = \{\tau_i \in \tau \mid \chi_i = HI \wedge U_i^H > 1\}$, $\tau_{Lu}^{LO} = \{\tau_i \in \tau \mid \chi_i = LO \wedge U_i^L \leq 1\}$ and $\tau_{Lu}^{HI} = \{\tau_i \in \tau \mid \chi_i = HI \wedge U_i^H \leq 1\}$. It follows that $\tau_{Hu}^{LO} \cup \tau_{Lu}^{LO} = \tau^{LO}$ and $\tau_{Hu}^{HI} \cup \tau_{Lu}^{HI} = \tau^{HI}$. Table I lists the most important and frequently used notions.

Following the system model established by Li et al. [6], a dual-criticality system has two system modes: typical state and critical state. The system always starts in the typical state. Because the system model does not assume clairvoyance in knowing the actual execution time of a job, a runtime monitor is required to control the mode transition. We assign a *virtual deadline* $D_i^{'}$ (specified in Section IV), which satisfies $D_i^{'} \leq D_i$, to each HI task. Virtual deadline is used to permit enough slack for HI task to complete C_i^H before its real deadline in case of mode transition. It has been introduced in EDF-VD [25] and adopted in other federated scheduling algorithms [6], [7]. During runtime, the monitor will trigger a **mode transition** from typical to critical state if it detects a job released by a HI task τ_i has either one of the following behaviors: 1) It does not complete within its virtual deadline $D_i^{'}$; and 2) It consumes more than C_i^L execution without signaling finish. Note that the other federated scheduling algorithms [6], [7] only use the first event to trigger mode transition. We include the second event as an additional mode transition triggering event since it is commonly used in MC scheduling research for non-parallel tasks [1]. Upon a mode transition from typical to critical state, LO tasks will be discarded immediately. In the critical state, the work and longest path length of any job released by a HI task τ_i will never exceed C_i^H and L_i^H, respectively.

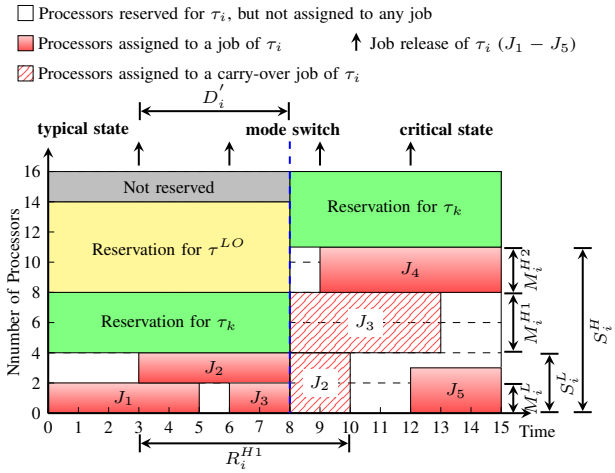


Fig. 1: An example of processor reservation in a dual-criticality system where $M = 16$ and $\tau^{HI} = \{\tau_i, \tau_k\}$. The processor allocation details for τ_i are provided. J_2 triggers the system transition, when it does not signal finish upon its virtual deadline $D_i^{'}$.

An MC task set τ is said to be correctly scheduled by a scheduler if: 1) All released jobs can meet their deadlines when the system stays in the typical state; 2) Any job released by a HI task can meet its deadline during stable system states (typical and critical states) and upon a mode transition from typical to critical state; and 3) Any job released by a LO task can meet its deadline when it has a deadline no later than the state transition time. For a task set with $\sum_{\tau_i \in \tau} U_i^H > M$ or $\sum_{\tau_i \in \tau} U_i^L > M$, or if there is any task in the task set such that $L_i^L \geq D_i$ or $L_i^H \geq D_i$, the task set is deemed unfeasible for any scheduling algorithm. Hence, in the following sections we will only consider task sets that satisfy the conditions: $\sum_{\tau_i \in \tau} U_i^L \leq M$, $\sum_{\tau_i \in \tau} U_i^H \leq M$, $\forall \tau_i \in \tau^{HI} : L_i^L \leq L_i^H \leq D_i$ and $\forall \tau_i \in \tau^{LO} : L_i^L \leq D_i$.

IV. Federated Scheduling for Dual-Criticality High-Utilization Tasks

In this section, we present an algorithm that schedules the high-utilization tasks. We will extend this algorithm to support both high- and low-utilization tasks in Section VI.

We classify jobs running in the critical state by their release time. A job released by a HI task before the system mode switch but not completed upon the mode switch is a *carry-over job*. Jobs released after the system mode switch are non-carry-over jobs. The scheduling of any high-utilization task τ_i follows these rules: 1) Each job has exclusive access to a number of processors during execution, where M_i^L is the number during typical state, M_i^{H1} is the number starts from the mode transition time if the job is a carry-over job, M_i^{H2} is the number if the job is a non-carry-over job; 2) Any job is scheduled by a work-conserving scheduler that never leaves a processor idle when there are vertices ready to run; and 3) Jobs release their processors immediately upon completion, making those processors available for the other upcoming jobs.

We denote the total number of processors a task τ_i requires in typical and critical states as S_i^L and S_i^H, respectively. Our idea is to set upper limits for S_i^L and S_i^H with each feasible tuple of $\{M_i^L, M_i^{H1}, M_i^{H2}\}$ so that we can measure the requirements of the whole task set in both system modes when the tuple is given for every task. Based on this knowledge, we can then choose $\{M_i^L, M_i^{H1}, M_i^{H2}\}$ for every task in a manner to keep the demands of the task set in both system modes lower than what the system can supply.

Figure 1 shows a high-level intuition for our processor reservation algorithm in a dual-criticality system. The behavior of a high-utilization HI task τ_i is highlighted. J_1-J_5 are jobs of τ_i. The system starts in the typical state and switches to the critical state at time 8 when J_2 remains unfinished upon its virtual deadline $D_i^{'}$. LO tasks are dropped immediately after time 8. J_2 and J_3 become carry-over jobs after the mode transition and J_4 and J_5 are non-carry-over jobs since they are released after the mode transition. The number of active jobs of τ_i at the same time is upper bounded by 2 in the typical state and 3 in the critical state. It follows that $S_i^L = 2M_i^L$. Since $M_i^{H1} > M_i^{H2}$, S_i^H reaches its upper bound $2M_i^{H1} + M_i^{H2}$ when the number of carry-over jobs is maximized as 2. R_i^{H1} is the Worst-Case Response Time (WCRT) of a carry-over job released by τ_i. R_i^{H1} and $D_i^{'}$ are crucial in bounding the maximum number of active jobs of τ_i, which will be explained formally in the following paragraphs.

Next, we consider a high-utilization task with $\chi_i = LO$ and $\chi_i = HI$ seperately. When $\chi_i = LO$, because such a task will be discarded upon mode transition, M_i^{H1}, M_i^{H2} and S_i^H are set to 0 and we only need to minimize the value of S_i^L. That is the same as what has to be done in a single-criticality system, and we simply let $\{M_i^L, S_i^L\} = \text{ReserveProcs}(C_i^L, L_i^L, D_i, T_i)$ (Algorithm 1 given by Guan et al. [13]). As for the case that $\chi_i = HI$, we want to answer the following questions:

Q1. How to get all tuples $\{M_i^L, M_i^{H1}, M_i^{H2}\}$ that guarantee the timing correctness in both typical and critical states for every job generated by a HI task τ_i? **(Lemma 4)**

Q2. Given any tuple $\{M_i^L, M_i^{H1}, M_i^{H2}\}$, can S_i^L and S_i^H of a HI task be upper bounded? **(Lemmas 6 and 7)**

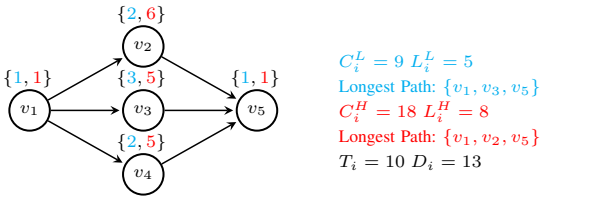
Q3. If S_i^L and S_i^H can be upper bounded with any feasible tuple of $\{M_i^L, M_i^{H1}, M_i^{H2}\}$, how do we choose the tuple for each HI task? **(Algorithm 2)**

Before addressing these questions, we define *key path* and introduce four additional notions (X, Y, $X^{'}$ and $Y^{'}$) to be used later in the proofs.

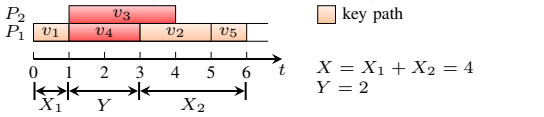
Definition 1. ***Key Path [26].*** *A key path λ_i of a job of τ_i is a complete path that satisfies the following condition. For any edge $(v, u) \in \lambda_i$, v has the latest completion time among all the predecessors of u.*

Note that the longest path and key path of a job may differ. By definition, a key path can only be known after a job's execution. The longest path, on the other hand, is static information independent of the execution sequence of the vertices. Figure 2 shows the key path of job J in different execution scenarios, where J is released by τ_i from Figure 2a.

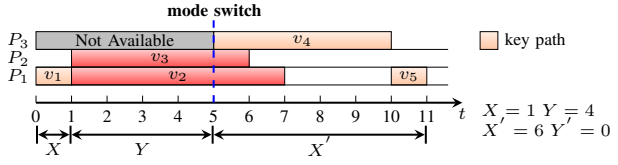
Figure 2b shows one possible execution sequence of J with total work C_i^L. The key path in the presented execution sequence is $\{v_1, v_2, v_5\}$ which is different from the longest path $\{v_1, v_3, v_5\}$. Similarly, in Figure 2c, τ_i's job J has total work equals C_i^H and triggers the mode transition when it consumes C_i^L without signaling finish. Here, the key path is $\{v_1, v_4, v_5\}$, while the longest path is $\{v_1, v_2, v_5\}$. Since a key path is also a complete path, its length will never exceed that of the longest path.



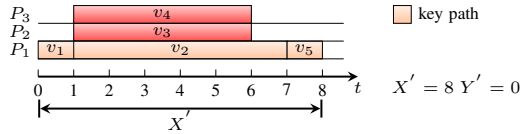
(a) A HI task τ_i. $\{c^L, c^H\}$ above each vertex represents two WCET estimates for that vertex. C_i^L is the sum of c^L and C_i^H is the sum of c^H.



(b) The system remains in the typical state during the entire execution of J.



(c) J is a carry-over job. It triggers the mode transition at time 5.



(d) J is a non-carry-over job.

Fig. 2: The scheduling of job J. $M_i^L = 2$ and $M_i^{H1} = M_i^{H2} = 3$. $P_1 - P_3$ are processors.

Next, we introduce four notions (X, Y, $X^{'}$ and $Y^{'}$) related to the key path. Consider a system switching to the critical state at time t_s. A job J is released at t_r and finishes at t_f. Define X as the key path execution duration within $[t_r, \min\{t_s, t_f\}]$, Y as the key path non-execution duration within $[t_r, \min\{t_s, t_f\}]$, $X^{'}$ as the key path execution duration within $[\max\{t_s, t_r\}, t_f]$, and $Y^{'}$ as the key path non-execution duration within $[\max\{t_s, t_r\}, t_f]$. We demonstrate X, Y, $X^{'}$ and $Y^{'}$ in the job execution examples in Figure 2. When $t_f < t_s$, i.e., the system remains in the typical state throughout J's execution, $X^{'} = Y^{'} = 0$ and the response time is $X + Y$ (Figure 2b). When $t_r \leq t_s \leq t_f$, i.e., J is a carry-over job, the response time is $X + Y + X^{'} + Y^{'}$ (Figure 2c). When $t_r > t_s$, i.e., J is a non-carry-over job released after the mode switch, $X = Y = 0$ and the response time is $X^{'} + Y^{'}$ (Figure 2d).

We now address Q1, Q2 and Q3 in the presented order. To

address Q1, the first thing is to set the virtual deadline for each HI task with the help of Lemma 1.

Lemma 1. *During the typical state, if a job J has total work and longest path length not exceeding C_i^L and L_i^L, respectively, its WCRT is upper bounded by $\frac{C_i^L - L_i^L}{M_i^L} + L_i^L$.*

Proof. As stated in Lemma 1 by Guan et al. [13], all M_i^L processors are busy during Y, making the total work done in Y equal to $M_i^L Y$. Since at least one processor is busy at any time point in X, the total work done in X is at least X and the total work done in $X+Y$ is at least $X + M_i^L Y$. Since the total work completed by J cannot exceed C_i^L, we have $M_i^L Y + X \leq C_i^L$, which leads to $Y \leq \frac{C_i^L - X}{M_i^L}$. By definition, X represents the length of the key path and must satisfy $X \leq L_i^L$. Therefore, we conclude: $X + Y \leq \frac{C_i^L}{M_i^L} + (1 - \frac{1}{M_i^L})X \leq \frac{C_i^L}{M_i^L} + (1 - \frac{1}{M_i^L})L_i^L$. □

We now safely set the virtual deadline of a HI task τ_i as $D_i^{'} = \frac{C_i^L - L_i^L}{M_i^L} + L_i^L$. By Lemma 1, no job of τ_i will miss this virtual deadline when the system stays in the typical state. We will discuss the relationship between the virtual deadline and the real deadline later. Next, we upper bound the response time for a job running in the critical state by Lemmas 2 and 3.

Lemma 2. *Assume that J is a carry-over job released by τ_i, then the WCRT of J is upper bounded by R_i^{H1} where*

$$R_i^{H1} = \begin{cases} \frac{C_i^L}{M_i^L} + \frac{C_i^H - C_i^L - L_i^H}{M_i^{H1}} + L_i^H & M_i^{H1} > M_i^L \\ \frac{C_i^H - L_i^H}{M_i^{H1}} + L_i^H & M_i^{H1} \leq M_i^L \end{cases} \quad (1)$$

Proof. By the definition of virtual deadline, it must be $X + Y \leq D_i^{'} = \frac{C_i^L}{M_i^L} + (1 - \frac{1}{M_i^L})L_i^L$. Since $X \geq 0$, $Y \leq \frac{C_i^L}{M_i^L} + (1 - \frac{1}{M_i^L})L_i^L$. As stated in Lemma 1 by Guan et al. [13], all M_i^L processors are busy during Y, so the total work done during Y is $M_i^L Y$. Since at least one processor is busy at any time point in X, the total work done in X is at least X and the total work done in $X+Y$ is at least $X + M_i^L Y$. Furthermore, the total work completed by J cannot exceed C_i^L upon the mode transition; otherwise, the transition would occur earlier. Thus, we have $M_i^L Y + X \leq C_i^L \Rightarrow Y \leq \frac{C_i^L - X}{M_i^L} \leq \frac{C_i^L}{M_i^L}$. Combining these two inequalities, we have $Y \leq \frac{C_i^L}{M_i^L}$.

From the analysis above, we also know that the work left to be finished and the remaining length of the key path after the mode transition are upper bounded by $C_i^H - (M_i^L Y + X)$ and $L_i^H - X$, respectively. Similar to the condition before the mode transition, the total work done in $X^{'} + Y^{'}$ is at least $X^{'} + M_i^{H1} Y^{'}$. We have $X^{'} + M_i^{H1} Y^{'} \leq C_i^H - (M_i^L Y + X) \Rightarrow Y^{'} \leq \frac{C_i^H - (M_i^L Y + X) - X^{'}}{M_i^{H1}}$. Another condition that must hold is $X^{'} \leq L_i^H - X$. Therefore, we have $X^{'} + Y^{'} \leq \frac{C_i^H - (M_i^L Y + X)}{M_i^{H1}} + (1 - \frac{1}{M_i^{H1}})(L_i^H - X)$. The response time of J is $X + Y + X^{'} + Y^{'} \leq \frac{C_i^H - L_i^H}{M_i^{H1}} + L_i^H + (1 - \frac{M_i^L}{M_i^{H1}})Y$. When $M_i^{H1} > M_i^L$, $1 - \frac{M_i^L}{M_i^{H1}} > 0$. Since $Y \leq \frac{C_i^L}{M_i^L}$, the response time is upper bounded by $\frac{C_i^L}{M_i^L} + \frac{C_i^H - C_i^L - L_i^H}{M_i^{H1}} + L_i^H$. When $M_i^{H1} \leq M_i^L$, $1 - \frac{M_i^L}{M_i^{H1}} \leq 0$. Since $Y \geq 0$, the response time is upper bounded by $\frac{C_i^H - L_i^H}{M_i^{H1}} + L_i^H$. □

Lemma 3. *Assume that J is a non-carry-over job released by τ_i, then the WCRT of J is upper bounded by $\frac{C_i^H - L_i^H}{M_i^{H2}} + L_i^H$.*

Proof. Since a non-carry-over job is solely influenced by M_i^{H2}, the proof closely resembles that of Lemma 1. $X^{'}$ as the length of the key path satisfies $X^{'} \leq L_i^H$. The work completed in $X^{'} + Y^{'}$ is at least $M_i^{H2} Y^{'} + X^{'}$ and at most C_i^H. Thus, we have $M_i^{H2} Y^{'} + X^{'} \leq C_i^H \Rightarrow Y^{'} \leq \frac{C_i^H - X^{'}}{M_i^{H2}}$. $X^{'} + Y^{'} \leq \frac{C_i^H}{M_i^{H2}} + (1 - \frac{1}{M_i^{H2}})L_i^H$ follows. □

Lemmas 1, 2 and 3 bound the WCRT of any job by a function of M_i^L, M_i^{H1} or M_i^{H2}. We can obtain the feasible solutions for $\{M_i^L, M_i^{H1}, M_i^{H2}\}$ by making each bound below D_i. However, the solution space would be three-dimensional, making it difficult to design the optimization algorithm. We want to reduce the number of variables at this stage to facilitate the optimization process that we will describe later. One way to achieve this is to express M_i^{H2} as a function of M_i^{H1}. Our intuition for deciding the value of M_i^{H2} is minimizing the number of active jobs during the critical state and avoiding high values for M_i^{H2}. According to Lemmas 2 and 3, the WCRT of any job in the critical state is $\max\{R_i^{H1}, \frac{C_i^H - L_i^H}{M_i^{H2}} + L_i^H\}$. Given a value of M_i^{H1}, setting M_i^{H2} such that $\frac{C_i^H - L_i^H}{M_i^{H2}} + L_i^H \leq R_i^{H1}$ limits the number of active jobs to a maximum of $\frac{R_i^{H1}}{T_i}$, where M_i^{H2} is minimized when the equal sign is taken. However, finding an integer value for M_i^{H2} to satisfy the equality might not always be possible for a given M_i^{H1}. To guarantee that the number of active jobs and M_i^{H2} remain upper bounded by $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil$ (as will be shown in the proof of Lemma 7) and $\left\lceil \frac{C_i^H - L_i^H}{R_i^{H1} - L_i^H} \right\rceil$ (as will be shown in the proof of Lemma 5), even when the aforementioned equal sign does not hold, we assign the value to M_i^{H2} as follows.

$$M_i^{H2} = \begin{cases} \left\lceil \frac{C_i^H - L_i^H}{\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\} - L_i^H} \right\rceil & M_i^{H1} > M_i^L \\ M_i^{H1} & M_i^{H1} \leq M_i^L \end{cases} \quad (2)$$

With Equation (2) the number of variables is reduced to two ($\{M_i^L, M_i^{H1}\}$). We are ready to answer Q1 with Lemma 4.

Lemma 4. *If M_i^L and M_i^{H1} satisfy Inequality Group* (3) *or* (4)*, a HI criticality task τ_i can complete within its deadline in both typical and critical states.*

$$\begin{cases} M_i^{H1} > M_i^L \\ \frac{C_i^L - L_i^L}{M_i^L} + L_i^L \leq D_i \\ \frac{C_i^L}{M_i^L} + \frac{C_i^H - C_i^L - L_i^H}{M_i^{H1}} + L_i^H \leq D_i \end{cases} \quad (3)$$

$$\begin{cases} M_i^{H1} \leq M_i^L \\ \frac{C_i^L - L_i^L}{M_i^L} + L_i^L \leq D_i \\ \frac{C_i^H - L_i^H}{M_i^{H1}} + L_i^H \leq D_i \end{cases} \quad (4)$$

Proof. In the typical state, by Lemma 1, if M_i^L satisfies Inequality Group (3) or (4), any HI task τ_i can finish its C_i^L time execution within its deadline. In the critical state, by Lemma 2, if M_i^{H1} and M_i^L satisfy Inequality Group (3) or (4), any carry-over job will complete its C_i^H time execution within its deadline. A non-carry-over job is assigned M_i^{H2} processors determined by Equation (2), which can guarantee a response time no more than $\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\} \leq D_i$ when $M_i^{H1} > M_i^L$ and D_i when $M_i^{H1} \leq M_i^L$, respectively. Concluding from both system modes, Lemma 4 is proved. □

With Lemma 4, we can find the solution region of $\{M_i^L, M_i^{H1}\}$ for any high-utilization HI task. Figure 3 provides three examples representing the solution regions of $\{M_i^L, M_i^{H1}\}$ when $C_i^H - C_i^L - L_i^H$ is < 0, $= 0$ and > 0.

For answering Q2, the dominance relationship between M_i^{H1} and M_i^{H2} is crucial and presented in Lemma 5.

Lemma 5. *Under the constraint of Lemma 4, $M_i^{H1} \geq M_i^{H2}$ holds for all the HI tasks.*

Proof. **Case 1:** $M_i^{H1} \leq M_i^L$. According to Equation (2), $M_i^{H1} = M_i^{H2}$. **Case 2:** $M_i^{H1} > M_i^L$. We consider two subcases. **Case 2a:** $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i < D_i$.

$$M_i^{H2} = \left\lceil \frac{C_i^H - L_i^H}{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i - L_i^H} \right\rceil \leq \left\lceil \frac{C_i^H - L_i^H}{R_i^{H1} - L_i^H} \right\rceil$$

$$\leq \left\lceil \frac{C_i^H - L_i^H}{\frac{C_i^L}{M_i^L} + \frac{C_i^H - C_i^L - L_i^H}{M_i^{H1}}} \right\rceil \quad \text{(by Equation (1))}$$

$$\leq M_i^{H1} \quad \left(\because M_i^L < M_i^{H1}\right)$$

Case 2b: $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i \geq D_i$. By Lemma 4, $R_i^{H1} \leq D_i$. By Equation (2), $M_i^{H2} = \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil \leq \left\lceil \frac{C_i^H - L_i^H}{R_i^{H1} - L_i^H} \right\rceil$. The rest of the proof is the same as in Case 2a.

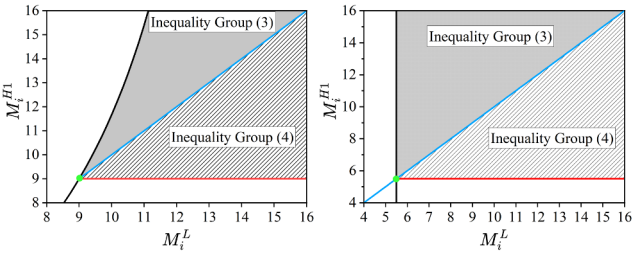
Concluding from both cases, Lemma 5 is proved. □

Now Q2 can be addressed easily by Lemmas 6 and 7. The proof for Lemma 6 is omitted because S_i^L is not related to any configuration made for the critical state and is determined by the response time (as indicated in Lemma 1) and M_i^L, which mirrors the proof of Lemma 6 by Guan et al. [13].

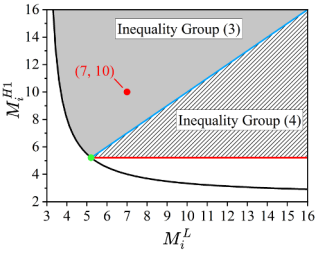
Lemma 6. *For a HI task τ_i, if M_i^L satisfies Lemma 4, S_i^L is upper bounded by $M_i^L \left\lceil \frac{\frac{C_i^L - L_i^L}{M_i^L} + L_i^L}{T_i} \right\rceil$.*

Lemma 7. *For a HI task τ_i, if M_i^L and M_i^{H1} satisfy Lemma 4 and M_i^{H2} is obtained by Equation (2), S_i^H is upper bounded by $M_i^{H1} \left\lceil \frac{D_i'}{T_i} \right\rceil + M_i^{H2} \left(\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil - \left\lceil \frac{D_i'}{T_i} \right\rceil \right)$.*

Proof. By Lemma 2 and Equation (2), the WCRT of a job running in the critical state is upper bounded by either R_i^{H1} or $\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\}$. Because $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i \geq R_i^{H1}$ and Lemma 4 guarantees $D_i \geq R_i^{H1}$, $\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\} \geq R_i^{H1}$. Hence, any job of HI task τ_i can complete within $\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\}$ in the critical state.



(a) $C_i^H = 20$, $C_i^L = 16$, $L_i^H = 11$, $L_i^L = 7$, $T_i = 5$ and $D_i = 12$. (b) $C_i^H = 30$, $C_i^L = 22$, $L_i^H = 8$, $L_i^L = 7$, $T_i = 5$ and $D_i = 12$.



(c) $C_i^H = 1500$, $C_i^L = 800$, $L_i^H = 15$, $L_i^L = 10$, $T_i = 200$ and $D_i = 300$.

Fig. 3: Solution space of $\{M_i^L, M_i^{H1}\}$, when $C_i^H - C_i^L - L_i^H$ is < 0 (a), $= 0$ (b) or > 0 (c). Black line: $\frac{C_i^L}{M_i^L} + \frac{C_i^H - C_i^L - L_i^H}{M_i^{H1}} + L_i^H = D_i$. Red line: $\frac{C_i^H - L_i^H}{M_i^{H1}} + L_i^H = D_i$. Blue line: $M_i^{H1} = M_i^L$. Green data point: $M_i^L = M_i^{H1} = \frac{C_i^H - L_i^H}{D_i - L_i^H}$.

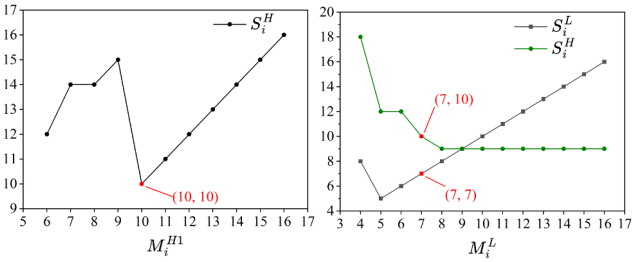
Let t be an arbitrary time during the critical state. When $M_i^{H1} > M_i^L$, any job of τ_i released before or at time $t - \min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\}$ has already been completed at t. Jobs that are released in the time interval $(t - \min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\}, t]$ can still be executing at t, and there are at most $\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil, \left\lceil \frac{D_i}{T_i} \right\rceil\}$ of them. By Lemma 4, $R_i^{H1} \leq D_i$. So we have $\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil, \left\lceil \frac{D_i}{T_i} \right\rceil\} = \left\lceil \frac{R_i^{H1}}{T_i} \right\rceil$, and the total number of currently running jobs in the critical state is upper bounded by $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil$.

According to Lemma 5, $M_i^{H1} \geq M_i^{H2}$. Therefore, the total number of processors required by τ_i will reach the peak value when the number of carry-over jobs is maximized. A carry-over job is released no earlier than $t - D_i'$; otherwise, it would have completed or triggered a system mode transition before the current mode transition time. The maximum number of carry-over jobs running at t is no more than $\left\lceil \frac{D_i'}{T_i} \right\rceil$. Thus, the total number of processors required by τ_i in the critical state is maximized at $M_i^{H1} \left\lceil \frac{D_i'}{T_i} \right\rceil + M_i^{H2} \left(\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil - \left\lceil \frac{D_i'}{T_i} \right\rceil \right)$. □

With the answer to Q2 in mind, we can now begin to delve into Q3. Assume the number of processors remaining for high-utilization tasks in the typical state is M^{Lr} and the critical state is M^{Hr}. When only high-utilization tasks exist in the task set, $M^{Lr} = M^{Hr} = M$. We want to decide if a $\{M_i^L, M_i^{H1}\}$ for every task exists, so that $\sum_{\forall \tau_i \in \tau} S_i^L \leq M^{Lr}$ and $\sum_{\forall \tau_i \in \tau_{Hu}^{HI}} S_i^H \leq M^{Hr}$. Recall that for a LO task $S_i^H = 0$, we minimize the value of S_i^L by ReserveProcs (Algorithm

Algorithm 1 ReservationPairs(τ_i, M^{Lr}, M^{Hr})

Input: τ_i, M^{Lr}, M^{Hr}
Output: Ψ_i
1: $\Pi^L \leftarrow$ all M_i^L that satisfy Lemma 4, $M_i^L \leq M^{Lr}$ and $M_i^L \in \mathbb{N}$.
2: $\Psi_i \leftarrow \varnothing$, $j \leftarrow 1$
3: **for** each $M_i^L \in \Pi^L$ **do**
4: $\quad \Pi_j^{H1} \leftarrow$ with the given M_i^L, all M_i^{H1} that satisfy Lemma 4, $M_i^{H1} \leq M^{Hr}$ and $M_i^{H1} \in \mathbb{N}$.
5: $\quad S_i^L \leftarrow M_i^L \left\lceil \frac{D_i^{'}}{T_i} \right\rceil$
6: $\quad f(M_i^{H1}) \leftarrow M_i^{H1} \left\lceil \frac{D_i^{'}}{T_i} \right\rceil + M_i^{H2} \left(\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil - \left\lceil \frac{D_i^{'}}{T_i} \right\rceil \right)$
7: $\quad S_i^H \leftarrow \min_{M_i^{H1} \in \Pi_j^{H1}} f(M_i^{H1})$
8: $\quad \Psi_{i,j} \leftarrow \{S_i^L, S_i^H\}$, $\Psi_i \leftarrow \Psi_i \cup \{\Psi_{i,j}\}$, $j \leftarrow j+1$
9: **end for**
10: **return** Ψ_i



(a) When $M_i^L = 7$, how S_i^H varies corresponding to all M_i^{H1} that satisfy the constraints of Lemma 4, $M_i^{H1} \leq M^{Hr}$ and $M_i^{H1} \in \mathbb{N}$.

(b) The final result of $\Psi_i = \{\{8, 18\}, \{5, 12\}, \{6, 12\}, \{7, 10\}, \{8, 9\}, \{9, 9\}, \{10, 9\}, \{11, 9\}, \{12, 9\}, \{13, 9\}, \{14, 9\}, \{15, 9\}, \{16, 9\}\}$.

Fig. 4: How Algorithm 1 generates Ψ_i for the high-utilization HI task in Figure 3c, when $M^{Lr} = M^{Hr} = 16$.

1 given by Guan et al. [13]). The problem is simplified to finding the values of $\{M_i^L, M_i^{H1}\}$ for every HI task. Given the constraints of Lemma 4, $M_i^L \leq M^{Lr}$, $M_i^{H1} \leq M^{Hr}$ and $M_i^L, M_i^{H1} \in \mathbb{N}$, the number of choices for $\{M_i^L, M_i^{H1}\}$ is finite, leading to a finite number of feasible pairs of $\{S_i^L, S_i^H\}$.

We use a set $\Psi_i = \{\Psi_{i,1}, ..., \Psi_{i,j}, ..., \Psi_{i,|\Psi_i|}\}$ to denote the feasible pairs of processor reservations for a HI task, i.e. pairs of $\{S_i^L, S_i^H\}$, where $\Psi_{i,j}$ is the j_{th} feasible pair in Ψ_i and $|\Psi_i|$ is the total number of feasible pairs in Ψ_i. Each Ψ_i corresponds to a HI task. For the convenience of explanation, when we refer to an overall processor reservation of a HI task in the typical state (or critical state), we also use $\Psi_{i,j}^L$ (or $\Psi_{i,j}^H$) when its order in Ψ_i need to be emphasized. Without loss of generality, we assume that τ_{Hu}^{HI} are the tasks in τ with the first $|\tau_{Hu}^{HI}|$ index numbers, i.e. $\tau_{Hu}^{HI} = \{\tau_1, \tau_2, ..., \tau_{|\tau_{Hu}^{HI}|}\}$. The set containing all the Ψ_i can be represented as $\{\Psi_i | i \in \{1, ..., |\tau_{Hu}^{HI}|\}\}$. We use Algorithm 1 to populate Ψ_i. For each HI task, Algorithm 1 first collects the set of M_i^L as Π^L (line 1). Next, it gathers the corresponding set of M_i^{H1} for each M_i^L as Π_j^{H1}, determines the minimum value of S_i^H given this specific M_i^L, and adds a new member to Ψ_i (lines 3-9).

Figure 4 illustrates the population of Ψ_i for the example task in Figure 3c. Based on the solution region of $\{M_i^L, M_i^{H1}\}$ (Figure 3c) and $M_i^L \leq M^{Lr}$, we have $\Pi^L = \{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$. For $M_i^L = 7$,

Algorithm 2 FedMixedCriticality(τ_{Hu}^{HI}, τ_{Hu}^{LO}, M^{Lr}, M^{Hr})

Input: τ_{Hu}^{HI}, τ_{Hu}^{LO}, M^{Lr}, M^{Hr}
Output: schedulability analysis result
1: **for** each $\tau_i \in \tau_{Hu}^{LO}$ **do**
2: $\quad \{M_i^L, S_i^L\} \leftarrow$ ReserveProcs(C_i^L, L_i^L, D_i, T_i) [13]
3: $\quad S_i^H \leftarrow 0$
4: **end for**
5: **for** each $\tau_i \in \tau_{Hu}^{HI}$ **do**
6: $\quad$ **if** $M^{Hr} < max\{\left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil, 1\}$ **then**
7: $\quad\quad$ **return** failure
8: $\quad$ **else**
9: $\quad\quad \Psi_i \leftarrow$ReservationPairs($\tau_i$, M^{Lr}, M^{Hr})
10: $\quad$ **end if**
11: **end for**
12: $\Gamma \leftarrow \{\Psi_i | i \in \{1, ..., |\tau_{Hu}^{HI}|\}\}$
13: **if** $\sum_{\forall \tau_i \in \tau_{Hu}^{LO}} S_i^L$+OptimizeReservation($\Gamma$, M^{Hr})$\leq M^{Lr}$ **then**
14: $\quad$ **return** success
15: **else**
16: $\quad$ **return** failure
17: **end if**

$\Pi_j^{H1} = \{6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$, considering the solution region of $\{M_i^L, M_i^{H1}\}$ and $M_i^H \leq M^{Hr}$. Algorithm 1 calculates the value of S_i^H for each M_i^{H1} in Π_j^{H1}, as shown in Figure 4a, and identifies the minimum S_i^H (which is 10 when $M_i^{H1} = 10$) to include in Ψ_i. The corresponding $\{M_i^L, M_i^{H1}\} = \{7, 10\}$ is marked in Figure 3c. Note that, as a result of Algorithm 1, not all pairs of $\{S_i^L, S_i^H\}$ are included in Ψ_i. For instance, in this example, $\{7, 11\}$, $\{7, 12\}$, $\{7, 13\}$, $\{7, 14\}$, $\{7, 15\}$ and $\{7, 16\}$ are left out because $\{7, 10\}$ has an equal typical state reservation and a lower critical state reservation. This ensures that we can achieve optimal results with a reduced Ψ_i in the subsequent process. The final Ψ_i after processing all M_i^L in Π^L is presented in Figure 4b, with the highlighted data representing the $\Psi_{i,j}$ obtained in Figure 4a. In this example $\Psi_{i,1}^H = 18 > M^{Hr}$, which is not useful for constructing feasible scheduling. However, we retain such a $\Psi_{i,j}$ as it does not affect the final result of our processor reservation algorithm. This will be further explained after introducing Algorithm 2.

We form the processer reservation problem as a multiple-choice knapsack problem in the following manner. There are $|\tau_{Hu}^{HI}|$ mutually disjoint sets of items to be packed into a knapsack of capacity M^{Hr}. Each item $\Psi_{i,j} \in \Psi_i$ has a cost $\Psi_{i,j}^L$ and a weight $\Psi_{i,j}^H$. The problem is to choose exactly one item from each set such that the cost sum is minimized without exceeding the capacity M^{Hr} in the corresponding weight sum. This problem can be solved in pseudopolynomial time through dynamic programming [27]. Let $Z_i(d)$ be a minimum cost sum to this problem defined on the first i sets and with restricted capacity d. Initially we set $Z_0(d) = 0$ for all $d = 0, ..., M^{Hr}$. To compute $Z_i(d)$ for $i = 1, ..., |\tau_{Hu}^{HI}|$, we use the recursion:

$$Z_i(d) = \min \begin{cases} Z_{i-1}(d - \Psi_{i,1}^H) + \Psi_{i,1}^L & d - \Psi_{i,1}^H \geq 0 \\ Z_{i-1}(d - \Psi_{i,2}^H) + \Psi_{i,2}^L & d - \Psi_{i,2}^H \geq 0 \\ ... & \\ Z_{i-1}(d - \Psi_{i,|\Psi_i|}^H) + \Psi_{i,|\Psi_i|}^L & d - \Psi_{i,|\Psi_i|}^H \geq 0 \end{cases} \quad (5)$$

If $d - \Psi_{i,j}^H < 0$ is true for all $j = 1, ..., |\Psi_i|$, the minimum

Algorithm 3 OptimizeReservation(Γ, M^{Hr})

Input: Γ, M^{Hr}
Output: $Z_{|\tau_{Hu}^{HI}|}(M^{Hr})$
1: $Z_0(d) \leftarrow 0$, $d \in \{0, 1, ..., M^{Hr}\}$
2: **for** each $i \in \{1, ..., |\tau_{Hu}^{HI}|\}$ **do**
3: obtain $Z_i(d)$ by Equation (5), $d \in \{0, 1, ..., M^{Hr}\}$
4: **end for**
5: **return** $Z_{|\tau_{Hu}^{HI}|}(M^{Hr})$

operator returns $+\infty$. The optimization result $Z_{|\tau_{Hu}^{HI}|}(M^{Hr})$ corresponds to the minimum number of processors required by τ_{Hu}^{HI} in the typical state without exceeding the capacity M^{Hr} in the critical state. By comparing $Z_{|\tau_{Hu}^{HI}|}(M^{Hr})$ with $M^{Lr} - \sum_{\forall \tau_i \in \tau_{Hu}^{LO}} S_i^L$, we can decide if the task set is schedulable.

Algorithm 2 shows the entire process of the schedulability analysis. Initially, it reserves processors for LO tasks using ReserveProcs [13] (lines 1-4). It then populates Ψ_i for each HI task using Algorithm 1 (lines 5-11). Finally, it solves the multiple-choice knapsack problem using Algorithm 3 (lines 12-17). Algorithm 2 has a time complexity of $O(NM^2)$. Algorithm 3 will discard any $\Psi_{i,j}$ with $\Psi_{i,j}^H > M^{Hr}$, rendering its presence in Ψ_i inconsequential. Similarly, any $\Psi_{i,j}$ with $\Psi_{i,j}^L > M^{Lr}$ will be ignored if it does not contribute to $Z_{|\tau_{Hu}^{HI}|}(M^{Hr})$ and a failure is inevitable if it does contribute, regardless of the presence or absence of this $\Psi_{i,j}$. In both cases, the final result remains unaffected.

V. The Capacity Augmentation Bound for Scheduling High-Utilization Tasks

The *capacity augmentation bound* is a quantitative metric that is widely used to analyze the worst-case performances of scheduling algorithms for real-time parallel tasks. It provides a linear-time schedulability test and serves as a good performance reference for the algorithm under evaluation. The definition of the capacity augmentation bound is as follows.

Definition 2. ***Capacity Augmentation Bound [23].*** *A scheduling algorithm S has a capacity augmentation bound ρ if it can schedule any task set that satisfies the following two conditions: the task set's total utilization is no greater than $\frac{M}{\rho}$, where M is the number of processors in the system; and the length of the longest path of each task is at most $\frac{1}{\rho}$ of its relative deadline.*

In the following, we provide a capacity augmentation bound for our algorithms in scheduling only high-utilization tasks. For simplicity, the term *high-utilization* is not always explicitly stated throughout the subsequent text within this section. Two *mathematical inequalities* will be used in the proofs: If $0 < c \leq \frac{a}{b}$ and $0 \leq x \leq y < bc$, then $\frac{a}{b} \leq \frac{a-x}{b-\frac{x}{c}} \leq \frac{a-y}{b-\frac{y}{c}}$; If $0 < \frac{a}{b} \leq c$ and $0 \leq x \leq y < bc$, then $\frac{a}{b} \geq \frac{a-x}{b-\frac{x}{c}} \geq \frac{a-y}{b-\frac{y}{c}}$.

LO **tasks:** As indicated in Algorithm 2, we reserve processors for every LO task by ReserveProcs [13]. The timing correctness of ReserveProcs has already been proved by Guan et al. [13]. It has also shown by Guan et al. [13] that ReserveProcs has a capacity augmentation bound of 3, which guarantees the total number of processors assigned to LO tasks will not exceed three times the total utilization of LO tasks, where each LO task τ_i has $L_i^L \leq \frac{1}{3}D_i$. Let's set this fact aside for now, and use it along with other facts of HI tasks to prove the final capacity augmentation bound later.

HI **tasks:** When considering HI tasks, we will start by considering a processor assignment strategy as shown in Table II and validate the timing correctness of each task with such an assignment in both system modes. Subsequently, we will prove that replacing the part related to HI tasks in Algorithm 2 with the setting in Table II can guarantee a capacity augmentation bound of 4. At last, we will prove the dominance of Algorithm 2 over Table II; this dominance relationship establishes a capacity augmentation bound of 4 for Algorithm 2.

In the above order, first we demonstrate the assignments in Table II can guarantee the timing correctness of HI tasks for any $\rho \geq \frac{3+\sqrt{5}}{2} \approx 2.62$ with Lemmas 8 and 9.

Lemma 8. *The processor assignments of Table II can guarantee the timing correctness of any Type II task in both critical state and typical state for any $\rho > 0$.*

Proof. We consider two possibilities for the value of M_i^L. **Case 1:** $\widehat{M_i^L} > \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$. According to [13], $\widehat{M_i^L} \geq 1$. The conditions $M_i^L = \widehat{M_i^L} \geq 1$, and $M_i^L \geq \max\{\left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil, 1\} = M_i^{H1}$ follow. **Case 2:** $\widehat{M_i^L} \leq \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$. $M_i^L = M_i^{H1} = \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$. In both cases, the processor assignment satisfies Inequality Group (4). According to Lemma 4, the task can complete its execution within the deadline in both system modes. □

Lemma 9. *The processor assignments of Table II can guarantee the timing correctness of any Type I task in both critical state and typical state for any $\rho \geq \frac{3+\sqrt{5}}{2} \approx 2.62$.*

Proof.

$$M_i^L \geq \left\lceil \frac{C_i^L}{(\frac{1}{\rho-1}+\frac{1}{\rho})D_i - L_i^H} \right\rceil > \left\lceil \frac{C_i^L - L_i^L}{(\frac{1}{\rho-1}+\frac{1}{\rho})D_i - L_i^L} \right\rceil > \left\lceil \frac{C_i^L - L_i^L}{D_i - L_i^L} \right\rceil \quad \left(\because \rho \geq \frac{3+\sqrt{5}}{2} \Rightarrow \frac{1}{\rho-1} + \frac{1}{\rho} \leq 1 \right)$$

The condition $\frac{C_i^L - L_i^L}{M_i^L} + L_i^L \leq D_i$ follows. For M_i^{H1}, there are two possible cases. **Case 1:** $M_i^L \geq \frac{C_i^H - L_i^H}{D_i - L_i^H}$. Because $M_i^L \geq \frac{C_i^L}{(\frac{1}{\rho-1}+\frac{1}{\rho})D_i - L_i^H}$ and $1 - \frac{1}{\rho-1} - \frac{1}{\rho} > 0$, $D_i - \frac{C_i^L}{M_i^L} - L_i^H \geq (1 - \frac{1}{\rho-1} - \frac{1}{\rho})D_i > 0$. Using the *mathematical inequalities* given in the previous text, we have $\left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil \geq \left\lceil \frac{C_i^H - L_i^H - C_i^L}{D_i - L_i^H - \frac{C_i^L}{M_i^L}} \right\rceil > 0$. Thus, $M_i^{H1} = \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$. M_i^L and M_i^{H1} satisfy Inequality Group (4). **Case 2:** $M_i^L < \frac{C_i^H - L_i^H}{D_i - L_i^H}$. Because $D_i - L_i^H - \frac{C_i^L}{M_i^L} > 0$, we have $0 < \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil < \left\lceil \frac{C_i^H - L_i^H - C_i^L}{D_i - \frac{C_i^L}{M_i^L} - L_i^H} \right\rceil$, and $M_i^{H1} = \left\lceil \frac{C_i^H - L_i^H - C_i^L}{D_i - L_i^H - \frac{C_i^L}{M_i^L}} \right\rceil$. M_i^L and M_i^{H1} satisfy Inequality Group (3). According to Lemma 4, the task can complete its execution within its deadline in both cases. □

TABLE II: A Processor Allocation Strategy for High-utilization HI Tasks

Task Type	Classification Criteria	M_i^L	M_i^{H1}
I	$C_i^H - C_i^L - L_i^H > 0$, $\frac{C_i^H}{D_i} > 1$, $1 < \frac{D_i}{T_i} \leq 2$ and $C_i^L \leq (T_i - L_i^L)\left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil + L_i^L$	$\max\{\left\lceil \frac{C_i^L}{(\frac{1}{\rho-1}+\frac{1}{\rho})D_i - L_i^H} \right\rceil, \lceil(\rho-1)U_i^L\rceil\}$	$\max\{\left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil, \left\lceil \frac{C_i^H - C_i^L - L_i^H}{D_i - \frac{C_i^L}{M_i^L} - L_i^H} \right\rceil\}$
II	Any HI task other than Task Type I	$\max\{\widehat{M_i^L}, \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil\}^a$	$\max\{\left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil, 1\}$

[a] $\widehat{M_i^L}$ is the first return value of ReserveProcs(C_i^L, L_i^L, D_i, T_i) [13]

Next, we provide the upper bounds of S_i^H and S_i^L for every HI task with Lemmas 10, 11, 12 and 13 where the processor mapping is compliance with the settings in Table II.

Lemma 10. *Under the configuration of Table II, if a Type II HI task τ_i satisfies the condition $\frac{1}{\rho}D_i \geq L_i^H \geq L_i^L$, $S_i^H \leq \rho U_i^H$ holds for any $\rho \geq 4$.*

Proof. We demonstrate Lemma 10 is correct by proving that $S_i^H \leq \rho U_i^H$ holds for any HI task with $\frac{1}{\rho}D_i \geq L_i^H \geq L_i^L$ if each its job is assigned the same number of processors as a Type II HI task in Table II. More specifically, we consider two complementary sets of HI tasks: $\frac{C_i^H}{D_i} > 1$ and $\frac{C_i^H}{D_i} \leq 1$. By Table II, $M_i^L \geq M_i^{H1}$. According to Equation (2), we have $M_i^{H2} = M_i^{H1} = \max\{\left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil, 1\}$. $S_i^H = M_i^{H1}\left\lceil \frac{D_i'}{T_i} \right\rceil + M_i^{H2}\left(\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil - \left\lceil \frac{D_i'}{T_i} \right\rceil\right) = M_i^{H1}\left\lceil \frac{\frac{C_i^H - L_i^H}{M_i^{H1}} + L_i^H}{T_i} \right\rceil$.

Case 1: $\frac{C_i^H}{D_i} > 1$. $M_i^{H1} = \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$.

$$S_i^H \leq \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil \left\lceil \frac{D_i}{T_i} \right\rceil \quad \left(\because M_i^{H1} \geq \frac{C_i^H - L_i^H}{D_i - L_i^H} > 0\right)$$
$$\leq \left(\frac{C_i^H - \frac{D_i}{\rho}}{D_i - \frac{D_i}{\rho}} + 1\right)\left\lceil \frac{D_i}{T_i} \right\rceil \quad \left(\because \frac{C_i^H}{D_i} > 1 \wedge L_i^H \leq \frac{D_i}{\rho}\right)$$
$$\leq \left(\frac{\rho}{\rho-1}\frac{C_i^H}{D_i} + \frac{\rho-2}{\rho-1}\right) \cdot 2\frac{D_i}{T_i} \quad \left(\because \frac{D_i}{T_i} > 1\right)$$
$$< \left(\frac{\rho}{\rho-1} + \frac{\rho-2}{\rho-1}\right)\frac{C_i^H}{D_i} \cdot 2\frac{D_i}{T_i} \quad \left(\because \frac{C_i^H}{D_i} > 1 \wedge \rho \geq 4\right)$$
$$< 2\frac{C_i^H}{D_i} \cdot 2\frac{D_i}{T_i} < \rho U_i^H$$

Case 2: $\frac{C_i^H}{D_i} \leq 1$. $M_i^{H1} = 1$. $S_i^H = \left\lceil \frac{C_i^H}{T_i} \right\rceil \leq \frac{C_i^H}{T_i} + 1 \leq 2U_i^H \leq \rho U_i^H$.

Concluding from both cases, the lemma is proved. □

Lemma 11. *Under the configuration of Table II, if a Type I HI task τ_i satisfies the condition $\frac{1}{\rho}D_i \geq L_i^H \geq L_i^L$, $S_i^H \leq \rho U_i^H$ holds for any $\rho \geq 4$.*

Proof. **Case 1:** $M_i^L \geq \frac{C_i^H - L_i^H}{D_i - L_i^H}$. As already demonstrated in the proof of Lemma 9, $M_i^{H1} = \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$. Similar to Case 1 in the proof of Lemma 10, $S_i^H \leq \rho U_i^H$. **Case 2:** $M_i^L < \frac{C_i^H - L_i^H}{D_i - L_i^H}$. In this case $M_i^{H1} = \left\lceil \frac{C_i^H - C_i^L - L_i^H}{D_i - \frac{C_i^L}{M_i^L} - L_i^H} \right\rceil$.

$$D_i' = \frac{C_i^L - L_i^L}{M_i^L} + L_i^L \leq \frac{C_i^L - L_i^L}{\lceil(\rho-1)U_i^L\rceil} + L_i^L \leq \frac{C_i^L}{(\rho-1)\frac{C_i^L}{T_i}} + L_i^L$$
$$\leq \frac{T_i}{\rho-1} + \frac{D_i}{\rho}\left(\because L_i^L \leq \frac{D_i}{\rho}\right) \leq \left(\frac{1}{\rho-1} + \frac{2}{\rho}\right)T_i \left(\because 1 < \frac{D_i}{T_i} \leq 2\right)$$

It can be derived that for any $\rho \geq 2 + \sqrt{2} \approx 3.41$, the inequality $D_i' \leq T_i$ holds true. According to Lemma 9, with the arrangement in Table II, any Type I task can be completed before its deadline. Therefore we have $R_i^{H1} \leq D_i$. Because $1 < \frac{D_i}{T_i} \leq 2$, $1 \leq \left\lceil \frac{R_i^{H1}}{T_i} \right\rceil \leq \left\lceil \frac{D_i}{T_i} \right\rceil \leq 2$. There are two possibilities: $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil = 1$ and $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil = 2$. We consider them separately. **Case 2a:** $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil = 1$.

$$S_i^H = M_i^{H1}\left\lceil \frac{D_i'}{T_i} \right\rceil + M_i^{H2}\left(\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil - \left\lceil \frac{D_i'}{T_i} \right\rceil\right)$$
$$= M_i^{H1} \quad \left(\because \left\lceil \frac{R_i^{H1}}{T_i} \right\rceil = \left\lceil \frac{D_i'}{T_i} \right\rceil = 1\right)$$
$$\leq \left\lceil \frac{C_i^H - \frac{C_i^L}{M_i^L} - L_i^H}{D_i - \frac{C_i^L}{M_i^L} - L_i^H} \right\rceil \leq \left\lceil \frac{C_i^H - \frac{D_i}{\rho-1} - \frac{D_i}{\rho}}{D_i - \frac{D_i}{\rho-1} - \frac{D_i}{\rho}} \right\rceil$$
$$\left(\because \frac{C_i^H}{D_i} > 1 \wedge M_i^L \geq (\rho-1)\frac{C_i^L}{D_i} \wedge L_i^H \leq \frac{D_i}{\rho}\right)$$
$$\leq \frac{\rho(\rho-1)}{\rho^2-3\rho+1}\frac{C_i^H}{D_i} - \frac{2\rho-1}{\rho^2-3\rho+1} + 1$$

For any ρ satisfies $3.41 \approx 2 + \sqrt{2} \leq \rho \leq \frac{5+\sqrt{17}}{2} \approx 4.56$, the conditions $\frac{\rho(\rho-1)}{\rho^2-3\rho+1} \leq \rho$ and $-\frac{2\rho-1}{\rho^2-3\rho+1} + 1 \leq 0$ hold. Consequently, the above value is upper bounded by $\rho\frac{C_i^H}{D_i} \leq \rho U_i^H$ ($\because D_i > T_i$). For any ρ that satisfies $\rho > \frac{5+\sqrt{17}}{2}$, it has $-\frac{2\rho-1}{\rho^2-3\rho+1} + 1 > 0$, and the above value is upper bounded by $\left(\frac{\rho(\rho-1)}{\rho^2-3\rho+1} - \frac{2\rho-1}{\rho^2-3\rho+1} + 1\right)\frac{C_i^H}{D_i} = 2\frac{C_i^H}{D_i} \leq 2U_i^H$ ($\because C_i^H > D_i > T_i$) $\leq \rho U_i^H$. **Case 2b:** $\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil = 2$. Because $\frac{D_i}{T_i} \leq 2$, $\min\{\left\lceil \frac{R_i^{H1}}{T_i} \right\rceil T_i, D_i\} = \min\{2T_i, D_i\} = D_i$. $M_i^{H2} = \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$.

$$S_i^H = M_i^{H1} + M_i^{H2} \quad \left(\because \left\lceil \frac{R_i^{H1}}{T_i} \right\rceil = 2, \left\lceil \frac{D_i'}{T_i} \right\rceil = 1\right)$$
$$\leq \left\lceil \frac{C_i^H - \frac{C_i^L}{M_i^L} - L_i^H}{D_i - \frac{C_i^L}{M_i^L} - L_i^H} \right\rceil + \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$$
$$\leq \left(\frac{C_i^H - \frac{D_i}{\rho-1} - \frac{D_i}{\rho}}{D_i - \frac{D_i}{\rho-1} - \frac{D_i}{\rho}} + 1\right) + \left(\frac{C_i^H - \frac{D_i}{\rho}}{D_i - \frac{D_i}{\rho}} + 1\right)$$
$$\left(\because \frac{C_i^H}{D_i} > 1 \wedge M_i^L \geq (\rho-1)\frac{C_i^L}{D_i} \wedge L_i^H \leq \frac{D_i}{\rho}\right)$$
$$\leq \left(\frac{\rho(\rho-1)}{\rho^2-3\rho+1} + \frac{\rho}{\rho-1}\right)\frac{C_i^H}{D_i} + \left(2 - \frac{2\rho-1}{\rho^2-3\rho+1} - \frac{1}{\rho-1}\right)$$

It can be derived that for any $\rho \geq 4$, the condition $2 - \frac{2\rho-1}{\rho^2-3\rho+1} - \frac{1}{\rho-1} > 0$ holds. As a result, the above value is upper bounded by

$\left(\frac{\rho(\rho-1)}{\rho^2-3\rho+1}+\frac{\rho}{\rho-1}\right)\frac{C_i^H}{D_i} + \left(2-\frac{2\rho-1}{\rho^2-3\rho+1}-\frac{1}{\rho-1}\right)\frac{C_i^H}{D_i} \leq 4\frac{C_i^H}{D_i} \leq \rho U_i^H \left(\because C_i^H > D_i > T_i \wedge \rho \geq 4\right)$. □

Lemma 12. *Under the configuration of Table II, if a Type II HI task τ_i satisfies the condition $\frac{1}{\rho}D_i \geq L_i^H \geq L_i^L$, $S_i^L \leq (\rho-1)U_i^L + U_i^H$ holds for any $\rho \geq 4$.*

Proof. If $\widehat{M_i^L} > \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$, $M_i^L = \max\{\widehat{M_i^L}, \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil\} = \widehat{M_i^L}$. According to [13], for any task with $D_i \geq 3L_i^L$, $S_i^L \leq 3U_i^L < (\rho-1)U_i^L + U_i^H$. If $\widehat{M_i^L} \leq \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil$, $M_i^L = \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil \geq 1 \left(\because \widehat{M_i^L} \geq 1\right)$, we prove the bound holds under four cases. Note that the first three cases will use the fact that $S_i^L = M_i^L \left\lceil \frac{\frac{C_i^L - L_i^L}{M_i^L} + L_i^L}{T_i} \right\rceil < M_i^L \left(\frac{\frac{C_i^L - L_i^L}{M_i^L} + L_i^L}{T_i} + 1 \right) < \frac{C_i^L}{T_i} - \frac{L_i^L}{T_i} + \left(1 + \frac{L_i^L}{T_i}\right) M_i^L$.

Case 1: $\frac{C_i^H}{D_i} \leq 1$.

$$\begin{aligned} S_i^L &< \frac{C_i^L}{T_i} - \frac{L_i^L}{T_i} + \left(1 + \frac{L_i^L}{T_i}\right) \quad \left(\because \frac{C_i^H}{D_i} \leq 1 \Rightarrow M_i^L = 1\right) \\ &< \frac{C_i^L}{T_i} + 1 < U_i^L + U_i^H \left(\because U_i^H > 1\right) < (\rho-1)U_i^L + U_i^H \end{aligned}$$

Case 2: $C_i^H - C_i^L - L_i^H \leq 0$. In this case, we have $M_i^L \leq \left\lceil \frac{C_i^L}{D_i - L_i^H} \right\rceil$. Therefore,

$$\begin{aligned} S_i^L &< \frac{C_i^L}{T_i} - \frac{L_i^L}{T_i} + \left(1 + \frac{L_i^L}{T_i}\right) \left\lceil \frac{C_i^L}{D_i - L_i^H} \right\rceil \\ &< \frac{C_i^L}{T_i} - \frac{L_i^L}{T_i} + \left(1 + \frac{L_i^L}{T_i}\right) \left(\frac{C_i^L}{D_i - L_i^H} + 1\right) \\ &< \frac{C_i^L}{T_i} + \left(1 + \frac{\frac{D_i}{\rho}}{T_i}\right) \frac{C_i^L}{D_i - \frac{D_i}{\rho}} + 1 \quad \left(\because L_i^L \leq L_i^H \leq \frac{D_i}{\rho}\right) \\ &< \frac{C_i^L}{T_i} + \frac{D_i + \frac{D_i}{\rho}}{T_i} \frac{C_i^L}{D_i - \frac{D_i}{\rho}} + 1 \quad (\because D_i > T_i) \\ &< \frac{2\rho}{\rho-1} U_i^L + U_i^H \quad \left(\because U_i^H > 1\right) \end{aligned}$$

For any ρ satisfies $\rho \geq 2 + \sqrt{3} \approx 3.73$, the condition $\frac{2\rho}{\rho-1} \leq \rho - 1$ holds. $\frac{2\rho}{\rho-1} U_i^L + U_i^H \leq (\rho-1)U_i^L + U_i^H$ follows.

Case 3: $\frac{C_i^H}{D_i} > 1$ and $C_i^L > (T_i - L_i^L) \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil + L_i^L$.

$$\begin{aligned} S_i^L &< S_i^L + 2\frac{C_i^L}{T_i} - 2\frac{(T_i - L_i^L)M_i^L + L_i^L}{T_i} \\ &< 3\frac{C_i^L}{T_i} - 3\frac{L_i^L}{T_i} + \left(3\frac{L_i^L}{T_i} - 1\right) M_i^L \\ &< 3\frac{C_i^L}{T_i} - 3\frac{L_i^L}{T_i} + 3\frac{L_i^L}{T_i} \left(\frac{C_i^H - L_i^H}{D_i - L_i^H} + 1\right) \\ &< 3\frac{C_i^L}{T_i} + 3\frac{\frac{D_i}{\rho}}{T_i} \frac{C_i^H - \frac{D_i}{\rho}}{D_i - \frac{D_i}{\rho}} \quad \left(\because L_i^L \leq L_i^H \leq \frac{D_i}{\rho} \wedge \frac{C_i^H}{D_i} > 1\right) \\ &< 3\frac{C_i^L}{T_i} + 3\frac{\frac{D_i}{\rho}}{T_i} \frac{\rho}{\rho-1} \frac{C_i^H}{D_i} \quad (\because \rho > 1) \\ &< 3U_i^L + \frac{3}{\rho-1} U_i^H < (\rho-1)U_i^L + U_i^H \quad (\because \rho \geq 4) \end{aligned}$$

Case 4: $\frac{C_i^H}{D_i} > 1$, $\frac{D_i}{T_i} > 2$ and $C_i^L \leq (T_i - L_i^L) \left\lceil \frac{C_i^H - L_i^H}{D_i - L_i^H} \right\rceil + L_i^L$. Because $C_i^L \leq (T_i - L_i^L)M_i^L + L_i^L$, we have $\frac{\frac{C_i^L - L_i^L}{M_i^L} + L_i^L}{T_i} \leq 1$ and $S_i^L = M_i^L$. Consequently,

$$\begin{aligned} S_i^L &< \frac{C_i^H - \frac{D_i}{\rho}}{D_i - \frac{D_i}{\rho}} + 1 \left(\because C_i^H > D_i \geq \rho L_i^H\right) < \frac{\rho}{\rho-1}\frac{C_i^H}{D_i} + \frac{\rho-2}{\rho-1} \\ &< \frac{\rho}{\rho-1}\frac{C_i^H}{D_i} + \frac{\rho-2}{\rho-1}\frac{C_i^H}{D_i} \quad \left(\because \frac{C_i^H}{D_i} > 1 \wedge \rho > 2\right) \\ &< 2\frac{C_i^H}{2T_i} \left(\because \frac{D_i}{T_i} > 2\right) < U_i^H < (\rho-1)U_i^L + U_i^H \end{aligned}$$

□

Lemma 13. *Under the configuration of Table II, if a Type I HI task τ_i satisfies the condition $\frac{1}{\rho}D_i \geq L_i^H \geq L_i^L$, $S_i^L \leq (\rho-1)U_i^L + U_i^H$ holds for any $\rho \geq 4$.*

Proof.

$$\begin{aligned} S_i^L &\leq M_i^L \left\lceil \frac{\frac{C_i^L - L_i^L}{\lceil (\rho-1)U_i^L \rceil} + L_i^L}{T_i} \right\rceil \quad \left(\because M_i^L \geq \lceil (\rho-1)U_i^L \rceil\right) \\ &\leq M_i^L \left\lceil \frac{\frac{C_i^L}{(\rho-1)U_i^L} + L_i^L}{T_i} \right\rceil \leq M_i^L \left\lceil \frac{\frac{1}{\rho-1}T_i + \frac{2}{\rho}T_i}{T_i} \right\rceil \left(\because L_i^L \leq \frac{D_i}{\rho} \leq \frac{2T_i}{\rho}\right) \\ &\leq M_i^L \quad \left(\because \frac{1}{\rho-1} + \frac{2}{\rho} < 1\right) \end{aligned}$$

If $M_i^L = \lceil (\rho-1)U_i^L \rceil$, $M_i^L < (\rho-1)U_i^L + 1 < (\rho-1)U_i^L + U_i^H (\because U_i^H > 1)$. If $M_i^L = \left\lceil \frac{C_i^L}{(\frac{1}{\rho-1} + \frac{1}{\rho})D_i - L_i^H} \right\rceil$,

$$\begin{aligned} M_i^L &\leq \left\lceil \frac{C_i^L}{(\frac{1}{\rho-1} + \frac{1}{\rho})D_i - \frac{D_i}{\rho}} \right\rceil \quad \left(\because L_i^H \leq \frac{D_i}{\rho}\right) \\ &< (\rho-1)\frac{C_i}{D_i} + 1 < (\rho-1)\frac{C_i}{T_i} + 1 \quad (\because D_i > T_i) \\ &< (\rho-1)U_i^L + U_i^H \quad \left(\because U_i^H > 1\right) \end{aligned}$$

□

Finally, we are able to establish the capacity augmentation bound by Theorems 1 and 2.

Theorem 1. *Replacing the processor assignment strategy for τ_{Hu}^{HI} in Algorithm 2 with Table II gives an algorithm that has a capacity augmentation bound of 4 in scheduling high-utilization relaxed-deadline tasks.*

Proof. If we can prove that under the assumptions of Theorem 1 a task set $\tau = \tau_{Hu}^{HI} \cup \tau_{Hu}^{LO}$ $(\forall \tau_i \in \tau, D_i > T_i)$ is always schedulable when it satisfies: $\sum_{\forall \tau_i \in \tau} U_i^L \leq \frac{M}{\rho}$, $\sum_{\forall \tau_i \in \tau_{Hu}^{HI}} U_i^H \leq \frac{M}{\rho}$, $\forall \tau_i \in \tau_{Hu}^{HI} : L_i^L \leq L_i^H \leq \frac{D_i}{\rho}$, $\forall \tau_i \in \tau_{Hu}^{LO} : L_i^L \leq \frac{D_i}{\rho}$ and $\rho = 4$, then according to Definition 2 this theorem is true.

According to Lemmas 10, 11, 12 and 13, if the above conditions are satisfied, the total number of processors is upper bounded by $\sum_{\tau_i \in \tau_{Hu}^{HI}} 4U_i^H \leq M$ in the critical state, and $\sum_{\tau_i \in \tau_{Hu}^{HI}} (3U_i^L + U_i^H) + \sum_{\tau_i \in \tau_{Hu}^{LO}} 3U_i^L \leq 3\sum_{\tau_i \in \tau} U_i^L + \sum_{\tau_i \in \tau_{Hu}^{HI}} U_i^H \leq \frac{3}{\rho}M + \frac{1}{\rho}M \leq M$ in the typical state. Therefore, the task set is schedulable in both system modes. □

Theorem 2. *The capacity augmentation bound of Algorithm 2 is 4 in scheduling high-utilization relaxed-deadline tasks.*

Proof. Under the assumption that only high-utilization tasks exist, if $\sum_{\forall \tau_i \in \tau_{Hu}^{HI}} \min_{M_i^{H1}} S_i^H \leq M$ under the constraint of Lemma 4, Algorithm 2 guarantees that $\sum_{\forall \tau_i \in \tau_{Hu}^{HI}} S_i^H \leq M$. Let's assume that the value of $\sum_{\forall \tau_i \in \tau_{Hu}^{HI}} S_i^L$ under the processor assignment strategy of Table II and Algorithm 2 are S^{LT} and S^{LA}, respectively. Algorithm 2 always makes processor assignments that minimize the value of $\sum_{\forall \tau_i \in \tau_{Hu}^{HI}} S_i^L$ under the constraint of Lemma 4, so we have $S^{LA} \leq S^{LT}$. Since in both cases ReserveProcs [13] assigns processors to

LO tasks in the typical state, the value of $\sum_{\forall \tau_i \in \tau_{Hu}^{LO}} S_i^L$ remains unchanged. If $S^{LT} + \sum_{\forall \tau_i \in \tau_{Hu}^{LO}} S_i^L \leq M$ holds true, $S^{LA} + \sum_{\forall \tau_i \in \tau_{Hu}^{LO}} S_i^L \leq M$ is also true. Therefore, as long as a task set is schedulable by adopting the processor assignment strategy of Table II, it is also schedulable under unaltered Algorithm 2. We can then safely conclude that the capacity augmentation bound of Algorithm 2 is at most 4. □

VI. Scheduling Dual-Criticality Tasks in General

Generalizing our algorithm to support both high- and low-utilization tasks is similar to the approach presented by Li et al. [6]. We reuse MC-Partition-UT-0.75 [28] to schedule low-utilization tasks. The whole scheduling algorithm is modified in the following manner. We first reserve the necessary number of processors for low-utilization tasks; and then, we assign processors to each high-utilization task and test if they are schedulable on the remaining processors with Algorithm 2.

The scheduling of low-utilization tasks is performed as follows: 1) Every low-utilization task τ_i is converted into an implicit-deadline task; and 2) All the low-utilization tasks are scheduled on a set of allocated processors by a partitioned scheduling algorithm MC-Partition-UT-0.75 [28], executed and scheduled as though they are sequential tasks. Note that every low-utilization task is converted into an implicit-deadline task before the partition process. This operation permits us to use the existing algorithms designed for implicit-deadline tasks, such as MC-Partition-UT-0.75. In this scenario, $M^{Lr} \leq M$ and $M^{Hr} \leq M$ when applying Algorithm 2.

We can obtain a capacity augmentation bound for the generalized algorithm by Theorem 3.

Theorem 3. *A joint use of Algorithm 2 and MC-Partition-UT-0.75 [28] in scheduling relaxed-deadline tasks in general has a capacity augmentation bound of* $\frac{4M}{M-1} \approx 4$ *for large* M.

Proof. Let's assume s is the utilization bound of the algorithm that is adopted in scheduling low-utilization tasks. The numbers of processors assigned to low-utilization tasks in typical state and critical state are upper bounded by $\left\lceil s \sum_{\forall \tau_i \in \tau_{Lu}^{LO} \cup \tau_{Lu}^{HI}} U_i^L \right\rceil < s \sum_{\forall \tau_i \in \tau_{Lu}^{LO} \cup \tau_{Lu}^{HI}} U_i^L + 1$ and $\left\lceil s \sum_{\forall \tau_i \in \tau_{Lu}^{HI}} U_i^H \right\rceil < s \sum_{\forall \tau_i \in \tau_{Lu}^{HI}} U_i^H + 1$, respectively. When $\rho = \frac{4M}{M-1} > 4$, Lemmas 10, 11, 12, and 13 still hold. When $s \leq 3$ ($s = \frac{8}{3} < 3$ for MC-Partition-UT-0.75), together with $\rho = \frac{4M}{M-1}$, Lemmas 12 and 13, the necessary processor reservation in the typical state is upper bounded by $3 \sum_{\tau_i \in \tau} U_i^L + \sum_{\tau_i \in \tau^{HI}} U_i^H + 1 \leq \frac{3}{\rho} M + \frac{1}{\rho} M + 1 \leq 4M \frac{M-1}{4M} + 1 \leq M$. Similarly, $s \leq 3$ and $\rho = \frac{4M}{M-1}$, together with Lemmas 10 and 11 guarantee an upper bound for the overall processor requirement in the critical state as $4 \sum_{\tau_i \in \tau^{HI}} U_i^H + 1 \leq \frac{4}{\rho} M + 1 \leq 4M \frac{M-1}{4M} + 1 \leq M$. Theorem 3 follows. □

Note that Li et al. [6], [7] proved that for implicit-deadline tasks, their algorithm achieves a capacity augmentation bound of $2 + \sqrt{2} \approx 3.4$ for high-utilization tasks and $\frac{11M}{3M-3} \approx 3.7$ for tasks in general. Our algorithm's capacity augmentation bound increases only slightly compared with their results, even as the tasks become significantly more complex.

VII. Experiments

We evaluate our algorithm in terms of the acceptance ratio (the ratio of schedulable task sets to total task sets generated) and computation time. The program is written in Python and runs on a computer with a $3.6GHz$ 8-Core Intel Core i7 CPU and $64GB$ of RAM.

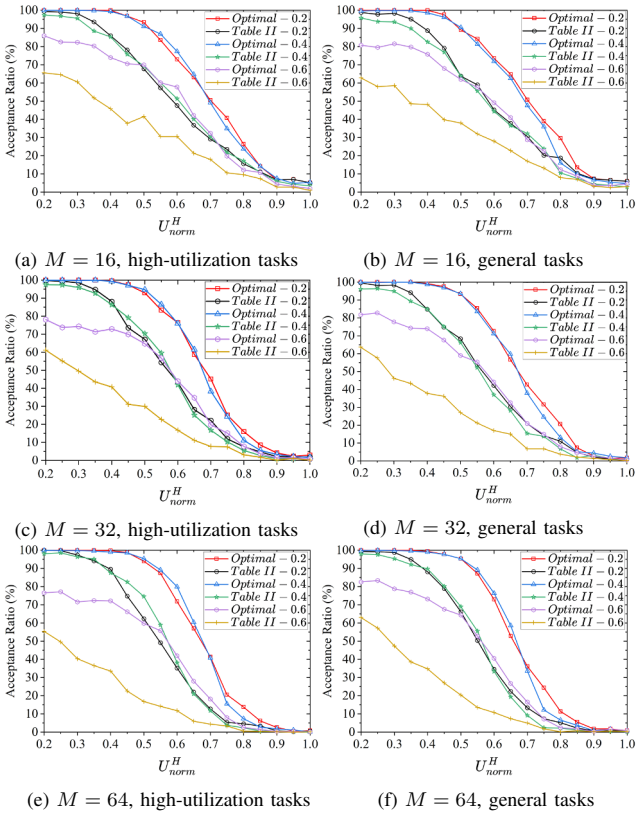
A. Task Generation Procedure

We define *normalized utilization* as the ratio of total utilization to M. Let $\{U_{norm}^L, U_{norm}^H\}$ and $\{U^L, U^H\}$ denote the normalized and total utilization in typical and critical states, respectively. Then, $U_{norm}^H = \frac{U^H}{M}$ and $U_{norm}^L = \frac{U^L}{M}$. During task set generation, $U_{norm}^L \in [0.2, 1.0]$, $U_{norm}^H \in [0.2, 1.0]$ and $M \in \{16, 32, 64\}$ are input parameters. For each combination of U_{norm}^L, U_{norm}^H and M, we generated 500 task sets.

We generate two types of task sets: those mainly consisting of high-utilization tasks and those composed of relaxed-deadline DAG tasks in general. The generation process is as follows. We choose integer N uniformly from $[2, \lfloor U^L \rfloor)$. After which, we choose integer $|\tau^{HI}|$ uniformly from $[1, min\{N, \lfloor U^H \rfloor\})$, to ensure a reasonable number of HI tasks for any $\{U^L, U^H\}$. Then, we have $|\tau^{LO}| = N - |\tau^{HI}|$. Without loss of generality, we let $\tau^{HI} = \{\tau_1, \tau_2, ..., \tau_{|\tau^{HI}|}\}$ and $\tau^{LO} = \{\tau_{|\tau^{HI}|+1}, \tau_{|\tau^{HI}|+2}, ..., \tau_N\}$ during the rest of the task generation process. After N, $|\tau^{LO}|$ and $|\tau^{HI}|$ are obtained, we use the Dirichlet-Rescale algorithm [29] to generate utilization vectors based on $\{U^L, U^H\}$. In the evaluation of high-utilization tasks, U_i^H for each HI task is obtained by calling $DRS(|\tau^{HI}|, U^H, upper\_bounds = None, lower\_bounds = \mathbf{u^{minH}})$ [29], where $\mathbf{u^{minH}} = (1, 1, ..., 1)$. After determining U_i^H, we let $(U_1^L, U_2^L, ..., U_N^L) = DRS(N, U^L, upper\_bounds = \mathbf{u^{maxL}}, lower\_bounds = \mathbf{u^{minL}})$. $\mathbf{u^{minL}} = (0, 0, ..., 0, 1, 1, ..., 1)$ with the first $|\tau^{HI}|$ elements as 0 and the last $|\tau^{LO}|$ elements as 1. This configuration ensures that every LO task has $U_i^L \geq 1$. $\mathbf{u^{maxL}} = (U_1^H, U_2^H, ..., U_{|\tau^{HI}|}^H, U^L, U^L, ...U^L)$, where the last $|\tau^{LO}|$ elements are set to U^L, and the ith ($i \in [1, |\tau^{HI}|]$) element is set to U_i^H since any HI task should satisfy $U_i^L \leq U_i^H$. In the evaluation of general task sets, we change $lower\_bounds$ to $None$ during the generation of utilization vectors. Next, we determine C_i^L, C_i^H, L_i^L, L_i^H, T_i, and D_i for each task. We uniformly choose integers from $[10, 1000]$ and $[1, D_i)$ as D_i and T_i, respectively. Subsequently, we have $C_i^L = U_i^L T_i$ and $C_i^H = U_i^H T_i$. For generating L_i^H of a HI task, we first select a random $\gamma \in [0.1, 0.5]$, then let $L_i^H = \min\{\gamma D_i, C_i^H\}$. For L_i^L of a HI task, we first select a random $\gamma^\bullet \in [0.1, 0.9]$, then let $L_i^L = \min\{\gamma^\bullet L_i^H, C_i^L\}$. Similarly, for L_i^L of a LO task, we first select a random $\beta \in [0.1, 0.5]$, then let $L_i^L = \min\{\beta D_i, C_i^L\}$.

B. Results

Figure 5 shows the acceptance ratios when Algorithm 2 is unaltered and when Algorithm 2 is modified by replacing



(a) $M = 16$, high-utilization tasks (b) $M = 16$, general tasks

(c) $M = 32$, high-utilization tasks (d) $M = 32$, general tasks

(e) $M = 64$, high-utilization tasks (f) $M = 64$, general tasks

Fig. 5: Acceptance ratios when scheduling relaxed-deadline MC tasks. $Optimal-$ and $Table\ II - U^L_{norm}$ represent the results when Algorithm 2 is unaltered and modified by replacing processor assignment strategy for τ^{HI}_{Hu} with Table II ($\rho = 4$), respectively.

processor assignment strategy for τ^{HI}_{Hu} with Table II, under different U^H_{norm}, U^L_{norm} and M. Limited by space, we only show the results when $U^L_{norm} = 0.2$, 0.4 and 0.6. For the scheduling of high-utilization tasks, Figures 5a, 5c and 5e show that Algorithm 2 always has higher acceptance ratios than the altered Algorithm 2 which is in line with our theory. Both algorithms admit fewer task sets when U^L_{norm} and U^H_{norm} get higher. For example, when $M = 32$, the acceptance ratios of $Optimal-0.4$ for $U^H_{norm} = 0.4$, 0.6 and 0.8 are approximately 100%, 76%, and 11%, respectively; and the acceptance ratios of $Table\ II - 0.4$ are around 86%, 42%, and 5.6% for the same values of U^H_{norm}. When $M = 32$ and $U^H_{norm} = 0.6$, the acceptance ratios of $Optimal - U^L_{norm}$ and $Table\ II - U^L_{norm}$ decrease by about 32% and 26%, respectively, when U^L_{norm} is increased from 0.2 to 0.6. The acceptance ratios of both Algorithms drop as M increases, while Algorithm 2 changes slower indicating a more steady performance. For instance, with $U^H_{norm} = 0.6$ and M increasing from 16 to 64, the acceptance ratio of $Optimal - 0.6$ decreases from 58% to 42%, while that of $Table\ II - 0.6$ lowers from 31% to 12%. Figures 5b, 5d and 5f show that the acceptance ratio for general task sets changes little compared to high-utilization task sets. In most cases, the acceptance ratios for general tasks are slightly higher, which can be attributed to low-utilization tasks. The experimental results show that our algorithm can maintain a relatively high schedulability ratio. This provides a foundation for future extensions of our algorithm to support multi-criticality systems. We also observe that, although our algorithms have a capacity augmentation bound of 4, a significant portion of task sets remains schedulable even when U^L_{norm} or U^H_{norm} exceeds 0.25. While this result does not necessarily imply a capacity augmentation bound lower than 4, it is worth investigation in future studies.

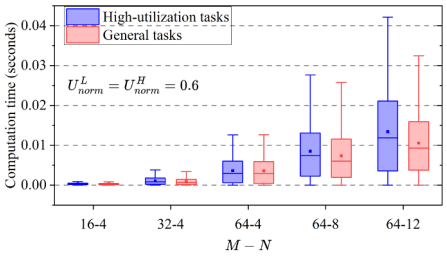


Fig. 6: Computation time of schedulability analysis. Each box displays the range between the first and third quartiles. The whiskers cover the range between the $5th$ and $95th$ percentiles. The solid square represents the mean and the horizontal bar is the median.

Figure 6 shows the analysis time of our algorithm (unaltered), with each box corresponding to a specific value of M and N. The task generation process is slightly adjusted by fixing N at 4, 8, or 12, and fixing U^L_{norm} and U^H_{norm} at 0.6, while the rest remains. We observe that the computation time increases with both M and N. For example, with $N = 4$, and M increasing from 16 to 64, the mean of the analysis time for high-utilization tasks increases from $0.3ms$ to $3.6ms$. With $M = 16$, and N increasing from 4 to 12, the mean of the analysis time for high-utilization tasks increases from $3.6ms$ to $13.4ms$. This result aligns with the time complexity. The analysis time for general tasks is consistently lower than that for high-utilization tasks. In all configurations tested, the maximum analysis time remains below $45ms$, demonstrating that our algorithm is computationally efficient.

VIII. Summary

In this paper, we have proposed a federated algorithm for scheduling relaxed-deadline DAG tasks in dual-criticality systems. We prove our algorithm has a capacity augmentation bound of 4 for high-utilization tasks and $\frac{4M}{M-1}$ for relaxed-deadline tasks in general. We show that the federated scheduling algorithm remains a useful strategy in reducing the difficulty and complexity of schedulability analysis. This advantage of federated scheduling is powerful in building real-time scheduling algorithms, especially when coping with complex systems like MC systems. With careful design, the 'resource waste' due to isolation can be negligible compared to the pessimism-induced waste in task interference analysis. We anticipate researchers working on the same task model could conduct comparisons to either confirm or refute our hypothesis. In our next step, we plan to extend our algorithm to support more than two criticality levels.

ACKNOWLEDGMENTS

This work was supported by Hong Kong GRF under grant No. 15206221 and 11208522, the National Natural Science Foundation of China under grant No. 62202093, and the Fundamental Research Funds for the Central Universities under grant No. 2572023CT16-06. This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2022R1A4A3018824, RS-2024-00438248).

REFERENCES

[1] A. Burns and R. Davis, *Mixed Criticality Systems - A Review: (13th Edition, February 2022)*, Feb. 2022, this is the 13th version of this review now updated to cover research published up to the end of 2021.

[2] S. Zhao, X. Dai, and I. Bate, "Dag scheduling and analysis on multi-core systems by modelling parallelism and dependency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4019–4038, 2022.

[3] G. Liu, Y. Lu, S. Wang, and Z. Gu, "Partitioned multiprocessor scheduling of mixed-criticality parallel jobs," in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014, pp. 1–10.

[4] R. Medina, É. Borde, and L. Pautet, "Scheduling multi-periodic mixed-criticality dags on multi-core architectures," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 254–264.

[5] R. Medina, É. Borde, and L. Pautet, "Generalized mixed-criticality static scheduling for periodic directed acyclic graphs on multi-core processors," *IEEE Transactions on Computers*, vol. 70, no. 3, pp. 457–470, 2021.

[6] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–12.

[7] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real-Time Systems*, vol. 53, no. 5, pp. 760–811, 2017.

[8] R. M. Pathan, "Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors," in *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*, ser. LIPIcs, S. Altmeyer, Ed., vol. 106. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 12:1–12:22.

[9] K. Agrawal and S. Baruah, "A Measurement-Based Model for Parallel Real-Time Tasks," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Altmeyer, Ed., vol. 106. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 5:1–5:19.

[10] T. Yang, Y. Tang, X. Jiang, Q. Deng, and N. Guan, "Semi-federated scheduling of mixed-criticality system for sporadic dag tasks," in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, 2019, pp. 163–170.

[11] A. Burns and R. I. Davis, "Schedulability analysis for adaptive mixed criticality systems with arbitrary deadlines and semi-clairvoyance," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 12–24.

[12] B. Yu, W. Hu, L. Xu, J. Tang, S. Liu, and Y. Zhu, "Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1067–1081.

[13] F. Guan, L. Peng, and J. Qiao, "A new federated scheduling algorithm for arbitrary-deadline dag tasks," *IEEE Transactions on Computers*, vol. 72, no. 8, pp. 2264–2277, 2023.

[14] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *2013 25th Euromicro Conference on Real-Time Systems*, 2013, pp. 225–233.

[15] A. Parri, A. Biondi, and M. Marinoni, "Response time analysis for g-edf and g-dm scheduling of sporadic dag-tasks with arbitrary deadline," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ser. RTNS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 205–214.

[16] S. Baruah, "Federated scheduling of sporadic dag task systems," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 179–186.

[17] N. Ueter, G. von der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 482–494.

[18] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *2014 26th Euromicro Conference on Real-Time Systems*, 2014, pp. 85–96.

[19] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 13–22.

[20] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *2012 IEEE 33rd Real-Time Systems Symposium*, 2012, pp. 63–72.

[21] X. Jiang, N. Guan, D. Liu, and W. Liu, "Analyzing gedf scheduling for parallel real-time tasks with arbitrary deadlines," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 1537–1542.

[22] K. Wang, X. Jiang, N. Guan, D. Liu, W. Liu, and Q. Deng, "Real-time scheduling of dag tasks with arbitrary deadlines," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 6, pp. 66:1–66:22, Oct. 2019.

[23] J. Li, K. Agrawal, C. Lu, and C. Gill, "Outstanding paper award: Analysis of global EDF for parallel tasks," in *2013 25th Euromicro Conference on Real-Time Systems*, 2013, pp. 3–13.

[24] F. Guan, L. Peng, and J. Qiao, "A fluid scheduling algorithm for dag tasks with constrained or arbitrary deadlines," *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1860–1873, 2022.

[25] S. Baruah, "Certification-cognizant scheduling of tasks with pessimistic frequency specification," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, 2012, pp. 31–38.

[26] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and Y. Wang, "Virtually-federated scheduling of parallel real-time tasks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 482–494.

[27] K. Dudziński and S. Walukiewicz, "Exact methods for the knapsack problem and its generalizations," *European Journal of Operational Research*, vol. 28, no. 1, pp. 3–21, 1987.

[28] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, vol. 50, no. 1, pp. 142–177, 2014.

[29] D. Griffin, I. Bate, and R. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *2020 IEEE Real-Time Systems Symposium (proceedings)*, Dec. 2020.