Criticality-Aware Partitioning for Multicore Mixed-Criticality Systems

Jian-Jun Han*, Xin Tao*, Dakai Zhu† and Hakan Aydin‡
*School of Computer Science and Technology, Huazhong University of Science and Technology, China
†Department of Computer Science, The University of Texas at San Antonio, USA
‡Department of Computer Science, George Mason University, USA
Email: {jasonhan@hust.edu.cn, xintao@hust.edu.cn, dakai.zhu@utsa.edu, aydin@cs.gmu.edu}

***Abstract*—The scheduling for mixed-criticality (MC) systems, where multiple activities have different certification requirements and thus different criticality on a shared hardware platform, has recently become an important research focus. In this work, considering that multicore processors have emerged as the *de facto* platform for modern embedded systems, we propose a novel and efficient *criticality-aware task partitioning algorithm (CA-TPA)* for a set of periodic MC tasks running on multicore systems. We employ the state-of-the art EDF-VD scheduler on each core. Our work is based on the observation that the utilizations of MC tasks at different criticality levels can have quite large variations; hence when a task is allocated, its *utilization contribution* on different processors may vary by large margins and this can significantly affect the schedulability of tasks. During partitioning, CA-TPA sorts the tasks according to their utilization contributions on individual processors. Several heuristics are investigated to balance the workload on processors with the objective of improving the schedulability of tasks under CA-TPA. The simulation results show that our proposed CA-TPA scheme is effective, giving much higher schedulability ratios when compared to the classical partitioning schemes.**

***Keywords*- Mixed-Criticality Systems; Real-Time Embedded Systems; Scheduling; Partitioning;**

I. Introduction

In modern real-time embedded systems, the ever-increasing complexity demands the integration of multiple functionalities on a common computing platform, in view of the space, power, and cost constraints. For example, the IMA (Integrated Modular Avionics) initiative provides guidelines for hosting multiple avionics components on shared systems to tackle increased complexity and cost [1]. In such integrated systems, various application activities with diverse certification requirements and different levels of importance (*criticality*) may coexist. For instance, in the avionics certification standard DO-178 B/C, there are five design assurance levels, ranging from A to E, categorized according to the degree of damage caused by run-time failures [21]. To incorporate different certification requirements and enable efficient management of application activities, the concept of *Mixed-Criticality (MC)* real-time systems has been proposed in [32]. Since that seminal work, numerous MC scheduling algorithms have been proposed for various task models and system settings [3], [4], [7], [12], [15], [17], [18], [20], [22], [31], [33], [35].

Unlike traditional real-time systems where the worst-case requirements of all tasks must be satisfied, the execution of an MC task will depend on its criticality level and the system's current *operation mode*. In fact, the basic principle of the mixed-criticality model is to have two or more criticality levels, where tasks at the k (> 1) criticality level normally have k different worst-case execution requirements [32]. Moreover, the $(k-1)^{th}$ level execution requirement of a task is no greater than its k^{th} level execution requirement. An MC system starts its execution at the lowest operation mode (i.e., level 1). As soon as a high criticality task with criticality level higher than i takes more time than its level-i execution requirement without indicating its completion at runtime, the system makes a *mode transition* to the $(i+1)^{st}$-level operation mode where only the tasks at criticality level $(i+1)$ or higher can be executed. Specifically, all current level-i tasks are discarded and no future level-i tasks are released to accommodate the increased execution requirements of high criticality tasks until the system goes back to level-1 operation mode when it becomes idle [32].

Following the above MC scheduling principle, most of the existing studies have focused on independent MC tasks running on single-processor systems. The algorithms proposed for uniprocessor MC scheduling can be generally classified into two categories: *Fixed-Priority* based scheduling algorithms (FP) and *Earliest-Deadline-First (EDF)* based scheduling algorithms. For FP, the common approach has been to resort to the *Response Time Analysis (RTA)* technique [7], [11], [18], [33], [35]. An alternative to the fixed-priority MC scheduling is *slack scheduling* that was first studied in [13], where low-criticality jobs can exploit slack generated by the high-criticality jobs when they only use their low-criticality execution budgets [14], [26], [29]. Moreover, by exploiting the *period transformation* technique [30], MC tasks can be split into two or more parts with each part having proportionally reduced timing parameters (e.g., execution time and period) [6], [18]. On the other hand, the dynamic priority MC scheduling based on EDF was first studied by Baruah and Vestal in [8]. Some recent analysis for deadline-based scheduling of MC tasks can be found in [12], [15], [17], in addition to the studies with reservation-based approach [25] and elastic task models [31].

The most notable EDF-based scheduling scheme for MC tasks is the recently proposed *EDF with virtual deadlines (EDF-VD)* algorithm [3], [4], [5]. The basic idea of EDF-VD is to have *virtual* (and *smaller*) deadlines (and thus higher priorities) for high-criticality tasks when the system operates at

low-criticality operation mode in order to improve the system schedulability. Several recent studies have been reported based on *demand-bound functions (DBF)* to further improve the analysis and virtual deadline assignment for MC tasks under EDF-VD on single processor systems [15], [16], [17], [34].

As multicore processors have become the *de facto* implementation platform for modern systems, there is a renewed interest in developing scheduling algorithms for multicore/multiprocessor systems. There are two traditional approaches to the multiprocessor scheduling problem: *partitioned* and *global* scheduling. A recent empirical study on multiprocessor scheduling shows that partitioned scheduling generally outperforms global scheduling in terms of the feasibility performance [9]; this is primarily due to the fact that the private processor ready queues and avoiding migration at runtime typically result in lower online overheads.

The first work that addresses mixed-criticality scheduling in the context of multiprocessor systems was reported in [2]. Based on global scheduling, a few studies have investigated the schedulability analysis for MC tasks running on multiprocessor systems [19], [24], [27]. For partitioned scheduling based studies, tasks are first sorted either according to their utilizations [20], [23], [28] or their criticality levels [22]. In general, the existing partitioned based MC scheduling usually adopts the traditional well-known heuristics and consider task/system *utilizations*, such as First-Fit Decreasing (FFD), Best-Fit Decreasing (BFD), and Worst-Fit Decreasing (WFD). It has been shown that a hybrid partitioned scheme, which allocates high-criticality tasks using WFD and low-criticality tasks using FFD, can effectively improve system schedulability [22], [28] when compared to the schemes that consider only either utilization or criticality. To achieve better schedulability, a partitioning scheme that exploits the DBF-based schedulability test (with a much higher complexity) was reported in [20]. A comprehensive review of mixed-criticality scheduling algorithms can be found in [10].

The existing partitioned based MC scheduling schemes normally consider only a task's utilization at its *highest criticality level* (i.e., its *maximum* utilization) [22], which usually results in pessimistic estimates of available system utilization and thus degraded system schedulability. However, from the schedulability conditions of the EDF-VD scheduler [3], [4], we can see that, in addition to its maximum utilization, an MC task's utilizations at other criticality levels play also an important role.

Based on this observation, in this work, we propose a *Criticality-Aware Task Partitioning Algorithm (CA-TPA)* for a set of mixed-criticality tasks running on a multicore system using the partitioned EDF-VD approach. Specifically, considering the significant variations of a task's utilizations at different criticality levels, we define the *utilization contribution* of an MC task at a given criticality level, which is exploited to guide the allocation of tasks to cores. CA-TPA adopts a probe-based approach to ensure that, when allocating an MC task, the overall system utilization increases by the smallest amount. Moreover, a *workload imbalance factor* is introduced with the aim of obtaining balanced workload partitioning on cores, in order to improve system schedulability. Hence, our work differs from the existing studies that rely on only the maximum utilizations of MC tasks at their corresponding criticality levels.

The remainder of this paper is organized as follows. Section II presents system models and reviews the schedulability conditions of the EDF-VD algorithm. Our novel CA-TPA scheme is presented in Section III. Simulation results are discussed in Section IV and Section V concludes the paper.

II. System Models and Preliminaries

In this section, we first present the system and task models. The schedulability conditions for MC tasks under EDF-VD on uniprocessor systems are briefly reviewed, followed by the description of the problem to be addressed in this work.

A. System and Task Models

We consider a multicore system that consists of M homogeneous processing cores, which are denoted as $\{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$ and have identical functions and capabilities. There are $K > 1$ criticality levels for application tasks running on the system, where the system starts its operations at level-1 criticality. There are a set of N mixed-criticality (MC) tasks $\Psi = \{\tau_1, \ldots, \tau_N\}$.

An MC task τ_i is characterized by three parameters: $\tau_i = \{\mathbf{C}_i, p_i, \ell_i\}$. Here, ℓ_i $(1 \leq \ell_i \leq K)$ indicates τ_i's criticality level (i.e., its *own* criticality) and p_i is its period. We consider *implicit-deadline* periodic tasks: p_i represents task τ_i's period as well as its relative deadline. The vector $\mathbf{C}_i =< c_i(1), \ldots, c_i(\ell_i) >$ represents the worst-case execution times (WCETs) of task τ_i at each criticality level, where the WCET at a higher level is generally no less than that at a lower level (i.e., $c_i(1) < c_i(2) < \ldots < c_i(\ell_i)$). Assuming that the first instance of each task arrives at time 0, the j^{th} task instance (or job) of task τ_i arrives at time $r_i^j = (j-1) \cdot p_i$ and must complete its execution by its absolute deadline $d_i^j = j \cdot p_i$. By focusing on partitioned scheduling, we assume that the subset of tasks allocated to core $\mathcal{P}_m$ is denoted as Ψ_m $(m = 1, \ldots, M)$. A partition of tasks to cores is represented by $\Gamma = \{\Psi_1, \ldots, \Psi_M\}$, where $\Psi = \bigcup_{m=1}^{M} \Psi_m$.

We assume that the *adaptive mixed criticality (AMC)* scheme is adopted to manage the executions of the MC tasks at runtime on each core, in conjunction with the EDF-VD algorithm [7]. Specifically, it is assumed that the system provides run-time support to monitor the execution of individual jobs. Under the AMC scheme, if the current operation mode of a core $\mathcal{P}_m$ is at level k $(1 \leq k \leq K)$ and a task τ_i $(\in \Psi_m)$ runs for a duration longer than its k-level WCET $c_i(k)$ $(k < \ell_i)$ without signaling completion, a *mode-switch* occurs on core $\mathcal{P}_m$ and its operation mode changes to level-$(k+1)$. At that moment, all tasks in Ψ_m with their criticality levels no more than k will be *dropped* to accommodate the additional execution requirements for tasks in Ψ_m with criticality levels k or higher [7].

B. Schedulability Conditions of EDF-VD

The EDF-VD scheduler for MC tasks running on a single processor was first proposed by Baruah et al. in [3], [5]. The key idea is to have *virtual* (and *smaller*) deadlines for high-criticality tasks to improve system schedulability. Specifically, based on EDF scheduling, each high-criticality task is assigned a virtual deadline (which is smaller than its original deadline) and thus has higher priority when the system runs at low-criticality mode. This helps high-criticality tasks complete their low-criticality executions earlier. Hence, when the system switches to high-criticality mode, the high-criticality execution requirements of high-criticality tasks can be ensured by restoring their original deadlines and discarding low-criticality tasks. In this paper, once tasks are partitioned to cores, we assume that EDF-VD is adopted on each core $\mathcal{P}_m$ to schedule its subset of MC tasks Ψ_m.

Before presenting our partitioned scheme, we first review the schedulability conditions for a set of MC tasks running on a single processor under the EDF-VD scheduler [3], [4], [5]. For the ease of presentation and discussion, some notations are defined as follows:

- $u_i(k) = \frac{c_i(k)}{p_i}$: the utilization of task τ_i at level k ($\leq \ell_i$);
- $\mathcal{L}_j = \{\tau_i \mid \ell_i = j\}$: the tasks at criticality level j;
- $U_j(k)$: the *level-k utilization* of tasks with criticality level j, which is defined as:

$$U_j(k) = \sum_{\forall \tau_i \in \mathcal{L}_j} u_i(k) \tag{1}$$

- $U(k)$: the *total level-k utilization* of tasks with criticality level k or higher. From Equation (1), we have:

$$U(k) = \sum_{j=k}^{K} U_j(k) \tag{2}$$

- $U_j^{\Psi_m}(k)$: the level-k utilization of tasks on core $\mathcal{P}_m$ with criticality level j. We have:

$$U_j^{\Psi_m}(k) = \sum_{\forall \tau_i \in (\Psi_m \bigcap \mathcal{L}_j)} u_i(k) \tag{3}$$

Based on the above definitions, a simple *sufficient* schedulability condition for the tasks that are allocated to core $\mathcal{P}_m$ under the EDF-VD scheduler can be given by the following equation (from Theorem 3.4 in [4]):

$$\sum_{k=1}^{K} U_k^{\Psi_m}(k) \leq 1 \tag{4}$$

In essence, the condition says that, if core $\mathcal{P}_m$ can accommodate the maximum utilization demands of all its tasks at their own criticality levels, the tasks are schedulable under EDF-VD (which actually reduces to EDF since no virtual deadline is needed for any task [4]). From the above equation, we can easily get the following proposition:

Proposition 1. *For a set of N tasks with K criticality levels running on a system with M cores, a given partition* $\Gamma = \{\Psi_1, \ldots, \Psi_M\}$ *is* feasible *if Equation (4) holds for all the cores* $\mathcal{P}_m$ $m = 1, \ldots, M$.

Although Proposition 1 provides a simple utilization-based *sufficient* schedulability condition for the partitioned EDF-VD scheduling, the condition represented by Equation (4) is rather *pessimistic* in that only the worst-case utilization demands of MC tasks on each core are considered. By incorporating tasks' utilizations at different (and lower) criticality levels, an improved *sufficient* schedulability condition for MC tasks under the EDF-VD scheduler was developed in [5], which can be summarized in the following theorem.

Theorem 1 (**Theorem** 3 **in [5]**). *The MC tasks allocated to core* $\mathcal{P}_m$ *are schedulable by the EDF-VD algorithm if, for* **any** $k = 1, \ldots, K-1$, *the condition below holds:*

$$\underbrace{\sum_{i=k}^{K-1} U_i^{\Psi_m}(i) + \min\left\{U_K^{\Psi_m}(K), \frac{U_K^{\Psi_m}(K-1)}{1 - \frac{U_K^{\Psi_m}(K)}{\prod_{j=1}^{k}(1-\lambda_j)}}\right\}}_{\mu^{\Psi_m}(k)} \leq \underbrace{\prod_{j=1}^{k}(1-\lambda_j)}_{\theta^{\Psi_m}(k)}, \tag{5}$$

where $\lambda_1 = 0$ *and for* $j > 1$,

$$\lambda_j = \frac{\sum_{x=j}^{K} U_x^{\Psi_m}(j-1)}{\prod_{x=1}^{j-1}(1-\lambda_x)} \Bigg/ \left(1 - \frac{U_{j-1}^{\Psi_m}(j-1)}{\prod_{x=1}^{j-1}(1-\lambda_x)}\right) \tag{6}$$

In Equation (6), λ ($0 < \lambda < 1$) is defined as a *reduction factor* for the virtual deadlines for high-criticality tasks to allow them to complete their low-criticality workload as soon as possible. In particular, suppose that the inequality (5) holds for a specific k, but does not hold for any smaller value. The system can operate as follows: when the system runs at level l ($\leq k-1$), we discard all jobs of tasks in $\mathcal{L}_1, \ldots, \mathcal{L}_{l-1}$. For any task $\tau_i \in \mathcal{L}_j$ ($j = l+1, \ldots, K$), its virtual (relative) deadline is set as $p_i(l) = \lambda_{l+1} \cdot p_i(l)$ where $p_i(1) = p_i$. As soon as the system switches to criticality level k, we cancel all jobs of tasks in $\mathcal{L}_1, \ldots, \mathcal{L}_{k-1}$; then we reset the original deadlines of task $\tau_i \in \mathcal{L}_j$ ($j = k, \ldots, K-1$). The deadlines of tasks in $\mathcal{L}_K$ can be set accordingly based on the values of the *min* term on the left side of Inequality (5). In this way, all tasks can meet their timing constraints accordingly. The detailed mechanism for setting tasks' virtual deadlines and formal feasibility proofs can be found in [5].

Based on Theorem 1, we can easily obtain the following proposition regarding the schedulability of MC tasks under the partitioned EDF-VD scheduling:

Proposition 2. *For a set of N tasks with K criticality levels running on a system with M cores, a given partition* $\Gamma = \{\Psi_1, \ldots, \Psi_M\}$ *is* feasible *if, for every core* $\mathcal{P}_m$ $m = 1, \ldots, M$, *there exists at least one* $k = 1, \ldots, K-1$, *such that Inequality (5) holds.*

As a special case, when a system has only two criticality levels (i.e., $K = 2$), which is denoted as a *dual-criticality*

system, the task set is schedulable under partitioned EDF-VD if, for every core $\mathcal{P}_m$ ($m = 1, \ldots, M$), the following holds [3]:

$$U_1^{\Psi_m}(1) + \min\left\{U_2^{\Psi_m}(2), \frac{U_2^{\Psi_m}(1)}{1 - U_2^{\Psi_m}(2)}\right\} \leq 1. \quad (7)$$

From Theorem 1, once tasks are partitioned to cores, we can see that there are $(K-1)$ conditions for each core in a K-level MC system. However, only one condition is required to hold on each core to satisfy the schedulability condition for the partitioned EDF-VD scheduling. Hence, we can define the *available utilization for condition-k* on core $\mathcal{P}_m$ as:

$$A^{\Psi_m}(k) = \theta^{\Psi_m}(k) - \mu^{\Psi_m}(k). \quad (8)$$

Above, $\theta^{\Psi_m}(k)$ and $\mu^{\Psi_m}(k)$ are defined as the terms on the left- and right-hand side of the Inequality (5), respectively.

The *core utilization* of $\mathcal{P}_m$ can be further defined as:

$$U^{\Psi_m} = \begin{cases} \infty, & \text{if } \forall A^{\Psi_m}(k) < 0,\ k = 1, \ldots, K-1 \quad (9a) \\ \max\limits_{\forall A^{\Psi_m}(k) \geq 0} \{1 - A^{\Psi_m}(k)\}, \text{otherwise.} & (9b) \end{cases}$$

Similarly, the *system utilization* U^{sys} and *average core utilization* U^{avg} can be defined, respectively, as follows:

$$U^{sys} = \max\{U^{\Psi_m} | m = 1, \ldots, M\}, \quad (10)$$

$$U^{avg} = \frac{\sum_{m=1}^{M} U^{\Psi_m}}{M}. \quad (11)$$

III. Criticality-Aware Task Partitioning

The problem that we address in this paper can be defined as: **The MC$^K(N, M)$ Partition Problem:** *For a set of N mixed-criticality tasks with K criticality levels running on a system with M homogeneous cores, find a task-to-core partitioning* Γ *where tasks on each core can be scheduled under the EDF-VD scheduler.*

Note that, when $K = 1$, the MC$^K(N, M)$ partition problem can be reduced to the classical real-time task partitioning problem, which is a well-known NP-hard problem. Therefore, the MC$^K(N, M)$ partition problem is NP-hard as well. Hence, in what follows, we focus on efficient partitioning heuristics.

In general, partitioning tasks to cores involves two main steps: a) sort the tasks according to certain criteria, and, b) find an appropriate core for each task by processing the tasks according to the order determined in a). Instead of exploiting the maximum utilizations of tasks and the simple (but quite pessimistic) schedulability condition in Equation (4), we focus on the improved schedulability condition as represented in Inequality (5).

In fact, instead of solely depending on tasks' maximum utilizations at their corresponding criticality levels, the schedulability conditions in Inequality (5) rely on tasks' utilizations at all criticality levels. Considering the potential large variations of tasks' utilizations at various criticality levels, it is crucial to take such variations into consideration when designing the two essential steps of partitioning heuristics.

A. Task Ordering and Utilization Contribution

To incorporate utilizations of tasks at different levels in the first step, we present the concept of *utilization contribution* of tasks. Specifically, a task τ_i's utilization contribution at level-k criticality is defined as:

$$\mathscr{C}_i(k) = \frac{u_i(k)}{U(k)}; \quad k = 1, \ldots, \ell_i. \quad (12)$$

where $U(k)$ is the total level-k utilization of all tasks with criticality levels k or higher (see Equation (2)). The utilization contribution of τ_i to the system (by considering all valid criticality levels) can be further defined as:

$$\mathscr{C}_i = \max\{\mathscr{C}_i(k) \mid k = 1, \ldots, \ell_i\}. \quad (13)$$

From the above definitions, we can see that the utilization contribution of a task essentially represents its largest weight in system utilizations among all its valid criticality levels. Hence, following the same principle as in the classical partitioning heuristics that order tasks based on their utilizations (e.g., Best-Fit-Decreasing -BFD- and Worst-Fit-Decreasing -WFD- heuristics), in this work, we sort tasks (i.e., determine the ordering priority of tasks) based on their utilization contributions in the first step before allocating them to processing cores. For this purpose, we define a relational operator $\triangleright$ with the following ordering priority rules:

- If task τ_i has larger utilization contribution than task τ_j, we say that τ_i has higher ordering priority than τ_j and this is denoted as $\tau_i \triangleright \tau_j$;
- If the two tasks have the same utilization contribution, the tie is broken in favor of the task with higher criticality level. That is, if $\mathscr{C}_i = \mathscr{C}_j \wedge \ell_i > \ell_j$, we have $\tau_i \triangleright \tau_j$;
- If there is still a tie, the task with smaller index is assigned higher ordering priority: $\tau_i \triangleright \tau_j$ if $i < j \wedge \mathscr{C}_i = \mathscr{C}_j \wedge \ell_i = \ell_j$.

Note that, in this step, all tasks in Ψ are considered together to calculate each task's utilization contribution. Based on the above ordering rules, each task will be assigned a *unique* ordering priority. The tasks are sorted according to their ordering priority before the partitioning step.

B. Target Core Selection and Utilization Increment

To generalize the ideas in the above example, the key point in our core selection heuristic is to take the utilization variations of tasks at different criticality levels into consideration. Specifically, from Condition (5), we can see that, the utilizations of tasks at *all* valid criticality levels can affect their schedulability on a certain core. Moreover, since each core has a distinct subset of MC tasks, the utilizations of cores at each criticality level can vary (see Equation (3)). Thus, the allocation of a task to different cores can result in quite different schedulability results and large variations in resulting average core utilizations, which is in sharp contrast to the scheduling of non-MC tasks.

To quantify the impact of allocating a task τ_i to core $\mathcal{P}_m$, the *increment of core utilization* on $\mathcal{P}_m$ is defined as:

$$\Delta^{\Psi_m \cup \{\tau_i\}} = \mathscr{U}^{\Psi_m \cup \{\tau_i\}} - U^{\Psi_m}. \quad (14)$$

Algorithm 1: Outline of CA-TPA

Input: Ψ (the task set); M (the number of cores);
Output: A feasible partitioning Γ or FAILURE;

1 Initiate $\Gamma = \{\Psi_m\}$, where $\Psi_m = \emptyset$ $(m = 1, \ldots, M)$;
2 Sort tasks in Ψ based on their utilization contributions;
3 **for** (each $\tau_i \in \Psi$) **do**
4 $\Delta = \infty$;
5 **for** (each $\mathcal{P}_m$) **do**
6 Calculate $\mathscr{U}^{\Psi_m \cup \{\tau_i\}}$ from Equation (15);
7 Calculate $\Delta^{\Psi_m \cup \{\tau_i\}}$ from Equation (14);
8 **if** *($\{\Psi_m \cup \{\tau_i\}\}$ is feasible and $\Delta^{\Psi_m \cup \{\tau_i\}} < \Delta$)* **then**
9 $\Delta = \Delta^{\Psi_m \cup \{\tau_i\}}$; $x = m$;
10 **end**
11 **end**
12 **if** *($\Delta == \infty$)* **then**
13 $\Gamma = \emptyset$; //not feasible on any core;
14 Return FAILURE;
15 **end**
16 $\Psi_x = \Psi_x \cup \{\tau_i\}$; //allocate τ_i to $\mathcal{P}_x$;
17 Update $U^{\Psi_x}(k)(k = 1, \ldots, K)$ and U^{Ψ_x};
18 **end**
19 Return Γ;

Here the *new* core utilization $\mathscr{U}^{\Psi_m \cup \{\tau_i\}}$ of core $\mathcal{P}_m$, by assuming that τ_i is allocated to the core, can be found as:

$$\mathscr{U}^{\Psi_m \cup \{\tau_i\}} = \begin{cases} \infty, & \text{if } \forall A^{\Psi_m \cup \{\tau_i\}}(k) < 0,\ k = 1, \ldots, K-1 \quad (15a) \\ \max\limits_{\forall A^{\Psi_m \cup \{\tau_i\}}(k) \geq 0} \{1 - A^{\Psi_m \cup \{\tau_i\}}(k)\}, & \text{otherwise} \quad (15b) \end{cases}$$

where $A^{\Psi_m \cup \{\tau_i\}}(k) = \theta^{\Psi_m \cup \{\tau_i\}}(k) - \mu^{\Psi_m \cup \{\tau_i\}}(k)$. Note that, if $\mathscr{U}^{\Psi_m \cup \{\tau_i\}} = \infty$, that indicates that task τ_i cannot be feasibly allocated to core $\mathcal{P}_m$ based on the condition in Inequality (5).

C. Criticality-Aware Task Partitioning Algorithm (CA-TPA)

Based on the above discussions, we adopt a probe-based approach when allocating a task to cores. Specifically, by checking all cores in the system, a task τ_i will be allocated to the core $\mathcal{P}_m$ that has the minimum increment for its core utilization, should τ_i be allocated to $\mathcal{P}_m$. That is, $\Delta^{\Psi_m \cup \{\tau_i\}} = \min\{\Delta^{\Psi_x \cup \{\tau_i\}} | x = 1, \ldots, M\}$. If more than one cores have the same minimum core utilization increment, the tie is broken by allocating the task to the core with smaller index.

The outline of our CA-TPA algorithm is summarized in Algorithm 1. First, the task-to-core partition Γ and the subset of tasks for each core are initialized (line 1). Then, all tasks are sorted according to the decreasing order of their utilization contributions (line 2). For each task in the above order, CA-TPA probes all cores by calculating its *new* core utilization and utilization increment assuming that task τ_i is allocated to it (lines 5 to 11). For all cores that can feasibly accommodate τ_i under the EDF-VD scheduler according to the schedulability condition in Inequality (5), the core $\mathcal{P}_x$ with the smallest utilization increment is identified (line 9). If τ_i cannot be feasibly allocated to any core, CA-TPA fails to obtain a feasible partitioning and quits (lines 12 – 14). Otherwise, task τ_i is allocated to the target core $\mathcal{P}_x$ by updating its subset of tasks Ψ_x and related parameters (lines 16 and 17). Once all tasks are successfully allocated to cores, the feasible partition Γ is found and returned (line 19).

An Example: To further illustrate how CA-TPA works, we consider a dual-criticality system with two cores and five MC tasks. The timing parameters of tasks are given in Table I.

TABLE I
THE TIMING PARAMETERS OF TASKS

	$c_i(1)$	$c_i(2)$	p_i	ℓ_i	$u_i(1)$	$u_i(2)$	$\mathscr{C}_i(1)$	$\mathscr{C}_i(2)$	$\mathscr{C}_i$
τ_1	24	-	61	1	0.394	-	0.257	-	0.257
τ_2	15	28	86	2	0.175	0.326	0.114	0.340	0.340
τ_3	30	-	96	1	0.313	-	0.204	-	0.204
τ_4	23	43	68	2	0.339	0.633	0.221	0.661	0.661
τ_5	20	-	63	1	0.318	-	0.207	-	0.207

For comparison, we first show the task-to-core mapping under the First-Fit-Decreasing (FFD) heuristic. The tasks are sorted in decreasing order of their maximum utilizations at their corresponding criticality levels (i.e., $u_i(\ell_i)$ for task τ_i), giving the order of τ_4, τ_1, τ_2, τ_5, τ_3. In FFD, a task is allocated to the first core that can feasibly accommodate it based on the pessimistic schedulability test in Equation (4). The results for core utilizations after mapping the first four tasks to cores are summarized in Table II. Here, we can see that FFD fails to allocate task τ_3 as the schedulability test in Equation (4) fails (actually, even the improved schedulability test in Equation (7) fails) on both cores.

TABLE II
THE TASK ALLOCATIONS UNDER FFD

	$\tau_4 \curvearrowright \mathcal{P}_1$	$\tau_1 \curvearrowright \mathcal{P}_2$	$\tau_2 \curvearrowright \mathcal{P}_1$	$\tau_5 \curvearrowright \mathcal{P}_2$	$\tau_3 \times$
Ψ_1	$\{\tau_4\}$	$\{\tau_4\}$	$\{\tau_4, \tau_2\}$	$\{\tau_4, \tau_2\}$	-
$U_1^{\Psi_1}(1)$	0	0	0	0	-
$U_2^{\Psi_1}(2)$	0.633	0.633	0.959	0.959	-
Ψ_2	$\emptyset$	$\{\tau_1\}$	$\{\tau_1\}$	$\{\tau_1, \tau_5\}$	-
$U_1^{\Psi_2}(1)$	0	0.394	0.394	0.712	-
$U_2^{\Psi_2}(2)$	0	0	0	0	-

In CA-TPA, tasks are ordered according to their utilization contributions, which can be calculated based on Equations (12) and (13) and are shown in Table I. Here, the order of tasks to be allocated can be easily found as τ_4, τ_2, τ_1, τ_5 and τ_3.

After task τ_4 is first allocated to core $\mathcal{P}_1$, we have $U_1^{\Psi_1}(1) = 0$, $U_2^{\Psi_1}(1) = 0.339$, $U_2^{\Psi_1}(2) = 0.633$ and $U^{\Psi_1} = 0 + \min\{0.633, \frac{0.339}{1-0.633}\} = 0.633$ based on Equations (7) to (9). There are two choices for mapping task τ_2: cores $\mathcal{P}_1$ and $\mathcal{P}_2$. If task τ_2 is assigned to core $\mathcal{P}_1$, we have $\mathscr{U}^{\Psi_1 \cup \{\tau_2\}} = 0 + \min\{0.633 + 0.326, \frac{0.175+0.339}{1-0.326-0.633}\} = 0.959$ and $\Delta^{\Psi_1 \cup \{\tau_2\}} = 0.959 - 0.633 = 0.326$ based on Equations (15) and (14). However, if task τ_2 is assigned to core $\mathcal{P}_2$, we can have $\mathscr{U}^{\Psi_2 \cup \{\tau_2\}} = 0 + \min\{0.326, \frac{0.175}{1-0.326}\} = 0.26$ and $\Delta^{\Psi_2 \cup \{\tau_2\}} = 0.26 - 0 = 0.26$. Therefore, task τ_2 is mapped onto core $\mathcal{P}_2$ according to CA-TPA. Following these steps,

TABLE III
THE TASK ALLOCATIONS UNDER CA-TPA

	$\tau_4 \curvearrowright \mathcal{P}_1$	$\tau_2 \curvearrowright \mathcal{P}_2$	$\tau_1 \curvearrowright \mathcal{P}_2$	$\tau_5 \curvearrowright \mathcal{P}_1$	$\tau_3 \curvearrowright \mathcal{P}_2$
Ψ_1	$\{\tau_4\}$	$\{\tau_4\}$	$\{\tau_4\}$	$\{\tau_4, \tau_5\}$	$\{\tau_4, \tau_5\}$
$U_1^{\Psi_1}(1)$	0	0	0	0.318	0.318
$U_2^{\Psi_1}(1)$	0.339	0.339	0.339	0.339	0.339
$U_2^{\Psi_1}(2)$	0.633	0.633	0.633	0.633	0.633
U^{Ψ_1}	0.633	0.633	0.633	0.951	0.951
Ψ_2	$\emptyset$	$\{\tau_2\}$	$\{\tau_2, \tau_1\}$	$\{\tau_2, \tau_1\}$	$\{\tau_2, \tau_1, \tau_3\}$
$U_1^{\Psi_1}(1)$	0	0	0.394	0.394	0.707
$U_2^{\Psi_1}(1)$	0	0.175	0.175	0.175	0.175
$U_2^{\Psi_1}(2)$	0	0.326	0.326	0.326	0.326
U^{Ψ_2}	0	0.26	0.654	0.654	0.967

all tasks can be feasibly allocated to cores according to the schedulability condition in Inequality (5). The resulting task-to-core mapping under CA-TPA is shown in Table III.

Complexity of CA-TPA: Recall that there are M cores and N tasks in the system. As there are normally only a few criticality levels (i.e., usually no greater than 6), K can be assumed to be a constant in the complexity analysis. Here, calculating $U_j(k)$ from Equation (1) can be done in time $O(N)$. The computation of $U(k)$ from Equation (2) can also be done in $O(N)$. Hence, sorting the tasks in decreasing order of their utilization contributions can be performed in time $O(N \cdot logN)$. Next, from Algorithm 1, we can see that finding a target core that can feasibly accommodate a given task can be done in time $O(M + N)$. Therefore, the time complexity of CA-TPA can be found as $O((M + N) \cdot N)$.

Workload Imbalance Factor: When tasks are allocated to cores under CA-TPA, it is possible to obtain a mapping with imbalanced workloads among cores, where a few cores can be over-loaded with remaining cores having large free capacities. To prevent our task partitioning algorithm from allocating most tasks to a few cores, we can define a *workload imbalance factor* Λ, which is defined as:

$$\Lambda = \frac{U^{sys} - \min\{U^{\Psi_m} | m = 1, \ldots, M\}}{U^{sys}}. \quad (16)$$

The parameter Λ can be utilized to control the variations of core utilizations. When it becomes larger and reaches a threshold α, which can be set prior to the task-to-core mapping, instead of selecting a target core according to CA-TPA, without considering utilization variations, the new task can be assigned directly to the core with the minimum core utilization (i.e., $\min\{U^{\Psi_m} | m = 1, \ldots, M\}$), subject to the schedulability conditions in Inequality (5). Hence, as shown in the simulation results, the imbalance factor can help obtain a mapping with relatively balanced workloads among all cores.

IV. EVALUATIONS AND DISCUSSIONS

To evaluate the performance of the proposed CA-TPA scheme experimentally, we constructed a simulator. For comparison, we also implemented the well-known partitioning heuristics Worst-Fit-Decreasing (WFD), First-Fit-Decreasing (FFD), Best-Fit-Decreasing (BFD), as well as the Hybrid scheme proposed in [28]. The Hybrid scheme first allocates the high-criticality tasks using WFD, and in the second step, allocates the low-criticality tasks using FFD. To assess the feasibility of a core with a new task, these schemes use first the utilization-based sufficient condition given in Equation (4). In case the outcome is negative, then they check the second and more involved sufficient schedulability condition as given in Theorem 2. Below, we first give the parameter settings for the simulations in Section IV-A and then, in Section IV-B, we present the experimental results for all the schemes under consideration.

A. Parameter settings

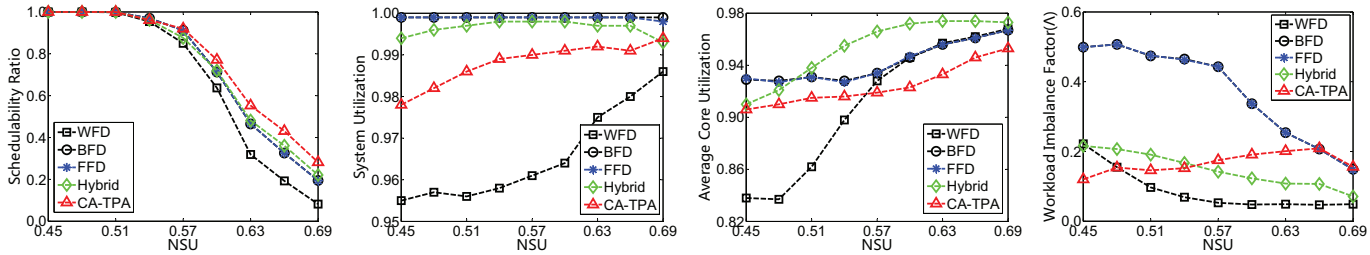
We compare these task partitioning schemes based on the following performance metrics: a) *Schedulability Ratio*, which is defined as the ratio of the number of task sets that satisfy the schedulability condition to the total number of tested task sets; b) *system utilization* (U^{sys}); c) *average core utilization* (U^{avg}) and d) *workload imbalance factor* (Λ). The last three metrics comprehensively evaluate the quality and workload balance of the partitions generated by the evaluated schemes.

In Table IV, we provide the full parameter range we considered in the experiments, including the number of cores (M), a threshold for workload imbalance factor Λ (α), the system criticality level (K), and the *normalized system utilization* (NSU) which is defined as the ratio of the aggregate raw utilization of tasks at level-1 criticality to the number of processors. The table also provides the ranges for the number of tasks (N), task periods (P) and the increment factor (IFC) defined as the ratio of WCETs for two consecutive criticality levels for a given task.

TABLE IV
SYSTEM PARAMETERS FOR THE SIMULATIONS

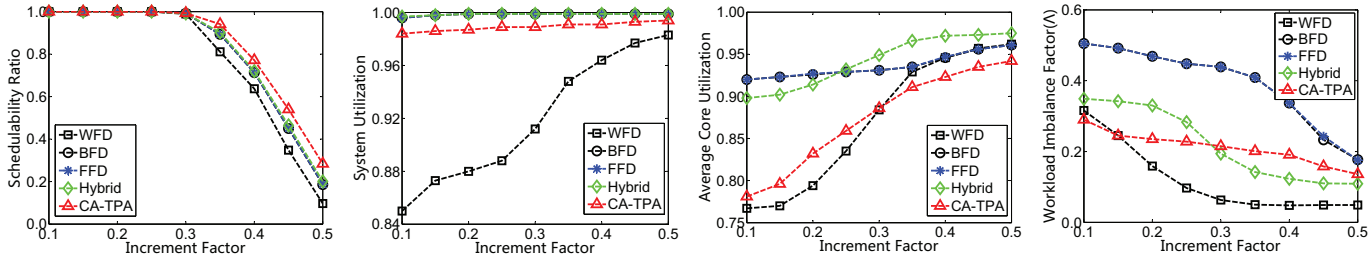
Parameters	Values/ranges
Number of cores (M)	$2, 4, 8, 16, 32$
System criticality level (K)	$[2, 6]$
Threshold for workload imbalance (α)	$[0.1, 0.5]$
Normalized system utilization (NSU)	$[0.4, 0.8]$
Number of tasks (N)	$[40, 200]$
Task periods (P)	[50, 200],[200, 500],[500, 2000]
Increment factor (IFC)	$[0.3, 0.7]$

In the simulations, the synthetic task sets are generated from the above parameters as follows. First, the system criticality level K is selected randomly in the range $[2, 6]$. For given M, N and NSU values, the base task utilization at level 1 is set as $u_{base}(1) = \frac{NSU \cdot M}{N}$. Then, for each task τ_i, the period p_i is randomly selected in one of the three period ranges given in Table IV. Next, the value of $c_i(1)$ is obtained uniformly in the range of $[0.2 \cdot p_i \cdot u_{base}(1), 1.8 \cdot p_i \cdot u_{base}(1)]$. Finally, the task τ_i's criticality level ℓ_i is selected randomly in the range $[1, K]$ and then the values of $c_i(k)(k = 2, \ldots, \ell_i)$ are obtained using $c_i(1)$ and the value of IFC.



(a) Schedulability ratio; (b) System utilization (c) Average core utilization (d) Workload imbalance factor

Fig. 1. Performance of the algorithms with varying NSU



(a) Schedulability ratio (b) System utilization (c) Average core utilization (d) Workload imbalance factor

Fig. 2. Performance of the algorithms with varying IFC

Unless otherwise specified, the default values for the parameters are: $M = 8$, $K = 4$, $NSU = 0.6$, $\alpha = 0.7$ and $IFC = 0.4$. For the results reported below, each data point corresponds to the average result of 50,000 task sets.

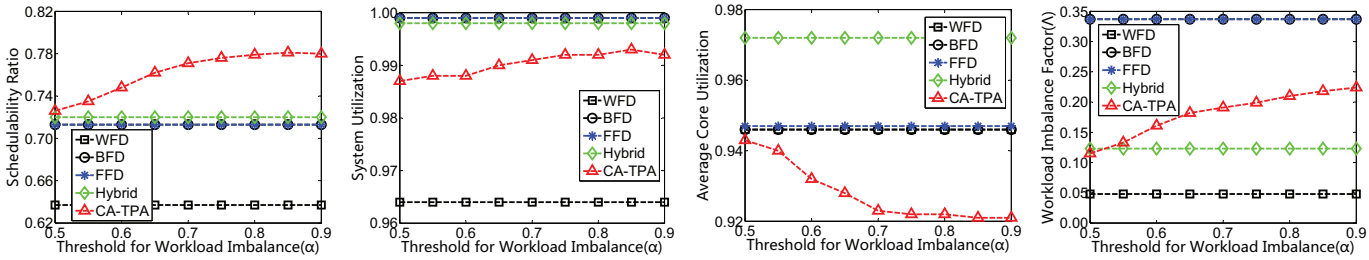
B. Performance of the Mapping Schemes

Impact of NSU. Figure 1 shows the impact of the normalized system utilization (NSU) on the performance of the mapping schemes. When other parameters (such as the number of cores M, threshold for workload imbalance factor α and increment factor IFC) are fixed, higher normalized system utilization generally means higher load and lower schedulability ratio for all mapping schemes. Not surprisingly, WFD heuristic usually yields the lowest schedulability ratio. As shown in Figure 1(a), CA-TPA can obtain the best schedulability performance compared to the others (about 5% to 25% more schedulable task sets compared to FFD, BFD and Hybrid mapping schemes) due to its effort to minimize core utilization during task-to-core assignments. Figures 1(b) to 1(d) further show the performance with regard to the workload balance generated by these schemes. These metrics are obtained by considering only the schedulable task sets. Recall that besides minimizing the increase in the core utilization, CA-TPA employs a threshold for workload imbalance factor to avoid as soon as possible severe imbalanced workload cases during task mappings. Therefore, CA-TPA can generate the task-to-core mapping with better or comparable system utilization, average utilization and workload balance when compared to FFD, BFD, and Hybrid heuristics.

Impact of the increment factor IFC. Next, we evaluate the schemes with varying increment factors (IFC). The results are shown in Figure 2. Usually, a greater IFC causes higher system workload and lower acceptance ratio from the definition of IFC and the schedulability test as given in Theorem 2. The results follow the similar trends as those for varying NSU: our CA-TPA based scheme performs best in terms of schedulability ratio and generates more balanced workload than FFD and BFD heuristics. More specifically, as CA-TPA scheme tries to bridge the gap between the total task utilizations at different levels of criticality on every processor core, it can typically obtain average core utilization comparable to WFD as shown in Figure 2(c).

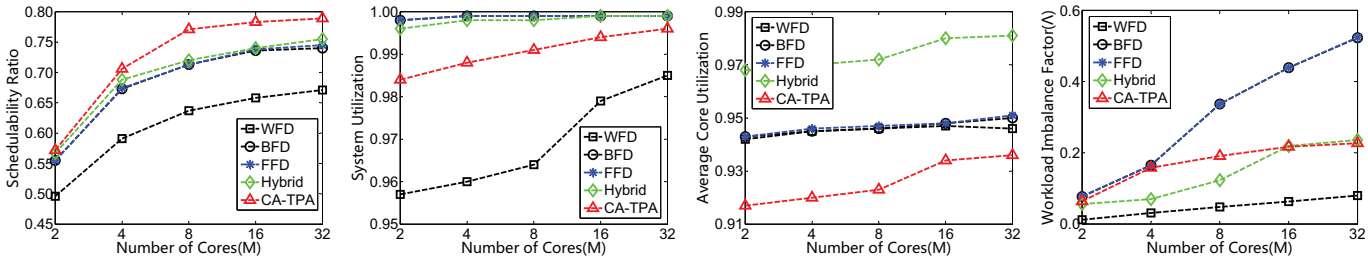
Impacts of the threshold for workload imbalance α. Figure 3 illustrates the performance comparison among all mapping schemes with different thresholds for workload imbalance. As this threshold is only employed by CA-TPA to tune workload imbalance during the task-to-core mapping, other schemes' performance remain constant with varying α as shown in Figures 3(a) to 3(d). A greater value of α usually implies larger tolerance of workload imbalance for CA-TPA. Consequently, when the value of α increases, CA-TPA attempts to allocate tasks to processing cores with the minimum increment in core utilizations without much consideration of the workload balance (i.e., in a way similar to FFD) and thus can effectively improve schedulability as shown in Figure 3(a). However, this behavior results in greater workload imbalance (i.e., higher system utilization and greater workload imbalance factor), but CA-TPA still manages to generate more balanced partitions compared to FFD and BFD schemes as shown in Figures 3(b) and 3(d). Moreover, CA-TPA can achieve the lowest average core utilization among all schemes as shown in Figure 3(c).

Impact of the number of processor cores M. We further evaluate the performance for all schemes with varying number



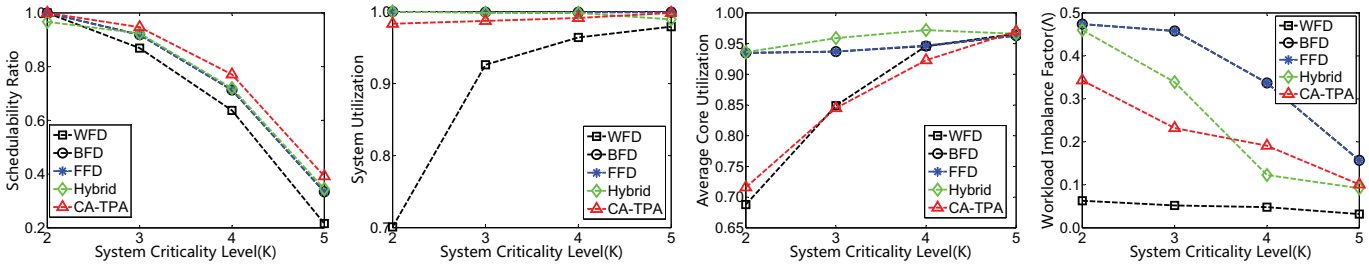
(a) Schedulability ratio (b) System utilization (c) Average core utilization (d) Workload imbalance factor

Fig. 3. Performance of the algorithms with varying α



(a) Schedulability ratio (b) System utilization (c) Average core utilization (d) Workload imbalance factor

Fig. 4. Performance of the algorithms with varying M



(a) Schedulability ratio (b) System utilization (c) Average core utilization (d) Workload imbalance factor

Fig. 5. Performance of the algorithms with varying K

of processor cores in the system and the results are shown in Figure 4. More processor cores generally can provide more capacity and more flexible processor core selections for tasks. Thus, when the value of M increases, all mapping schemes can obtain better schedulability performance as shown in Figure 4(a). Similarly, CA-TPA yields better workload balance than BFD and FFD and the lowest average core utilization among all schemes as shown in Figures 4(b) to 4(d).

Impacts of the number of criticality levels K. Finally, the performance evaluation for all schemes with different system criticality levels is shown in Figure 5. Recall that the normalized system utilization NSU represents the system's raw utilization at level 1. When the value of NSU is fixed, a greater value of K implies more execution times for tasks with highest criticality level running at level K. Thus, the schedulability ratios of all schemes decrease quickly with the increasing value of K as shown in Figure 5(a), but CA-TPA still obtains the best acceptance ratio among all schemes as explained above. Furthermore, CA-TPA generates better or comparable workload balance compared to BFD, FFD and Hybrid heuristics as shown in Figures 5(b) to 5(d).

V. Conclusions

For periodic implicit-deadline mixed-criticality (MC) tasks running on multicore systems, in this paper, we proposed an efficient *criticality-aware task partitioning algorithm (CA-TPA)*, where tasks on each core are scheduled by the EDF-VD algorithm. Observing large variations of execution requirements of MC tasks at different criticality levels, we first introduced the concept of utilization contributions of tasks to order tasks. Then, we developed a criticality-aware task partitioning approach with the objective of minimizing the utilization increment on processing cores. Moreover, we explored the workload imbalance factor together with a threshold to balance system workload on the cores. We evaluated the proposed CA-TPA mapping scheme through extensive simulations. The results show that, when compared to the existing task mapping schemes, CA-TPA can yield better balanced partitions and offer improved schedulability ratios.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (NSFC) Awards 61472150 and 61173045, the Fundamental Research Funds for the Central Universities (China) Awards HUST: 2016YXMS081 and 2015TS072, US National Science Foundation Awards CNS-1422709 and CNS-1421855.

References

[1] ARINC Specification 651: Design guidance for integrated modular avionics, 1991.

[2] James H Anderson, Sanjoy Baruah, and Björn B Brandenburg. Multicore operating-system support for mixed criticality. In *Proc. of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009.

[3] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proc. of the 24th Euromicro Conference on Real-Time Systems*, pages 145–154, 2012.

[4] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM*, 62(2):14, 2015.

[5] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo DAngelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proc. of the 19th Annual European Symposium on Algorithm (ESA)*, pages 555–566, 2011.

[6] Sanjoy Baruah and Alan Burns. Fixed-priority scheduling of dual-criticality systems. In *Proc. of the 21st International conference on Real-Time Networks and Systems*, pages 173–181, 2013.

[7] Sanjoy Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *Proc. of the 32nd IEEEReal-Time Systems Symposium*, pages 34–43, 2011.

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

[9] Andrea Bastoni, Björn B Brandenburg, and James H Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers. In *Proc. of the 31st IEEE Real-Time Systems Symposium*, pages 14–24, 2010.

[10] A. Burns and R. Davis. Mixed criticality systems-a review. In *Department of Computer Science, University of York, Tech. Rep.*, 2016.

[11] A Burns and R I Davis. Adaptive mixed criticality scheduling with deferred preemption. In *Proc. of the 35th IEEE Real-Time Systems Symposium*, pages 21–30, 2014.

[12] Yao Chen, Qiao Li, Zheng Li, and Huagang Xiong. Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling. *Chinese Journal of Aeronautics*, 27(4):856–866, 2014.

[13] Dionisio De Niz, Karthik Lakshmanan, and Ragunathan Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proc. of the 30th IEEE Real-Time Systems Symposium*, pages 291–300, 2009.

[14] Dionisio de Niz and Linh TX Phan. Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms. In *Proc. of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 111–122, 2014.

[15] Arvind Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Proc. of the 34th IEEE Real-Time Systems Symposium*, pages 78–87, 2013.

[16] Pontus Ekberg and Wang Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Proc. of Euromicro Conference on Real-Time Systems*, 2012.

[17] Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-time systems*, 50(1):48–86, 2014.

[18] Thomas Fleming. *Extending mixed criticality scheduling*. PhD thesis, University of York, 2013.

[19] R Gratia, T Robert, and L Pautet. Adaptation of run to mixed-criticality systems. *JRWRTC*, page 25, 2014.

[20] Chuancai Gu, Nan Guan, Qingxu Deng, and Wang Yi. Partitioned mixed-criticality scheduling on multiprocessor platforms. In *Proc. of Design, Automation and Test in Europe Conference and Exhibition*, pages 1–6, 2014.

[21] RTCA Inc. DO-178C: Software considerations in airborne systems and equipment certification, 2011.

[22] Owen R Kelly, Hakan Aydin, and Baoxian Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *Proc. of the International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1051–1059, 2011.

[23] Karthik Lakshmanan, Dionisio De Niz, Ragunathan Rajkumar, and Gines Moreno. Resource allocation in distributed mixed-criticality cyber-physical systems. In *Proc. of the 30th International Conference on Distributed Computing Systems*, pages 169–178, 2010.

[24] Haohan Li and Sanjoy Baruah. Outstanding paper award: Global mixed-criticality scheduling on multiprocessors. In *Proc. of the 24th Euromicro Conference on Real-Time Systems*, pages 166–175, 2012.

[25] Giuseppe Lipari and G Buttazzo. Resource reservation for mixed criticality systems. In *Proc. of the Workshop on Real-Time Systems: the past, the present, and the future*, pages 60–74, 2013.

[26] Moritz Neukirchner, Philip Axer, Thomas Michaels, and Rolf Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *Proc. of the 34th IEEE Real-Time Systems Symposium*, pages 88–96, 2013.

[27] Risat Mahmud Pathan. Schedulability analysis of mixed-criticality systems on multiprocessors. In *Proc. of the 24th Euromicro Conference on Real-Time Systems*, pages 309–320, 2012.

[28] Paul Rodriguez, Laurent George, Yasmina Abdeddaïm, and Joël Goossens. Multi-criteria evaluation of partitioned edf-vd for mixed-criticality systems upon identical processors. In *Proc. of the Workshop on Mixed Criticality Systems*, 2013.

[29] Vincent Sciandra, Pierre Courbin, and Laurent George. Application of mixed-criticality scheduling model to intelligent transportation systems architectures. *ACM SIGBED Review*, 10(2):22–22, 2013.

[30] Lui Sha, John P Lehoczky, and Ragunathan Rajkumar. Task scheduling in distributed real-time systems. In *Proc. of Robotics and IECON Conferences*, pages 909–917. International Society for Optics and Photonics, 1987.

[31] Hang Su and Dakai Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proc. of the Conference on Design, Automation and Test in Europe*, pages 147–152, 2013.

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

[33] N Zhang, C Xu, J Li, and Peng M. A sufficient response-time analysis for mixed criticality systems with pessimistic period. *Journal of Computational Information Systems*, 11(6):1955–1964, 2015.

[34] T. Zhang, N. Guan, Q. Deng, and W. Yi. On the analysis of edf-vd scheduled mixed-criticality real-time systems. In *Proc. of the 9th IEEE International Symposium on Industrial Embedded Systems*, 2014.

[35] Qingling Zhao, Zonghua Gu, and Haibo Zeng. Pt-amc: Integrating preemption thresholds into mixed-criticality scheduling. In *Proc. of the Conference on Design, Automation and Test in Europe*, pages 141–146, 2013.