Received October 9, 2018, accepted October 26, 2018, date of publication November 12, 2018,
date of current version December 7, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2879717

Semi-Slack Scheduling Arbitrary Activation Patterns in Mixed-Criticality Systems

BIAO HU[1], GANG CHEN[2], AND KAI HUANG[2], (Member, IEEE)
[1]College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China
[2]School of Data and Computer Science, Sun Yat-sen University, Xiaoguwei Island, Panyu District, Guangzhou 510006, China

Corresponding author: Gang Chen (gangchen1170@foxmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grants 61802013, 61702085, and 61872393, in part by the Talent Foundation of the Beijing University of Chemical University under Grant buctrc201811, in part by the Open Research Project of the State Key Laboratory of Synthetical Automation for Process Industries under Grant H2018294, and in part by the Fundamental Research Funds for the Central Universities under Grant XK1802-4.

ABSTRACT This paper proposes a semi-slack scheduling framework for mixed-criticality systems with arbitrary task activation patterns. In particular, based on the schedulability test of arbitrary task activation model, we present a tight slack-reclaim scheme that can adaptively make use of system slack to improve the quality of service to low critical tasks. This scheme works at the moment that low critical tasks are supposed to be degraded by the offline schedulability test, while at other moments the system is scheduled by the earliest deadline first with virtual deadlines. The semi-slack scheduling scheme is first discussed in uniprocessor and later extended to multiprocessor. Extensive simulation results demonstrate that compared with some state-of-the-art scheduling approaches, the proposed semi-slack scheduling scheme efficiently reduces deadline misses of low critical tasks without jeopardizing the guarantee to critical tasks.

INDEX TERMS Mixed-criticality system, semi-slack scheduling.

I. INTRODUCTION

Today's emerging embedded systems for intelligent vehicles and robots are performing more complex and diverse tasks on a single platform than ever before, as in this way costs and space can be reduced. Those tasks in a safety-critical system often have different criticalities and thus should be guaranteed to different degrees [1]. This type of system is called *mixed criticality system* (MCS). A typical example of MCS is the unmanned aerial vehicle system that needs to guarantee the correctness more for tasks related to vehicle safety than for tasks only related to some vehicle missions.

A. MOTIVATIONS

A common assumption adopted by the real-time system committee [1] is to guarantee the schedulability of all tasks when no tasks overrun their less rigorous execution thresholds (normal mode) and the schedulability of only critical tasks when they overrun (critical mode). In such certifiable mixed-criticality real-time systems, tasks are often modeled as sporadic activation pattern that only defines a minimum inter-activation interval (also called period) [1], [2]. Although results based on the analysis of sporadic tasks can provide some insights for classic activation patterns like periodic activations, those results are indeed pessimistic to accommodate complex activation tasks. A simple example is illustrated in Fig. 1, where a task is activated every 4 time units, while there is a jitter of 1 time unit in every activation. In this case, the minimum distance between two task activations is 2. If this task is modeled as a sporadic task, the estimated number of task activations within 14 is 8. However, the fact is that at most 5 activations would happen within 14. The overestimation on activation events would make a system schedulability test more pessimistic. To fully exploit the system schedulability towards tasks with arbitrary activation patterns, a more accurate task activation model should be explored.

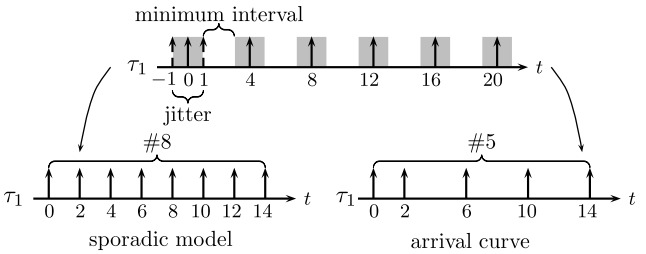


FIGURE 1. Illustration of difference by modeling the same task activation pattern as different models.

Recently, a more powerful schedulability test that can cope with arbitrarily activated tasks has been proposed in [3]. This schedualbility test is more practical because complex or arbitrary activation patterns are very common in automotive systems [4]–[10], where tasks in a processing unit are often triggered by external events coming from other devices. Arrival patterns of most external event streams are neither periodic nor periodic with a jitter, but with some peculiarities that are difficult to be captured by the sporadic task model. In order to precisely analyze effects of event arrival patterns on the system behavior, the framework of *Real-Time Calculus* [11] adopts a more general task activation model, *arrival curve*, to compute the task response time or maximum backlogged events. Stemming from the theory of network calculus [12], arrival curve effectively expresses the maximum arrival events within any interval, which provides more accurate information on task activations than sporadic model [13]. Network calculus is also a very effective theory to bound the system processing delay. In this article, arrival curve is adopted to model task arbitrary activation patterns.

A scheduling routine in classic mixed-criticality model is to degrade or completely remove low critical tasks once a high critical task overruns its given execution budget. However, this is a pessimistic scheduling routine as the system may have free slack to allow tasks overrun without degrading the service to low critical tasks. This free slack arises because tasks may not be activated as frequently as the worst-case assumption, or because most activated tasks use less time than the given budgets to finish. Being aware of free slack, we propose a semi-slack scheduling scheme to exploit such free slack as much as possible. In detail, this scheme performs two functionalities. The first function is to postpone the system entering critical mode by allowing high-critical tasks to overrun. The second function is to schedule low critical jobs as in normal mode after the system enters into the critical mode. In contrast with previous scheduling scheme, this new scheduling scheme relaxes the system mode-switch and fully exploit the system free slack to improve the quality of service to lo-critical tasks. This scheme is called semi-slack scheduling scheme because system free slack is exploited only when high critical tasks overrun in the normal mode or when the system is in critical mode.

B. RELATED WORK

The slack scheduling in mixed-criticality systems is not new. Existing slack scheduling approaches with relation to four system properties are generally summarized in Tab 1. The runtime adaptability denotes that the scheduling capacity for both high critical and low critical tasks can be adaptively updated based on runtime scheduling. An essential difference between mixed-criticality systems and traditional real-time systems is the criticality guarantee, where different criticality tasks are guaranteed to different levels. Scheduling approaches that can handle arbitrary task activations and multiprocessor are two important but seldom studied topics.

TABLE 1. Comparison of the existing results.

	Runtime Adaptability	Criticality Guarantee	Arbitrary Activations	Multiprocessor
[14]–[18]	✓	✓	×	×
[19]	×	✓	×	×
[8], [20], [21]	×	×	✓	×
[22], [23]	✓	×	✓	×
[24]	×	✓	✓	×
this paper	✓	✓	✓	✓

Some of previous slack scheduling approaches adopt the scheme in which low critical tasks are run in the slack generated by high critical jobs when they only use their low-level execution budgets [14]–[16]. The main difficulty of such slack scheduling scheme is to incorporate sporadic tasks. It is unclear at what point that slack of a non-appearing sporadic task can be allocated to low critical jobs [1]. The problem of handling sporadic tasks has been partly resolved by two recent approaches. In [17], a dynamic scheduling model is proposed to determine execution budgets online for individual high critical tasks. In [18], an adaptive task dropping scheme is proposed to minimize execution penalty of low critical tasks by adaptively determining which task to drop based on the runtime analysis. In [19], a model called interference constraint graph is proposed to specify allowed interferences between tasks. Although this model can generalize many of existing mixed-criticality scheduling conditions, it does not take the runtime adaptability into consideration. The drawback of all aforementioned approaches is that they are not capable of handling arbitrary activation tasks.

There are some other scheduling approaches that are effective for arbitrary activation tasks. Under the interpretation of considering the mixed-criticality system as a system mixed with soft and hard real-time tasks, some shaping approaches are proposed to shape arrival events to conform to the offline computed bound, where the offline computed bound is often too pessimistic for serving low critical tasks [8], [20], [21]. To improve the system utilization, the online information should be used to calculate a tighter bound. The runtime adaptability has been addressed in [22] and [23] where the inflow workload of low critical tasks is regulated based on the actual demand of high critical tasks. Although this shortens the response time of low critical tasks, it is not clear at what level of guarantee that the system can provide to low critical tasks. In a nutshell, it is still an open problem on how to adaptively explore system slack to schedule general activation task models while sufficiently providing the basic service guarantee to low critical tasks.

C. CONTRIBUTIONS

In [24], we have partly analyze how to sufficiently guarantee system schedulability at runtime by exploring free slack to run low critical tasks. Based on the schedulability analysis result from [24], this article proposes a semi-slack scheduling scheme that can fully make use of system slack to improve the scheduling performance for a dual-criticality system.

Compared to previous scheduling approaches, the proposed approach covers the four properties. First, the system schedulability towards arbitrary activation tasks is first verified so that the basic service guarantee is provided to low critical tasks. After ensuring the system is schedulable, we proceed to apply the slack reclaim scheme to improve quality of service to low critical tasks. With the existence of free slack, we use the task procrastination techniques [25], [26] to detect runtime slack that can be safely used to run low critical tasks without reducing the guarantee to critical tasks. The detected slack can be either used to postpone the system switching to critical mode or to not degrade the service to low critical jobs even when the system is in critical mode. In this way, the free slack can be aggressively explored to improve the system scheduling capability. In addition, we extend the semi-slack scheduling from uniprocessor to multiprocessor. In order to balance workload among different processors, a global scheduler is devised to cross-use system free slack. The global scheduler is on top of the local scheduler in each individual processor, dispatching or migrating tasks. The results confirm the high effectiveness of our proposed global scheduler.

The remainder of this article is structured as follows. In Section II, we introduce the system model. Then we explain how the semi-slack scheduling works for uniprocessor platform (Section IV). In Section V, we present how to extend this semi-slack scheduling to multiprocessor platform. We provide extensive simulation results (Section VI). We finally conclude this article in Section VII.

II. SYSTEM MODEL AND SCHEDULABILITY ANALYSIS

A. EVENT MODEL

We consider that a task is activated by an event [9]. Task activation sequence in the system can be expressed as an event stream. A trace of such an event stream is described by means of a function $R[s,t)$ that denotes the sum of events arrived in the time interval $[s,t)$, with $R[s,s) = 0, \forall s,t \in \mathbb{R}$. While any R always describes one concrete trace, $\overline{\alpha}^u(\Delta)$ provides an abstract event stream model that represents the maximum number of events that are seen in a time interval with Δ length.

Definition 1 (Arrival Curve [9]): Denote $R[s,t)$ as the number of events that arrive on an event stream in the time interval $[s,t)$. Then, R, $\overline{\alpha}^u$ represents the upper and lower bound on the number of event in any interval $t-s$, that is,

$$R[s,t) \leq \overline{\alpha}^u(t-s), \quad \forall t \geq s \geq 0,$$

with $\Delta = t-s$ and $\overline{\alpha}^u(\Delta) \geq 0$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$.

The concept of arrival curve substationally generalizes conventional event stream models, such as sporadic, periodic, periodic with jitter, and arbitrary event streams. For instance, for the arbitrary events modeled with the period p, the jitter j, and the minimum inter arrival distance d between successive two events, its upper arrival curve is $\overline{\alpha}^u(\Delta) = \min\{\lceil \frac{\Delta+j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}$. This arrival pattern is called *pjd* pattern that is often used as a typical example of the complex arrival pattern in many previous works [22], [26], [27].

The arrival curve only provides an upper bound on the number of events. In order to know specific minimal distances among any events, we rely on a concept similar to the arrival curve: minimum distance function [4].

Definition 2 (Minimum Distance Function): The minimum distance function $\delta(q)$ is a pseudo super-additive[1] function, which returns a lower bound on the time interval between the first and the last event of any sequence of $q+1$ event occurrences.

The minimum distance function is an inverse description of upper arrival curve. For example, $\delta(k) = \Delta_k$ denotes that, the first and the last event of any sequence of $k+1$ events is at least Δ_k time units apart, i.e., $\overline{\alpha}(\delta(k)) = k+1$.

Analogous to the arrival curve that provides an abstract event stream model, the service curve $\beta^l(\Delta)$ denotes the minimum number of execution time units available over any time interval of fixed length Δ, where the superscript l means the lower bound. In this article, the uniprocessor is supposed to be a unit-speed platform. For example, for a task set τ, if the processing is not interrupted, the service curve provided to τ can be represented by $\beta^l(\Delta) = \Delta$.

B. MIXED-CRITICALITY SYSTEM SETTINGS

This paper assumes a *dual-criticality* uniprocessor or multiprocessor system with two distinct criticality levels: HI (high) and LO (low). For abbreviation, we use LO mode and HI mode to represent the system at LO and HI criticality level, respectively. A task set $\tau = \{\tau_1, \ldots, \tau_n\}$ is given to be scheduled. The task WCETs are modeled on both criticalities. Therefore, each task τ_i can be characterized by ($\overline{\alpha}^u(\Delta)$ or $\delta_i(q)$, $\mathbf{C_i}$, L_i), where

- $\overline{\alpha}^u(\Delta)$ denotes the upper bound of task activations within a certain time interval Δ; $\delta_i(q)$ denotes the minimum time interval of $q+1$ activations.
- $\mathbf{C_i} = (C_i^L, C_i^H)$ represents a task two level WCETs, where C_i^L is a low-confidence WCET and C_i^H is a high-confidence WCET.
- $L_i \in \{LO, HI\}$ represents a task criticality.

For HI-critical tasks, the WCETs on the HI criticality are non-decreasing when compared to their WCETs on the LO criticality, i.e., $\forall \tau_i \in \tau^H : C_i^L \leq C_i^H$ where τ^H denotes the subset of HI-critical tasks in τ. Based on the basic scheduling scheme of EDF on MCS, the deadline of HI-critical tasks should be shortened when system is in LO mode, which means there will be two deadlines mounted on each HI-critical task. We use $\mathbf{D_i} = (D_i^L, D_i^H)$ to represent the deadline of a HI-critical task in LO mode and in HI mode, respectively. In contrast, a LO-critical task only has a deadline because it is not supposed to meet its deadline in HI mode.

III. PRELIMINARY KNOWLEDGE

In this section, we present the online and offline schedulability test. We first present the schedulability conditions and

[1]For pseudo super-additive we denote the property of a function δ that $\forall a,b \in \mathbb{N}^+ : \delta(a+b) \geq \delta(a) + \delta(b)$. It corresponds to the property of "good" arrival curve in [12].

then present how to derive the demand-bound function offline and online because the schedulability test relies on them.

A. SCHEDULABLE CONDITIONS

In order to test the schedulability of a system scheduled by Earliest Deadline First-Virtual Deadline (EDF-VD) [2], we now introduce the demand-bound function.

Definition 3 (Demand-Bound Function [2], [28]): The demand-bound function $\mathrm{dbf}(\tau_i, \Delta)$ *gives an upper bound on the maximum possible execution demand of the task* τ_i *in any time interval of length* Δ*, where demand is calculated as the total amount of required execution time of events with their whole scheduling windows within the time interval.*

For a non-mixed-criticality system with a task set τ, the following inequality guarantees the schedulability of this task set:

$$\sum_{\tau_i \in \tau} \mathrm{dbf}(\tau_i, \Delta) \leq \beta^l(\Delta), \ \forall \Delta \geq 0.$$

For a dedicated unit-speed uniprocessor that constantly keeps working in active state, its supply $\beta^l(\Delta) = \Delta$. Analogously, in MCSs, the task set is schedulable if DBFs in LO and HI modes are smaller than the provided supply of this platform [2], [29], [30], i.e.,

$$\text{Condition 1} : \forall \Delta \geq 0 : \sum_{\tau_i \in \tau} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta) \leq \beta^l(\Delta) = \Delta,$$

$$\text{Condition 2} : \forall \Delta \geq 0 : \sum_{\tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \leq \beta^l(\Delta) = \Delta,$$

where *Condition* 1 guarantees that all tasks meet their deadlines in LO mode (normal mode) and *Condition* 2 guarantees that HI-critical tasks meet their deadlines in HI mode (critical mode). The two conditions were first used to test the schedulability of a system with sporadic tasks. However, according to the definition of demand-bound function, it is also applicable to test the schedulability of a system with arbitrarily activated tasks, because $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$ indicates the lower bound of supply bound to meet its deadline. Therefore, to determine the schedulability of a task set, one only needs to derive DBFs in both modes.

The definition 3 defines $\mathrm{dbf}(\tau_i, \Delta)$ to denote the worst-case demand that the demand of τ_i will never surpass it at runtime. In order to represent the demand-bound function at runtime more tightly, we use $\mathrm{dbf}(\tau_i, \Delta, t)$ and $\beta^l(\Delta, t)$ to denote the task demand and system service curve over any Δ from time t and onward. Analogous to the schedulable conditions offline, the schedulable conditions at runtime would be

$$\text{Condition 1} : \forall \Delta \geq 0 : \sum_{\tau_i \in \tau} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \leq \beta^l(\Delta, t),$$

$$\text{Condition 2} : \forall \Delta \geq 0 : \sum_{\tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t) \leq \beta^l(\Delta, t).$$

B. DEMAND-BOUND FUNCTIONS OFFLINE

For the task with arbitrary activation pattern with dual-criticality, it has been derived in [24] that

$$\begin{cases} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta) = \bar{\alpha}_i^u(\Delta - D_i^L) \cdot C_i^L \\ \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) = (k+1) \cdot C_i^H - [\![C_i^L - (\Delta - \delta_i'(k))]\!]_0, \end{cases} \tag{1}$$

where

$$\delta_i'(k) = \begin{cases} k \cdot C_i^L, & k \leq h \\ h \cdot C_i^L + \delta_i(k) - \delta_i(h), & k > h, \end{cases} \tag{2}$$

and the denotation $[\![x]\!]_0$ is the abbreviation of $\max(0, x)$; $\delta_i(h+1) - \delta_i(h) > C_i(LO)$, $k \in \mathbb{N}^+$, $h = \min\{q | \delta_i(q+1) - \delta_i(q) > C_i^L\}$. It has also been presented in [24] how to construct the pseudo-polynomial bound such that the above two conditions can hold. For completeness, we present the derivation of Eq. 1 and how to decide D_i^L to pass the schedulability test of arbitrary activation tasks in Appendix VII.

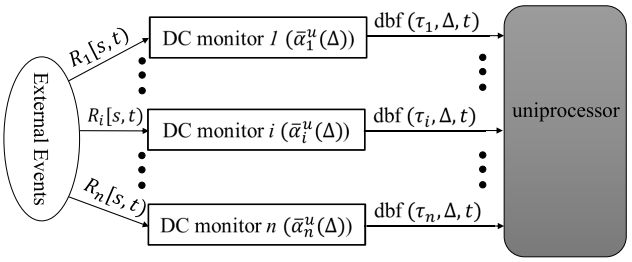


FIGURE 2. System monitored by dynamic counters, which can provide the demand-bound function of each task at runtime dbf(τ_i, Δ, t).

C. DEMAND-BOUND FUNCTIONS AT RUNTIME

To get $\mathrm{dbf}(\tau, \Delta, t)$, event streams have to be monitored. The system monitoring setup is shown in Fig. 2, where external events activate n tasks and are thus monitored by n monitors. By constantly monitoring arriving events, the monitor is able to provide the demand-bound function of each task at runtime to system schedule.

1) FUTURE EVENTS AND THEIR DEMAND BOUND

In principle, any complex arrival pattern $\overline{\alpha}^u(\tau_i, \Delta)$ of task τ_i can be bounded by a set of upper staircase functions [31], i.e.,

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \overline{\alpha}^u(\tau_i, \Delta) \leq \min_{j=1..n'} \{N_j^u + \lfloor \frac{\Delta}{\delta_j^u} \rfloor\},$$

where N_j^u is the initial value of a staircase function, δ_j^u is the stair length and n' is the number of staircase functions that need to be approximate $\overline{\alpha}^u(\tau_i, \Delta)$. As shown in Fig. 2, dynamic counters (DC) can be used to predict the upper bound of activations of task τ_i from time t and onward [27], which is

$$\overline{\alpha}^u(\tau_i, \Delta, t) = \min_{j=1..n'} (\mathcal{U}_j(\tau_i, \Delta, t)). \tag{3}$$

where

$$\mathcal{U}_j(\tau_i, \Delta, t) = DC_j(t) + \begin{cases} \lfloor \frac{\Delta + (t - k_j\delta_j^u)}{\delta_j^u} \rfloor & \text{if } DC_j(t) < N_j^u \\ \lfloor \frac{\Delta}{\delta_j^u} \rfloor & \text{if } DC_j(t) = N_j^u \end{cases} \quad (4)$$

Then, it can be derived that the demand bound function of future events is that

$$\text{dbf}^{\text{F}}(\tau_i, \Delta, t) = \overline{\alpha}^u(\tau_i, \Delta - D_i, t) \cdot c_i, \quad (5)$$

where D_i and c_i are relative deadline and WCET of task τ_i. The detailed derivation of Eq. 3 is presented in Appendix VIII.

2) BACKLOGGED EVENTS AND THEIR DEMAND

During the runtime, if events arrive more frequently than the rate that they can be processed, some events may be backlogged. We denote the set of unfinished events of τ_i in the backlog at time t as $\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. Then, the number of backlogged events can be denoted as $|\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|$. We denote the j-th event in the backlogged events as $e_{i,j}$. For each event $e_{i,j} \in \mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$, we use $D_{i,j}$ to denote its absolute deadline. Let $D_{i,j}$ denote the absolute deadline for event $e_{i,j} \in \mathbf{E}(\tau_\mathbf{i}, \mathbf{t})$. A backlogged demand for this task is defined as [26]

$$\text{dbf}^{\text{B}}(\tau_i, \Delta, t) = c_i \cdot \begin{cases} (j-1), & D_{i,j} - t < \Delta < D_{i,j+1} - t, \\ |\mathbf{E}(\tau_\mathbf{i}, \mathbf{t})|, & \Delta \geq D_{i,|\mathbf{E}(\tau_\mathbf{i},\mathbf{t})|} - t, \end{cases} \quad (6)$$

in which $D_{i,0}$ is defined as t for brevity.

3) CARRY-ON EVENT AND ITS DEMAND

A carry-on event is an event that has been released but not finished. Suppose $C(\tau_i, t)$ is used to denote the left time for finishing a carry-on event of τ_i at time t, and the demand for the carry-on event is that

$$\text{dbf}^{\text{C}}(\tau_i, \Delta, t) = c_i \cdot \begin{cases} 0, & \Delta < D_c - t, \\ C(\tau_i, t), & \Delta \geq D_c - t, \end{cases} \quad (7)$$

where D_c is the absolute deadline of this carry-on event.

4) DEMAND-BOUND FUNCTION IN LO MODE

Based on the definition of demand bound function, it can be concluded that the demand bound function $\text{dbf}(\tau_i, \Delta, t)$ of task τ_i is that,

$$\text{dbf}(\tau_i, \Delta, t) = \text{dbf}^{\text{F}}(\tau_i, \Delta, t) + \text{dbf}^{\text{B}}(\tau_i, \Delta, t) + \text{dbf}^{\text{C}}(\tau_i, \Delta, t). \quad (8)$$

In the dual-criticality system, the system mode is divided into LO or HI mode. In LO mode, a task τ_i is assigned to be with execution time of C_i^L and relative deadline of D_i^L. Straightforwardly, the demand bound function of a task τ_i in LO mode is

$$\text{dbf}_{\text{LO}}(\tau_i, \Delta, t) = \text{dbf}_{\text{LO}}^{\text{F}}(\tau_i, \Delta, t) + \text{dbf}_{\text{LO}}^{\text{B}}(\tau_i, \Delta, t) + \text{dbf}_{\text{LO}}^{\text{C}}(\tau_i, \Delta, t), \quad (9)$$

where the WCET is C_i^L instead of c_i and the relative deadline is D_i^L instead of D_i in above equations from Eq. 3 to Eq. 8.

IV. SEMI-SLACK SCHEDULING IN UNIPROCESSOR

A schedulability test has been presented to verify an arbitrary activation task set is schedulable in LO and HI modes. This schedulability test adopts the scheme that the system should immediately enter into HI mode once a HI-critical task overruns its given LO WCET, where in HI mode LO-critical tasks are not guaranteed schedulable. While this scheme is sufficient in providing the basic level of guarantee to both LO-critical and HI-critical tasks, it fails to fully make use of system resource to improve the quality of service to LO-critical tasks. In reality, the task often demands less than the worst-case assumption because the task will neither be activated as frequently as the arrival curve nor need the given execution budgets to finish at runtime. Due to this lesser demand on the system resource, there will be some free slack that can be exploited to loosen the mode-switch scheme or even run LO-critical jobs as normal when system is in HI mode. In light of this, we present the semi-slack scheduling scheme in this section to make use of system free slack to postpone the mode-switch as long as possible and furthermore not degrade the service to LO-critical jobs when system being in HI mode.

The semi-slack scheduling is inspired by the task procrastination technique in hard real-time system [25], [26], where task executions are deliberately delayed by forcing processor to *sleep* state without missing any deadlines. Analogously, the overrun of HI-critical tasks is considered as resources procrastination on all tasks in the sense that overrun executions obstruct executions of all other tasks. We adopt such task procrastination technique to explore the system free slack at runtime. Because system free slack is exploited only when high critical tasks overrun or when the system is in critical mode, the proposed approach is called semi-slack scheduling approach.

A. TASK PROCRASTINATION APPROACH

Before we present the semi-slack scheduling scheme, we give a brief introduction on how to detect the free slack by using task procrastination approach [25], [26]. Typically, the system free slack comes from the case that tasks demand fewer resources than what the system can provide. In order to compute the system free slack at runtime, the task online demand-bound function has to be derived. The computation of online demand-bound function has been presented in the previous section. More details can be found in [23] and [27]. Under EDF schedule, the demand-bound function of a task set τ is the sum of every task demand-bound function, i.e.,

$$\text{dbf}(\tau, \Delta, t) = \sum_{\tau_i \in \tau} \text{dbf}(\tau_i, \Delta, t). \quad (10)$$

Based on $\text{dbf}(\tau, \Delta, t)$, the maximum free slack can be computed in the following way.

$$\rho^*(t) = \max \left\{ \rho : [\![\Delta - \rho]\!]_0 \geq \text{dbf}(\tau, \Delta, t),\ \forall \Delta \geq 0 \right\}, \quad (11)$$

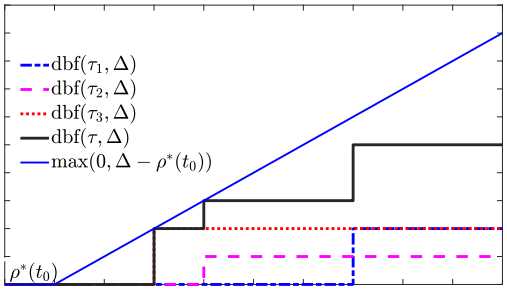


FIGURE 3. Illustration of maximum free slack by task procrastination approach.

where $\rho^*(t)$ is the maximum free slack that can be exploited for other purposes. Fig. 3 conceptually shows how to compute the free slack by Eq. 11.

Theorem 1 (From [25]): Suppose $\mathrm{dbf}(\tau, \Delta, t)$ denote DBF of a task set τ from time t. If there is a $\rho\,(\rho > 0)$ that satisfies

$$\forall \Delta > 0 : \mathrm{dbf}(\tau, \Delta, t) \leq [\![\Delta - \rho]\!]_0, \tag{12}$$

then executions of all tasks can be immediately delayed for ρ and there will be no deadline misses after t.

The complexity of computing $\rho^*(t)$ has been demonstrated in [23] to be $O(n{\cdot}log(n))$ (n is the number of tasks). It was also evaluated that the computation time is 36.1 μs for computing one task and 164.7 μs for computing ten tasks in [23]. For practical implementation, we propose setting a certain time budget for computing free slack. When this time budget is used up, the free slack updating would be stopped and the system would be forcefully switched to HI mode. Another drawback of this approach is that the computation time will increase with task set size. A possible solution could be to take the largest overhead among all tested task sets into scheduling. In this article, the practical influence of computation overhead is not deeply studied because we want to focus on the proof-of-concept results by our proposed approaches.

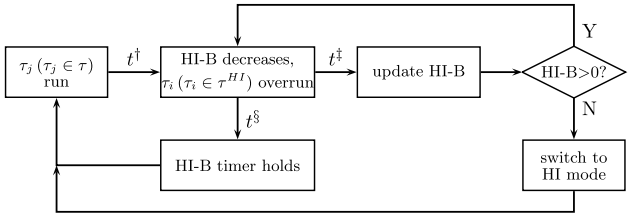


FIGURE 4. Workflow of the mode-switch procrastination.

B. MODE-SWITCH PROCRASTINATION

The workflow of mode-switch procrastination is shown in Fig. 4, where we introduce a term called HI-critical Budget (HI-B) to denote the extent that HI-critical tasks can be allowed to overrun without switching system to HI mode. First of all, we set up a HI-B timer that is used to limit the overrun margin within HI-B. HI-B is a variable that will change with the time. So we denote HI-B as *HI-B*(t). The system runs tasks without using *HI-B*(t) when all HI-critical jobs run within their LO WCETs. Once a HI-critical job overruns ($t^\dagger$), *HI-B*(t) starts to decrease. If this overrun job is suspended or finished before *HI-B*(t) decreases to 0 ($t^\S$), *HI-B*(t) will stop decreasing and the system returns to 'no overrun' state. If this job does not finish before *HI-B*(t) elapses to 0 ($t^\ddagger$), *HI-B*(t) will be updated. In this case this overrun job will still be allowed to overrun if the updated *HI-B*(t) is greater than 0. Otherwise the system will be switched to HI mode.

A key problem of such mode-switch scheme is how to find a feasible *HI-B*(t) that can safely allow tasks to overrun without violating the basic guarantee level to tasks, i.e., LO-critical tasks should be schedulable in LO mode and HI-critical tasks should be schedulable in both modes. To get a feasible *HI-B*(t), we have to first derive system online demand-bound function in LO mode, where all tasks are given LO WCET budgets. The derivation of demand-bound function in LO mode is similar to Eq.10, as shown below:

$$\begin{aligned}\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t) &= \sum_{\tau_i \in \tau} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \\ &= \sum_{\tau_i \in \tau} \{\mathrm{dbf}^{\mathrm{F}}_{\mathrm{LO}}(\tau_i, \Delta, t) + \mathrm{dbf}^{\mathrm{B}}_{\mathrm{LO}}(\tau_i, \Delta, t) \\ &\quad + \mathrm{dbf}^{\mathrm{C}}_{\mathrm{LO}}(\tau_i, \Delta, t)\},\end{aligned} \tag{13}$$

where the subscript of *LO* in each dbf means that a task τ_i only demands LO WCET for each job.

Theorem 2: Suppose the LO mode demand-bound function of a task set τ at time t is $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t)$, then by giving $\rho^*(t)$ computed by Eq. 11 to HI-B(t) at every updating step, all tasks will meet their LO deadlines.

Proof: In order to prove this theorem, we separate system runtime behavior to 'no overrun state' and 'overrun state', which represents whether a job is overrunning or not. On the one hand, the system demand in 'no overrun state' is bounded within $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t)$. Since the provided system resource to 'no overrun state' is $[\![\Delta - \rho^*(t)]\!]_0$. Because of $[\![\Delta - \rho^*(t)]\!]_0 \geq \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t)$, there will be no deadline misses in 'no overrun state'. On the other hand, according to theorem 1, $[\![\Delta - \rho^*(t)]\!]_0 \geq \mathrm{dbf}(\tau, \Delta, t)$ also denotes that the system can be shutdown for $\rho^*(t)$ without missing any deadline, which also indicates that any task can be allowed to overrun for $\rho^*(t)$. Hence, there will also no deadline miss during 'overrun state'. □

Corollary 1: Suppose t_s is the mode-switch moment of a system under the mode-switch procrastination scheme of Fig. 4. Then, the following inequality is guaranteed:

$$\forall \tau_i \in \tau^H, \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta), \tag{14}$$

where $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s)$ is the demand-bound function of a HI-critical task τ_i that demands HI WCET *at t_s, and $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ is the offline derived demand-bound function.*

Proof: Theorem 2 has showed that τ_i will be schedulable even when its job may overrun before mode-switch. There are

two cases to HI-B in Fig. 4, which are HI-B=0 and HI-B>0. In the case that HI-B=0, the mode-switch will be triggered immediately after a job overruns, which is equivalent to the system without deploying the semi-slack slack scheduling scheme. Then, according to the derivation of $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s)$ of [3], we have $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) = \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$. In the other case, if a job has overrun for a certain time before mode-switch, τ_i will demand less at t_s than that of not overrunning because a part of execution of τ_i has been processed in LO mode. Thus, $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) < \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$. □

From the above corollary, we can further prove that

$$\sum_{\tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) \leq \sum_{\tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \leq \Delta. \quad (15)$$

According to the Condition 2, Corollary 1 and the above equation show that the mode-switch procrastination scheme can guarantee the system will be schedulable after the system enters into HI mode.

C. SHAPING APPROACH IN HI MODE

After the system enters into HI mode, a lot of previous scheduling approaches abandon LO-critical tasks in order to sufficiently meet HI-critical tasks timing requirements. However, abandoning all LO-critical tasks is unnecessary due to the existence of free slacks. In this section, we present a shaping approach to let LO-critical jobs run in HI mode.

The shaping approach also depends on the task procrastination technique to exploit free slack. The shaping principle is to run LO-critical jobs only if their executions will not violate the feasible procrastination interval. With the shaping approach, LO-critical tasks are not prevented from executing in HI mode, but are constrained within a certain execution extent. To specify such extent, a term named LO-critical Budget (LO-B) is introduced. *LO-B*(t) at some time t is a safe upper bound on the total amount of time that the processor can work on LO-critical tasks after time t. In other words, all HI-critical tasks can meet their deadlines after time t if the processor execute HI-critical tasks for no more than LO-B time units.

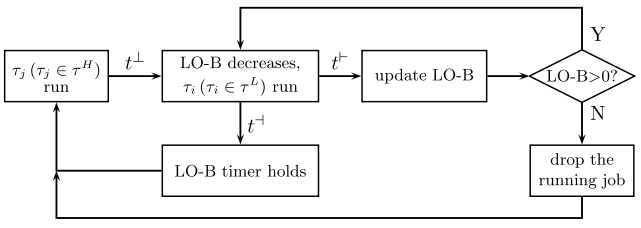


FIGURE 5. Workflow of the shaping approach in HI mode.

Analogous to Fig. 4, the system in HI mode will schedule HI- and LO-critical tasks as Fig. 5. First of all, we set up a LO-B timer. Based on the principle of earliest-deadline-first schedule, the task with the earliest deadline will be executed prior to other tasks. If this task is HI-critical, the system allows it to run without the constraint of *LO-B*(t). If it is LO-critical, its execution will be constrained by *LO-B*(t). Suppose at a time $t^{\perp}$ (see Fig. 5), the system starts to run a LO-critical task; meanwhile the *LO-B*(t) timer starts to decrease. The execution extent of this task depends on *LO-B*(t) timer. In a case that this task does not finish till LO-B timer times out (time instant $t^{\vdash}$), the system will update *LO-B*(t). The new updated *LO-B*(t) will either allow this task to run further if the new *LO-B*(t) is greater than zero, or drop it otherwise. In another case that this task finishes before LO-B timer times out (time instant $t^{\dashv}$), *LO-B*(t) will hold its current value at the task finishing time and be used for shaping future LO-critical tasks. Compared to the previous shaping approaches, the key advantage of our shaping approach is able to detect and use runtime processing slack as much as possible by updating *LO-B*(t) whenever it runs out.

The updating on *LO-B*(t) is similar to updating *HI-B*(t) in the mode-switch procrastination approach. First, we have to derive the total demand-bound function of HI-critical tasks because only they need to be sufficiently guaranteed schedulable in HI mode. The total demand-bound function of HI-critical tasks is

$$\mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta, t) = \sum_{\tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t) \quad (16)$$

Then by using Eq. 11 with $\mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta, t)$, we get the corresponding $\rho^*(t)$, which will be used to update *LO-B*(t).

Theorem 3: Suppose the HI mode demand-bound function of HI-critical tasks τ^H at time t is $\mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta, t)$, then by giving $\rho^*(t)$ computed by Eq. 11 to LO-B(t) at every updating step, all HI-critical tasks will meet their deadlines in HI mode.

Proof: Straightforwardly, since $\forall t \geq 0$, $[\![\Delta - \rho^*(t)]\!]_0 \geq \mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta, t)$, i.e., the provided processing resource is always greater than the demand of all HI-critical tasks, there will be no HI-critical task deadline miss. □

An example is presented below to illustrate how the proposed semi-slack scheduling works on a uniprocessor.

Example 1: Suppose there are three sporadic tasks in a uniprocessor system. Task parameters are shown as Tab. 2. This task set is tested to be schedulable with the assigned LO deadlines in table. $\delta_i(1)$ denotes the minimum interval between any two task activations.

TABLE 2. Task parameters.

τ_i	L_i	C_i^L	C_i^H	D_i^L	D_i	$\delta_i(1)$
τ_1	LO	4	-	8	8	8
τ_2	HI	2	4	9	11	11
τ_3	HI	1	6	5	14	14

The system runtime behavior is shown in Fig. 6. Suppose the three tasks are activated at the beginning. We assume that actual execution times for the three tasks are constant number, which are 3, 4, 5 from τ_1 to τ_3. From Tab. 2, we know that τ_2 and τ_3 have run over their LO WCET*s.*

We analyze three scheduling policies, as presented in Fig. 6. The first one presents the schedule of system driven by EDF with LO deadlines but without switching to HI mode.

In this situation, interfered by the LO-critical task τ_1, the first job of HI-critical task τ_2 misses its deadline because both τ_2 and τ_3 have executed over their LO WCETs*. The second Gante diagram presents the task execution with the mode-switch scheme. It shows that after the system enters into HI mode, HI-critical tasks will be scheduled by HI deadlines and LO-critical tasks will be dropped. In this situation, the first three jobs of τ_1 have been dropped. The third Gante diagram shows task executions with mode-switch scheme and together with the proposed semi-slack scheduling. In this situation, the mode-switch is postponed for 2 time units compared with the second diagram. By using HI-$B(t)$ and LO-$B(t)$, jobs of τ_1 are given some time budgets to run in HI mode. The schedule result demonstrates that all tasks can meet their deadlines.*

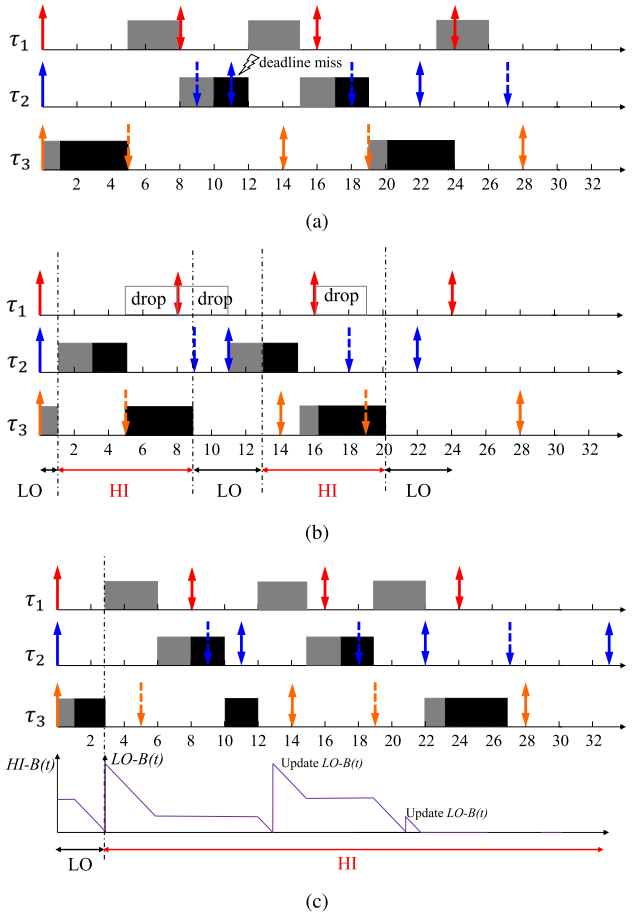


FIGURE 6. Example showing the runtime behavior of system under three scheduling policies. The dashed downward arrow denotes LO deadlines in LO mode. The black part denotes the time that a HI-critical task runs over its LO WCET. The change of *HI-B(t)* and *LO-B(t)* at runtime is also presented in the third diagram. The denotations of 'LO' and 'HI' at the bottom of each diagram represent LO and HI modes, respectively. (a) Without mode-switch. (b) With mode-switch but without deploying semi-slack scheduling. (c) With mode-switch and semi-slack scheduling.

From Fig. 6(a), we observe that the interference from LO-critical tasks results in the deadline miss of the HI-critical task. By switching the system mode, HI-critical tasks can meet their deadlines by sacrificing QoS of LO-critical tasks, as demonstrated in Fig. 6(b). To improve the QoS of LO-critical tasks as well as guaranteeing timing requirements of HI-critical tasks, we can deploy the semi-slack scheduling, because the two budgets $HI - B(t)$ and $LO - B(t)$ allow the system to stay in LO mode as long as possible and provides execution budget to LO-critical tasks in HI mode. The two features can significantly improve QoS of LO-critical tasks.

V. SEMI-SLACK SCHEDULING IN MULTICORES

This section presents how to extend the semi-slack scheduling to cope with multicores. The scheduling in multicore platform can be generally categorized into partitioned scheduling and global scheduling. The partitioned scheduling statically allocates tasks to individual cores and does not allow migration of tasks among processors, while global scheduling allows a task to execute on any processor at any time. The key problem of partitioned scheduling is how to allocate tasks among processors so that the system performance can be maximized, and global scheduling focuses on workload balance among processors. In this article, we suppose tasks have been allocated to processors and tested to be mixed-criticality schedulable with EDF+SSS in each core beforehand and propose a global scheduling strategy in order to fully use system free slack.

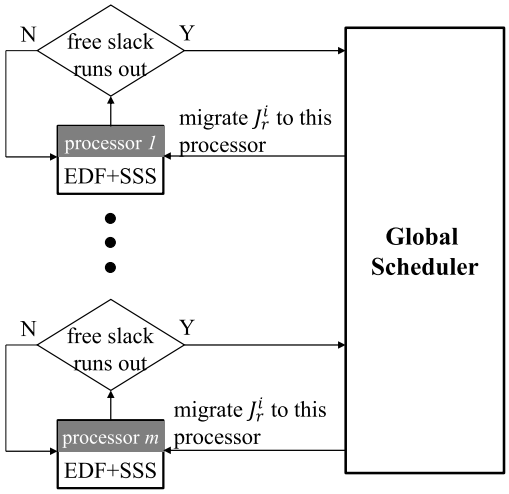


FIGURE 7. Scheduling structure in multicores.

The scheduling structure in multicores is shown in Fig. 7, where there are m identical cores in a multicore platform, and the scheduling framework is composed of one local scheduler that works on individual processor and one global scheduler that works on all processors. Each core has already been allocated LO-critical and HI-critical tasks and is scheduled by a local scheduler composed of the EDF algorithm assisted with our proposed semi-slack scheduling (SSS) approaches, where mode-switch is independent from each other. On top of all local schedulers, there is a global scheduler to manage the migration of jobs among all cores so that the free slack on each core can be used to serve jobs from other cores (we call it cross-use). It should be noted that in most of time, processors are scheduled by the local scheduler. A local scheduler can only be interfered by the depletion of free slack on a processor or by the job migration from another processor. The local

scheduler has been discussed in the previous section and our focus in this section is to design an efficient global scheduling approach that can cross-use free slack by migrating jobs among cores.

A. PROCEDURES

The main idea of applying global scheduler is to utilize other core's free slack when one core runs out its own free slack, in which way a core has less possibility to be switched to HI mode or to abandon its LO-critical jobs. The role of global scheduler is thus to migrate an overrun job or a LO-critical job from a core without free slack to another core with free slack.

Algorithm 1 Semi-Slack Scheduling Algorithm on Multicores

Input: A task set $\tau = \{\tau_1, \ldots, \tau_n\}$, the parameters $\forall \tau_i \in \tau$, (α_i or $\delta_i(q)$, $\mathbf{D_i}$, $\mathbf{C_i}$, L_i), the processors $P = \{P_1, \ldots, P_m\}$

Output: Task Executions

1: Schedule tasks on each core independently by the semi-slack scheduling scheme in uniprocessor; ▷ Local scheduling
2: **if** $\exists P_i \in P$, $P_i.free\_slack$ runs out **then**
3: $ChosenProID = 0, free\_slack\_Max = 0$;
4: **for** $k = 1..m$ **do**
5: **if** $P_k.free\_slack$ is not occupied && $P_k.free\_slack > free\_slack\_Max$ **then**
6: $ChosenProID = k$;
7: $free\_slack\_Max = P_k.free\_slack$;
8: **end if**
9: **end for**
10: **if** $ChosenProID > 0$ **then**
11: Migrate the job J_r that causes free slack running out in P_i to the processor $P_{ChosenProID}$, and assign the highest scheduling priority to it;
12: **else if** J_r is LO-critical **then**
13: J_r is dropped;
14: **else**
15: J_r is returned to its birth core and forces its birth core to HI mode.
16: **end if**
17: **end if** ▷ Global scheduling

By deploying the global scheduler, we still need to guarantee the system schedulability property, i.e., on each core all tasks can meet their deadlines in LO mode and HI-critical tasks will meet their deadlines in HI mode. Towards this goal, the scheduling algorithm is designed as Algo. 1. In the first line, tasks are locally scheduled by the semi-slack scheduling of uniprocessor on each core. The global scheduler starts to work only when a job uses up a core's free slack. In detail, the procedures of global scheduler in handling the case that a core runs out its free slack ($HI\text{-}B(t)$ or $LO\text{-}B(t)$) are designed as the following steps.

- Step 1: The global scheduler is activated whenever a job uses up a core's free slack. This job is marked as J_r, which can be either a HI-critical overrun job or a LO-critical job when the system is in HI mode (line 2).
- Step 2: Then the global scheduler finds out another core that satisfies the below two conditions. First, this core is not using its free slack at this moment. Second, its free slack is the largest among cores that meet the first condition (lines 3-9).
- Step 3: The global scheduler migrates J_r to the core found out by step 2 and executes J_r ahead of all other jobs in this core until J_r finishes or free slack in the chosen core runs out.
- Step 4: If J_r does not finish before free slack in the chosen core runs out, repeat step 2,3.
- Step 5: If there is not an available core to run J_r from step 2,3, J_r will be processed depending on its criticality. If J_r is a LO-critical job, it will be dropped immediately. If it is a HI-critical job, it will be returned to its birth core (the core that produces the job), in which case system of its birth core will be switched to HI mode (lines 12-16).

From above procedures, we note that the local scheduler, i.e., EDF+SSS, will not be interfered by global scheduler unless its free slack runs out or it is chosen to run a job migrated from another core. Whenever J_r finishes or is migrated to another place, the system that releases this job will be scheduled by EDF+SSS again.

The computation complexity of semi-slack scheduling in multicores includes two parts, i.e., local scheduling complexity and global scheduling complexity. The local scheduling complexity is $O(n \cdot log(n))$ because the local scheduler is the semi-slack scheduling algorithm in uniprocessor. The computation of global scheduler is to find the maximum free slack among all cores, which has $O(m)$ complexity. In this article, the overhead of migrating a job to another core is neglected as the aim of this article is to produce the proof-of-concept results. Nevertheless, in actual implementation, it can be accounted as an extra task execution time and consume the free slack.

We provide a simple example to illustrate how the global scheduler works.

Example 2: We deploy our proposed scheduling strategy in a dual-core system. Suppose each core has been allocated two tasks whose properties have been presented in the following table, where τ_i^j denotes the i-th task in core j. It can be observed that the two cores have the same tasks. We further suppose the actual execution time of each released job from those tasks are $\tau_1^0.act = \{4, 4, 4, 4\}$, $\tau_2^0.act = \{2, 6, 6\}$, $\tau_1^1.act = \{4, 4, 4, 4\}$, and $\tau_2^1.act = \{6, 4, 6\}$, where the k-th element in the set $\tau_i^j.act$ denotes the actual execution time of k-th job released from τ_i^j.

Core	task	L_i	C_i^L	C_i^H	D_i^L	D_i	$\delta_i(1)$
core 0	τ_1^0	LO	4	-	8	8	8
	τ_2^0	HI	2	6	6	10	10
core 1	τ_1^1	LO	4	-	8	8	8
	τ_2^1	HI	2	6	6	10	10

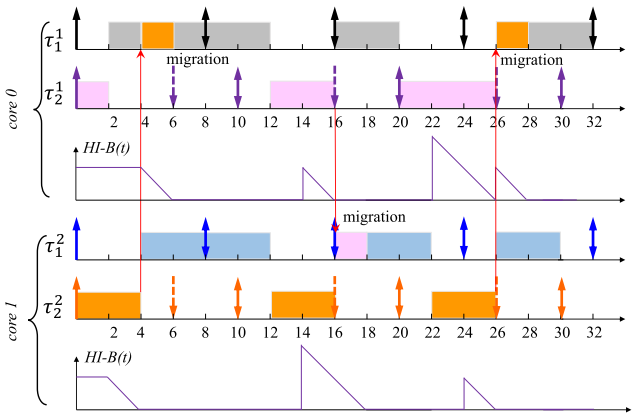


FIGURE 8. Example showing the runtime behavior of a dual-core system with our proposed global scheduler and local scheduler. The dashed downward arrow denotes LO deadlines in LO mode.

We find that the two cores have been overloaded at different moments. If each core works independently with the local scheduler, i.e., EDF+SSS, τ_1^0 and τ_1^1 must abandon one or two jobs. Considering that the overload in two cores happens in different moments, we can apply the global scheduler to cross-use the free slack so that the workload in the two cores can be balanced. As shown in Fig. 8, at $t = 4$, core 1 has used up its free slack. Because core 0 has remaining free slack, τ_2^2 will be migrated to core 0 and be assigned with the highest priority. This task will preempt other tasks and execute until its finish. Analogously, when core 0 has run out its free slack, the overrun task will be migrated to core 1. Our proposed global scheduler performs very well in maintaining the timing requirement and keeping system in LO mode.

B. SYSTEM PROPERTY ANALYSIS

In Section IV, we have proved that EDF+SSS can guarantee all tasks meet deadlines in LO mode and HI-critical tasks meet deadlines in both modes. Now we analyze the influence of global scheduler on the individual system with EDF+SSS. After deploying global scheduler, there are two main changes to each core. First, a core may execute a job from another core. Second, a core may release a job to another core and perhaps retake the released job. For the first change, since the execution on a migrated job is constrained within the free slack, the system schedulability property will not be changed. For the second change, after a job has been migrated to another place, its birth core will be continuously scheduled by EDF+SSS and its schedulability is also guaranteed until the back of this migrated job. Then, we only need to analyze the migrated job.

There are three cases that could happen to the migrated job. First, the migrated job finishes in step 3. Second, when it goes to step 5, it is dropped if it is a LO-critical job. In the third case if it is a HI-critical job, it will be returned to its birth core and forces the system on the birth core to switch to HI mode. Here we should note that if the criticality of migrated job is LO-critical, this job must come from the HI-mode system, because LO-critical jobs do not need free slack when the system is in LO mode. Since the LO-critical job does not need to be guaranteed meeting its deadline in HI mode, the migrated LO-critical job can be dropped in the second case without violating the mixed-criticality schedulability property.

Hence, we only need to prove that the HI-critical migrated job will meet its deadline in the first case, and in the third case the system of its birth core can still meet the schedulability in HI mode. Formally, we denote a job J_r^i as a HI-critical migrated job from core i and we suppose its execution finishes in another core j ($i \neq j$). In addition, we further denote the LO WCET, HI WCET, LO deadline, HI deadline and actual execution time of J_r^i as $C_{J_r^i}^L$, $C_{J_r^i}^H$, $D_{J_r^i}^L$, $D_{J_r^i}^H$ and $J_r^i.act$, respectively, where $J_r^i.act \leq C_{J_r^i}^H$.

Corollary 2: If tasks in i-th core is mixed-criticality schedulable, the inequality $D_{J_r^i}^H - D_{J_r^i}^L \geq C_{J_r^i}^H - C_{J_r^i}^L$ holds.

Proof: We prove it by contradiction. Suppose $D_{J_r^i}^H - D_{J_r^i}^L < C_{J_r^i}^H - C_{J_r^i}^L$. There could be a case that J_r^i has finished $C_{J_r^i}^L$ execution and this moment is just at $D_{J_r^i}^L$. Then the system will switch to HI mode, and it further needs $C_{J_r^i}^H - C_{J_r^i}^L$ execution to finish. Because $D_{J_r^i}^H - D_{J_r^i}^L < C_{J_r^i}^H - C_{J_r^i}^L$, this task may miss its HI mode deadline, which is contradicted with the mixed-criticality schedulable result. □

Corollary 3: Once J_r^i leaves its birth core, J_r^i will continuously execute until its finish.

Proof: From the line 11 of Algo. 1, except in its birth core, the migrated job will be always assigned the highest priority in any core. Therefore, before J_r^i returns to its birth core, J_r^i will continuously execute until its finish. □

Lemma 1: J_r^i will meet its HI deadline.

Proof: Theorem 2 has showed that all tasks will meet their LO deadlines, which means that the execution of J_r before it leaves its birth core is more than C_{Jr}^L. As $J_r^i.act - C_{J_r^i}^L \leq C_{J_r^i}^H - C_{J_r^i}^L$, J_r will need execution less than $C_{Jr}^H - C_{Jr}^L$ to finish. According to Corollary 3, once J_r^i leaves its birth core, J_r^i will be continuously processed until its finish. By constantly executing J_r after it leaves its birth core, it can meet D_{Jr}^H because we have $D_{Jr}^H - D_{Jr}^L \geq C_{Jr}^H - C_{Jr}^L$ according to Corollary 2. □

We have proved that the HI-critical migrated job will meet its deadline if this job finishes in step 3. We next prove that if the HI-critical migrated job returns to its birth core and forces the system of this birth core to HI mode, this system can still be schedulable.

Lemma 2: Under the global schedule, if the migrated job J_r^i returns to its birth core and forces the system of its birth core to HI mode, this system can still be schedulable after entering into HI mode.

Proof: For representation simplicity, we denote the task that releases J_r^i as $\tau_{J_r^i}$. We have proved in Corollary 1 that the local semi-slack scheduling is able to guarantee $\forall \tau_i \in \tau^H$, $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$, where t_s is the mode-switch time. Hence, in the birth core of J_r^i, for the task

$\forall \tau_i \in \tau^H \backslash \{\tau_{J_r^i}\}$, we also have $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ because those tasks $\tau_i \in \tau^H \backslash \{\tau_{J_r^i}\}$ are only under the local semi-slack scheduler. For the task $\tau_{J_r^i}$, since J_r^i constantly executes before it returns to its birth core, the HI mode demand of $\tau_{J_r^i}$ at t_s will also be less than its offline demand-bound function $\mathrm{dbf}_{\mathrm{HI}}(\tau_{J_r^i}, \Delta)$. As a result, we have $\forall \tau_i \in \tau^H, \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ and further prove that $\sum_{\tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, t_s) \leq \sum_{\tau_i \in \tau^H} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \leq \Delta$. Therefore, the system is still schedulable afer entering into HI mode. □

VI. EVALUATIONS

We now evaluate the performance of our proposed semi-slack scheduling approach with state-of-art scheduling techniques by extensive simulations. To this aim, two types of task sets are randomly generated for the evaluation, which are sporadic task sets and arbitrarily activated tasks.

A. COMPARED APPROACHES

In uniprocessor platform, the semi-slack scheduling has been presented to perform two functionalities: postponing the mode-switch and shaping LO-critical jobs in HI mode. In order to compare the performance of semi-slack scheduling on the two functionalities respectively, we evaluate approaches with the following combinations:

- EDF+MSP: the EDF approach assisted with mode-switch procrastination.
- EDF+Shaping: the EDF approach assisted with shaping approach in HI mode.
- EDF+SSS: the EDF approach assisted with full semi-slack scheduling.

In order to evaluate the performance of our proposed approaches on uniprocessor, we choose seven existing scheduling techniques, including two popular scheduling approaches as baselines, i.e., EDF-VD and AMCmax; and seven state-of-art approaches that are also able to improve the QoS of LO-critical tasks, i.e., MEBA, BPSG, Service-Adaption, Elastic, Isolation-Scheme, ICG and MC-ADAPT. In the following, we compare their performance separately, because these approaches cannot deal with burst arrival pattern.

- EDF-VD: The basic EDF-VD scheduling that forces mode-switch whenever a HI-critical task overruns [2].
- AMCmax: The basic fixed-priority scheduling with the AMCmax schedulability test. Details of this test can be seen in [32].
- MEBA (short for Maximum Execution-based Budget Allocation): The dynamic mixed-criticality model that works under EDF-VD. Based on the system runtime behavior, MEBA dynamically determines the budget to C_i^L of a HI-critical task at runtime. Compared to fixing C_i^L offline, MEBA has been proved performing better in postponing mode-switch [17].
- BPSG (short for Bailout Protocol with Slack and Gain Time): This is an enhanced bailout protocol by increasing the execution budgets of C_i^L offline, and via reclaiming gain time as well. This protocol works under fixed-priority schedule and its main purpose is to timely switch system from HI mode back to LO mode, instead of waiting for an idle tick. Details of this approach are in [33]. The effectiveness of this protocol is demonstrated to be more effective than most of other state-of-art scheduling approaches in [33].
- Service-Adaption: The service to LO-critical tasks is degraded after the system enters into HI mode [34].
- Elastic: The periods and deadlines of LO-critical tasks are increased after the system enters into HI mode. This is called elastic task model [35].
- Isolation-Scheme: A mechanism is designed to provide the necessary isolation to support the execution of low critical tasks [36].
- ICG (short for interference constraint graph): ICG specifies the allowed interferences between tasks and makes it possible to systematically reduce the number of tasks that can be dropped [19].
- MC-ADAPT: Analogous to our proposed approach, MC-ADAPT adaptively drop LO-critical tasks based on the runtime schedulability analysis [18].
- Offline-Shaping: This approach considers the mixed-criticality system as a system mixed with soft and hard rea-time tasks. An offline calculated bound is used to regulate the execution of soft real-time tasks so that hard real-time tasks can always meet their timing requirements [21].
- Online-Shaping: Under the same interpretation as offline-shaping on mixed-criticality system, the online-shaping updates the bound on running LO-critical tasks at runtime so that the dynamic free slack can be used [22].

B. RANDOM TASK SET GENERATION

We generate the task set in the same way as [2]. In order to generate tasks including sporadic tasks, a random task set is generated as a *pjd* activation task by starting with an empty task set $\tau = \emptyset$, where random tasks are successively added and *pjd* activation pattern is defined as $\overline{\alpha}_i^u(\Delta) = \min\{\lceil \frac{\Delta + j_i}{p_i} \rceil, \lceil \frac{\Delta}{d_i} \rceil\}$.

- The task set utilization is a value of $(x + 0.5)/30$, where $x \in \{0, 1, \ldots, 29\}$.
- The probability of a random task being HI-critical is 0.5.
- C_i^L is drawn from the uniform distribution over $\{1, 2, \ldots, 10\}$.
- C_i^H is drawn from the uniform distribution over $\{C_i^L, C_i^L + 1, \ldots, 4 \cdot C_i^L\}$ if $L_i = HI$.
- The period p_i is drawn from the uniform distribution over $\{C_i^H, C_i^H + 1, \ldots, 200\}$.
- The jitter j_i is set as $\mathcal{J} \cdot p_i$.
- The minimum inter distance d_i is set as $\mathcal{D} \cdot p_i$.
- The relative deadline is set to $D_i = p_i$.

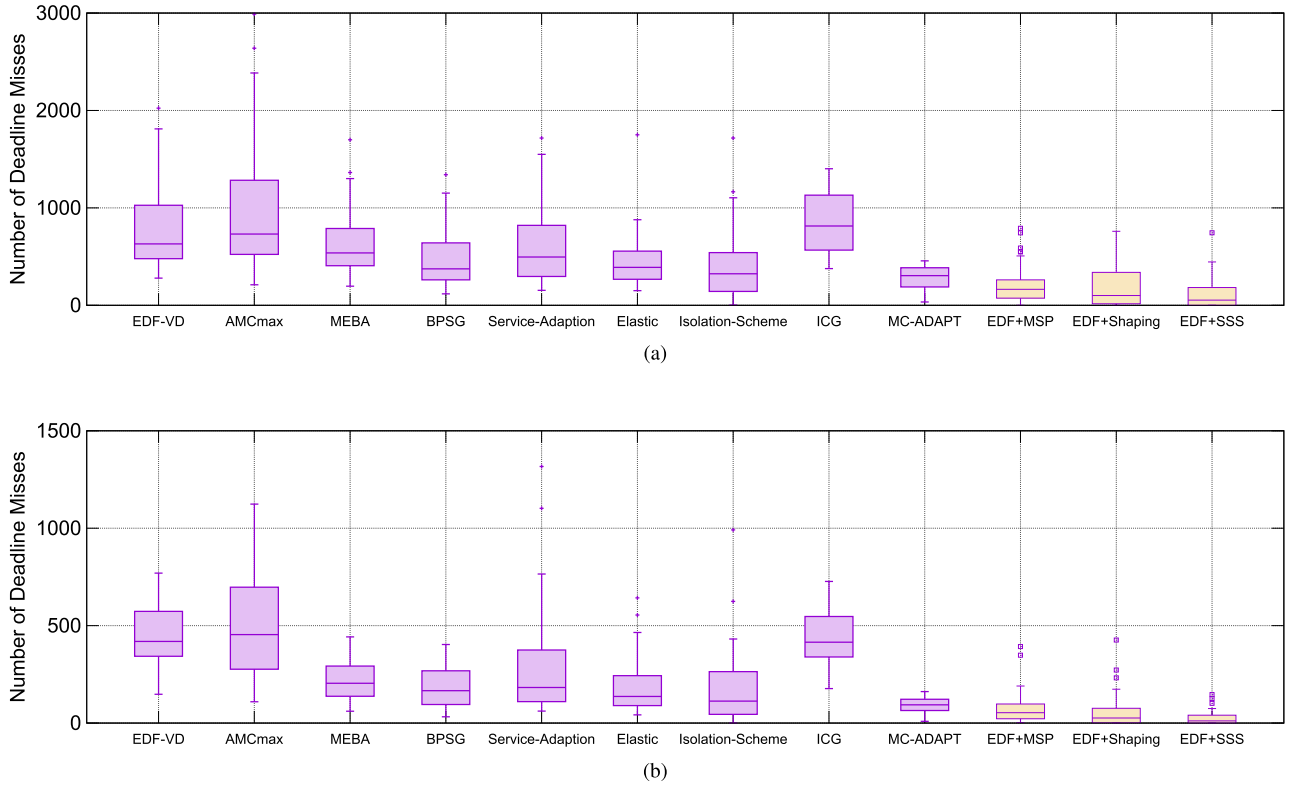


FIGURE 9. The boxplot of the deadline misses in 10^7 time units simulation. Tasks in simulation are sporadically activated. (a) $f_a = 1.2$, $f_e = 0.8$. (b) $f_a = 1.8$, $f_e = 0.2$.

We define $U_{LO}(\tau) = \sum_{\tau_i \in \tau} \left(C_i^L/p_i\right)$, and $U_{HI}(\tau^H) = \sum_{\tau_i \in \tau^H} \left(C_i^H/p_i\right)$, where τ includes both LO-critical and HI-critical tasks, and τ^H only include HI-critical tasks. The task set utilization is defined as $U(\tau) = \Big(U_{HI}(\tau^H) + U_{LO}(\tau)\Big)/2$. For every task set generation, the utilization is allowed to be located in $[U^* - 0.005, U^* + 0.005]$, where U^* is a targeted utilization. If the generated utilization is smaller than $U^* - 0.005$, a new random task is added. If the generated utilization is greater than $U^* + 0.005$, this task set is discarded, and a new empty task set is started, until a task set with the allowed utilization is found.

C. SEMI-SLACK SCHEDULING PERFORMANCE

1) SEMI-SLACK SCHEDULING IN UNIPROCESSOR

Conceptually, the advantage of adopting the semi-slack scheduling over other approaches lies in making use of the system free slack. Hence, the more free slack there is, the better that system performance can be improved. The free slack can be categorized into two types: static slack and dynamic slack. Due to the under-utilization of system, there exits some free static slack. The dynamic slack arises from the actual execution time or activation frequency being less than the worst-case assumption. In this section, we investigate the effect of dynamic slack on scheduling sporadic tasks and arbitrarily activated tasks.

By configuring $\mathcal{J} = 0$ and $\mathcal{D} = 1$, tasks generated by above procedures will be sporadically activated. We randomly generate 50 feasible task sets, where in every task set its $U_{LO}(\tau) = 0.75$, $U_{HI}(\tau) = 0.95$. A task set is feasible in the sense that they are schedulable by our proposed scheduling approaches and all the compared approaches. At runtime, a HI-critical job has 0.01 probability to overrun its LO WCET and a LO-critical job will always be finished by its LO WCET. If a job does not overrun, its actual execution time is evenly distributed in $[f_e \cdot C_i^L,\ C_i^L]$, where $f_e \in \{0.2, 0.4, 0.6, 0.8\}$. The runtime activation frequency is also relaxed. Suppose d_i^t represent the theoretically minimum distance between two events from τ_i at time t. Then during the runtime simulation, the actual distance of two events at t is a random number within $[d_i^t,\ f_a \cdot d_i^t]$, where $f_a \in \{1.2, 1.4, 1.6, 1.8\}$. From the above setting, we know that f_e and f_a represent the extent of free slack that can be generated at runtime. By configuring $f_a = 1.2$, $f_e = 0.8$ and $f_a = 1.8$, $f_e = 0.2$, the least free slack and the most free slack can be generated during simulation. For every task set, we simulate 10^7 time units and collect the number of deadline misses (dropped jobs are also counted as deadline misses).

The simulation results are presented in Fig. 9. In both configurations, the result presents that the number of deadline misses with our proposed approaches is 3-4 folds less than other state-of-art approaches. The performance between EDF+MSP and EDF+Shaping is very close because the

functionalities of both approaches are to exploit the free slack in system to serve LO-critical tasks. The approach EDF+SSS performs slightly better than the two other our proposed approaches. This is because the free slack in system has been exploited by both approaches so efficiently that not much free slack can be left by the counterpart approach. Besides, considering that the number of deadline misses in Fig. 9(b) is almost half of deadline misses of Fig. 9(a), we find that less LO-critical jobs will miss their deadlines when the system free slack increases, which is in accordance with our expectation.

Regarding to results of other existing approaches, EDF-VD and AMCmax perform the worst because they directly drop all LO-critical tasks whenever a task overruns, without taking any free slack into consideration. Although MEBA works under EDF-VD, it dynamically determines the value of C_i^L at runtime, in which way the mode-switch can be postponed and thus save LO-critical jobs. BPSG works under fixed-priority schedule and saves the LO-critical jobs by switching the system from HI mode back to LO mode as soon as possible. Both MEBA and BPSG rely on the offline schedulability analysis and can only use free slack to a limited extent. The two approaches of Service-Adaption and Elastic improve the system performance by tuning parameters of running LO-critical tasks in HI mode. The Isolation-Scheme proposes a component-based scheduling strategy that supports the execution of LO-critical tasks while simultaneously protects HI-critical tasks. Although this approach provides a very tight schedulability test, it fails to use the system free slack at runtime and leads to a big loss in running LO-critical tasks. ICG approach only provides a graph-based schedulability analysis, which is also not able to use the free slack. The inner working scheme of MC-ADAPT is very close to our proposed EDF+MSP approach and performs better than other existing approaches. There are two main differences between MC-ADAPT and EDF+MSP. One is that MC-ADAPT relies on an utilization-based schedulability test, and EDF+MSP relies on demand-based schedulability test. As demand-based schedulability test is tighter but more complex than utilization-based schedulability test, EDF+MSP performs better in using free slack but has larger complexity compared to MC-ADAPT. The other difference is that EDF+MSP drops the overrun job, while MC-ADAPT drops the LO-critical task. As a result, EDF+MSP allows tasks to constantly release jobs, but the dropped tasks in MC-ADAPT will not release jobs until its recovery.

In above experiments where sporadic workload is considered, we have demonstrated that our approach outperforms all the existing approaches because the system free slack is aggressively exploited to run LO-critical jobs. As semi-slack scheduling approaches are superior in scheduling arbitrary activation tasks, we next investigate the scheduling effect of our proposed approaches on complex activated tasks by configuring $\mathcal{J} = 3$ and $\mathcal{D} = 0.2$. MC-ADAPT is chosen to compare with our approaches because it has a relatively higher performance than other existing approaches. We use

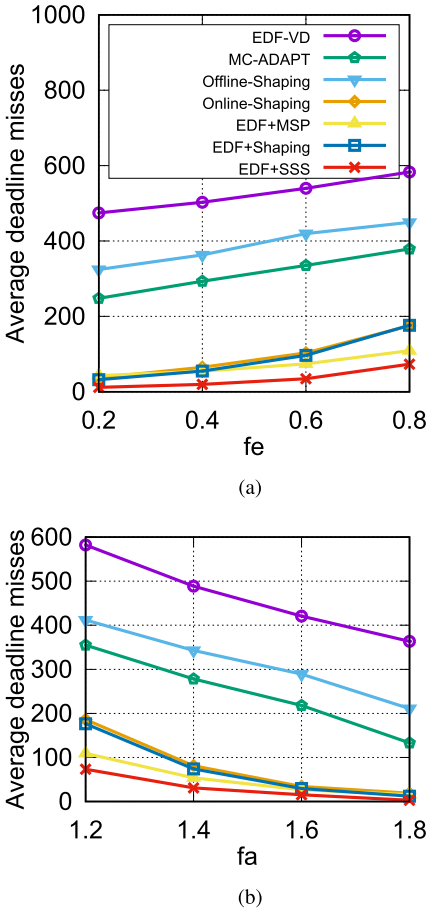


FIGURE 10. Average number of deadline misses in two different settings, where both figures share the same line scheme. (a) $f_a = 1.2$. (b) $f_e = 0.8$.

the minimum distance D between two successive activations to define the sporadic task model in order to accommodate MC-ADAPT scheduling model. The deadline misses with regard to actual execution time (f_e) and activation (f_a) are individually investigated. As shown in Fig. 10, we either fix f_a to investigate the effect of f_e or fix f_e to investigate the effect of f_a. Fig. 10 shows that all three approaches of deploying our proposed slack scheduling outperform the EDF-VD, MC-ADAPT and offline-shaping to a large extent, which confirms that the semi-slack scheduling is very effective to improve the system performance. Fig. 10 also shows that online-shaping, EDF+MSP and EDF+shaping can achieve almost the same performance on supporting LO-critical service, but are inferior to EDF+SSS. This is expected because EDF+SSS reclaims the free slack no matter which mode the system is. The drawback of EDF+Shaping is that it considers LO-critical tasks as soft real-time tasks without providing a certain guarantee to LO-critical tasks.

2) SEMI-SLACK SCHEDULING IN MULTIPROCESSORS

We now present the semi-slack scheduling results in multiprocessor platforms. Since it is not clear how to extend existing approaches to multiprocessor in a global scheduling way, we can only evaluate their performance in multipro-

cessor in the partitioned scheduling. As the performance of partitioned scheduling is the same as the scheduling in uniprocessor, we skip it. In multiprocessors our proposed approaches can be either deployed as the global schedule to cross-use the free slack among different cores or as the local schedule to implement the semi-slack scheduling in every core. Hence we compare these two approaches: EDF schedule assisted with full semi-slack scheduling in every core (E+S); on top of the first approach (E+S), the system is deployed with global schedule to cross-use free slack (E+S+C) in the second approach.

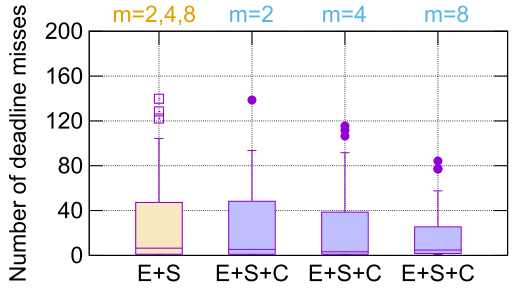


FIGURE 11. Boxplot of deadline misses per core with regard to the number of cores, where *m* represents core number.

We present our results in Fig. 11, where 2-core, 4-core and 8-core multiprocessors are investigated. Here we list the distribution of deadline misses per core among 50 task sets when the task set in every core satisfies $f_a = 1.2$, $f_e = 0.8$, $U_{LO}(\tau) = 0.75$, $U_{HI}(\tau) = 0.95$. For the approach E+S, the scheduler of every core works independently from each other. Thus the scheduling performance of each core only depends on the allocated tasks by itself. We observe that the deadline miss distributions on every core is the same across different multiprocessors because the system utilization in each core keeps the same. While for the approach E+S+C, the number of deadline misses decreases with increasing core number, because the extra free slack of one core can be used to compensate the lack of free slack on other cores. When the core number increases, there will be more extra free slack to be compensated on different cores. As a result, the number of deadline misses decreases.

VII. CONCLUSION AND FUTURE WORKS

In this article, we presented a semi-slack scheduling framework that is able to reclaim system free slack to serve the overrun HI-critical job or LO-critical job, in which way the system mode-switch is postponed as much as possible and LO-critical jobs can still run in HI mode. This framework has been extended to work in multiprocessor systems. We also theoretically prove that the semi-slack scheduling can maintain the mixed-criticality schedulability property, i.e., all tasks can meet their deadlines in LO mode and HI-critical tasks meet their deadlines in HI mode. At last, the effectiveness of our proposed schedulability analysis and semi-slack scheduling has been confirmed by extensive simulations.

In the future, we are going to extend the semi-slack scheduling framework from dual-criticality systems to multiple-criticality systems. We can also extend the independent task model to a task graph model. Besides, the scheduling performance can be evaluated by implementing it in real-world platform.

APPENDIX A
SCHEDULABILITY TEST

A. SCHEDULABLE CONDITIONS

The schedulable conditions are,

$$\text{Condition 1} : \forall \Delta \geq 0 : \sum_{\tau_i \in \tau} \text{dbf}_{\text{LO}}(\tau_i, \Delta) \leq \Delta,$$

$$\text{Condition 2} : \forall \Delta \geq 0 : \sum_{\tau_i \in HI(\tau)} \text{dbf}_{\text{HI}}(\tau_i, \Delta) \leq \Delta.$$

Therefore, to determine the schedulability of a task set, one only needs to derive the DBFs in both modes.

B. DEMAND-BOUND FUNCTION IN LO MODE

If the system is in LO mode, every task behaves as a normal task with parameters $(\bar{\alpha}_i^u(\Delta)$ or $\delta_i(q)$, C_i^L, $D_i^L)$. According to the framework of real-time calculus [11], a tight demand bound function (DBF) of a task τ_i is that

$$\text{dbf}_{\text{LO}}(\tau_i, \Delta) = \bar{\alpha}_i^u(\Delta - D_i^L) \cdot C_i^L. \tag{17}$$

C. DEMAND-BOUND FUNCTION IN HI MODE

The derivation of DBF in HI mode needs to know the maximum required execution times within any time interval in HI mode to guarantee that events within this interval can meet their deadlines, where events of an interval may include the carry-over jobs (released but not finished at the mode-switching time). In order to know $\text{dbf}_{\text{HI}}(\tau_i, \Delta)$, we need to upper-bound the maximum number of deadlines of τ_i within any interval of a length Δ in HI mode. To do so, the as-early-as-possible event arrivals are assumed because this assumption is able to accumulate the most number of deadlines in the shortest time. Hence, when given a time length Δ in HI mode, the maximum number of deadlines is $\bar{\alpha}_i^u(\Delta)$ and the demand is $\bar{\alpha}_i^u(\Delta) \cdot C_i^H$. However, this demand ignores that events may have been finished earlier than their deadlines if those events need to keep schedulable. To explore those finished executions, we introduce the effective deadlines to define the timing constraints that are able to guarantee the system schedulable in LO mode.

1) EFFECTIVE DEADLINES

The timing constraints require that the finishing time of any event must not exceed its absolute deadline, where the absolute deadline of an event is the time point that is the sum of its arrival time and its relative deadline.

For some event streams, their events may be required to be finished earlier than their absolute deadlines so that there will be enough slacks for processing following events. The effective deadline is thus defined to show this timing constraint.

Definition 4 (Effective Deadline): The effective deadline of an event is referred to its allowable largest finish time that guarantees other events of this task meet their absolute deadlines.

We use $AD(e)$ and $ED(e)$ to respectively denote the absolute deadline and the effective deadline of an event e.

Example 3: An example is used to explain the differences between effective deadlines and absolute deadline. For a task τ_i activated by a pjd event stream with $p = 10, j = 30, d = 2$, the as-early-as-possible arrival events are shown in Fig. 12. Suppose the relative deadline D_i^L is 7 and every event needs 3 time units to be processed, then the absolute deadlines of e_1, e_2, e_3, e_4 are 7, 9, 11, 13, while their effective deadlines are 4, 7, 10, 13 as every released event should be given 3 processing time units. Hence, e_1, e_2, e_3 are required to be finished before $ED(e_1)$, $ED(e_2)$, $ED(e_3)$.

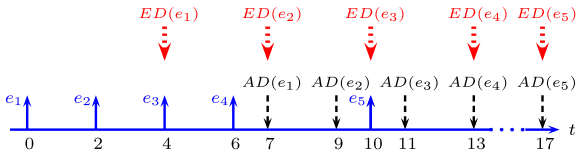


FIGURE 12. Event trace, absolute deadlines, and effective deadlines of a task τ_1 activated in the *pjd* pattern of $p = 10, j = 30, d = 2$.

From this example, we find that the latest finishing times of events are constrained by the effective deadlines, instead of absolute deadlines. Therefore, to explore the exact executions of events in LO mode, we rely on the information of effective deadlines.

2) MINIMUM DISTANCE FUNCTION OF EFFECTIVE DEADLINES

First, the MDF of event trace can be generally categorized to be two types. One type fulfills that $C_i^L \leq \delta_i(1)$, which means that the WCET is not greater than the minimum inter-activation interval. The effective deadlines are the same as the absolute deadlines for this type. The other type fulfills that $C_i^L > \delta_i(1)$, which means that the WCET is greater than the minimum inter-activation interval. The effective deadlines will be different with absolute deadlines. The MDF of event trace is generalized as follows

$$h = \min\{q | \delta_i(q+1) - \delta_i(q) > C_i^L\}. \tag{18}$$

where $\delta_i(0) = 0$, $q \in \mathbb{N}$. If $h = 0$, this is the type of $C_i^L > \delta_i(1)$. Otherwise, it is the type of $C_i^L \leq \delta_i(1)$.

In Fig. 13, $\delta_i'(k)$ represents the MDF of effective deadlines. Since the effective deadlines of first h events are separated by C_i^L and the effective deadlines of later events are the same as the absolute deadlines, the $\delta_i'(k)$ can be generalized as follows:

$$\delta_i'(k) = \begin{cases} k \cdot C_i^L, & k \leq h \\ h \cdot C_i^L + \delta_i(k) - \delta_i(h), & k > h, \end{cases} \tag{19}$$

where $\delta_i(h+1) - \delta_i(h) > C_i^L, k \in \mathbb{N}^+$.

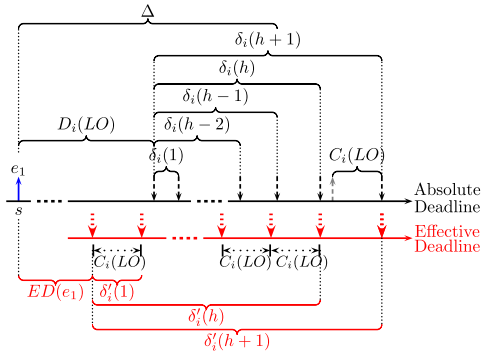


FIGURE 13. The absolute deadlines and effective deadlines corresponding to the as-early-as-possible event trace.

3) DERIVING DEMAND-BOUND FUNCTION

The idea of deriving the DBF of HI mode is based on the condition that the task set is schedulable in LO mode (i.e., Condition 1 holds). When we derive the DBF of HI mode, we therefore assume that all effective deadlines are met in LO mode.

The maximum number of effective deadlines within an interval of Δ depends on the MDF $\delta_i'(k)$ of effective deadlines. We set $D_i^H = D_i^L$ to keep the relative deadlines unchanged after the mode-switch. Hence, the effective deadlines are also not changed after mode-switch. As shown in Fig. 14, within an interval Δ that $\delta_i'(k) \leq \Delta < \delta_i'(k+1)$, the DBF is at most $(k+1) \cdot C_i^L$. In this interval, there might be an event that has already been executed before the mode-switch.

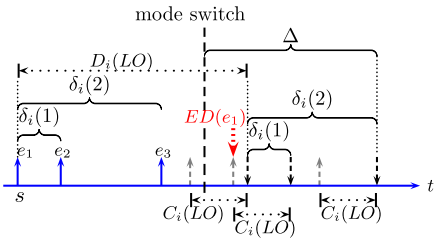


FIGURE 14. Bounding the demand of an event trace.

Lemma 3: In the interval of Δ that satisfies $\delta_i'(k) \leq \Delta < \delta_i'(k+1)$ in HI mode, if the system needs to schedule $k+1$ events, the execution time of these events before the mode-switch is not smaller than $[\![C_i^L - (\Delta - \delta_i'(k))]\!]_0$.

Proof: For the time interval of length Δ, there are at most $x = \Delta - \delta_i'(k)$ time units left for the "first" event (the first event in HI mode) if the system schedules $k+1$ events in HI mode. If $x \geq C_i^L$, the "first" event may not be executed before the mode-switch because there is enough interval for scheduling the "first" event. If $0 \leq x < C_i^L$, the execution time of the "first" event is at least $C_i^L - x$ because the "first" event is supposed to meet its effective deadline in LO mode. In general, $[\![C_i^L - (\Delta - \delta_i'(k))]\!]_0$ represents the least execution time. □

According to Lem. 3, the DBF of HI mode is thus concluded as follows:

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) = (k+1)\cdot C_i^H - [\![C_i^L - (\Delta - \delta_i'(k))]\!]_0,$$

where $\delta_i'(k) \le \Delta < \delta_i'(k+1)$ and $\delta_i'(0) = 0$.

D. DEMAND-BOUND FUNCTION TUNING

Inspired by the EDF-VD [2], by artificially decreasing the LO mode deadline D_i^L, the DBF of HI mode can be decreased. We can apply the following theorem to tune $dbf_{HI}(\tau_i, \Delta)$.

Theorem 4: If a HI-critical task τ_i is schedulable in LO mode, its DBF *of HI mode is that:*

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) = \begin{cases} 0, & \text{if } 0 \le \Delta < D_i^H - D_i^L \\ (k+1)\cdot C_i^H - [\![C_i^L - x]\!]_0, & \\ \text{where } x = \Delta - \delta_i'(k) - \left(D_i^H - D_i^L\right) & \\ \text{and } \delta_i'(k) \le x + \delta_i'(k) < \delta_i'(k+1). & \end{cases} \tag{20}$$

Proof: After the mode-switch, there is at least an interval of $D_i^H - D_i^L$ within which there is no deadline. Hence, if $0 \le \Delta < D_i^H - D_i^L$, $dbf_{HI}(\tau_i, \Delta) = 0$. If $\Delta \ge D_i^H - D_i^L$, within an interval of Δ that $\delta_i'(k) \le \Delta - \left(D_i^H - D_i^L\right) < \delta_i'(k+1)$, there are at most $k+1$ effective deadlines. Hence, $dbf_{HI}(\tau_i, \Delta) \le (k+1)\cdot C_i^H$. If the system needs to schedule $k+1$ events during this interval, the "first" event will probably have been executed before the mode-switch. According to Lem. 3, the execution time is not smaller than $[\![C_i^L - (\Delta - \delta_i(k))]\!]_0$ in the case that $D_i^H = D_i^L$. In the case that $D_i^L < D_i^H$, $x = \Delta - \delta_i'(k) - \left(D_i^H - D_i^L\right)$ is the time interval for scheduling the "first" event to meet the D_i^L. As D_i^L is met, the execution time is $[\![C_i^L - x]\!]_0$. Hence, $dbf_{HI}(\tau_i, \Delta) = (k+1)\cdot C_i^H - [\![C_i^L - x]\!]_0$ where $\delta_i'(k) \le x + \delta_i'(k) < \delta_i'(k+1)$. □

Based on Theorem 4, $dbf_{HI}(\tau_i, \Delta)$ can be tuned by changing the D_i^L. In order to make this dual-criticality system schedulable, conditions 1, 2 should hold.

VIII. FUTURE EVENT PREDICTION BY DYNAMIC COUNTERS

In principle any (discrete) complex arrival pattern can be bounded by a set of upper and lower staircase functions of the form $\overline{\alpha}_i^u(\Delta) = N_i^u + \lfloor \frac{\Delta}{\delta_i^u} \rfloor$ [31]:

$$\forall \Delta \in \mathbb{R}_{\ge 0} : \overline{\alpha}^u(\Delta) \le \min_{i=1..n} \{\overline{\alpha}_i^u(\Delta)\}.$$

A dynamic counter (DC_i) can be used to track a single upper staircase function ($\overline{\alpha}_i^u$). The detail tracking algorithm can be seen in Algo. 2 [27]. DC_i tracks the potential burst capacity, and the auxiliary variable k_i in Algo. 2 tracks the offset between the current time t and the last δ_i^u. Below shows an example of using dynamic counters for event prediction.

Example 4: As shown in Fig. 15, for the PJD task with $(P, J, D) = (100, 300, 20)$, two staircase functions, $\overline{\alpha}_1$ and $\overline{\alpha}_2$, are used to approximate the arrival curve of this PJD task. Every staircase function is tracked by a counter. Assume the real arrival event trace is shown in the event trace R^{act}. By applying Algo. 2 [27], DC_1 tracks the arrival event trace based on $\overline{\alpha}_1$, and DC_2 tracks the arrival event trace based on $\overline{\alpha}_2$. The minimum of $DC_1(t)$ and $DC_2(t)$ is the potential activations of this task at time t.

Algorithm 2 Implement a Dynamic Counter to Track a Staircase Function

Input: signal s, $\triangleright tuple < DC_i, CLK_i >$;

```
if s = eventArrival then
    if DC_i = N_i^u then
        reset_timer(CLK_i, δ_i^u)
        k_i = 0
    end if
    DC_i ← DC_i − 1
end if
if s = CLK_iTimeout then
    DC_i ← min(DC_i+1, N_i^u)
    reset_timer(CLK_i, δ_i^u)
    k_i = k_i + 1
end if
if DC_i < 0 then
    report_exception
end if
```

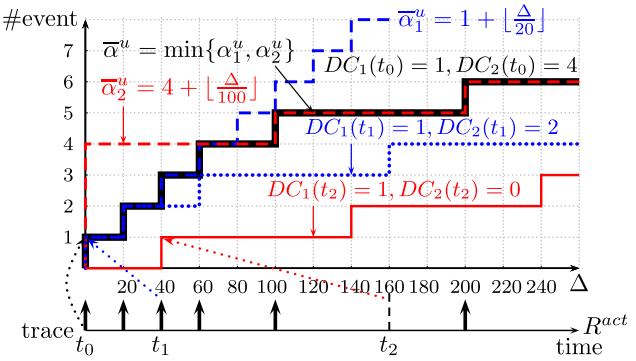


FIGURE 15. An example for using dynamic counters to predict the future events.

As shown in Fig. 15, the bold line shows the worst-case arrival pattern at the beginning, $DC_1(t_0) = N_1^u = 1$, $DC_2(t_0) = N_2^u = 4$. At time t_1, two events have been recorded, and dynamic counters are updated to that $DC_1(t_1) = 1$, $DC_2(t_1) = 2$. The future event prediction is shown in the dotted line, which is less than the worst-case assumption. At time t_2, dynamic counters are updated to that $DC_1(t_2) = 1$, $DC_2(t_2) = 0$ since five events arrived during $[t_0, t_2]$. The prediction shown in the solid line is further less than the one at time t_1.

The potential burst capacity $DC_i(t)$, together with staircase function, yields the following future events prediction [27]:

$$\mathcal{U}_i(\Delta, t) = DC_i(t) + \begin{cases} \lfloor \dfrac{\Delta + (t - k_i\delta_i^u)}{\delta_i^u} \rfloor & \text{if } DC_i(t) < N_i^u \\ \lfloor \dfrac{\Delta}{\delta_i^u} \rfloor & \text{if } DC_i(t) = N_i^u \end{cases} \tag{21}$$

where N_i^u is the absolute burst. The above function bounds future event arrivals. By using the monitor to track the events

trace for high-critical tasks, DC_i and k_i can be known during the runtime. Therefore, $\mathcal{U}_i$ is also known during the runtime. For bounding the number of future event arrivals w.r.t. complex task activations, one can simply take the minimum over all the $\mathcal{U}_i$:

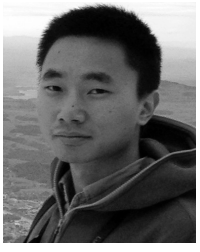
$$\overline{\alpha}^u(\tau_i, \Delta, t) = \min_{i=1..n} (\mathcal{U}_i(\Delta, t)). \tag{22}$$

ACKNOWLEDGMENT

The authors would like to express their gratitude to the associate editor and two anonymous reviewers for their constructive comments which have helped to improve the quality of the paper.

REFERENCES

[1] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–37, 2017.
[2] P. Ekberg and W. Yi, "Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Proc. 24th Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2012, pp. 135–144.
[3] B. Hu, "Schedulability analysis of general task model and demand aware scheduling in mixed-criticality systems," Ph.D. dissertation, Dept. Informatiks, Technische Universität München, 2017.
[4] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis—The SymTA/S approach," *IEE Proc.-Comput. Digit. Techn.*, vol. 152, no. 2, pp. 148–166, Mar. 2005.
[5] X. Chen, J. Feng, M. Hiller, and V. Lauer, "Application of software watchdog as a dependability software service for automotive safety relevant systems," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 618–624.
[6] N. N. Stoimenov, "Compositional design and analysis of distributed, cyclic, and adaptive embedded real-time systems," Ph.D. dissertation, Dept. Inf. Technol. Elect. Eng., Univ. Adelaide, Adelaide, SA, Australia, 2011.
[7] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst, "Monitoring arbitrary activation patterns in real-time systems," in *Proc. IEEE 33rd Real-Time Syst. Symp. (RTSS)*, Dec. 2012, pp. 293–302.
[8] M. Neukirchner, K. Lampka, S. Quinton, and R. Ernst, "Multi-mode monitoring for mixed-criticality real-time systems," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Sep./Oct. 2013, pp. 34:1–34:10.
[9] E. Wandeler, "Modular performance analysis and interface-based design for embedded real-time systems," Ph.D. dissertation, Dept. Inf. Technol. Elect. Eng., ETH Zurich, 2006.
[10] G. Carvajal, M. Salem, N. Benann, and S. Fischmeister, "Enabling rapid construction of arrival curves from execution traces," *IEEE Des. Test*, vol. 35, no. 4, pp. 23–30, Aug. 2018.
[11] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2000, pp. 101–104.
[12] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. New York, NY, USA: Springer, 2001.
[13] S. Schliecker, J. Rox, M. Ivers, and R. Ernst, "Providing accurate event models for the analysis of heterogeneous multiprocessor systems," in *Proc. 6th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Oct. 2008, pp. 185–190.
[14] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Proc. 30th IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2009, pp. 291–300.
[15] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks," in *Proc. IEEE 18th Real Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2012, pp. 23–32.
[16] D. De Niz, L. Wrage, A. Rowe, and R. R. Rajkumar, "Utility-based resource overbooking for cyber-physical systems," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 5s, pp. 162–188, 2014.
[17] X. Gu and A. Easwaran, "Dynamic budget management with service guarantees for mixed-criticality systems," in *Proc. IEEE Real-Time Syst. Symp.*, Nov./Dec. 2017, pp. 47–56.
[18] J. Lee, H. S. Chwa, L. T. Phan, I. Shin, and I. Lee, "MC-ADAPT: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–21, 2017.
[19] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "Interference constraint graph—A new specification for mixed-criticality systems," in *Proc. IEEE 18th Conf. Emerg. Technol. Factory Automat. (ETFA)*, Sep. 2013, pp. 1–8.
[20] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst, "Monitoring of workload arrival functions for mixed-criticality systems," in *Proc. IEEE 34th Real-Time Syst. Symp. (RTSS)*, Dec. 2013, pp. 88–96.
[21] S. Tobuschat, M. Neukirchner, L. Ecco, and R. Ernst, "Workload-aware shaping of shared resource accesses in mixed-criticality systems," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Oct. 2014, pp. 1–10.
[22] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll, "Adaptive runtime shaping for mixed-criticality systems," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, Oct. 2015, pp. 11–20.
[23] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll, "Adaptive workload management in mixed-criticality systems," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 1, p. 14, 2016.
[24] B. Hu and K. Huang, "Scheduling and shaping of complex task activations for mixed-criticality systems," in *Proc. 23rd Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2018, pp. 58–63.
[25] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Adaptive dynamic power management for hard real-time systems," in *Proc. 30th IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2009, pp. 23–32.
[26] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Applying real-time interface and calculus for dynamic power management in hard real-time systems," *Real-Time Syst.*, vol. 47, no. 2, pp. 163–193, 2011.
[27] K. Lampka, K. Huang, and J.-J. Chen, "Dynamic counters and the efficient and effective online power management of embedded real-time systems," in *Proc. 9th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Oct. 2011, pp. 267–276.
[28] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. 11th Real-Time Syst. Symp. (RTSS)*, Dec. 1990, pp. 182–190.
[29] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *Proc. IEEE 34th Real-Time Syst. Symp. (RTSS)*, Dec. 2013, pp. 78–87.
[30] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Syst.*, vol. 50, no. 1, pp. 48–86, 2014.
[31] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: A hybrid method for analyzing embedded real-time systems," in *Proc. Int. Conf. Embedded Softw. (EMSOFT)*, 2009, pp. 107–116.
[32] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Proc. IEEE 32nd Real-Time Syst. Symp. (RTSS)*, Nov./Dec. 2011, pp. 34–43.
[33] I. Bate, A. Burns, and R. I. Davis, "An enhanced bailout protocol for mixed criticality embedded software," *IEEE Trans. Softw. Eng.*, vol. 43, no. 4, pp. 298–320, Apr. 2017.
[34] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," in *Proc. 19th Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2014, pp. 125–130.
[35] H. Su, D. Zhu, and S. Brandt, "An elastic mixed-criticality task model and early-release edf scheduling algorithms," *ACM Trans. Design Automat. Electron. Syst.*, vol. 22, no. 2, pp. 1–25, 2016.
[36] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *Proc. 27th Euromicro Conf. Real-Time Syst.*, Jul. 2015, pp. 13–24.



BIAO HU received the B.Sc. degree in control science and engineering from the Harbin Institute of Technology in 2010, the M.Sc. degree in control science and engineering from Tsinghua University in 2013, and the Ph.D. degree from the Department of Computer Science, Technische Universität Müchen, Germany, in 2017. He is currently an Associate Professor with the College of Information Science and Technology, Beijing University of Chemical Technology. His research interests include the autonomous driving, OpenCL computing in heterogeneous system, the scheduling theory in real-time systems, and safety-critical embedded systems. He is a Handling Editor of the *Elsevier Journal of Circuits, Systems, and Computers*.

GANG CHEN received the B.E. degree in biomedical engineering, the B.S. degree in mathematics and applied mathematics, and the M.S. degree in control science and engineering from Xi'an Jiaotong University, China, in 2008, 2008, and 2011, respectively, and the Ph.D. degree in computer science from Technische Universität Müchen, Germany, in 2016. He is currently an Associate Professor with Sun Yat-sen University, China. His research interests include mixed-criticality system, energy-aware real-time scheduling, certifiable cache architecture design, and high-performance computing.

KAI HUANG received the B.Sc. degree from Fudan University, China, in 1999, the M.Sc. degree from the University of Leiden, The Netherlands, in 2005, and the Ph.D. degree from ETH Zurich, Switzerland, in 2010. He was a Research Group Leader with fortiss GmbH, Munich, Germany, in 2011, and a Senior Researcher with the Computer Science Department, Technische Universität Müchen, Germany, from 2012 to 2015. He joined Sun Yat-sen University as a Professor in 2015. In 2016, he joined the School of Data and Computer Science, Institute of Autonomous Cyber Physical Systems, as the Director. His research interests include techniques for the analysis, design, and optimization of embedded systems, particularly in the automotive domain. He has been serving as a member for the Technical Committee on Cybernetics for Cyber-Physical Systems of the IEEE SMC Society since 2015. He was a recipient of the the Program of Chinese Global Youth Experts in 2014 and the Chinese Government Award for Outstanding Self-Financed Students Abroad in 2010. He received the Best Paper Award from ESTIMedia in 2013 and SAMOS in 2009, and the General Chairs Recognition Award for Interactive Papers from CDC in 2009. He is a Regional Editor of the *Elsevier Journal of Circuits, Systems, and Computers*.