# On-the-Fly Fast Overrun Budgeting for Mixed-Criticality Systems

Biao Hu[1], Kai Huang[1,2], Pengcheng Huang[3], Lothar Thiele[3], and Alois Knoll[1]
[1]Tech. Univ. Muenchen TUM, [2]Sun Yat-Sen University, [3]ETH Zurich
[1]{hub, huangk, knoll}@in.tum.de, [3]{lothar.thiele, pengcheng.huang}@tik.ee.ethz.ch

## ABSTRACT

In mixed-criticality scheduling, the widely assumed mode-switch scheme assumes that both high- and low-criticality tasks are schedulable when no tasks overrun (normal mode) and all high-criticality tasks are schedulable even when they overrun (critical mode, where low-criticality tasks are abandoned/degraded). However, this scheme triggers a mode-switch immediately after any task overruns, which can be abrupt and pessimistic. In this paper, we tackle dual-criticality systems scheduled by earliest-deadline-first, and propose light-weight mode-switch schemes that are effective in keeping the system "away" from the critical mode. Our main idea is to perform overrun budgeting for all tasks as a whole, by monitoring task executions and updating a common overrun budget. This way, the overrun budget is shared among all tasks, and adaptively replenished leveraging run-time information; consequently, mode-switch can be postponed as much as possible. Experimental results demonstrate that the proposed mode-switch schemes outperform existing solutions to a large extent, in reducing the abandoned jobs and mode-switch frequencies, as well as in increasing the time ratio that all tasks are scheduled in the system.

## 1. INTRODUCTION

Nowadays, to reduce "SWaP" (Size, Weight, and Power) related costs, integrating tasks of different criticality levels into a shared computing platform has become a prevalent design paradigm in real-time embedded systems. An important feature of such *mixed-criticality systems* (MCSs) is that tasks need to be certified to different criticality levels. For example, the avionic DO-178B software standard [23] defines 5 criticality levels from A to E, each of which requires a different certification. In detail, the failure of A-criticality tasks is catastrophic, whereas the failure of E-criticality tasks has no effect on the airplane safety. Thus, A-criticality tasks have more stringent certification requirements.

From a timing perspective, mixed-criticality systems often model *worst-case execution times* (WCETs) on different criticality levels, with the one on a higher criticality level being more pessimistic [12]; for the same piece of code, it will have a higher WCET if it is safety-critical than it would if it is non-critical. Furthermore, by allowing multiple level WCETs of one task, mixed-criticality systems can deploy the commonly assumed mode-switch scheme to improve resource efficiency [12]: For dual-criticality systems composed of low-criticality (LO-critical) and high-criticality (HI-critical) tasks, the system starts with normal (LO) mode where all tasks can guarantee their deadlines assuming low level WCETs; if any HI-critical task overruns its low level WCET, the system switches to the critical (HI) mode, where HI-critical tasks can still meet their deadlines assuming high level WCETs and LO-critical tasks are abandoned or their services are degraded.

*EMSOFT'16,* October 01-07, 2016, Pittsburgh, PA, USA
Copyright 2016 ACM 978-1-4503-4485-2/16/10 ...$15.00
http://dx.doi.org/10.1145/2968478.2968491.

A plethora of scheduling techniques [3, 6, 10, 13–15] have been proposed following such a mode-switch scheme, see [12] for an excellent review. A common feature of them is that they assume HI mode is immediately entered whenever any HI-critical task overruns its low level WCET. Although such a mode-switch scheme is effective in guaranteeing timeliness of critical tasks, it is abrupt and pessimistic – abrupt in the sense that LO-critical tasks are suddenly dropped/degraded after a single HI-critical task overruns, and pessimistic in the sense that the system may naturally have an *overrun budget* due to free slacks. Such slacks exist either because the system is underloaded (static slacks), or because tasks will most likely finish before their WCETs at runtime, giving space for other tasks to execute (dynamic slacks). This, however, is not fully exploited in existing mode-switch schemes, to keep the system "away" from the critical mode where LO-critical tasks are abandoned or degraded.

**Related Work.** In fact, some "static" mechanisms were already proposed by previous researches, allowing the delay of mode-switch or switching back from HI to LO mode. Santy *et al.* [24] presented a method to compute offline the margins that HI-critical tasks are allowed to overrun without triggering the mode-switch. This method was further refined in [11] by using sensitivity analysis [9] to more effectively utilize the statically available resources in the system. In [6], a bailout protocol was developed to timely switch the system back to the LO mode by relying on a so-called bailout fund, instead of a system *idle tick* [12]. However, all those techniques exploit only static slacks and do not efficiently make use of dynamic slacks to postpone the mode-switch.

**Contributions.** In this paper, we propose an *on-the-fly fast overrun budgeting* (FFOB) mode-switch scheme. FFOB relies on the run-time information of tasks to compute available slacks (both static and dynamic, denoted as the overrun budget), which all tasks can spend on overrun without triggering the switch to the critical mode. The design and analysis of FFOB mode-switch scheme, however, is nontrivial. The reason is multi-fold. First, the mode-switch scheme should exploit free run-time slacks as much as possible in order to increase its efficiency. Second, the procrastination of mode-switch should not hamper *dynamic guarantees* in MCSs, i.e. all tasks must be schedulable in LO mode and HI-critical tasks must be schedulable in both modes. Finding the maximal overrun allowance and computing when to conduct the mode-switch with runtime information are more involved. Last, the timing overhead to compute the mode-switch decision should be kept a minimum. Any mode-switch scheme would be useless if its timing overhead is more than the allowance of task overrun.

Our proposed FFOB mode-switch scheme is inspired by the task procrastination techniques in dynamic power management [1, 20, 22], where the processing of incoming tasks are deliberately postponed such that the processor can reside in a sleep mode to reduce energy consumption. Analogously, task overrun in this paper is considered as a procrastination on the system, in the sense that it delays resources available to other tasks. While a lot of effective procrastination techniques are proposed for conventional real-time systems, none of them can provide dynamic timing guarantees for mixed-criticality systems. To solve this problem, a distinguishing feature that makes the existing procrastination techniques applicable in MCSs is explored. This feature, called *automatic schedulability guarantee*, can guarantee that if the system is schedulable in both modes by offline analysis, then the schedulability of LO mode at runtime automatically guarantees the schedulability of HI mode. This way, the schedulability guarantee of dual-criticality systems is transformed to the schedulability guarantee of conventional real-time systems. Besides, FFOB only needs to use a timer to manage the overrun budget, which can be efficiently implemented in many embedded systems. This timer can be renewed once it times out, which can further explore the existing slack in the system to schedule overrun tasks.

In the MCS scheduled by the EDF (earliest-deadline-first), FFOB computes an overrun budget by using the task procrastination technique and we theoretically prove that the computed overrun budget still guarantees system schedulability in both LO and HI modes. The detailed contributions of this paper are as follows:

- We propose an on-the-fly mode-switch scheme for the MCS scheduled by EDF to adaptively postpone system mode-switch. Specifically, we develop a scheme to manage system overrun budget, allowing all tasks to overrun in normal mode.
- We explore the automatic schedulability guarantee feature, reducing the dual-criticality schedulability guarantee to the schedulability guarantee of conventional systems. The automatic schedulability guarantee feature enables us to apply existing task procrastination techniques for conventional real-time systems to the MCS.
- An industrial task set and extensive generated task sets are used to test the performance of the known scheduling approaches and the proposed FFOB. Experimental results show that FFOB outperforms the static overrun allowance [24] and the bailout protocol [11] to a large extent in reducing the deadline misses, mode-switch frequencies, and in increasing the time ratio that all tasks are scheduled.

The remainder of this paper is structured as follows. Section 2 presents our system settings. Section 3 provides an overview of our techniques and proves the correctness of the FFOB mode-switch scheme in EDF scheduled MCS. Experimental results are presented in Section 4, and Section 5 concludes the paper.

## 2. MODELS AND PRELIMINARIES

In this section, we introduce the related models and present the preliminary knowledge of EDF-VD that will be used in this paper.

### 2.1 Models and Notations

**Mixed-Criticality System Model.** We adopt the classic dual-criticality system model [3, 6, 10, 12–15] in this paper. A dual-criticality task set $\tau = \{\tau_1, ..., \tau_n\}$ is given to be scheduled on a uniprocessor. All tasks are independent. Each task, $\tau_i$, is characterized by a minimal inter-arrival time $T_i$, relative deadline $D_i$, WCET $\mathbf{C_i}$ and criticality $\mathcal{X}_i$, where $\mathbf{C_i} = (C_i^L, C_i^H)$ and $D_i = T_i$. Each LO-critical task only has a LO WCET $C_i^L$, and each HI-critical task has a LO WCET $C_i^L$ and a HI WCET $C_i^H$. For HI-critical tasks, their HI WCETs are not smaller compared to their LO WCETs, i.e., $C_i^L \leq C_i^H \leq D_i$. This corresponds to the assumption that the execution time estimation on a higher criticality level is more conservative. In the system, since $C_i^L$ is less conservative, some tasks (including LO-critical tasks) may overrun their given $C_i^L$. However, we suppose no HI-critical tasks can overrun their given $C_i^H$. The rationale behind this is that $C_i^H$ is obtained under very strict and conservative assumptions and no task will overrun this WCET.

Although the research on mixed-criticality system is quite new (stemming from a seminar paper [25]), a standard model already exists [3, 10, 12–15].

- The system starts in LO mode, where all tasks are assumed to not exceed/overrun their LO WCETs and are guaranteed to meet their deadlines.
- If any HI-critical job exceeds its LO WCET, then the system transits immediately to the HI mode, where all LO-critical jobs are abandoned and HI-critical tasks are guaranteed to meet their deadlines if they do not exceed their HI WCETs.
- If any LO-critical job executes for its LO WCET without completion, it is immediately aborted.
- When the system is in HI mode, an idle tick will trigger the system to switch back to the LO mode.

This standard system model is called *mixed-criticality schedulable* if the following two properties are guaranteed:

- Property 1: All jobs that are released and complete in LO mode, are guaranteed to meet their deadlines.
- Property 2: HI-critical jobs released at any time are guaranteed to meet their deadlines.

**Task Demand Model.** In order to analyze the system schedulability of a task set, we need to know the *demand bound function* (DBF) [4] of each task in this set.

Definition 1 (Demand Bound Function). *A demand bound function* dbf$(\tau_i, \Delta)$ *is the maximum required execution time of a task $\tau_i$ over any interval of length $\Delta$ to guarantee that any job of this task that is released and has deadline within this interval $\Delta$ will not miss its deadline.*

For a sporadic task $\tau_i$ with relative deadline $D_i$, its DBF is already provided in [4]

$$\mathrm{dbf}(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i - D_i}{T_i} \right\rfloor \cdot C_i, \ \forall \ \Delta \geq 0. \quad (1)$$

**Task Resource Model.** The resource that the system provides to a task is modeled by the *supply bound function* (SBF) that specifies the minimum number of execution time units available over any time interval of length $\Delta$. In this paper, the system resource is modeled as a dedicated uniprocessor with a unit-speed.

Analogous to the denotation of DBF, the SBF of a unit-speed processor system for the task set $\tau$ is thus denoted as

$$\mathrm{sbf}(\tau, \Delta) = \Delta, \ \forall\ \Delta \geq 0. \tag{2}$$

**Short Notations.** For ease of expression in the sequel, we adopt some short notations. We denote the subset of all LO-critical tasks and all HI-critical tasks in $\tau$ as $\tau^L = \{\tau_i \in \tau | L_i = \mathrm{LO}\}$ and $\tau^H = \{\tau_i \in \tau | L_i = \mathrm{HI}\}$. Besides, for simplicity, we use $[\![a]\!]_b$ to represent $\max(a, b)$ and $[\![a]\!]^c$ to represent $\min(a, c)$.

## 2.2 EDF-VD Technique

Earliest-Deadline-First Virtual-Deadlines (EDF-VD) [2, 14] is a scheduling technique that makes the conventional EDF applicable in the MCS. In this paper, the proposed F-FOB mode-switch scheme relies on the EDF-VD algorithm.

A key feature of EDF-VD is to artificially shorten the deadlines of HI-critical tasks when the system is in LO mode. In this way, HI-critical tasks will finish earlier so that there is enough time slack for them to catch their actual deadlines after switching to the HI mode.

### 2.2.1 DBF *in LO and HI modes*

In order to schedule HI-critical tasks in MCSs, the relative deadline $D_i$ of a HI-critical task is artificially shortened in LO mode and returns to $D_i$ after the system switches to the HI mode. We name the deadline in LO mode as the LO mode deadline, and denote it as $D_i^L$. Note that, for LO-critical tasks, their deadlines do not need to be shortened, thus $D_i^L = D_i$, $\forall \tau_i \in \tau^L$.

When the system is in LO mode, each task $\tau_i$ behaves as a normal sporadic task with parameters $C_i^L$, $D_i^L$ and $T_i$. A DBF of such a task is known [4]:

$$\mathrm{dbf_{LO}}(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i - D_i^L}{T_i} \right\rfloor C_i^L. \tag{3}$$

When the system is in HI mode, LO-critical tasks are abandoned, thus only the demands of HI-critical tasks need to be considered. The DBF of a HI-critical task $\tau_i$ in HI mode is that [14]:

$$\begin{aligned} &\mathrm{dbf_{HI}}(\tau_i, \Delta) = \left\lfloor \frac{\Delta + T_i - (D_i - D_i^L)}{T_i} \right\rfloor C_i^H - \mathrm{done}(\tau_i, \Delta), \\ &\mathrm{done}(\tau_i, \Delta) = \begin{cases} [\![C_i^L - l + D_i - D_i^L]\!]_0, & if D_i > l \geq D_i - D_i^L \\ 0, \text{otherwise}, \end{cases} , \end{aligned} \tag{4}$$

where $l = \Delta \bmod T_i$.

In EDF-VD scheduled MCS, the DBF of a system is the sum of DBFs of all tasks in this system [14]. That is,

$$\begin{aligned} \mathrm{dbf_{LO}}(\tau, \Delta) &= \sum_{\forall \tau_i \in \tau} \mathrm{dbf_{LO}}(\tau_i, \Delta), \\ \mathrm{dbf_{HI}}(\tau^{\mathrm{H}}, \Delta) &= \sum_{\forall \tau_i \in \tau^H} \mathrm{dbf_{HI}}(\tau_i, \Delta). \end{aligned} \tag{5}$$

### 2.2.2 *Schedulability analysis*

The following proposition presents the sufficient conditions that can guarantee all tasks to meet their deadlines in LO mode and all HI-critical tasks to meet their deadlines in both LO and HI modes.

Proposition 1. *[From [14]]: In* MCS*s, the taskset is schedulable if the* DBF*s of LO and HI modes are not greater than the* SBF*s of this system, i.e.,* $\forall \Delta \geq 0$,

$$\text{Condition LO}: \quad \mathrm{dbf_{LO}}(\tau, \Delta) \leq \mathrm{sbf}(\tau, \Delta) = \Delta, \tag{6a}$$

$$\text{Condition HI}: \quad \mathrm{dbf_{HI}}(\tau^{\mathrm{H}}, \Delta) \leq \mathrm{sbf}(\tau^H, \Delta) = \Delta. \tag{6b}$$

## 3. FFOB OF EDF SCHEDULE

In this section, we present the FFOB mode-switch scheme in EDF scheduled MCS. We first present the schedulability analysis with the task procrastination technique at runtime, on top of which we compute an overrun budget and theoretically prove the correctness of FFOB in guaranteeing the system mixed-criticality schedulable.

## 3.1 Schedulability Analysis at Runtime

We have presented the schedulability analysis of the MCS offline in Section 2.2. Now, we present the schedulability analysis with the task procrastination online.

At runtime, the task information may be different with the offline worst-case assumption. To denote the task information at runtime, we use a form of $\mathrm{f(obj, \Delta, t)}$ to represent the function of a related object over any time interval of length $\Delta$ after time $t$. The online DBF and SBF of a task $\tau_i$ are thus denoted as $\mathrm{dbf(\tau_i, \Delta, t)}$ and $\mathrm{sbf(\tau_i, \Delta, t)}$.

**Task Procrastination.** Suppose at a time $t$ when the MCS is in LO mode, all tasks in $\tau$ are delayed for a time length $\rho(t)$ to be executed, then the SBF of $\tau$ after $t$ is that [21]

$$\mathrm{sbf_{LO}}(\tau, \Delta, t) = [\![\Delta - \rho(t)]\!]_0. \tag{7}$$

Denote $t_{ms}$ as the time instant of a mode-switch and $\mathrm{sbf_{HI}}(\tau^H, \Delta, t_{ms})$ as the SBF of $\tau^H$ in HI mode after $t_{ms}$. Straightforwardly extended from Proposition 1, we have the following schedulability conditions.

Proposition 2. *The schedulability conditions at runtime are that,* $\forall\ t$, $t_{ms}$, $\Delta \geq 0$,

$$\text{Condition LO-t}: \mathrm{dbf_{LO}}(\tau, \Delta, t) \leq \mathrm{sbf_{LO}}(\tau, \Delta, t), \tag{8a}$$

$$\text{Condition HI-t}: \mathrm{dbf_{HI}}(\tau^H, \Delta, t_{ms}) \leq \mathrm{sbf_{HI}}(\tau^H, \Delta, t_{ms}). \tag{8b}$$

*where* $\mathrm{dbf_{LO}(\tau, \Delta, t)}$ *gives the upper bound on the maximum possible execution demand of a task set* $\tau$ *over any time interval of length* $\Delta$ *after time* $t$ *in LO mode. Similarly,* $\mathrm{dbf_{HI}(\tau^H, \Delta, t_{ms})}$ *gives the upper bound on the maximum possible execution demand of tasks* $\tau^H$ *over any time interval of length* $\Delta$ *after time* $t_{ms}$ *in HI mode.*

Note that the two conditions correspond to the two properties of being mixed-criticality schedulable in Section 2.1. The Condition LO-t corresponds to Property 1. The Condition LO-t and Condition HI-t together correspond to Property 2.

**Intuition.** In the conventional hard real-time system, task executions can be delayed for a certain time length without missing any deadlines. To get a feasible time length, we have the following lemma.

Lemma 1. *[From [19]] Suppose* $\mathrm{dbf}(\tau, \Delta, t)$ *denote the* DBF *of a task set* $\tau$ *from time* $t$*. If there is a* $\rho$ $(\rho > 0)$ *that satisfies*

$$\forall \Delta > 0 : \mathrm{dbf}(\tau, \Delta, t) \leq [\![\Delta - \rho]\!]_0, \tag{9}$$

*then the executions of all tasks can be immediately delayed for* $\rho$ *and there will be no deadline misses after* $t$*.*

Inspired by Lemma 1, our idea is to allow tasks to overrun for a time boundary, by ensuring that this time boundary is not greater than the feasible task procrastination time length (a time length that task executions are delayed). Such idea is also exploited in [7, 16] to manage the runtime workload, in which way the task executions are regulated and no tasks will miss their deadlines [17, 18].

To dynamically manage the time boundary that a task is allowed to overrun, the overrun budget is introduced. In

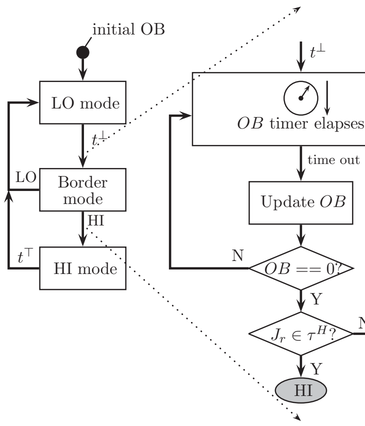


Figure 1: Overview of FFOB in EDF scheduled MCS

the following, we design a mode-switch scheme called FFOB that relies on the overrun budget to allow tasks overrun. The FFOB mode-switch scheme can adaptively postpone the mode-switch, while sufficiently guaranteeing the two properties of being mixed-criticality schedulable.

## 3.2 FFOB Mode-Switch Scheme

We now present the FFOB mode-switch scheme. First, we give an overview of FFOB, followed by a running example to further explain it. Then, we provide how to initialize, update an overrun budget, and how to choose appropriate virtual deadlines. Last, we theoretically prove the correctness of FFOB that it can guarantee the two properties of being mixed-criticality schedulable.

### *3.2.1 FFOB Overview*

In the standard model of MCSs presented in Section 2.1, no LO-critical job is allowed to exceed its LO WCET and a HI-critical job's overrun of its LO WCET immediately triggers the mode-switch. However, in FFOB, by relying on an overrun budget, all jobs are allowed to run over their LO WCETs, without being dropped or triggering the mode-switch. To denote the system state that a job overruns and system is not in HI mode, the Border mode is introduced.

FFOB mainly relies on the overrun budget to schedule the overrun jobs. The overrun budget denotes a capacity that the MCS allow tasks to overrun, which is defined as follows.

Definition 2 (Overrun Budget). *The overrun budget $OB$ at some time $t$ is a safe upper bound on the total amount of time that the processor can work on any overrun task after time $t$. In other words, all LO mode deadlines can be guaranteed after time $t$ if the processor does not execute overrun tasks for more than $OB$ time units.*

Based on the overrun budget $OB$, the system's runtime behavior and the working overflow of FFOB mode-switch scheme is depicted in Fig. 1. At the beginning, the system sets up the $OB$ timer with an initial value. Then, the system may go through the following modes.

*LO mode:*

(i) The scheduler maintains the overrun budget $OB$. If there is an idle tick, $OB$ is reset to the initial value.

(ii) While all jobs do not execute for more than their LO WCETs, the MCS remains in LO mode.

(iii) If any job overruns its LO WCET (at time instant $t^\perp$), the system goes to Border mode.

*Border mode:*

(iv) The $OB$ timer elapses, as an overrun job (denoted as $J_r$ in Fig. 1) runs.

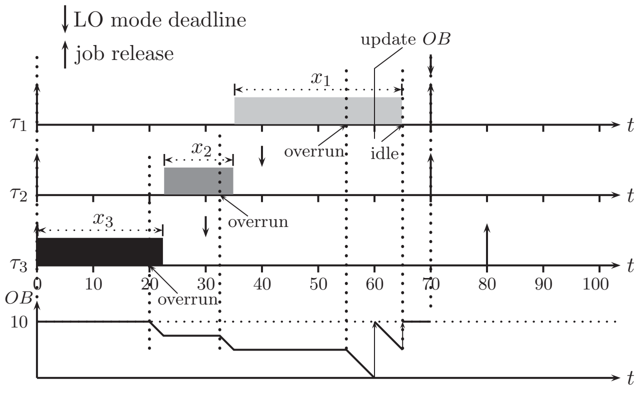


Figure 2: Illustration of task execution with FFOB

(v) If $J_r$ stops running before $OB$ times out, OB timer stops and the system returns to LO mode.

(vi) Once $OB$ timer times out, the scheduler calls a procedure to update $OB$, then the overrun budget $OB$ is updated based on the current state of tasks' executions. The updating of $OB$ will be detailed in Section 3.2.3. If the updated $OB$ is not 0, the $OB$ timer (with this new value) continues to elapse as $J_r$ continues to run. If the updated $OB$ is 0, a decision procedure is called.

(vii) In this decision procedure, if the overrun job is LO-critical, it is dropped and the system goes back to LO mode. Otherwise, the system goes to HI mode.

*HI mode:*

(viii) Only HI-critical tasks run and all released LO-critical jobs will be abandoned.

(ix) An idle tick (denoted as time instant $t^\top$) will set $OB$ to the initial value and trigger the system to switch to the LO mode.

Even with the Border mode, the MCS is still called mixed-criticality schedulable if the two properties described in Section 2.1 are guaranteed, because the Border mode is an extension of LO mode in the sense that all tasks in Border mode are scheduled according to their LO mode deadlines.

The following example illustrates how the FFOB mode-switch scheme works.

Example 1. *In a uniprocessor system, three tasks are scheduled by the EDF-VD algorithm. Task properties are shown as follows.*

| $\tau_i$ | $\mathcal{X}_i$ | $C_i^L$ | $C_i^H$ | $D_i^L$ | $D_i$ | $T_i$ |
|---|---|---|---|---|---|---|
| $\tau_1$ | *LO* | 20 | - | 70 | 70 | 70 |
| $\tau_2$ | *HI* | 10 | 20 | 40 | 70 | 70 |
| $\tau_3$ | *HI* | 20 | 40 | 30 | 80 | 80 |

*Fig. 2 illustrates the system runtime behavior under FFOB. Before the system runs, the OB timer is initialized to 10. Suppose that the first jobs of all tasks are released at time zero. Task $\tau_3$ executes first. Once $\tau_3$ overruns at $t = 20$, the OB timer starts to elapse. When $\tau_3$ finishes, the OB timer will hold its current value and stop elapsing. $\tau_2$ starts to execute. After $\tau_2$ executes over 10, it overruns and triggers the OB timer to elapse. Similarly, the OB timer will stop when $\tau_2$ finishes. Then $\tau_1$ runs and further overruns to the extent that OB timer elapses to 0. The OB updating procedure is called and OB is updated to 10 by the approach in Section 3.2.3. After that, $\tau_1$ runs until it finishes. After $\tau_1$ finishes, the system returns to LO mode as there is an idle tick; in the meanwhile OB is reset.*

The FFOB mode-switch scheme outperforms the other

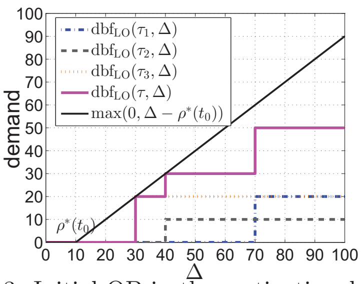


Figure 3: Initial OB in the motivational example

known methods that fairly increase the allowance of task overrun offline in two aspects. First, $OB$ can be flexibly used by all tasks. In the static method, the overrun allowance is assigned to each individual task, and the mode-switch is triggered once a task exceeds its own overrun allowance. Such a scheme is not flexible as it cannot use the remaining allowances of other tasks. Second, $OB$ is updated at runtime in FFOB, which can often replenish the overrun allowances and postpone the mode-switch by exploring dynamic slacks. Such slacks naturally exist as tasks will most likely take less than their WCETs to finish. The updated $OB$ is able to collect those slacks to postpone the mode-switch. Furthermore, since the remaining $OB$ is still valid with the time going on and $OB$ will be automatically replenished to the initial value whenever an idle tick emerges, FFOB does not need to use a complex way to update $OB$ every time a task overruns.

### 3.2.2 *Initialize* $OB$

In order to get the initial $OB$, we compute the largest procrastination interval that the system at the beginning can accept. Suppose there is a initial procrastination interval $\rho$ on the processor when the system starts. Then, the service bound function becomes $[\![\Delta - \rho]\!]_0$. The longest procrastination interval is defined as follows [20].

Definition 3 (Longest Procrastination Interval). *The longest procrastination interval $\rho^*$ with respect to a given* DBF $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta)$ *is*

$$\rho^* = \max\Big\{\rho : [\![\Delta - \rho]\!]_0 \geq \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta),\ \forall \Delta \geq 0\Big\}. \qquad (10)$$

Therefore, $OB$ is initialized to the longest procrastination interval with respect to $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta)$. This longest procrastination interval is denoted as $\rho^*(t_0)$. For the task set in the motivational example, the initial $OB$ is set to 10 based on Eq. 10, as shown in Fig. 3.

### 3.2.3 *Update* $OB$ *at runtime*

While tasks are overrunning, $OB$ may elapse to 0. The system may be still able to postpone the mode-switch, because the actual overrun allowance at this moment may be greater than 0 based on the current tasks' execution state, i.e., dynamic slacks are available. We now derive the actual DBF of a task $\tau_i$ at time $t$.

Lemma 2. *At any time $t$, for a task $\tau_i$ that has no backlogged job at $t$, its LO mode* DBF *is*

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta). \qquad (11)$$

*For a task $\tau_i$ that has one backlogged job at $t$, its LO mode* DBF *is*

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \max\big(\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta),\ \mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t)\big), \qquad (12)$$

*where* $\mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t)$ *is derived as follows*

$$\begin{aligned}\mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t) = &\left[\!\!\left[\left\lfloor \frac{\Delta}{r_i(t) + D_i^L - t} \right\rfloor\right]\!\!\right]^1 \cdot [\![C_i^L - e_i(t)]\!]_0 \\ &+ \left[\!\!\left[\left\lfloor \frac{\Delta + \min\big(T_i, t - r_i(t)\big) - D_i^L}{T_i} \right\rfloor\right]\!\!\right]_0 C_i^L,\end{aligned} \qquad (13)$$

*and $r_i(t)$, $e_i(t)$ are the release time and the actual execution time of the latest released job of $\tau_i$ at $t$, respectively.*

Proof. Since the DBF for a task $\tau_i$ must upper-bound the maximum execution demand of jobs from $\tau_i$ within any scheduling interval after time $t$, the DBF will include the demand from jobs that are backlogged and the future jobs. As shown in Fig. 4, the release time of the latest released job is $r_i(t)$, and at time $t$, the released job may have finished or may not. Therefore, a task may have backlogged job or may not. We consider the demand of the two cases, respectively.

First, we consider that the released job has been finished. Since the released job has been finished, there is no demand from this job in future. The DBF will only bound the demand of future jobs. We assume future jobs are released as early as possible. At time $r_i(t) + T_i$, the jobs' release pattern is the same as the offline assumption. Then, the maximum demand within an interval will be the same as the DBF within the same interval in the offline analysis, as the demand within the interval $\Delta'$ seen in Fig. 4. Therefore, we prove that $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) = \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$.

Second, we consider that there is a backlogged job. The demand of task $\tau_i$ within an interval may include the demand of the backlogged job, or may not include this demand. If the demand of the backlogged job is not included, the upper bound of such a demand is the same as $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$. Therefore, $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \geq \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$. Now we consider the upper bound on the demand that includes the demand of the backlogged job. To include the demand of this backlogged job, an interval $\Delta$ should start from $t$ and end at $t + \Delta$. We use $\mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, \mathrm{t})$ to denote the upper bound of the demand that includes the backlogged job. The backlogged job may overrun or may not overrun. If the backlogged job does not overrun, it will demand $C_i^L - e_i(t)$. Otherwise, its demand is 0, because instead of contributing to the system demand, overrun in our technique is considered as the processing procrastination. In short, we use $[\![C_i^L - e_i(t)]\!]_0$ to denote the demand of the backlogged job. This backlogged job should be given $[\![C_i^L - e_i(t)]\!]_0$ before $r_i(t) + D_i^L$. Future jobs are assumed to be released as early as possible in order to maximize its demand. The request demand of every future job is $C_i^L$. Within $\Delta$, the maximum number of arrival events is $\lfloor\big(\Delta + \min\big(T_i, t - r_i(t)\big)\big)/T_i\rfloor$, and those jobs should be given their requested demand no later than their LO mode deadlines. In summary, $\mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, \mathrm{t})$ is represented by Eq. 13, where its first part is the demand of the backlogged job and its second part is the demand of its future jobs.

Since $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \geq \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta)$ and $\mathrm{dbf}_{\mathrm{LO}}(\tau_i, \Delta, t) \geq \mathrm{Dmd}^{\mathrm{bk}}(\tau_i, \Delta, t)$, Eq. 12 holds. □

With Eqs. 11 and 12, we can get the $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, \mathrm{t})$ at runtime. By applying the following equation (similar to Eq. 10)

$$\rho^*(t) = \max\Big\{\rho : [\![\Delta - \rho]\!]_0 \geq \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, \mathrm{t}),\ \forall \Delta \geq 0\Big\}, \qquad (14)$$

we get $\rho^*(t)$. This $\rho^*(t)$ is used to renew the overrun budget $OB$ at runtime whenever it elapses to zero.

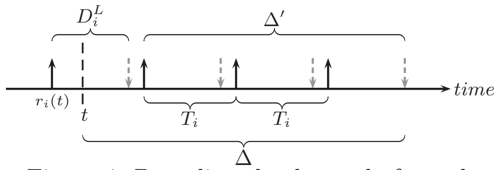


Figure 4: Bounding the demand of a task

### 3.2.4 Setting LO mode deadlines

From Eqs. 10 and 14, we can see that the initial $OB$ and the updated $OB$ depend on the task offline and online demand bound functions, hence their LO mode deadlines. We proceed now to optimize $OB$ by configuring tasks LO mode deadlines.

There can be a lot of options in setting $D_i^L$ for every HI-critical task. Different options may have different runtime effects. To illustrate this problem, we consider the task set in the Example 1. There are two options of $D_i^L$, as shown in the following.

- Option 1: $D_2^L = 40$ and $D_3^L = 30$, which is the same as the example.

- Option 2: $D_2^L = 60$ and $D_3^L = 40$, which is different with the example.

In both options, the schedulability of this system in the offline analysis is guaranteed. However, the second option can initialize $OB$ to 20, while the first option can only initialize $OB$ to 10. Option 2 is expected to have a better performance because the initial $OB$ is greater and $D_2^L, D_3^L$ of option 2 are greater than those of option 1. When we update an $OB$, the $\rho^*(t)$ of using option 2 must be equal to or greater than that of using option 1.

Motivated by this example, we need to carefully choose $D_i^L$ for every HI-critical task so that $OB$ will be replenished the most in every update.

We propose three targets to configure $D_i^L$. The first target is to make the initial procrastination interval $\rho^*(t_0)$ the largest. $\rho^*(t_0)$ is important because $OB$ will be updated to $\rho^*(t_0)$ whenever there is an idle tick and $OB$ is constrained within $\rho^*(t_0)$ in every $OB$ update. Therefore, among all $D_i^L$ configurations, those initializing $OB$ the most are preferable. If there exists more than one options that have the same initial $OB$, we need to consider the second target. The second target is to make the sum of all LO mode deadlines the largest because a larger $D_i^L$ makes $\mathrm{dbf_{LO}}(\tau_i, \Delta, t)$ smaller, potentially leading to a larger $\rho^*(t)$. There may still exist a lot of options that meet the two targets. For fairness, we prefer that the LO mode deadlines have not much difference. Thus, the third target is to make the variance of LO mode deadlines the least. If there are more than one options that meet the above three targets, we randomly pick one among them. Note that, in choosing $D_i^L$, the first consideration is the first target, then the second target, and the last is the third target. If there is only one option from the first target, the second and third targets need not to be considered. The principles of choosing LO mode deadlines only come from our intuitions that those principles may improve system performance.

In the search of feasible $D_i^L$, $D_i^L$ should not be smaller than $C_i^L$ in order to keep the schedulability in LO mode. $D_i^L$ should not be greater than $D_i^H - (C_i^H - C_i^L)$ in order to leave enough slack for processing the *carry-on job* (released but not finished at the mode switch) [14].

In general, we formalize the problem of choosing $D_i^L$ as the following optimization problem.

$$\begin{aligned}
&\textbf{Constraint}: \ Eqs.\ 6a,\ 6b,\\
&Target\ 1: \ \textbf{maximize} \ \ \rho^*,\\
&Target\ 2: \ \textbf{maximize} \ \ \sum_{\tau_i \in \tau^H} D_i^L,\\
&Target\ 3: \ \textbf{minimize} \ \ \mathrm{variance}(D_i^L),
\end{aligned}$$

where Eqs. 6a, 6b (Section 2.2) guarantee the schedulability of the system in both modes.

### 3.2.5 Correctness of FFOB

This section proves the correctness of the FFOB mode-switch scheme, i.e., the two properties of being mixed-criticality schedulable are sufficiently guaranteed.

First of all, we need to clarify some denotations on SBF. $\mathrm{sbf_{LO}}(\tau, \Delta, t)$ and $\mathrm{sbf_{HI}}(\tau, \Delta, t)$ define the system SBF for the task set $\tau$ in LO mode and in HI mode after time $t$, respectively. The system SBF for the task set $\tau$ in any mode is denoted as $\mathrm{sbf}(\tau, \Delta)$, and $\mathrm{sbf}(\tau, \Delta) = \Delta$ as the processor constantly provides full processing service.

**Property 1 is guaranteed.**

We prove that Property 1 is guaranteed by dividing the time interval into the busy interval and the idle interval. The busy interval is an interval during which the system is executing jobs. The idle interval is an interval in which no jobs are executed. In the idle interval, no jobs will miss their deadlines as no jobs exist with a work-conserving scheduler, i.e., EDF-VD. We only need to guarantee Property 1 in any busy interval.

For ease of the proof, we provide some denotations on the time instants. Denote $t_0$ as the starting time of the system. Denote $[t_{sn}, t_{en})$ as the $n$-th busy interval where $t_{sn}$ and $t_{en}$ are the starting and ending times of this busy interval, respectively. We have $t_{s1} = t_0$. In any busy interval, the system will be switched to HI mode at most once because otherwise there must be an idle tick to switch the system from HI to the LO mode during this interval. If the system switches to HI mode, we denote $t_{ms,n}$ as the time instant within $[t_{sn}, t_{en})$ at which time the system switches to the HI mode. If the system does not switch to HI mode within $[t_{sn}, t_{en})$, we set $t_{ms,n} = t_{en}$ for completeness.

We prove that Property 1 holds in the first busy interval. The proof of other busy intervals are similar. During the first busy interval, $OB$ is initialized to $\rho^*(t_0)$ at $t_{s1}$ and the system will be in LO mode only within $[t_0,\ t_{ms,1}]$.

LEMMA 3. *No tasks in LO mode will miss their LO mode deadlines during* $[t_0,\ t_{ms,1}]$.

PROOF. According to FFOB mechanism in Fig. 1, LO mode is sometimes interfered by Border mode, while such interference is constrained by $OB$. Based on $OB$, we consider the following cases that may happen during $[t_0,\ t_{ms,1}]$.

- Case 1: $OB$ does not elapse to 0.

If $OB$ does not elapse to 0, the maximum accumulated time of the system in Border mode within $[t_0,\ t_{ms,1}]$ is less than $\rho^*(t_0)$, which means that the lower bound of resources available within this interval for LO mode is greater than $[\![\Delta - \rho^*(t_0)]\!]_0$. Since

$$\mathrm{dbf_{LO}}(\tau, \Delta) \leq [\![\Delta - \rho^*(t_0)]\!]_0, \forall \Delta \geq 0, \qquad (15)$$

we conclude that the the lower bound of provided resources to the task set $\tau$ within $[t_0,\ t_{ms,1}]$ is greater than its demand, thus no tasks in $\tau$ will miss their deadlines in this case.

- Case 2: $OB$ first elapses to 0 at a time $t_{c1,1}$. At the time $t_{c1,1}$, $OB$ is updated to a value $OB(t_{c1,1})$.

According to the analysis of case 1, we know that no tasks will miss their deadlines from time $t_0$ to time $t_{c1,1}$. At the time $t_{c1,1}$, $OB = OB(t_{c1,1})$ that satisfies

$$\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t_{c1,1}) \leq [\![\Delta - OB(t_{c1,1})]\!]_0, \ \forall \Delta > 0. \tag{16}$$

Analogously, from time $t_{c1,1}$ to the next time that $OB$ elapses to 0 or the system is idled, the maximum accumulated time that the system is in Border mode is less than $OB(t_{c1,1})$. The lower bound of resources available for the task set $\tau$ in LO mode is also greater than the LO mode DBF of the task set $\tau$. Tasks will not miss their LO mode deadlines in this case.

Other cases just repeat case 1 and case 2.

Therefore, all LO mode deadlines can be met in LO mode during the interval $[t_0, \min(t_{ms,1}, t_{e1})]$. □

LEMMA 4. *The* DBF *of a task set $\tau$ in LO mode at time $t_{sn}$, $n \in \mathbb{N}^+$ is the same as the* DBF *of a task set $\tau$ in LO mode at time $t_{s1}$.*

PROOF. At time $t_{sn}$, there are no backlogged jobs. According to Lemma 2, the LO mode DBF is the same, as shown in the offline analysis. □

THEOREM 1. *At any time $t_{sn}$, $n \in \mathbb{N}^+$, $OB$ can be reset to a value $\rho^*(t_0)$ that guarantees all tasks can meet their LO mode deadlines in LO mode.*

PROOF. With Lemma 4, we know that $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t_{s1}) = \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t_{sn})$, where $n \in \mathbb{N}^+$. Therefore, we have

$$\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t_{sn}) = \mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t_{s1}) \leq [\![\Delta - \rho^*(t_0).]\!]_0 \tag{17}$$

Following a similar procedure used in proving Lemma 3, we conclude that all tasks can meet their LO mode deadlines in LO mode during the $n-th$ busy interval, $n \in \mathbb{N}^+$. □

**A property in Border mode.**

An important property in Border mode is that the overrun job in Border mode can also meet its LO mode deadline. Such property will be used in proving Property 2 in the following.

THEOREM 2. *Any job finished in Border mode meets its LO mode deadline.*

PROOF. In LO mode, we guarantee that at any time $t$, we have $\mathrm{dbf}_{\mathrm{LO}}(\tau, \Delta, t) \leq [\![\Delta - OB(t)]\!]_0$. From Lemma 1, we know that the task execution can be delayed for $OB(t)$ without missing any deadline, which also means that any task can be allowed to overrun for $OB(t)$. Therefore, there will be no job missing its deadline in Border mode. □

**Property 2 is guaranteed.**

To prove Property 2, we need to know some features in the MCS scheduled with FFOB.

PROPOSITION 3. *In both LO and Border modes, FFOB guarantees that HI-critical jobs will not be dropped, and all HI-critical jobs can meet their LO mode deadlines even if they overrun. If a HI-critical job runs across its LO mode deadline without stopping, the system will be switched to HI mode.*

PROOF. This comes from Property 1 and the Property in Border mode. □

THEOREM 3. *FFOB guarantees that* $\mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta, t_{ms}) \leq \mathrm{dbf}_{\mathrm{HI}}(\tau^H, \Delta)$.

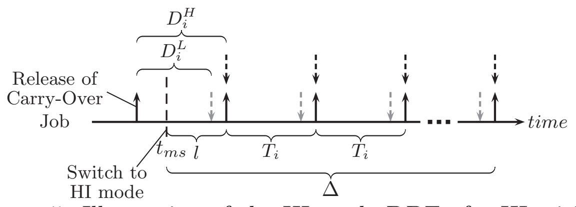


Figure 5: Illustration of the HI mode DBF of a HI-critical task

PROOF. In this proof, we need to introduce the *carry-over* jobs. A carry-over job is a job from a HI-critical task that is active (released, but not finished) at the time of the switch to HI mode [14].

Since any HI-critical task will not miss its LO mode deadline before the system switches to the HI mode, there must be at least a slack of $D_i^H - D_i^L$ for the carry-over job to meet its HI mode deadline after the mode-switch. Therefore, the smallest time interval that a carry-over job's demand must be met is $D_i^H - D_i^L$. Except the carry-over jobs, any other jobs in HI mode have an interval of $D_i^H$ ($D_i^H = T_i$) from its release to meet their deadlines. It indicates that the smallest time interval in which the demand of $k$ jobs must be met is $(k-1) \cdot T_i + D_i^H - D_i^L$. Then, for any HI-critical task $\tau_i$, its HI mode DBF is bounded by

$$\mathrm{Dmd}_{\mathrm{HI}}^{\mathrm{full}} = \left\lfloor \frac{\Delta + T_i - (D_i - D_i^L)}{T_i} \right\rfloor C_i^H, \ \forall \Delta \geq 0. \tag{18}$$

For a carry-over job, its demand may not be $C_i^H$ because it may have been executed for some time before the mode-switch. Now we derive the least time that a carry-over job must have executed within a time interval in the demand $\mathrm{Dmd}_{\mathrm{HI}}^{\mathrm{full}}$. The following observation is presented.

- Fact 1: If a job meets its LO mode deadline $D_i^L$ before the system switches to the HI mode and this job has $n$ time units left until its HI mode deadline $D_i^H$ at the moment of the mode-switch, then (1): this job has finished if $l < D_i^H - D_i^L$; (2): this job has finished at least $[\![C_i^L - \big(l - (D_i^H - D_i^L)\big)]\!]_0$ if $l \geq D_i^H - D_i^L$.

This fact comes from Lemma III.3 of [14]; it is proposed for the carry-over job that does not overrun its LO WCET. For a carry-over job that has overrun its LO WCET, however, its execution time units before system switching to the HI mode is more than the execution time in Fact 1. Suppose a carry-over job exceeds its LO WCET by $\epsilon$ ($\epsilon > 0$) before switching to the HI mode, we conclude that (1): this job has finished if $l < D_i^H - D_i^L$. (2): this job has finished at least $[\![\epsilon + C_i^L - \big(l - (D_i^H - D_i^L)\big)]\!]_0$ if $l \geq D_i^H - D_i^L$. For completeness, we set $\epsilon \geq 0$ to represent the carry-over job that either overruns or not.

Now, we analyze how to get such $l$ in Fact 1. As shown in Fig. 5, for a time interval $\Delta$, time units left for the carry-over job are at most $l = \Delta \bmod T_i$ ($0 \leq l < D_i^H = T_i$). If $l < D_i^H - D_i^L$, this carry-over job must have finished. Otherwise, it has finished $[\![\epsilon + C_i^L - \big(l - (D_i^H - D_i^L)\big)]\!]_0$. The lower bound of finished execution time units in $\Delta$ is then,

$$\mathrm{Ext}_{\mathrm{HI}}^{\mathrm{done}} = \begin{cases} [\![\epsilon + C_i^L - l + D_i - D_i^L]\!]_0, \ if \ D_i > l \geq D_i - D_i^L \\ 0, \mathrm{otherwise}, \end{cases} \tag{19}$$

where $\epsilon \geq 0$ and $l = \Delta \bmod T_i$.

Therefore, the HI mode DBF of a task $\tau_i$ from time $t_{ms}$ is that

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, \mathrm{t_{ms}}) = \mathrm{Dmd}_{\mathrm{HI}}^{\mathrm{full}} - \mathrm{Ext}_{\mathrm{HI}}^{\mathrm{done}}. \tag{20}$$

Compared to $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta)$ in Eq. 4, we know for any HI-critical task $\tau_i$, $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta) \geq \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \Delta, \mathrm{t_{ms}})$ because

$\epsilon \geq 0$. Therefore, for a set of HI-critical tasks, we have $\text{dbf}_{\text{HI}}(\tau^H, \Delta, t_{ms}) \leq \text{dbf}_{\text{HI}}(\tau^H, \Delta)$. □

Theorem 4. *If Condition LO and Condition HI hold, Condition HI-t can be met by only guaranteeing Condition LO-t.*

Proof. Since the system provides full resource in HI mode, $\text{sbf}_{\text{HI}}(\tau^{\text{H}}, \Delta, \text{t}_{\text{ms}}) = \Delta$. From Theorem 3, we know that $\text{dbf}_{\text{HI}}(\tau^H, \Delta) \geq \text{dbf}_{\text{HI}}(\tau^H, \Delta, t_{ms})$. Since the constraint of Eq. 6b has guaranteed $\text{dbf}_{\text{HI}}(\tau^H, \Delta) \leq \Delta$, we have $\text{dbf}_{\text{HI}}(\tau^H, \Delta, t_{ms}) \leq \text{sbf}_{\text{HI}}(\tau^{\text{H}}, \Delta, \text{t}_{\text{ms}}) = \Delta$. □

**Automatic Schedulability Guarantee.** From the above we conclude that, if the schedulability of HI mode needs to be guaranteed, we only need to choose appropriate LO mode deadlines offline that can make Conditions LO and HI hold. At runtime, task overrun will not change the schedulability of HI mode if Condition LO-t holds. Therefore, the FFOB mode-switch scheme only needs to choose appropriate LO mode deadlines at the beginning. At runtime, tasks are allowed to overrun by only guaranteeing LO mode schedulability, because this can automatically guarantee HI mode schedulability.

## 4. EVALUATION

We now evaluate the FFOB mode-switch scheme with an avionic task set and extensive simulations on synthetic task sets. In particular, we evaluate the performance of the proposed FFOB in this paper and some other known approaches; the compared approaches are:

1. EDF-B: The basic EDF-VD scheduling that forces the mode-switch whenever a HI-critical task overruns.
2. EDF-FFOB-S: The EDF-VD scheduling with a simple FFOB mode-switch scheme that resets $OB$ to its initial value $\rho^*(t_0)$ whenever there is an idle tick. This approach does not update $OB$ at runtime when $OB$ elapses to 0, in order to have $O(1)$ online complexity.
3. EDF-FFOB-A: The EDF-VD scheduling with an advanced FFOB mode-switch scheme that updates $OB$ at runtime when $OB$ elapses to 0 and resets to its initial value $\rho^*(t_0)$ when an idle tick emerges.
4. FP-B: The basic FP scheduling that applies AMC-rtb and Audsley algorithm to assign task priorities and forces the mode-switch whenever a HI-critical task overruns, see [3].
5. FP-SOA: The FP scheduling with the static overrun allowance. $C_i^L$ of HI-critical tasks are fairly increased while ensuring the MCS provably schedulable [24].
6. FP-SOA-B: The bailout protocol enhanced by offline increasing $C_i^L$ [6].

The following metrics are used to evaluate the performance of the listed approaches above.

- Dropped Jobs of LO-Critical Tasks ($DJ_{LO}$): This metric represents the number of dropped jobs of LO-critical tasks in an interval. Note that, no HI-critical jobs will be dropped in all compared approaches.
- Time Ratio of HI-mode ($TR_{HI}$): This metric represents the time ratio within an interval that the system stays in HI mode. The smaller $\text{TR}_{\text{HI}}$ is, the longer that the system stays in LO mode where all tasks can run.
- Number of Mode-Switch ($N_{ms}$): This metric represents the number of mode-switches in an interval.

Table 1: Flight management system parameters

| $\tau_i$ | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $\tau_5$ |
|---|---|---|---|---|---|
| T/D | 200 | 1000 | 1600 | 100 | 200 |
| $C_i^L$ | [0,20] | [0,20] | [0,20] | [0,20] | [0,20] |
| $\mathcal{X}_i$ | HI | HI | HI | HI | HI |
| $\tau_i$ | $\tau_6$ | $\tau_7$ | $\tau_8$ | $\tau_9$ | |
| T/D | 1000 | 1000 | 1000 | 1000 | |
| $C_i^L$ | [0,200] | [0,200] | [0,200] | [0,200] | |
| $\mathcal{X}_i$ | LO | LO | LO | LO | |

Table 2: Number of dropped jobs (FMS)

| $OP$ | EDF-FFOB-S | EDF-FFOB-A | FP-SOA-B |
|---|---|---|---|
| $10^{-5}$ | 0 | 0 | 1 |
| $5 * 10^{-5}$ | 8 | 2 | 17 |
| $10^{-4}$ | 24 | 18 | 40 |
| $5 * 10^{-4}$ | 92 | 58 | 196 |
| $10^{-3}$ | 189 | 136 | 414 |
| $5 * 10^{-3}$ | 924 | 616 | 1949 |
| $10^{-2}$ | 1896 | 1356 | 3971 |
| $5 * 10^{-2}$ | 12153 | 7873 | 19854 |
| $10^{-1}$ | 29520 | 18848 | 40309 |

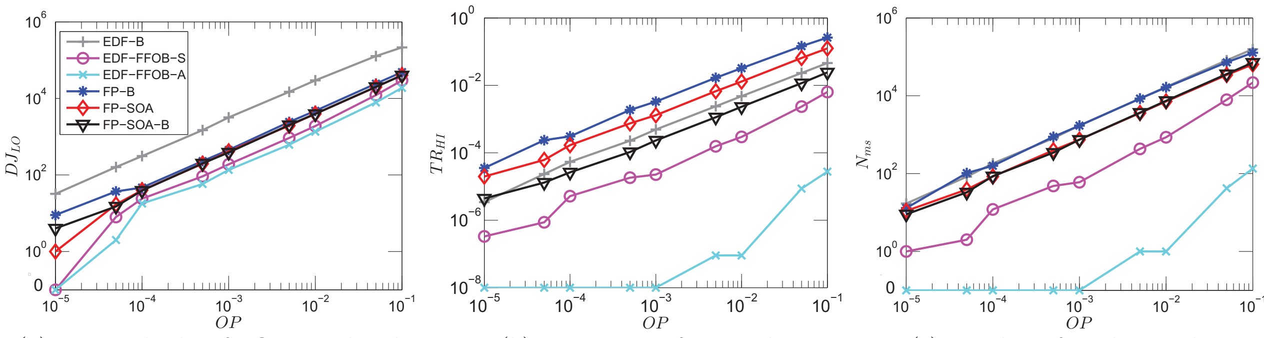
### 4.1 Flight management system

The first set of experiments is conducted on a subset of the flight management system (FMS), which consists of 5 HI-critical tasks and 4 LO-critical tasks. Task parameters come from an industrial partner and are shown in Tab. 1 with timing units of $ms$. We generate random LO WCETs conforming to Tab. 1 and use a criticality factor ($CF$) to get HI WCETs: $C_i^H = CF \cdot C_i^L$ for HI-critical tasks.

In the experiments, we simulate the system for $10^8\,ms$ and record the $TR_{HI}$, $N_{ms}$, $DJ_{LO}$. We set $CF = 7$ so that the task set is unschedulable by the conventional EDF or FP algorithms. The probability that any task may overrun its LO WCET is denoted as $OP$. As we want to investigate the effect of $OP$ on the performance of the compared approaches, $OP$ is set to one of $\{10^{-5},\ 5 \cdot 10^{-5},\ 10^{-4},\ 5 \cdot 10^{-4},\ 10^{-3},\ 5 \cdot 10^{-3},\ 10^{-2},\ 5 \cdot 10^{-2},\ 10^{-1}\}$. $OP$ is deliberately set to a relatively high value as we want to evaluate the FFOB performance w.r.t. $OP$. Indeed, this can happen if LO WCETs are estimated in a loose way. Once a task (including LO-critical tasks) overruns, the actual execution time is set as a random number in $\left(C_i^L,\ CF \cdot C_i^L\right]$; otherwise, the actual execution time is set as a random number in $[0.6 \cdot C_i^L,\ C_i^L]$.

The simulation results are presented in double logarithmic coordinates in Fig. 6. All figures are best seen online in color. Fig. 6(a) shows the number of dropped jobs w.r.t. $OP$. From it, we get the overview that EDF-FFOB-A and EDF-FFOB-S have the least number of dropped jobs. EDF-B has the largest number of dropped jobs. FP-B, FP-SOA and FP-SOA-B have similar number of dropped jobs. This is because in EDF-B, all LO-critical jobs are dropped whenever an overrun of a HI-critical job triggers the mode-switch. However, in FP-B, FP-SOA and FP-SOA-B, the LO-critical jobs at the moment of mode-switch can still run until they use up their execution budget, i.e., $C_i^L$. Among FP-B, FP-SOA and FP-SOA-B, FP-SOA-B has the least number of dropped jobs as FP-SOA-B statically increases $C_i^L$ and applies the bailout protocol to timely switch back to LO mode. The concrete number of dropped jobs of EDF-FFOB-A, EDF-FFOB-S and FP-SOA-B are shown in Tab. 2. Compared to FP-SOA-B, we find that EDF-FFOB-A has around half dropped jobs and EDF-FFOB-S has three quarters of dropped jobs.

Fig. 6(b) and Fig. 6(c) show the time ratio of HI mode $TR_{HI}$ and mode switch times $N_{ms}$. We find that EDF-



(a) Dropped jobs of LO-critical tasks (b) Time ratio of HI mode (c) Number of mode switches

Figure 6: FMS: simulation results of the compared approaches (subfigures share the same color scheme)

FFOB-A is two orders of magnitude lower than FP-SOA-B in both $TR_{HI}$ and $N_{ms}$, even though FP-SOA-B has the least $TR_{HI}$ and $N_{ms}$ among FP-B, FP-SOA and FP-SOA-B. Furthermore, EDF-FFOB-S with an $O(1)$ online complexity also performs well in saving dropped jobs, reducing the mode-switch times and increasing the time ratio that all tasks are scheduled.

## 4.2 Extensive simulations

In order to validate our proposed techniques on general task sets, we now apply them to randomly generated task sets. We adopt a similar random task generator as used in [6]. A task set consists of 8 tasks; the parameters of each task are generated as follows.

- *Periods and Deadlines* - The period of each task is chosen at random from a set of periods, {20, 25, 40, 50, 80, 100 200, 250, 400, 800, 1000}. Those periods are typically found in automotive and avionics systems [5]. Task deadlines are the same as their periods.
- *Criticality* - A task is generated as a HI-critical task with 50% probability.
- *Execution Times* - The LO WCETs are determined according to the UUniFast algorithm [8], which ensures that utilizations are distributed on tasks without bias. We set the sum utilization of a task set as 0.7. Once a task gets a utilization, its LO WCET is $C_i^L = U_i \cdot T_i$, where $U_i$ is its utilization. If this task is HI-critical, its HI WCET is $C_i^H = CF \cdot C_i^L$, where $CF = 2$. During simulations, the actual execution time is within $[0.6 \cdot C_i^L, CF \cdot C_i^L]$. Each job has a probability $OP$ to overrun its LO WCET. If a task does not overrun, its actual execution time is a random number in $[0.6 \cdot C_i^L, C_i^L]$. If a task overruns, its actual execution time is a random number in $(C_i^L, CF \cdot C_i^L]$. $OP$ is set as one of $\{0.0001,\ 0.001,\ 0.01\}$.

*Simulation:* We generate 50 task sets and apply the compared approaches to schedule them w.r.t. $OP$. The simulation time on each task set is $10^7$ time units.

*Results:* The simulation results are shown in Figs. 7, 8 and 9 by boxplot. The results of approaches without FFOB and the approach with a simple implementation of FFOB, i.e., EDF-FFOB-S, are presented together in order to highlight the difference between the state-of-art approaches and our proposed approach. Since the metrics of EDF-FFOB-S and EDF-FFOB-A are far smaller than other approaches, we present the results of EDF-FFOB-S and EDF-FFOB-A together to highlight the further improvement of the FFOB mechanism by making use of dynamic slacks.

In the three figures, we find that EDF-FFOB-S outperforms other approaches to a large extent in reducing the three metrics. In particular, for the median of the number of dropped jobs across all approaches in Fig. 7, with regard to OP $= 0.0001/0.001/0.01$, EDF-FFOB-S achieves 22/34/27 folds reduction compared to FP-B, 14/21/16 folds reduction compared to FP-SOA, 10/16/12 folds reduction compared to FP-SOA-B, and 21/31/23 folds reduction compared to EDF-B. The right plots of Fig. 7 demonstrate that, by exploiting the dynamic slack at runtime, FFOB can further reduce the number of dropped jobs by 5/4.9/5.4 folds compared to FFOB only making use of the static slack. The big advantages of FFOB are also confirmed by investigating the time ratio in HI mode and the mode-switch times, as presented in Figs. 8 and 9.

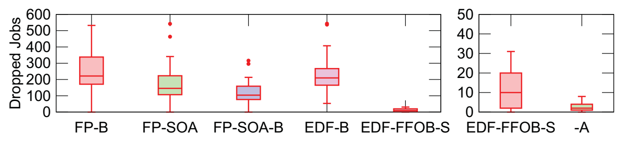
In addition, by comparing the results of the EDF-FFOB-S and EDF-FFOB-A under $OP = 0.001$ to the results of other approaches under $OP = 0.0001$, we find that the two FFOB approaches still perform better. This result is quite useful for guiding us on how to assign tasks WCETs. The overrun probability often represents the level of pessimism of the used WCET, i.e., the WCET under $OP = 0.001$ is less pessimistic and smaller than the WCET under $OP = 0.0001$. Then, if we artificially shorten tasks LO WCETs and meanwhile apply the FFOB mechanism to postpone the mode-switch, the system may schedule more tasks without lowering the QoS for LO-critical tasks. Besides, by lowering the LO WCETs, the system can harvest more static slacks, which can be flexibly used to allow all tasks to overrun and thus make the system switch the mode less frequent.

In summary, the FFOB mechanism can greatly improve the MCS performance as it makes use of the possible slack existing in the system to schedule the overrun jobs, without jeopardizing the basic properties of MCSs.

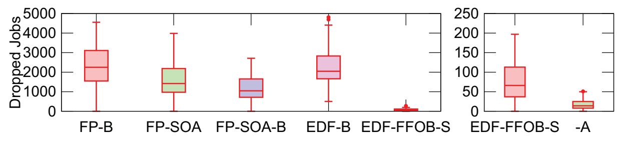
## 5. CONCLUSIONS

We propose in this paper an on-the-fly fast overrun budgeting scheme (FFOB) to allow tasks to overrun their LO WCETs, while guaranteeing that the system is mixed-criticality schedulable. This scheme is lightweight as it only needs a timer in the system to manage the overrun allowance. At runtime, the overrun budget can be replenished if it runs up, which further extends the overrun allowance and postpones the mode-switch. Experimental results show that FFOB, even with a simple implementation, outperforms other known methods to a large extent.

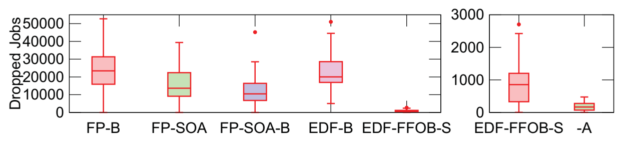
The proposed FFOB scheme provides an interesting possibility. Since FFOB can make use of the system slack at runtime to allow tasks to overrun their LO WCETs and effectively postpone the mode-switch, it may be a good option to artificially decrease the the tasks' LO WCETs. Compared to the system without FFOB, this can make the system schedule more tasks, without lowering the QoS for LO-critical tasks and the guarantee to all critical tasks.



(a) OP = 0.0001

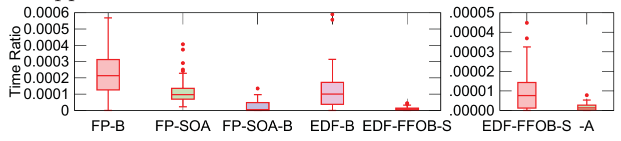


(b) OP = 0.001

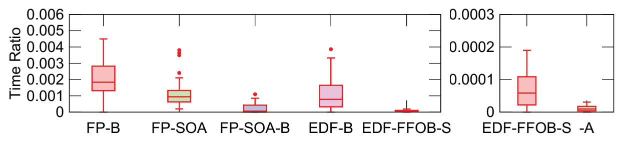


(c) OP = 0.01

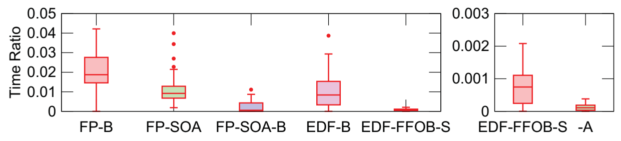
Figure 7: Box-Plot – number of dropped jobs with different overrun probabilities (OP); -A in the right plots represents the approach EDF-FFOB-A



(a) OP = 0.0001

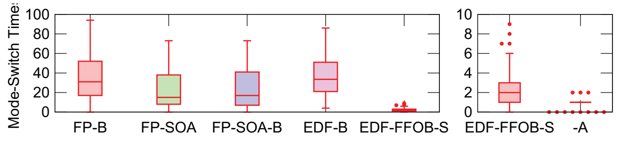


(b) OP = 0.001

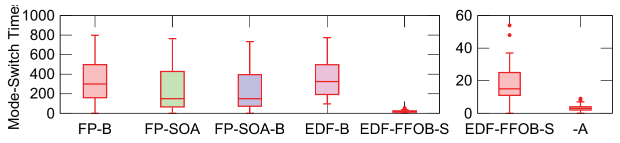


(c) OP = 0.01

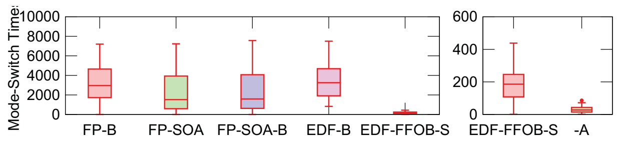
Figure 8: Box-Plot – HI mode time ratio with different overrun probabilities (OP); -A in the right plots represents the approach EDF-FFOB-A



(a) OP = 0.0001



(b) OP = 0.001



(c) OP = 0.01

Figure 9: Box-Plot – mode-switch times with different overrun probabilities (OP); -A in the right plots represents the approach EDF-FFOB-A

## 6. REFERENCES

[1] M. A. Awan and S. M. Petters. Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems. In *Real-Time Systems (ECRTS)*, 2011.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[3] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS)*, 2011.

[4] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium (RTSS)*, 1990.

[5] I. Bate and A. Burns. An integrated approach to scheduling in safety-critical embedded control systems. *Real-Time Systems*, 25(1):5–37, 2003.

[6] I. Bate, A. Burns, and R. I. Davis. A bailout protocol for mixed criticality systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015.

[7] H. Biao, H. Kai, C. Gang, C. Long, and K. Alois. Adaptive runtime shaping for mixed-criticality systems. In *Conference on Embedded Software (EMSOFT)*, 2015.

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

[9] E. Bini, M. Di Natale, and G. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39(1-3):5–30, 2008.

[10] A. Burns and S. Baruah. *Timing faults and mixed criticality systems*. Springer, 2011.

[11] A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems (colocated with RTSS)*, 2013.

[12] A. Burns and R. Davis. Mixed criticality systems: A review. *University of York, Tech. Rep*, 2015.

[13] A. Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Real-Time Systems Symposium (RTSS)*, 2013.

[14] P. Ekberg and W. Yi. Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

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

[16] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Adaptive workload management for mixed-criticality systems. *ACM Transactions in Embedded Computing Systems*, 2016.

[17] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Evaluation and improvements of runtime monitoring methods for real-time event streams. *ACM Transactions in Embedded Computing Systems*, 2016.

[18] B. Hu, K. Huang, G. Chen, and A. Knoll. Evaluation of runtime monitoring methods for real-time event streams. 2015.

[19] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Adaptive dynamic power management for hard real-time systems. In *Real-Time Systems Symposium (RTSS)*, 2009.

[20] 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 Systems*, 47(2):163–193, 2011.

[21] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[22] Y.-H. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[23] R. RTCA-DO-178B. Software considerations in airborne systems and equipment certification. In *Radio Technical Commission for Aeronautics (RTCA)*, 1992.

[24] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[25] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium (RTSS)*, 2007.