Mixed-Criticality Scheduling on Multiprocessors using Task Grouping

Jiankang Ren* Linh Thi Xuan Phan†
*School of Software Technology, Dalian University of Technology, China
†Computer and Information Science Department, University of Pennsylvania, U.S.A.

***Abstract*—Real-time systems are increasingly running a mix of tasks with different criticality levels: for instance, unmanned aerial vehicle has multiple software functions with different safety criticality levels, but runs them on a single, shared computational platform. In addition, these systems are increasingly deployed on multiprocessor platforms because this can help to reduce their cost, space, weight, and power consumption. To assure the safety of such systems, several mixed-criticality scheduling algorithms have been developed that can provide mixed-criticality timing guarantees. However, most existing algorithms have two important limitations: they do not guarantee strong isolation among the high-criticality tasks, and they offer poor real-time performance for the low-criticality tasks.**

In this paper, we present a partitioned scheduling scheme for mixed-criticality tasks on multiprocessor platforms that addresses both issues. Our scheduling scheme consists of (i) a task-to-processor packing algorithm that takes into account the demands of tasks with respect to their criticality levels, and (ii) a mixed-criticality uniprocessor scheduling strategy that is based on task grouping. Our strategy associates each high-criticality task with a subset of the low-criticality tasks and encapsulates them in a task group, which is scheduled with the other task groups under the Earliest Deadline First (EDF) policy. Within each task group, the low-criticality task and the high-criticality tasks are scheduled using a server-based strategy, so as to enable more of the former to meet their deadlines without affecting the latter. We present a schedulability analysis for our scheduling strategy, and we show how tasks can be grouped using Mixed Integer Nonlinear Programming. Our evaluation shows that our proposed scheme significantly outperforms existing partitioned mixed-criticality scheduling algorithms, in terms of both the fraction of schedulable task sets and its ability to schedule low-criticality tasks.

I. Introduction

Due to recent advances in chip technology, multiprocessor platforms are increasingly applied in safety-critical domains (such as avionics, automotive, and industrial control) to accommodate the increasing demand for computationally-intensive workloads. Many systems in these domains are mixed-criticality (MC) real-time systems – that is, they integrate multiple functionalities (tasks) with different safety criticality levels, such as flight-critical and mission-critical tasks, on a single, shared hardware platform. This is typically done to meet stringent requirements on space, weight, and power consumption. Therefore, it is important to provide strong isolation and timing guarantees for the mixed-criticality tasks with respect to their criticalities.

Since MC real-time systems run a mix of tasks with different degrees of criticality, these systems are more complex and unpredictable than traditional real-time systems. In order to certify the correctness of a MC system, it is common to make certain assumptions about the worst-case behavior of the system – for instance, some task parameters are allowed to depend on the criticality levels of the tasks. A typical approach is to assume more pessimistic worst-case execution times (WCETs) for higher criticality levels for each task; in systems with two levels of criticality, a task is assumed to be in the high-criticality mode if its execution time exceeds the low-criticality WCET. A major challenge in this approach is to guarantee mixed-criticality schedulability (i.e., high-criticality tasks always meet their deadlines and low-criticality tasks meet their deadlines if the high-criticality tasks are always in the low-criticality mode) of the tasks while achieving efficient resource use.

There is a rich literature on scheduling mixed-criticality real-time systems on both uniprocessor and multiprocessor platforms (e.g., [1]–[11]); however, most existing solutions have at least one of the following three key limitations. First, they assume that, whenever a job of a high-criticality task exhibits high-criticality behavior – that is, its execution time exceeds the task's WCET as estimated at the low-criticality level – all current and future jobs of every high-criticality task in the system will also exhibit high-criticality behavior, which does not always happen in general. Second, existing techniques cannot fully capture the dynamic behavior of the system: each task can dynamically transition back and forth between the high-criticality and the low-criticality mode. Third, existing solutions do not offer good performance for the low-criticality tasks: since these are killed as soon as a high-criticality job in the system enters its high-criticality mode, it is possible that no low-criticality job can ever meet its deadline – for instance, when the criticality change happens shortly after the system begins its execution. This is undesirable because low-criticality tasks require a certain timing performance as well (although with weaker guarantees than the high-criticality tasks) [12].

In this paper, we propose a compositional approach to mixed-criticality task scheduling that can solve these problems. Our approach, called TG-PEDF, provides strong isolation among the high-criticality tasks, and it enables more low-criticality tasks to be scheduled while still ensuring the mixed-criticality schedulability guarantee of the entire system. TG-PEDF is based on task grouping and compositional scheduling: each high-criticality task is encapsulated in a separate task group (to enable isolation) together with a subset of the low-criticality tasks (to enable overbooking of resources). Within each task group, the tasks are scheduled with a server-based mixed-criticality scheduling strategy, and the different task groups are scheduled on the platform as conventional tasks under the EDF scheduling policy. This ensures that the high-criticality behavior of a high-criticality task cannot affect other high-criticality tasks, and it also enables more low-criticality jobs in the system to meet their deadlines. This is because (i) the interference of a high-criticality task on low-criticality tasks beyond its local task group is significantly reduced, and because (ii) low-criticality tasks are not always discarded, but rather scheduled according to best-effort service when the high-criticality task in the same task group is in the high-criticality mode. Furthermore, our scheduling algorithm can dynamically adapt to the criticality mode of each job (by enabling tasks to return to the low-criticality mode, depending on the execution time of their current jobs), and can thus save resources at runtime.

Contributions. This paper makes the following contributions:

- A partitioned mixed-criticality scheduling approach based on task grouping for implicit-deadline periodic MC tasks on multiprocessor platforms. In contrast to existing solutions, our approach provides strong isolation between high-criticality tasks and better timing performance for low-criticality tasks;
- A schedulability analysis for a set of mixed-criticality tasks that have been assigned to a single processor under our task group based scheduling algorithm;
- A Mixed Integer Nonlinear Programming (MINLP) formulation for the task grouping, based on the schedulability constraints, that can improve the schedulability of the system; and
- A task packing algorithm that can save resources by taking into account the demand of both low-criticality tasks and high-criticality tasks during deployment.

Our evaluation on randomly generated task systems shows that the schedulability of our proposed scheme outperforms the state-of-the-art MC scheduling algorithms.

The rest of the paper is organized as follows. In Section II, we summarize the related work. Section III presents the system model. The detailed description of the task group based MC scheduling on a uniprocessor is given in Section IV. Section V presents the task packing algorithm. We discuss our evaluation in Section VI and conclude the paper in Section VII.

II. Related Work

Since Vestal's initial work on mixed-criticality systems [1], a rich literature on mixed-criticality scheduling of real-time systems has developed; see [13] for a survey. Traditionally, work in this domain has focused on uniprocessor platforms [2]–[7], [14]–[21], but recently efforts have shifted towards multiprocessor platforms.

Several multiprocessor mixed-criticality scheduling algorithms have already been developed [8]–[11], [22]–[25], including both global and partitioned approaches. For instance, Pathan [22] derived a sufficient schedulability condition of a global fixed-priority scheduling algorithm on preemptive multiprocessors based on response time analysis. Baruah et al. [10] proposed global and partitioned scheduling algorithms for mixed-criticality implicit-deadline sporadic task systems by combining fpEDF [23] with EDF-VD, and they showed that partitioned scheduling gives better system schedulability than global scheduling. For harmonic task systems, Mollison et al. [8] proposed an architecture to use appropriate scheduling techniques for tasks with different criticalities on different processors, which provides timing isolation between these levels through a bandwidth reservation server. Kelly et al. [9] studied the schedulability of various partitioning techniques inspired by bin packing, as well as traditional partitioning heuristics under fixed-priority scheduling for MC systems. To address the effects of the physical environment on the system, Niz et al. [11] presented a partitioned multiprocessor scheduling scheme based on a multi-mode extension of the zero slack rate-monotonic scheduling. Gu et al. [24] extended the demand-based single-processor mixed-criticality scheduling [25] (called EY in this paper) to multiprocessor platforms and proposed two enhancements based on heavy low-criticality task awareness and balance factor to improve the system schedulability. However, these existing solutions assume that, whenever a job of a high-criticality task exhibits high-criticality behavior, all current and future jobs of every high-criticality task in the system will exhibit high-criticality behavior and thus all low-criticality tasks will be dropped. Our work removes this assumption by considering the specific mode of each high-criticality tasks in the scheduling.

There has also been work on offering degraded service to low-criticality tasks when the system is in the high-criticality mode. For instance, Su et al. [17], [26] studied an elastic mixed-criticality task model that allows low-criticality task to have different periods in different system modes. To maximize the execution rate of low-criticality tasks, Jan et al. [18] introduced stretching factors and proposed an associated online decision algorithm based on the existing elastic task model. Santy et al. [15] proposed a technique that aims to avoid or delay the rise in criticality level in order to minimize the number of tasks that will miss their deadlines. From the implementation perspective, Burns et al. [19] pointed out the limitations of the existing standard mixed-criticality models, and they proposed a more practical model that considers the service adaption of low-criticality tasks and the system mode swathing problem from the low-criticality mode to the high-criticality mode. Huang et al. [20], [27] studied the service reconfiguration and resetting for low-criticality tasks by quantifying the degraded service of low-criticality tasks. Gu et al. [21] proposed a hierarchical execution model that can isolate the impact of high-criticality tasks from low-criticality tasks, and that can allow low-criticality executions in the high-criticality behavior by using component boundaries. Our work differs from these approaches in that it can reduce the propagation of high-criticality job behavior for mixed-criticality systems through the isolation of different task groups. Further, our approach enables the system to switch back to low-criticality mode from high-criticality mode once all high-criticality tasks exhibit low-criticality job behavior. To the best of our knowledge, the propagation reduction of high-criticality job behavior has not been investigated in existing MC scheduling solutions.

III. Mixed-Criticality System Model

The system consists of a set τ of implicit-deadline periodic mixed-criticality tasks that need to be scheduled on a multiprocessor platform with p identical, unit-capacity processors. We follow the same task model as in [25]: Each task τ_i is characterized by a tuple $\tau_i = (\zeta_i, C_{\tau_i}(\mathrm{LO}), C_{\tau_i}(\mathrm{HI}), T_{\tau_i})$, where

- $\zeta_i \in \{\mathrm{LO}, \mathrm{HI}\}$ denotes the criticality level of τ_i;
- $C_{\tau_i}(\mathrm{LO})$ and $C_{\tau_i}(\mathrm{HI})$ specify the WCET estimates of τ_i at the low and high criticality levels, respectively, where $C_{\tau_i}(\mathrm{HI}) = C_{\tau_i}(\mathrm{LO})$ if $\zeta_i = \mathrm{LO}$, and $C_{\tau_i}(\mathrm{HI}) \geq C_{\tau_i}(\mathrm{LO})$ if $\zeta_i = \mathrm{HI}$ (that is, the WCET at the high criticality level is more pessimistic than the WCET at the low criticality level); and
- T_{τ_i} denotes the period (relative deadline) of τ_i.

We refer to $C_{\tau_i}(\mathrm{LO})$ and $C_{\tau_i}(\mathrm{HI})$ as the LO-WCET and HI-WCET of τ_i, respectively. We assume that all tasks are initially released simultaneously (at time $t = 0$).

Mixed-criticality schedulability. A task set τ is considered mixed-criticality schedulable under a scheduling policy if the following conditions are met:

- *Low-criticality guarantee*: If all jobs run for at most their LO-WCETs, all high-criticality and low-criticality jobs must complete before their deadlines.
- *High-criticality guarantee*: If all high-criticality jobs run for at most their HI-WCETs and at least one runs for more than its LO-WCET, all high-criticality jobs must complete before their deadlines (whereas low-criticality jobs may be dropped).

We note that although our system model and algorithms consider only two levels of criticality, it should be possible to extend them to more than two criticality levels. One potentially efficient approach is to perform task grouping in a hierarchical fashion according to the criticality levels of tasks, which we plan to explore in future work.

In the next section, we present the algorithm for scheduling mixed-criticality tasks on a (uni)processor. The algorithm for assigning tasks to processors is discussed in Section V. Due to space constraints, we omit the proofs of lemmas and theorems here; they can be found in our technical report [28].

IV. Task-Group-Based Mixed-Criticality Scheduling on Uniprocessors

To provide isolation between high-criticality tasks and to reduce their interference with low-criticality tasks, we associate each high-criticality task with a subset of the low-criticality tasks and encapsulate both in a separate task group. Tasks within each task group are scheduled using a server-based strategy to maximize mixed-criticality schedulability, and the task groups themselves are scheduled on the processor under the Earliest Deadline First (EDF) policy. (In a special case where all tasks assigned to a processor have the same criticality level, they are scheduled as conventional tasks using EDF.)

A. Task groups

A *task group* consists of one high-criticality task and several low-criticality tasks. It is specified as $TG_j = \{\ \tau_1^{LO},\ \tau_2^{LO},\ ...,\ \tau_n^{LO},\ \tau_j^{HI}\ \}$, where the τ_i^{LO} $(1 \le i \le n)$ are the n low-criticality tasks, and τ_j^{HI} is the single high-criticality task.

The period of a task group TG_j is a common divisor of the periods of all tasks in the task group, i.e., $T_{TG_j} = \text{CD}\big(T_{\tau_1^{LO}},\ T_{\tau_2^{LO}},\ ...,\ T_{\tau_n^{LO}},\ T_{\tau_j^{HI}}\big)$, where $T_{\tau_i^{LO}}$ $(1 \le i \le n)$ is the period of the low-criticality task τ_i^{LO} and $T_{\tau_j^{HI}}$ is the period of the high-criticality task τ_j^{HI}. (Since it is common for real-time systems to have harmonic periods in practice, we make this choice to simplify the analysis; however, extensions to other period values are possible.)

All task groups are simultaneously activated at time $t = 0$. By definition, the number of task group budget replenishments during a period of a low-criticality task τ_i^{LO} is given by $l_i^j = T_{\tau_i^{LO}}/T_{TG_j}$ $(1 \le i \le n)$, and the number of task group budget replenishments during a period of the high-criticality task τ_j^{HI} is given by $h_j = T_{\tau_j^{HI}}/T_{TG_j}$.

We denote by B_j the minimum budget (to be computed) that a task group TG_j must receive in each of its periods to ensure all of its tasks meet the mixed-criticality schedulability. In our scheduling strategy, each task group TG_j will be scheduled with other task groups in the system as a conventional periodic task with period (relative deadline) T_{TG_j} and execution time B_j. As usual, we define the utilization of a task group to be its budget divided by its period. For ease of presentation, we refer to a period of a task group as a *TG-period*.

B. Three-phase Scheduling Strategy within Task Groups

Basic ideas. The scheduling within each task group TG_j is done in *rounds*; each round corresponds to a period of the high-criticality task τ_j^{HI}. Within each round, there can be three different scheduling behaviors: one is used while the actual execution time of τ_j^{HI}'s current job has not yet reached its LO-WCET; another is used when τ_j^{HI}'s job *might* reach its LO-WCET during the current period of the task group; and a third is used when τ_j^{HI}'s job has either finished or exceeded its LO-WCET, i.e., switched to high-criticality job mode. This corresponds to three distinct phases, which are illustrated in Figure 1:

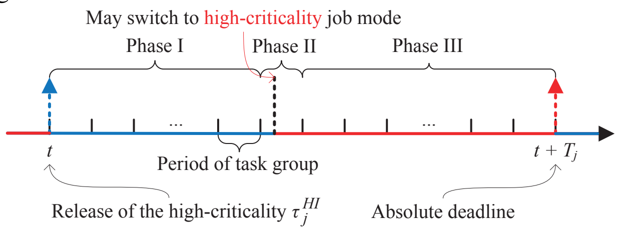

May switch to high-criticality job mode
Phase I
Phase II
Phase III
...
...
t
Period of task group
$t + T_j$
Release of the high-criticality τ_j^{HI}
Absolute deadline

Fig. 1: A scheduling round of a task group.

Phase I (low-criticality mode): In this phase, the current job of τ_j^{HI} has not yet reached its LO-WCET and is thus in low-criticality mode. All jobs in the task group are scheduled to ensure the low-criticality guarantee.

Phase II (switching mode): In this phase, the current job of τ_j^{HI} can potentially exceed its LO-WCET – and thus switch to high-criticality mode – during the current period of the task group. All low-criticality tasks in the task group are suspended to ensure the high-criticality guarantee.

Phase III (low-criticality or high-criticality mode): In this phase, the current job of τ_j^{HI} has either finished or exceeded its LO-WCET, so it is known whether the task group is in low- or high-criticality mode. If the job has indeed finished, all the low-criticality tasks are scheduled; otherwise the group is in high-criticality mode, and the low-criticality tasks continue to be suspended.

When the high-criticality job completes (in phase II or III) and the task group has not yet used up its budget, any suspended low-criticality jobs will continue to be scheduled; however, if they do not complete by the end of the round (i.e., when a new high-criticality job is released) or by their deadlines, whichever is sooner, they will be dropped.

Scheduling mechanism for each round. Recall that the budget of the task group TG_j in each TG-period is B_j and the number of task group budget replenishments in any period of τ_j^{HI} (i.e., in each scheduling round of TG_j) is h_j. Without loss of generality, we suppose that Phase I lasts for k_j TG-periods, i.e., the current job of τ_j^{HI} either switches to the high-criticality mode or has completed its execution in the (k_j+1)th TG-period, where $0 \le k_j \le h_j - 1$. In each scheduling round, tasks are scheduled based on their pre-computed budget values, x_j,

b_i^1 and b_i^2 $(1 \le i \le n)$ as follows. (The method for computing these parameters is presented in Section IV-D.)

Phase I (the first k_j TG-periods): To achieve the low-criticality guarantee, in each of the first k_j TG-periods, tasks are scheduled as follows: if there are some pending low-criticality jobs, the current job of τ_j^{HI} is executed first for a specific period of time $x_j \ge 0$ (at most) and then the low-criticality jobs are executed in ascending order of the task index, where each low-criticality task τ_i^{LO} is allocated a budget of $b_i^1 \ge 0$. If there are no pending low-criticality jobs, the current job of τ_j^{HI} is executed until it completes or the budget of the task group is used up.

The budget parameters should satisfy two conditions: First, the total budget allocated to all tasks in the task group should not exceed the budget B_j of the task group, i.e.,

$$x_j + \sum_{i=1}^{n} b_i^1 \le B_j \tag{1}$$

Second, since the current job of the high-criticality task is always in the low-criticality mode in this phase, the total budget received by τ_j^{HI} in this phase should not exceed its LO-WCET, i.e.,

$$k_j \times x_j \le C_{\tau_j^{HI}}(\text{LO}) \tag{2}$$

Phase II (the (k_j+1)th TG-period): To achieve the high-criticality guarantee, the current job of τ_j^{HI} is executed first with an allocated budget of x_j. If this job completes its execution, the server will schedule the low-criticality tasks in the same manner as in the previous phase. Otherwise, the job will switch from the low-criticality mode to the high-criticality mode and will continue to be executed until it completes or the budget of the task group is used up; in the former case, the server will schedule the low-criticality jobs in the same manner as before until the budget of the task group is used up (some low-criticality jobs may be dropped).

As above, inequality (1) should hold to ensure the budget x_j for τ_j^{HI} and the budget b_i^1 for each low-criticality task τ_i^{LO}. Moreover, since the current job of τ_j^{HI} will switch to the high-criticality mode if it cannot complete with the budget x_j, the budget received by τ_j^{HI} before the mode switch should be at least equal to its LO-WCET. Therefore,

$$(k_j+1) \times x_j \ge C_{\tau_j^{HI}}(\text{LO}) \tag{3}$$

Phase III (the last $h_j - k_j - 1$ TG-periods): To achieve the high-criticality guarantee, the scheduling in each of the last $h_j - k_j - 1$ periods of the task group is as follows: if the current job of τ_j^{HI} has not completed in the considering TG-period, it will be executed until it completes or the budget of the task group is used up. Otherwise, the low-criticality tasks are scheduled in ascending order of the task index, where each task τ_i^{LO} is allocated a budget of $b_i^2 \ge 0$.

Because all budget of the task group will be allocated to the low-criticality tasks in the task group if the task group is in the low-criticality job mode in this last phase, we can allocate more budget to each low-criticality task. In other words,

$$0 \le b_i^1 \le b_i^2 \tag{4}$$

Further, the total budget allocated to all the low-criticality tasks should not exceed the budget of the task group, i.e.,

$$\sum_{i=1}^{n} b_i^2 \le B_j \tag{5}$$

TABLE I: An example task group: task parameters

Task	ζ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	T_i	$U_i(\text{LO})$	$U_i(\text{HI})$
τ_1^{HI}	HI	0.6	2.4	3	0.2	0.8
τ_1^{LO}	LO	0.8	0.8	2	0.4	0.4
τ_2^{LO}	LO	0.6	0.6	3	0.2	0.2

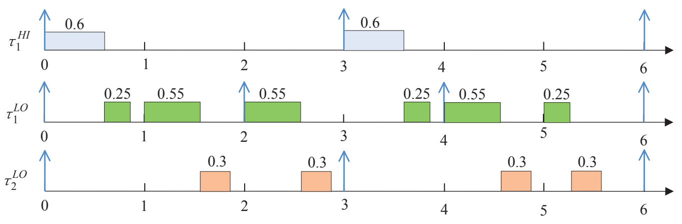


Fig. 2: A schedule of the example in the low-criticality mode.

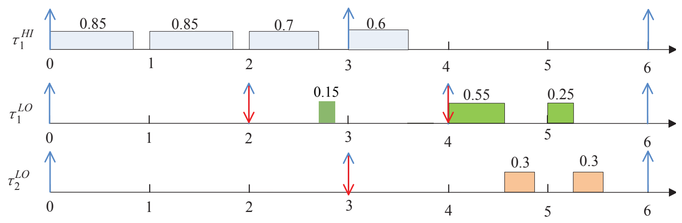


Fig. 3: A schedule of the example in the high-criticality mode.

We illustrate the scheduling mechanism using an example.

Example 1. Consider the task group TG_1 depicted in Table I, with period $T_{TG_1} = GCD(2,3,3) = 1$. Suppose the budget given to TG_1 is $B_1 = 0.85$ and the scheduling parameters are $k_1 = 0$, $x_1 = 0.6$, $b_1^1 = 0.25$, $b_1^2 = 0.55$, $b_2^1 = 0$ and $b_2^2 = 0.3$. Figures 2 and 3 each show two scheduling rounds for TG_1: one in which the high-criticality task always exhibits low-criticality behavior, and one in which it exhibits high-criticality behavior. Note that each round contains $h_j = T_{\tau_1^{HI}} / T_{TG_1} = 3$ TG-periods and consists of only Phase II and Phase III (since $k_1 = 0$).

First, consider Figure 2. In the first TG-period (Phase II), the high-criticality task is first executed for $x_1 = 0.6$ time units and completes; hence, the low-criticality task τ_1^{LO} is then executed for $b_1^1 = 0.25$ time units (τ_2^{LO} is not executed since $b_2^1 = 0$). Since the task group is in low-criticality mode, the low-criticality tasks τ_1^{LO} and τ_2^{LO} are executed in the second and third TG-period (Phase III) for $b_1^2 = 0.55$ and $b_2^2 = 0.3$ time units, respectively. The second scheduling round works similarly, except that in the last TG-period, τ_1^{LO} finishes its execution before using all of its allocated budget.

Now consider Figure 3. Here, τ_1^{HI} exceeds its LO-WCET in the first round, so the task group is switched to high-criticality mode, and both low-criticality tasks are suspended (until τ_1^{HI} completes). In the second round, however, τ_1^{HI} remains in low-criticality mode, so τ_1^{LO} and τ_2^{LO} are executed as usual, based on their allocated budgets. (Note that in the first phase of this round, i.e., time interval [3,4], neither τ_1^{LO} nor τ_2^{LO} is executed. This is because the current job of τ_1^{LO} has been dropped at time 3 when τ_1^{HI} is released, whereas the budget allocated to τ_2^{LO} is $b_2^1 = 0$.) We observe that only some jobs of the low-criticality tasks miss their deadlines: the low-criticality jobs released in the first round are suspended when τ_1^{HI} is in the high-criticality mode, but the low-criticality jobs released in the second round are still schedulable. We can also validate that the task group is mixed-criticality schedulable in the example schedules.

C. Schedulability Analysis

Consider any task group $TG_j = \{ \tau_1^{LO}, \tau_2^{LO}, ..., \tau_n^{LO}, \tau_j^{HI} \}$. We first establish the mixed-criticality schedulability conditions for TG_j under the three-phase scheduling strategy with a given budget B_j and given scheduling parameters k_j, x_j, b_i^1, b_i^2 $(1 \le i \le n)$, where x_j, b_i^1 and b_i^2 are non-negative real numbers and k_j is a non-negative integer. We then derive the schedulability conditions for a set of task groups on a processor.

Recall that $l_i^j = T_{\tau_i^{LO}}/T_{TG_j}$ is the number of task group budget replenishments during each period of τ_i^{LO}, and $h_j = T_{\tau_j^{HI}}/T_{TG_j}$ is the number of task group budget replenishments in each period of the high-criticality task τ_j^{HI} (i.e., in a scheduling round). Lemma IV.1 gives the scheduling conditions for each low-criticality task τ_i^{LO}:

Lemma IV.1. *Each low-criticality task τ_i^{LO} $(1 \le i \le n)$ can be successfully scheduled if all of the inequalities* (1)–(6) *hold, where the inequality* (6) *is given by*

$$N_i^{\mathsf{maxOL}} \times b_i^1 + (l_i^j - N_i^{\mathsf{maxOL}}) \times b_i^2 \ge C_{\tau_i^{LO}}(\mathrm{LO}), \tag{6}$$

where $N_i^{\mathsf{maxOL}} = \lfloor \frac{l_i^j}{h_j} \rfloor \times (k_j + 1) + \min\{l_i^j \mod h_j, k_j + 1\}$ *is the maximum number of TG-periods in a period of τ_i^{LO} that overlap with phases I and II of a scheduling round.*

Example 2. Consider the task group depicted in Table II, with budget $B_1 = 0.3$, $k_1 = 1$, $b_1^1 = 0$ and $b_1^2 = 0.3$. The maximum number of TG-periods that overlap with the first $k_1 + 1 = 2$ TG-periods during a period of the high-criticality task is $\min\{3 \mod 5, 1+1\} = 2$. As demonstrated in Figure 4, there are two overlapping TG-periods for the 1st and 4th releases of τ_1^{LO}, one for the 2nd and 3rd releases, and zero for the 5th release. We can validate that all jobs of τ_1^{LO} can finish before their deadlines.

TABLE II: An example task group: task parameters

Task	ζ_i	C_i(LO)	C_i(HI)	T_i	U_i(LO)	U_i(HI)
τ_1^{HI}	HI	0.6	1.5	5	0.12	0.3
τ_1^{LO}	LO	0.3	0.3	3	0.1	0.1

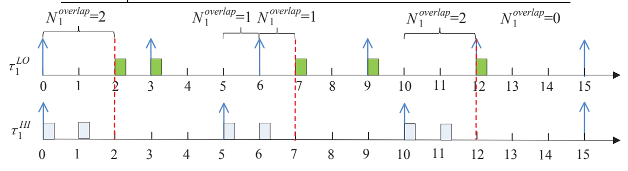


Fig. 4: Number of overlapping TG-periods in Example 2.

Lemma IV.2. *Every job of the high-criticality task τ_j^{HI} receives a budget of at least $C_{\tau_j^{HI}}$(LO) when it exhibits low-criticality behavior and at least $C_{\tau_j^{HI}}$(HI) when it exhibits high-criticality behavior by its deadline if the inequalities* (1), (3) *and the following inequality hold:*

$$k_j \times x_j + (h_j - k_j) \times B_j \ge C_{\tau_j^{HI}}(\mathrm{HI}) \tag{7}$$

The next theorem is derived directly from Lemmas IV.1 and IV.2.

Theorem IV.1. *Suppose each task group $TG_j = \{ \tau_1^{LO}, \tau_2^{LO}, ..., \tau_n^{LO}, \tau_j^{HI} \}$ is guaranteed to receive a budget of B_j in each of its period. Then, TG_j is mixed-criticality schedulable under the three-phase scheduling strategy with scheduling parameters k_j, x_j, b_i^1 and b_i^2 $(1 \le i \le n)$ if all of the inequalities* (1)–(7) *are satisfied.*

Remarks. We note that Theorem IV.1 holds regardless of when exactly the task group is scheduled, because all tasks are guaranteed to receive their allocated budgets as long as the task group as a whole receives a budget of B_j in each period. In addition, the high-criticality behavior of a job of τ_j^{HI} only affects some low-criticality jobs that need to be executed during the period of this τ_j^{HI} job but not the ones that are released after its deadline. Moreover, since each round of a task group corresponds to a job of its high-criticality task, each task group can switch from the high-criticality mode to the low-criticality mode independently of each other, and the system can switch to the low-criticality mode once all high-criticality jobs in the system exhibit low-criticality behavior.

Let S_k be the set of task groups scheduled on a processor k. Recall that each task group $TG_j \in S_k$ is scheduled with other task groups in S_k as a conventional implicit-deadline periodic task with period T_{TG_j} and execution time B_j under EDF. Based on the schedulable utilization bound of EDF [29], all task groups in S_k are schedulable under EDF if their total utilization is less than or equal to 1, i.e., $U_k = \sum_{TG_j \in S_k} B_j/T_{TG_j} \le 1$. In other words, every task group TG_j is guaranteed a budget of B_j in each of its period if $U_k \le 1$. The next theorem follows directly from this observation and Theorem IV.1.

Theorem IV.2. *Let S_k be the set of task groups scheduled on processor π_k (according to some task grouping algorithm). If for all $TG_j = \{ \tau_1^{LO}, \tau_2^{LO}, ..., \tau_n^{LO}, \tau_j^{HI} \}$ in S_k, there exist a non-negative budget value $B_j \in \mathbb{R}_+$ and non-negative scheduling parameters $k_j \in \mathbb{N}$ $(k_j < h_j)$ and $x_j, b_i^1, b_i^2 \in \mathbb{R}_+$ for all $1 \le i \le n$ such that $U_k = \sum_{TG_j \in S_k} B_j/T_{TG_j} \le 1$ and the inequalities* (1)–(7) *are satisfied, then all the tasks assigned to processor k are mixed-criticality schedulable.*

We note that, although Theorem IV.2 only gives a sufficient analysis, our simulation shows that our partitioned scheduling based on task grouping outperforms an existing partitioned algorithm based on EDF-VD, and EDF-VD has been shown to be optimal (if all deadlines are reduced by the same ratio) on single processors for two-level MC systems from the perspective of speedup bound [10].

Next, we present an algorithm based on Mixed Integer Nonlinear Programming (MINLP) for grouping tasks into task groups, as well as for computing the budget and scheduling parameters of each task group.

D. Mixed Integer Nonlinear Programming for Task Grouping

Let $\tau(\pi_k) = \{\tau_1^{LO}, \ldots, \tau_n^{LO}, \tau_1^{HI}, \ldots, \tau_m^{HI}\}$ be the set of tasks that are assigned to the processor π_k based on some tasks-to-processors packing algorithm. The goal of the task grouping is to determine (a) for each high-criticality task τ_j^{HI} $(1 \le j \le m)$ a subset of the low-criticality tasks in $\tau(\pi_k)$ that will be grouped with τ_j^{HI} into the task group TG_j, and (b) for each obtained task group TG_j, the values of the task group's budget B_j and the following scheduling parameters: k_j, the number of TG-periods in scheduling Phase I; x_j, the scheduling budget of the high-criticality task τ_j^{HI}; and b_i^1 and b_i^2, the scheduling budgets for each low-criticality task τ_i in TG_j. Recall that the period T_{TG_j} of TG_j is a common divisor of the periods of all tasks in TG_j; to reduce the computation complexity, we fix T_{TG_j} to be the greatest common divisor of all tasks in $\tau(\pi_k)$. Thus, $l_i^j = T_{\tau_i^{LO}}/T_{TG_j}$ and $h_j = T_{\tau_j^{HI}}/T_{TG_j}$ also become constants.

To this end, we define a MINLP for task grouping based on the schedulability conditions of the task groups. Since the task groups TG_j $(1 \le j \le m)$ are scheduled under EDF as conventional implicit-deadline periodic tasks with periods T_{TG_j} and budgets B_j, the smaller their total utilization $\sum_{j=1}^m B_j/T_{TG_j}$ is, the better their schedulability will be. Therefore, the objective of the proposed MINLP is to minimize the total utilization, $\sum_{j=1}^m B_j/T_{TG_j}$, of the task groups so as to improve their schedulability. The constraints of the MINLP are derived directly from the mixed-criticality schedulability conditions of the task groups (c.f. Theorem IV.2).

To enable a low-criticality task to use the overbooking of multiple high-criticality tasks, we allow each low-criticality task τ_i^{LO} to appear in multiple task groups, i.e., each job of τ_i^{LO} can receive budget from more than one task group and the job is executed whenever it is scheduled in a task group that it belongs. Since all task groups are executed sequentially on a single processor, τ_i^{LO} is never executed at the same time by more than one task group. Hence, if the total budget that τ_i^{LO} is guaranteed to receive in each of its periods from all of its task groups is greater than or equal to its LO-WCET (= HI-WCET), then τ_i^{LO} is schedulable.

Based on the above idea, we use m pairs of non-negative real variables $(b_i^{j,1}, b_i^{j,2})$ for each low-criticality task τ_i^{LO} $(1 \le i \le n, 1 \le j \le m)$ to represent the budgets that τ_i^{LO} receives from task group TG_j in the three scheduling phases of TG_j (instead of using a single pair of variables (b_i^1, b_i^2)). These variables are also used to indicate whether τ_i^{LO} belongs to the task group TG_j: if $b_i^{j,1} = b_i^{j,2} = 0$, then τ_i^{LO} does not belong to TG_j, otherwise it does.

Based on Theorem IV.1, the task set $\tau(\pi_k)$ is mixed-criticality schedulable if all of the inequalities (1)–(7) are satisfied for every task group TG_j. Due to the above encoding of low-criticality tasks, the constraints for each task group TG_j can be obtained by replacing b_i^1 with $b_i^{j,1}$, b_i^2 with $b_i^{j,2}$, N_i^{maxOL} with $N_{i,j}^{\mathsf{maxOL}}$ in the inequalities and rewritting the inequality (7) as $\sum_{j=1}^m \big(b_i^{j,1} \times N_{i,j}^{\mathsf{maxOL}} + b_i^{j,2} \times (l_i^j - N_{i,j}^{\mathsf{maxOL}})\big) \ge C_{\tau_i^{LO}}(\mathrm{LO})$. To reduce the computation complexity, instead of using the inequality (1) (i.e., $x_j + \sum_{i=1}^n b_i^{j,1}$), we set $B_j = x_j + \sum_{i=1}^n b_i^{j,1}$.

By merging all constraints, we obtain the MINLP formulation shown in Figure 5. Constraints (1)–(7) correspond to inequalities (1)–(7), and the remaining constraints are conditions on the variables, as was discussed earlier.

Given an MC task set, if the total task group utilization obtained from solution of the MINLP is more than one, then this task set is unschedulable by the task group based scheduling and will be rejected by the scheduler.

Complexity. If n and m are the number of low- and high-criticality tasks in the task set, respectively, then the MINLP has $2mn + 6m$ constraints, m integer variables and $m + 2mn$ real variables. Hence, the number of variables and constraints is polynomially bounded in the size of the input problem (i.e., the task set size), and it can be solved by a fully polynomial-time approximation scheme. The latter is therefore sufficient to obtain all the scheduling parameters.

Minimize $\sum_{j=1}^m((\sum_{i=1}^n b_i^{j,1} + x_j)/T_{TG_j})$
Subjected to
(1) $B_j = x_j + \sum_{i=1}^n b_i^{j,1}$ $(j = 1, ..., m)$
(2) $k_j \times x_j \le C_{\tau_j^{HI}}(\mathrm{LO})$ $(j = 1, ..., m)$
(3) $(k_j + 1) \times x_j \ge C_{\tau_j^{HI}}(\mathrm{LO})$ $(j = 1, ..., m)$
(4) $0 \le b_i^{j,1} \le b_i^{j,2}$ $(i = 1, ..., n; j = 1, ..., m)$
(5) $\sum_{i=1}^n b_i^{j,2} \le B_j$ $(j = 1, ..., m)$
(6) $\sum_{j=1}^m \big(b_i^{j,1} \times N_{i,j}^{\mathsf{maxOL}} + b_i^{j,2} \times (l_i^j - N_{i,j}^{\mathsf{maxOL}})\big) \ge C_{\tau_i^{LO}}(\mathrm{LO})$ $(i = 1, ..., n; j = 1, ..., m)$
(7) $k_j \times x_j + (h_j - k_j) \times B_j \ge C_{\tau_j^{HI}}(\mathrm{HI})$ $(j = 1, ..., m)$
(8) $0 \le k_j \le h_j - 1$ $(j = 1, ..., m)$
(9) $x_j \in \mathbb{R}_+$ $(j = 1, ..., m)$
(10) $b_i^{j,1} \in \mathbb{R}_+$ $(i = 1, ..., n; j = 1, ..., m)$
(11) $b_i^{j,2} \in \mathbb{R}_+$ $(i = 1, ..., n; j = 1, ..., m)$
(12) $k_j \in \mathbb{N}$ $(j = 1, ..., m)$

Fig. 5: MINLP formulation of the task grouping

Discussions. The MINLP formulation given in Figure 5 can easily be modified to consider other types of constraints as well – e.g., constraints on the number of task groups that each low-criticality task can belong, constraints on the number of low-criticality tasks in each task group, etc. Due to space constraints, we omit them here.

E. EDF-based Two-phased Scheduling Strategy

The proposed budget-driven three-phased scheduling algorithm within a task group (c.f. Section IV-B) may lead to high context switch overhead when there is a large number of low-criticality tasks in a task group. To avoid this drawback, we propose a two-phased variant of the algorithm that schedules low-criticality tasks based on EDF. (This version of the algorithm is used in our evaluation.) Given a task group TG_j with budget B_j and scheduling parameters k_j and x_j, the scheduling during each round (i.e., period of the high-criticality task τ_j^{HI} in the task group) works as follows:

Phase I (the first k_j $(0 \le k_j \le h_j - 1)$ TG-periods): In each TG-period, if there are some pending low-criticality jobs in the task group, the current job of τ_j^{HI} is executed first for a specific period of up to $x_j \ge 0$, and then the low-criticality jobs in the task group are executed under the EDF policy. Otherwise, the current job of τ_j^{HI} is executed until it completes or the budget of the task group expires.

Phase II (the last $h_j - k_j$ period(s) of task groups): In each TG-period, the job of τ_j^{HI} is executed first until it finishes and then the pending low-criticality jobs in the task group are executed under the EDF policy. If the current job of τ_j^{HI} exhibits high-criticality behavior, the low-criticality jobs that cannot finish when their deadlines are reached or when a new job of τ_j^{HI} is released are dropped.

Note that, unlike in the budget-driven three-phased scheduling strategy, the $(k_j + 1)$th TG-period and the last $h_j - k_j$ TG-periods are considered as one phase in the EDF-based two-phased scheduling strategy. The reason for this is that under the three-phased strategy, different budgets (b_i^1 and b_i^2) may be used for each low-criticality task τ_i^{LO} in the $(k_j + 1)$th TG-period and in the last $h_j - k_j - 1$ TG-periods, whereas under the EDF-based scheduling strategy, low-criticality tasks are always scheduled using the EDF policy.

The schedulability under the EDF-based two-phased scheduling strategy is given by the next theorem.

Theorem IV.3. *If a task group $TG_j = \{\ \tau_1^{LO},\ \tau_2^{LO},\ ...,\ \tau_n^{LO},\ \tau_j^{HI}\ \}$ is mixed-criticality schedulable under the three-phased scheduling strategy with budget B_j and scheduling parameters k_j, x_j, b_i^1, b_i^2, then TG_j is also mixed-criticality schedulable under the EDF-based two-phased scheduling strategy with budget B_j and scheduling parameters k_j and x_j.*

Example 3. Consider the task group shown in Table III, with period 1, budget 0.9, $k_1 = 0$ and $x_1 = 0.6$. As illustrated in Figure 6, all low-criticality tasks can be successfully scheduled by the three-phased server with budgets $b_1^1 = 0.25$, $b_1^2 = 0.55$, $b_2^1 = 0.05$ and $b_2^2 = 0.35$. From Figure 7, we can see that an EDF-based schedule for the low-criticality tasks can be obtained by interchanging some executions of τ_1^{LO} and τ_2^{LO}, resulting in a smaller number of context switches.

TABLE III: An example task group: task parameters

Task	ζ_i	C_i(LO)	C_i(HI)	T_i	U_i(LO)	U_i(HI)
τ_1^{HI}	HI	0.6	2.7	3	0.2	0.9
τ_1^{LO}	LO	0.8	0.8	2	0.4	0.4
τ_2^{LO}	LO	0.75	0.75	3	0.25	0.25

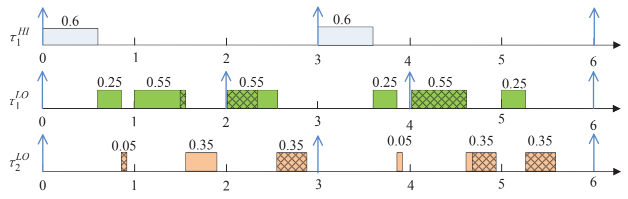


Fig. 6: A budget-driven schedule.

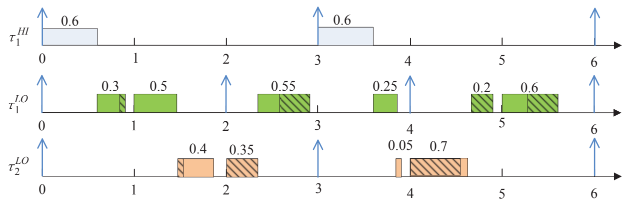


Fig. 7: An EDF-based schedule obtained by transposition.

Due to Theorem IV.3, we can determine the task groups and their scheduling parameters for the EDF-based two-phased scheduling algorithm by solving the MINLP formulated in the previous section. A simple case study that illustrates the end-to-end behavior and the benefits of our task-grouping-based scheduling algorithm compared to an existing algorithm based on virtual deadline [7] is provided in [28].

Remarks. The context switch overhead of EDF-based two-phased scheduling can further be mitigated by considering the slicing number of low-criticality tasks as a constraint in the MINLP formulation and by extending analysis to allow other values for the task group period and/or to determine the time of the task group budget replenishment dynamically. In addition, we can also enable service guarantee for low-criticality tasks using a constraint on the number of task groups that each low-criticality task can belong in the MINLP formulation.

V. Task-Group-Based Partitioned MC Scheduling

In this section, we present a partitioning algorithm for assigning a set of mixed-criticality tasks τ to processors. We consider a platform with p identical, unit-capacity processors, denoted as $\pi = \{\pi_1,\ ...,\ \pi_p\}$. Our algorithm aims to balance the loads across all processors, while selecting high-criticality (low-criticality) tasks to assign based on decreasing high-criticality (low-criticality) utilization. Let $\tau^{HI} = \{\tau_1, ..., \tau_{n_1}\}$ be the set of high-criticality tasks and $\tau^{LO} = \{\tau_{n_1+1}, ..., \tau_n\}$ be the set of low-criticality tasks of the task set τ.

Algorithm 1 is our partitioning algorithm, written in pseudocode. In this algorithm, the set of tasks allocated to each processor π_j is denoted by $\tau(\pi_j)$. The low-criticality utilization $U_{LO}(\tau_i)$ and the high-criticality utilization $U_{HI}(\tau_i)$ of a task τ_i are defined as the task's LO-WCET and HI-WCET divided by its period, respectively. For brevity, we simply refer to the high-criticality utilization of a high-criticality task (and, similarly, the low-criticality utilization of a low-criticality task) as that task's utilization. Further, the cumulative task group utilization for a set of tasks is the total utilization (i.e., budget divided by period) of all task groups of the task set (determined using the MINLP method in Section IV-D).

Algorithm 1 Task group based MC partitioned algorithm.

```
1: $\tau(\pi_j) \leftarrow \emptyset$, for all $j = 1, ..., p$.
2: Sort high-criticality tasks $\tau^{HI}$ in decreasing order of high-criticality
   utilization and then in decreasing order of low-criticality utilization.
3: Sort low-criticality tasks $\tau^{LO}$ in decreasing order of low-criticality utiliza-
   tion and then in increasing order of period.
4: **while** $\tau^{HI} \neq \emptyset$ and $\tau^{LO} \neq \emptyset$ **do**
5:     $\tau_{\text{selected}} \leftarrow$ NIL
6:     **if** $\tau^{LO} \neq \emptyset$ **then**
7:         $\tau_{\text{selected}} \leftarrow$ The first task in $\tau^{LO}$
8:     **if** $\tau^{HI} \neq \emptyset$ **then**
9:         $\tau'_{\text{selected}} \leftarrow$ The first task in $\tau^{HI}$
10:        **if** $\tau_{\text{selected}} =$ NIL **or**
              $(\tau_{\text{selected}} \neq$ NIL **and** $U_{LO}(\tau_{\text{selected}}) < U_{HI}(\tau'_{\text{selected}}))$ **then**
11:            $\tau_{\text{selected}} \leftarrow \tau'_{\text{selected}}$
12:    Remove $\tau_{\text{selected}}$ from its set
13:    Find a processor $\pi_j \in \pi$ that has a minimal cumulative task group
       utilization for the tasks $\tau(\pi_j) \cup \{\tau_{\text{selected}}\}$
14:    **if** the cumulative task group utilization for tasks $\tau(\pi_j) \cup \{\tau_{\text{selected}}\}$
       does not exceed 1 **then**
15:        $\tau(\pi_j) \leftarrow \tau(\pi_j) \cup \{\tau_{\text{selected}}\}$
16:    **else**
17:        **return** FAILURE
18: **return** SUCCESS
```

The algorithm starts by initializing $\tau(\pi_j)$ to null (Line 1) and by sorting high-criticality tasks and low-criticality tasks (Lines 2–3). High-criticality tasks are sorted in decreasing order of high-criticality utilization, and tasks with the same high-criticality utilization are further ordered by decreasing low-criticality utilization (Line 2). Low-criticality tasks are sorted in decreasing order of low-criticality utilization, and tasks with the same low-criticality utilization are further ordered by increasing period (Line 3). Our reason for this sorting is as follows: for high-criticality tasks with the same high-criticality utilization, a task with a higher low-criticality utilization typically has less overbooking; and for low-criticality tasks with the same low-criticality utilization, the task with a smaller period tends to have a lower utilization rate of overbooking, since the task group period is chosen as the common divisor of the periods of the tasks within the task group. However, the algorithm can easily be extended to other sorting methods (which we plan to investigate in future work).

The algorithm then selects the task τ_{selected} that has the largest utilization (Lines 5–11), and removes it from its task set (Line 12). Next, the algorithm selects a processor π_j for this task, such that the cumulative task group utilization for this task along with the existing tasks on this processor is minimal (Line 13); this is done to balance the load across different processors. If the resulting minimum cumulative task group utilization on the selected processor π_j does not exceed one, τ_{selected} is assigned to π_j (Lines 14–15). Otherwise, the algorithm aborts with a failure (Line 17). If every task is successfully allocated to a processor, the algorithm reports success (Line 18).

Based on Theorem IV.2, the resulting task allocation obtained by Algorithm 1 always ensures that tasks on each processor can be successfully scheduled by the task group based scheduling strategy. It should be noted that the criticality levels of the tasks on a processor may be the same; in this case, all tasks on the processor are directly scheduled under EDF.

VI. Evaluation

In this section, we experimentally compare the performance of our proposed scheme, TG-PEDF, with the following partitioned mixed-criticality scheduling algorithms:

- DC-RM: a partitioned scheduling algorithm from [9] that is based on RM priority assignment;
- DC-Audsley: a partitioned scheduling algorithm from [9] that is based on Audsley's optimal priority assignment;
- MC-Partition: a partitioned scheduling algorithm based on EDF-VD (from [10]);
- EY-FF: a straightforward extension of EY [25] for partitioned scheduling with the FF packing strategy from [24];
- MPVD: an extension of EY for partitioned scheduling with the hybrid packing strategy from [24];
- MPVD-HA: MPVD with the heavy low-criticality task aware allocation policy from [24]; and
- MPVD-HA-BF: MPVD-HA with the optimized virtual deadline tuning from [24].

Objectives. We have three main goals for our evaluation: (1) to compare the performance of TG-PEDF in terms of mixed-criticality schedulability to that of the above algorithms; (2) to evaluate how well TG-PEDF can protect low-criticality jobs when a high-criticality task is in high-criticality mode; (3) to evaluate the deadline miss ratios of low-criticality tasks under different frequencies of high-criticality tasks being in high-criticality mode; (4) to evaluate the computational complexity of TG-PEDF; and (5) to evaluate the effect of overbooking by allowing a low-criticality task to appear in multiple task groups in terms of schedulability on a uniprocessor.

For Objective (2), our metric is the *impact ratio* of low-criticality tasks with respect to different percentages of high-criticality tasks being in the high-criticality mode. We define the impact ratio of low-criticality tasks as the fraction of such tasks for which at least one job is dropped or misses its deadline because a high-criticality task is in high-criticality mode. The deadline miss ratio of low-criticality jobs is defined as the fraction of low-criticality jobs that miss their deadlines. Due to space constraints, some additional results are presented in our technical report [28].

A. Workload

For our experiments, we used randomly generated task sets, which we created using the algorithm from [7]. We set the probability *pCriticality* that a task is a high-criticality task to 0.5. The period (relative deadline) T_i of each task τ_i was an integer that was drawn uniformly at random from the interval [5, 100]. The low-criticality WCET $C_i(\text{LO})$ of task τ_i was drawn from $[0.02 \times T_i,\ 0.25 \times T_i]$ and, if τ_i was a high-criticality task, its high-criticality WCET $C_i(\text{HI})$ was drawn from $[2 \times C_i(\text{LO}),\ 4 \times C_i(\text{LO})]$. We define the normalized average utilization $U_{\text{avg}}(\tau)$ of a task set τ to be:

$$U_{\text{avg}}(\tau) = \frac{U_{LO}(\tau) + U_{HI}(\tau)}{2 \times p} \tag{8}$$

where $U_{LO}(\tau)$ and $U_{HI}(\tau)$ are the cumulative low-criticality utilization and the cumulative high-criticality utilization of a task set τ, respectively. p is the number of processors.

Using the above parameters, we generated the tasks one at a time until the following conditions on system utilization were satisfied: (i) $U^*_{\text{avg}} - 0.005 \le U_{\text{avg}}(\tau) \le U^*_{\text{avg}} + 0.005$; (ii) $U_{LO}(\tau) \le p$; and (iii) $U_{HI}(\tau) \le p$, where $U^*_{\text{avg}} \in \{0.6,\ 0.65,\ 0.7,\ 0.75,\ 0.8,\ 0.85,\ 0.9,\ 0.95,\ 0.975\}$ and $p \in \{4, 8, 16\}$. For each U^*_{avg} and each p value, we generated 1,000 task sets for each of the above eight algorithms.

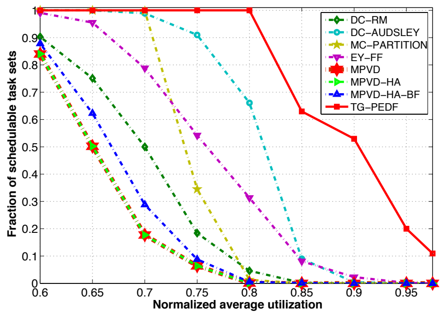
Since the evaluation of the impact ratio of low-criticality tasks and the deadline miss ratio of low-criticality jobs involves expensive simulations, we used for this purpose 100 task sets randomly chosen from the generated task sets with $U^*_{\text{avg}} \in \{0.6, 0.7, 0.8, 0.9\}$ that can be scheduled by TG-PEDF. We ran each simulation for 1,000,000 time units. To evaluate the effect of resource overbooking on the schedulability for different maximum numbers of high-criticality task groups that a low-criticality task can belong to, we set *pCriticality* to 0.8, and we randomly drew the low-criticality task utilization from [0.01, 0.2] to generate more high-criticality tasks.

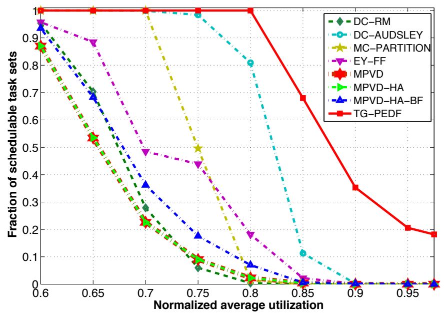
B. Results

Mixed-criticality schedulability. Figures 8a–8c show the fraction of schedulable task sets versus the normalized average utilization of the different algorithms in 4-processor, 8-processor, and 16-processor systems, respectively.

We begin with a few observations about the existing algorithms. First, the results for 8- and 16-processor systems show that the performance of EY-FF decreased as the number of processors increased. This is because the space for EY to tune the virtual deadlines is reduced due to the unbalanced allocation of high-criticality tasks. Second, the results in Figure 8c show that, in 16-processor systems, MPVD, MPVD-HA and MPVD-HA-BF outperformed EY-FF by balancing the demand of high-criticality tasks among different processors. MPVD-HA enhances MPVD by assigning the heavy low-criticality tasks to processors before the high-criticality ones, but the performance of the two algorithms is similar when the task sets contain few low-criticality tasks. MPVD-HA-BF aims to improve schedulability using a new virtual deadline tuning algorithm, and its schedulability is indeed better than MPVD and MPVD-HA in most cases; however, the virtual deadline tuning algorithm does not take the low-criticality tasks into account, so it can affect schedulability even if the demand in low-criticality mode is low. For this reason, MPVD-HA-BF did not outperform EY-FF on the systems with 4 and 8 processors, and it performed worse than MPVD and MPVD-HA on 16-processor systems with higher average utilization.

Our main result is that TG-PEDF *consistently outperformed* all existing partitioned MC scheduling algorithms. The performance gap tends to widen as the average utilization of the


(a) 4-processor systems


(b) 8-processor systems

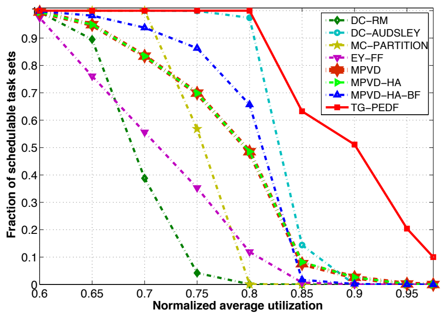

(c) 16-processor systems

Fig. 8: Fraction of mixed-criticality schedulable task sets for different algorithms.

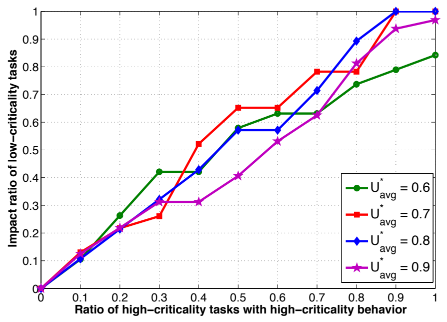


Fig. 11: Impact ratio of low-criticality tasks.

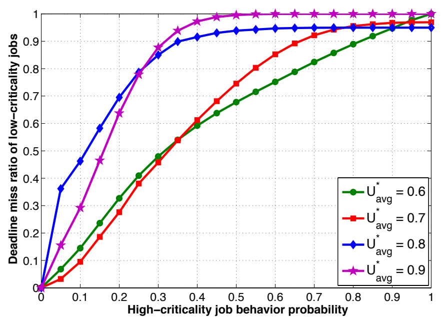


Fig. 12: Deadline miss ratio of low-criticality jobs.

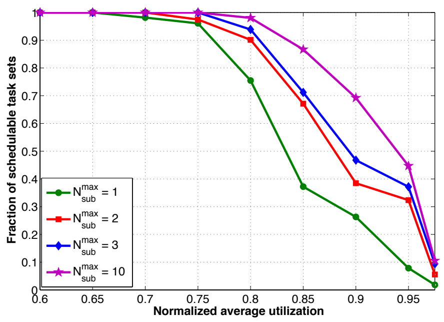


Fig. 13: Effect of overbooking on schedulability.

task sets increases; the reason is that, as the number of tasks increases, there are more and more opportunities for TG-PEDF to find an appropriate task group for each low-criticality task.

Impact of high-criticality behavior on low-criticality tasks. Figure 11 shows the impact ratio of the low-criticality tasks for different percentages of high-criticality tasks with high-criticality behavior on 4-processor systems. The results show that *only a certain fraction of low-criticality tasks are affected*, and that this fraction grows with the fraction of high-criticality tasks that exhibit high-criticality behavior. This is because, with TG-PEDF, the high-criticality behavior of a high-criticality task can only affect low-criticality tasks that are in the same task group with this high-criticality task, but not low-criticality tasks in other task groups.

We make three additional observations. First, the relationship is not exactly linear because some low-criticality tasks are grouped together with more than one high-criticality task, and can thus be affected by any one of these tasks. Second, when all high-criticality tasks exhibit high-criticality behavior (i.e., the ratio of high-criticality tasks with high-criticality behavior is one), there are still some low-criticality tasks that are not affected. The reason for this is that there can be processors with only low-criticality tasks and no high-criticality tasks, and these cannot be influenced by the high-criticality behavior on other processors. Finally, the results are not sensitive to different U^*_{avg} values (the curves corresponding to different system utilization values cross). This is expected because the fraction of low-criticality tasks that are affected depends primarily on the task grouping and the fraction of high-criticality tasks that exhibit high-criticality behavior instead of the system utilization.

Real-time performance of low-criticality tasks. Figure 12 shows the effect of high-criticality job behavior on the deadline miss ratio of low-criticality jobs, using the 4-processor system as an example. Not surprisingly, there are no deadline misses when the probability of high-criticality job behavior is zero: when the system exhibits low-criticality behavior, the task-group-based scheduling can ensure that the system is schedulable. As the probability of high-criticality job behavior rises, the deadline miss ratio rises as well, but only gradually; the reason is, again, that the high-criticality behavior of a job can only affect low-criticality jobs with which it shares a task group, but not low-criticality jobs in other task groups. Additionally, since the isolation between task groups enables the system to switch back to low-criticality mode once all high-criticality tasks exhibit low-criticality job behavior (or, at a task group level, even when only the local high-criticality task exhibits low-criticality behavior), the deadline miss ratio of the low-criticality jobs can be kept low even for long-running systems.

We also observe that the deadline miss ratio curves corresponding to different averge system utilization values cross each other, i.e., a task set with a higher average utilization may have a smaller deadline miss ratio of low-criticality jobs than that of a task set with a lower average utilization. This is not surprising, because the deadline miss ratio of low-criticality jobs can become smaller as the total number of low-criticality jobs increases, which can be the case when the average utilization increases.

It is worth noting that the existing mixed-criticality algorithms drop all of the low-criticality tasks in the system as soon as a high-criticality job exhibits high-criticality behavior, so they provide no real-time performance guarantee for low-criticality tasks in the high-criticality mode.

Effect of resource overbooking on schedulability. Figure 13 illustrates the effect of enabling low-criticality tasks to use the overbooking of multiple high-criticality tasks on the schedulability of the task set on a uniprocessor. Here, N_{sub}^{max} is the maximum number of task groups for each low-criticality task. The results show that the more high-criticality tasks that a low-criticality task is executed with, the better the schedulability. In addition, the schedulability gap tends to widen as the average utilization of the task set increases, which further indicates the effectiveness of using resource overbooking of high-criticality tasks. We note that this can also lead to more high-criticality tasks interfering with a low-criticality task; however, since the low-criticality schedulability guarantee also reflects the schedulability of low-criticality tasks, the resulting benefits outweigh the additional interference.

Computation complexity. We also evaluated the time needed per task set to group the tasks and to compute the task groups' scheduling parameters in our experiments, when using the APMonitor optimization suite [30] for solving the MINLP problem. The results show that the formulated MINLP problem can be solved efficiently (by a fully polynomial-time approximation scheme): it took less than 105 ms across all task sets in our experiments.

VII. Conclusion

We have proposed TG-PEDF, a partitioned scheduling technique for mixed-criticality multiprocessor real-time systems that is based on task grouping. TG-PEDF works by associating each high-criticality task with a subset of the low-criticality tasks and by encapsulating them in separate task groups. Task groups are scheduled with other task groups using the EDF policy, while the tasks within each group are scheduled using a server-based strategy that can ensure the mixed-criticality guarantees. This strategy not only guarantees isolation among high-criticality tasks, it also enables more low-criticality tasks to meet their deadlines. TG-PEDF is also compositional: new tasks can easily be composed with existing ones via task groups. We have presented a schedulability analysis for the system under the proposed scheduling strategy and an MINLP formulation for the task grouping, as well as a packing algorithm that considers criticality in task deployment to optimize resource use. Our evaluation shows that TG-PEDF consistently outperforms existing multiprocessor MC scheduling algorithms in terms of system schedulability, while enabling better timing performance for low-criticality tasks. As a future direction, we plan to implement the proposed technique on a real platform and study its real-time performance and scheduling overhead, as well as to extend it to multiple criticality levels.

Acknowledgments

The authors would like to thank Arvind Easwaran, Nan Guan and Chuancai Gu for their inputs on the implementation of existing algorithms, Andreas Haeberlen for his useful feedbacks on the paper, and Guowei Wu for his support.

This work was supported in part by the NSF grants CNS 1117185, ECCS 1135630 and CNS 1329984, the ONR grant N00014-13-1-0802, the National Natural Science Foundation of China under Grant No. 61173179 and 61202441, and the Fundamental Research Funds for the Central Universities (No. DUT13JS10).

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, 2007.

[2] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *RTSS*, 2009.

[3] H. Li and S. Baruah, "Load-based schedulability analysis of certifiable mixed-criticality systems," in *Proceedings of the tenth ACM international conference on Embedded software*, 2010.

[4] S. K. Baruah, V. Bonifaci, G. DAngelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Algorithms–ESA*, 2011.

[5] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *RTSS*, 2011.

[6] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *ECRTS*, 2012.

[7] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *RTSS*, 2013.

[8] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *CIT*, 2010.

[9] O. R. Kelly, H. Aydin, and B. Zhao, "On partitioned scheduling of fixed-priority mixed-criticality task sets," in *TrustCom*, 2011.

[10] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, 2014.

[11] D. de Niz and L. T. X. Phan, "Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms," in *RTAS*, 2014.

[12] "ISO/DIS 26262 road vehicles - functional safety," http://www.iso.org.

[13] A. Burns and R. Davis, "Mixed criticality systems: A review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[14] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *RTSS*, 2011.

[15] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *ECRTS*, 2012.

[16] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks," in *RTAS*, 2012.

[17] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *DATE*, 2013.

[18] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low criticality tasks in mixed criticality system," in *WMC*, 2013.

[19] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *WMC, RTSS*, 2013.

[20] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," Technical Report 350, ETH Zurich, Tech. Rep., 2013.

[21] X. Gu, A. Easwaran, K. M. Phan, and I. Shin, "Compositional mixed-criticality scheduling," in *CRTS*, 2014.

[22] R. M. Pathan, "Schedulability analysis of mixed-criticality systems on multiprocessors," in *ECRTS*, 2012.

[23] S. K. Baruah, "Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors," *IEEE Transactions on Computers*, 2004.

[24] C. Gu, N. Guan, Q. Deng, and W. Yi, "Partitioned mixed-criticality scheduling on multiprocessor platforms," in *DATE*. IEEE, 2014.

[25] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-time systems*, 2014.

[26] H. Su, D. Zhu, and D. Mossé, "Scheduling algorithms for elastic mixed-criticality tasks in multicore systems," 2013.

[27] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," in *ASP-DAC*, 2014.

[28] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," Technical Report, University of Pennsylvania, Tech. Rep., 2015. [Online]. Available: http://www.cis.upenn.edu/~linhphan/papers/phan-ecrts15-tr.pdf

[29] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, 1973.

[30] J. D. Hedengren. (2014) Apmonitoroptimization suite. [Online]. Available: http://apmonitor.com