Real-Time Systems manuscript No.
(will be inserted by the editor)

A Novel View on Bounding Execution Demand under Mixed-Criticality EDF

Mitra Mahdiani · Alejandro Masrur

Received: date / Accepted: date

Abstract In this paper, we are concerned with scheduling a mix of high-criticality (HI) and low-criticality (LO) tasks under Earliest Deadline First (EDF) on one processor. To this end, the system implements two operation modes, LO and HI mode. In LO mode, HI tasks execute for no longer than their *optimistic* execution budgets and are scheduled together with the LO tasks. The system switches to HI mode, where all LO tasks are prevented from running, when one or more HI tasks run for longer than expected. Since these mode changes may happen at arbitrary points in time, it is difficult to find an accurate bound on *carry-over jobs*, i.e., those HI jobs that were released before, but did not finish executing at the point of the transition. To overcome this problem, we propose a technique that works around the computation of carry-over execution demand. Basically, the proposed technique separates the schedulability analysis of the transition between LO and HI mode from that of *stable* HI mode. We prove that a transition from LO to HI mode is feasible, if an equivalent task set derived from the original is schedulable under plain EDF. On this basis, we can apply approximation techniques such as, e.g., the well-known Devi's test to derive further tests that trade off accuracy versus complexity/runtime. Finally, we perform a detailed comparison with respect to weighted schedulability on synthetic data illustrating benefits by the proposed technique.

1 Introduction

There is increasingly important trend in domains such as automotive systems, avionics, and medical engineering towards integrating functions with different levels of criticality onto a common hardware platform. This allows for a reduction of costs and complexity, however, it leads to mixed-criticality (MC) systems that

Mitra Mahdiani and Alejandor Masrur
Department of Computer Science
TU Chemnitz, Germany
E-mail: mitra.mahdiani@informatik.tu-chemnitz.de

require careful design and analysis. In particular, it must be guaranteed that high-criticality (HI) functions/tasks are not affected by low-criticality (LO) tasks that share the same resources.

In this paper, we are concerned with scheduling a mix of HI and LO tasks under EDF and on one processor — however, a discussion for more levels of criticality is presented in the appendix. In particular, we make use of Vestal's task model [20]. That is, LO tasks are modeled by only one worst-case execution time (WCET) (apart from inter-arrival time and deadline), while HI tasks are characterized by an optimistic and by a conservative WCET to account for potential increases in execution demand [20]. In this context, a standard real-time scheduling requires guaranteeing that LO and HI tasks meet their deadlines when HI tasks' conservative WCETs are considered. However, since HI tasks run for at most their optimistic WCETs almost all the time, this leads to an inefficient design. On the other hand, if only optimistic WCETs are considered, HI tasks may occasionally miss their deadlines.

To overcome this predicament, a common approach is to implement two operation modes.[1] In LO mode, HI tasks run for their optimistic WCETs and are scheduled within *virtual deadlines* together with all LO tasks. Virtual deadlines are given by $x_i \cdot D_i$ and are usually shorter than real deadlines D_i with $x_i \in (0, 1]$ being referred to as *deadline scaling factor*. The system switches to HI mode, when one or more HI tasks require running for longer (up to their conservative WCETs). HI tasks are then scheduled within their *real deadlines* and LO tasks are stopped from running (i.e., discarded).

In such a setting, apart from guaranteeing schedulability of individual modes, it is necessary to guarantee schedulability of transitions between modes. However, since transitions between LO and HI mode can happen at arbitrary points in time, *carry-over jobs* cannot be avoided, i.e., HI jobs that are released prior to, but have not finished executing at the moment of the transition.

Approaches from the literature such as GREEDY [12] and ECDF [11] focus on bounding execution demand by carry-over jobs, resulting in relatively complex algorithms. In this paper, we propose a novel technique that works around the computation of carry-over execution demand. We prove that transitions between LO and HI mode are feasible, if an equivalent task set derived from the original one is schedulable under plain EDF. This not only reduces the overall complexity, but it also improves our understanding on MC scheduling. Moreover, we can apply well-known approximation techniques such as Devi's test [10] to trade off accuracy for complexity/runtime. We present a large set of experiments based on synthetic data illustrating the benefits of the proposed approach in term of weighted schedulability and runtime compared to the most prominent approaches from the literature.

The rest of the paper is structured as follows. Related work is briefly discussed in Section 2. Section 3 introduces the task model and assumptions used, whereas Section 4 deals with the proposed technique for bounding execution demand under mixed-criticality EDF and perform an analytical comparison with the GREEDY and the ECDF algorithm in Section 5. Further, we apply Devi's test in Section 6 to derive two approximated variants of our proposed approach, i.e., based on *per-task*

[1] If more than two criticality levels are to be regarded, there will be one operation mode per criticality level.

and on *uniform* deadline scaling. In Section 7, we present extensive experimental results evaluating the proposed technique, whereas Section 8 concludes the paper. Finally, in the appendix, we briefly investigate how to extend the proposed technique to more than two levels of criticality.

2 Related Work

In this section, we briefly revise the rich literature concerning scheduling MC tasks on one processor and, in particular, under EDF — a complete overview can be found in [8].

The problem around MC systems was first addressed by Vestal in [20], who proposed modeling HI tasks with multiple WCET parameters to account for potential increases in execution demand. Based on this model, Baruah *et al.* later analyzed per-task priority assignments and the resulting worst-case response times [3].

In [2], Baruah *et al.* also proposed the EDF-VD algorithm to schedule a mix of HI and LO tasks. EDF-VD introduces two operation modes and uses a *priority-promotion* scheme by uniformly scaling deadlines of HI tasks. The resulting speed-up factor was shown to be equal to $(\sqrt{5}+1)/2$ [2]. Later this speed-up factor was improved to $4/3$ [4]. Baruah *et al.* further proposed extensions to EDF-VD, where, in particular, a per-task deadline scaling is used [5]. However, they also concluded that the speed-up factor of $4/3$ cannot be improved [5]. Similarly, Müller presented a more general per-task deadline scaling technique that allows improving schedulability [18].

Improvements to the original EDF-VD have also been proposed by other authors. In [19], Su and Zhu used an elastic task model [15] to improve resource utilization in MC systems. In [22], Zhao et al. applied preemption thresholds [21] in MC scheduling in order to better utilize the processing unit. In [17], a technique consisting of two scaling factors is proposed for admission control in MC systems.

A more flexible approach referred to as GREEDY with per-task deadline scaling was presented by Ekberg and Yi for the case of two criticality levels [12]. Ekberg and Yi characterized the execution demand of MC systems under EDF by deriving demand bound functions for the LO and the HI mode. Later, they extended this work to the case of more than two criticality levels [13]. In [11], Easwaran presented a similar technique called ECDF also for the case of two criticality levels and showed that it strictly dominates the GREEDY approach.

In contrast to GREEDY and ECDF, in [16], we proposed working around the computation of carry-over execution demand at transitions between LO and HI mode. In particular, testing schedulability of transitions from LO to HI reduces to testing schedulability of an equivalent task set under plain EDF [16]. In this paper, we extend this work and propose using approximation techniques, particularly, Devi's test to derive schedulability tests for MC systems under EDF that trade off accuracy versus performance as discussed later in detail.

3 Models and Assumptions

In this section, we discuss most of our notation. Similar to [12] and [11], we basically adopt the task model originally proposed in [20]. We consider a set τ of MC tasks that are independent, preemptable and sporadic running on one processor under preemptive EDF scheduling. Each individual task τ_i in τ is characterized by its minimum inter-release time T_i, i.e., the minimum distance between two consecutive jobs or instances of a task, and by its relative deadline D_i with $D_i \leq T_i \forall i$. Further, we assume that tasks do not self-suspend and that the overhead by context switches has been accounted for in tasks' WCETs or can be neglected.

As already stated, we are concerned with dual-criticality systems with two levels of criticality, namely LO and HI.[2] The *criticality* of a task τ_i is denoted by χ_i with:

$$\chi_i \in \{LO, HI\}.$$

A LO task is associated with only one WCET value/estimate denoted by C_i^{LO}. Opposed to this, a HI task is characterized by its optimistic WCET estimate C_i^{LO} and its conservative WCET estimate C_i^{HI} with:[3]

$$C_i^{LO} < C_i^{HI} \leq D_i \leq T_i.$$

The system basically distinguishes two operation modes denoted by m: LO and HI mode. In LO mode, HI tasks execute for no longer than C_i^{LO}, whereas these might require executing for up to C_i^{HI} in HI mode. The system initializes in LO mode where all LO and HI tasks need to meet their deadlines. As soon as one job of a HI task executes for longer than its C_i^{LO}, the system switches to HI mode discarding all LO tasks. Similar to context switches, we assume that the overhead by mode switches has been accounted for in C_i^{HI}.

Finally, we denote the *utilization* by LO and HI tasks in the LO and HI mode respectively as follows:

$$U_\chi^m := \sum_{\chi_i = \chi} \frac{C_i^m}{T_i},$$

where again χ and m can assume values in $\{LO, HI\}$. Note that only U_{LO}^{LO}, U_{HI}^{LO} and U_{HI}^{HI} exist. U_{LO}^{HI} is effectively zero, since LO tasks are dropped and, hence, do not run in HI mode.

Mixed-Criticality EDF. A common approach when scheduling MC systems is to shorten the deadlines of HI jobs in LO mode. This way, processor capacity can be *reserved* for a potential switch to HI mode — where HI tasks require more execution demand. In other words, we assign a *virtual deadline* equal to $x_i \cdot D_i$ with $x_i \in (0, 1]$ to all τ_i with $\chi_i = HI$.

This virtual deadline is used instead of D_i — the *real* deadline — to schedule HI tasks in LO mode. The parameter x_i is the so-called *deadline scaling factor*. There is no deadline scaling for LO tasks such that they are scheduled (only in LO mode) using their D_i.

When the system switches to HI mode, HI tasks start being scheduled according to their real deadlines D_i whereas LO tasks are discarded immediately. In this

[2] See the appendix for an extension to more than two levels of criticality.

[3] Note that either real or integer numbers can be used for C_i^{LO}, C_i^{HI}, D_i and T_i.

paper, we consider that tasks are scheduled under EDF in both modes and refer to this scheme as *mixed-criticality* EDF.

Clearly, whereas schedulability of separated modes can be easily tested, i.e., when the system is stable in either LO or HI mode, it is difficult to test schedulability of transitions between modes. In particular, careful analysis is required when the system switches from LO to HI mode.

In this paper, similar to other approaches from the literature, transitions from HI back to LO mode are disregarded. The reason is that, in contrast to changes from LO to HI, a change from HI to LO mode can be programmed or postponed to a suitable point in time, e.g., at which the processor idles, and does not require further analysis.

The EDF-VD Algorithm. EDF with Virtual Deadlines (EDF-VD) is a special case of mixed-criticality EDF for the case $D_i = T_i \ \forall i$. Under EDF-VD, virtual deadlines are obtained by $x \cdot D_i$, where $x \in (0, 1]$ is a uniform deadline scaling factor for all HI tasks [2].

Under EDF-VD, the LO and HI tasks need to be schedulable with their corresponding C_i^{LO} under EDF in LO mode. Similarly, in HI mode, the HI tasks also need to be schedulable with their corresponding C_i^{HI} under EDF. As a result, the following two schedulability conditions are necessary:

$$U_{LO}^{LO} + U_{HI}^{LO} \leq 1 \tag{1}$$

$$U_{HI}^{HI} \leq 1 \tag{2}$$

Note that these two conditions do not ensure schedulability in the transition phase. Hence, they are indeed only necessary, but not sufficient conditions. In [4], Baruah *et al.* also obtained a sufficient schedulability condition for EDF-VD in the form of a utilization bound: $\max\left(U_{LO}^{LO} + U_{HI}^{LO}, U_{HI}^{HI}\right) \leq 3/4$. They also proposed a more accurate schedulability test based on whether feasible lower and upper bounds can be obtained on x [4]:

$$\frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}} \leq x, \tag{3}$$

$$x \leq \frac{1 - U_{HI}^{HI}}{U_{LO}^{LO}}. \tag{4}$$

That is, if the value of x obtained with (3) is less than or equal to the value obtained with (4), then it is possible to find a valid x for the considered system rendering it schedulable under EDF-VD.

4 Bounding Execution Demand

In this section, we are concerned with characterizing execution demand by τ under mixed-criticality EDF. In [12] and [11], a demand bound function was presented for each mode in the system, i.e., one for the LO mode and one for the HI mode. As mentioned above, we derive a third demand bound function for the transition between modes. This allows working around the computation of carry-over execution demand, reducing the amount of pessimism and, hence, relaxing schedulability

conditions in HI mode [16].

Schedulability in LO mode. In LO mode, LO tasks need to scheduled together with HI tasks, while the latter are assigned virtual deadlines $x_i \cdot D_i$. As a consequence, the resulting demand bound function $\mathrm{dbf}_{LO}(t)$ in LO mode is given by:

$$\mathrm{dbf}_{LO}(t) = \sum_{\chi_i=LO} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} + \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}. \quad (5)$$

Here $t \geq 0$ is a (real) number representing time, i.e., $\mathrm{dbf}_{LO}(t)$ returns a τ's maximum execution demand in LO mode in an interval of length t. Note that $\mathrm{dbf}_{LO}(t)$ is always greater than or equal to zero, since $D_i \leq T_i$ holds for all τ_i and x_i has values in $(0, 1]$.

The system is schedulable in LO mode, if $\mathrm{dbf}_{LO}(t) \leq t$ holds for all possible t until the processor first idles [1]. That is, until a point in time $\hat{t}_{LO}$ for which the following holds:

$$\hat{t}_{LO} = \sum_{\chi_i=LO} \left(\left\lfloor \frac{\hat{t}_{LO} - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} + \sum_{\chi_i=HI} \left(\left\lfloor \frac{\hat{t}_{LO} - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO},$$

where removing the floor function leads to:

$$\hat{t}_{LO} \leq \sum_{\chi_i=LO} \left(\frac{\hat{t}_{LO} - D_i}{T_i} + 1 \right) C_i^{LO} + \sum_{\chi_i=HI} \left(\frac{\hat{t}_{LO} - x_i \cdot D_i}{T_i} + 1 \right) C_i^{LO},$$

and reshaping to:

$$\hat{t}_{LO} \leq \hat{t}_{LO} \left(\sum_{\chi_i=LO} \frac{C_i^{LO}}{T_i} + \sum_{\chi_i=HI} \frac{C_i^{LO}}{T_i} \right) + \sum_{\chi_i=LO} (T_i - D_i) \frac{C_i^{LO}}{T_i} + \sum_{\chi_i=HI} (T_i - x_i \cdot D_i) \frac{C_i^{LO}}{T_i}.$$

Finally, considering that $U_{LO}^{LO} = \sum_{\chi_i=LO} \frac{C_i^{LO}}{T_i}$ and $U_{HI}^{LO} = \sum_{\chi_i=HI} \frac{C_i^{LO}}{T_i}$, we obtain an upper bound on $\hat{t}_{LO}$:

$$\hat{t}_{LO} \leq \frac{\sum_{\chi_i=LO} (T_i - D_i) \frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{HI}^{LO}} + \frac{\sum_{\chi_i=HI} (T_i - x_i \cdot D_i) \frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{HI}^{LO}}. \quad (6)$$

The bound in (6) depends on the values of x_i, which are not known a priori. On the other hand, note that this bound maximizes when all x_i tend to 0, which then leads to the following:

$$\hat{t}_{LO} \leq \frac{\sum\limits_{\chi_i=LO}(T_i - D_i)\frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{HI}^{LO}} + \frac{\sum\limits_{\chi_i=HI} C_i^{LO}}{1 - U_{LO}^{LO} - U_{HI}^{LO}}, \tag{7}$$

Clearly, $U_{LO}^{LO} + U_{HI}^{LO}$ — the utilization in LO mode — must be strictly less than one in order that (6) and (7) return valid and finite values.

Schedulability in stable HI mode. In this case LO tasks do not run and HI tasks run for their corresponding C_i^{HI} leading to the following demand bound function:

$$\mathrm{dbf}_{HI}(t) = \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{HI}, \tag{8}$$

where again $t \geq 0$ is a (real) number representing time, i.e., $\mathrm{dbf}_{HI}(t)$ returns the maximum execution demand in a time interval of length t.

The system is schedulable in stable HI mode, if $\mathrm{dbf}_{HI}(t) \leq t$ for all possible t until the processor first idles, i.e., until a point in time $\hat{t}_{HI}$ is reached where $\mathrm{dbf}_{HI}(\hat{t}_{HI}) = \hat{t}_{HI}$. We again can remove the floor function in (8) to obtain an upper bound on $\hat{t}_{HI}$:

$$\hat{t}_{HI} \leq \frac{\sum\limits_{\chi_i=HI}(T_i - D_i)\frac{C_i^{HI}}{T_i}}{1 - U_{HI}^{HI}}. \tag{9}$$

U_{HI}^{HI} — the utilization in HI mode — must be strictly less than one in order that (9) returns a valid and finite upper bound on $\hat{t}_{HI}$.

Schedulability in the transition from LO to HI mode. The transition from LO to HI mode may happen at an arbitrary point in time when one HI job exceeds its LO execution budget C_i^{LO}. Unfinished LO jobs are discarded at that time; however, the problem arises with HI jobs that were released before but have not finished executing their C_i^{LO}, i.e., carry-over jobs. Since it is difficult to accurately bound the execution demand by carry-over jobs, usually, pessimistic assumptions need to taken.

The following theorem, is a generalization of a theorem in [17] and allows working around carry-over jobs and characterizing the *additional* execution demand in the transitions from LO to HI mode in a more effective manner. In other words, this theorem allows us to guarantee schedulability without computing carry-over execution demand at the point of switching from LO to HI mode.

Theorem 1 *Given a set τ of MC tasks, let us assume that (i) $dbf_{LO}(t) \leq t$ and (ii) $dbf_{HI}(t) \leq t$ hold for $0 < t \leq \hat{t}_{LO}$ and $0 < t \leq \hat{t}_{HI}$ respectively, i.e., τ is schedulable in LO and stable HI mode. The transition from LO to HI mode is schedulable under mixed-criticality EDF, if $dbf_{SW}(t) \leq t$ also holds for $0 < t \leq \hat{t}_{SW}$, where $dbf_{SW}(t)$ is given by:*

$$dbf_{SW}(t) = \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - \Delta D_i}{T_i} \right\rfloor + 1 \right) \Delta C_i, \tag{10}$$

with $\Delta D_i = D_i - x_i \cdot D_i$, $\Delta C_i = C_i^{HI} - C_i^{LO}$, *and* $\hat{t}_{SW}$ *is upper bounded by the following expression:*

$$\hat{t}_{SW} \leq \frac{\sum\limits_{\chi_i=HI} \Delta C_i}{1 - U_{HI}^{SW}}, \tag{11}$$

where U_{HI}^{SW} *is given by* $\sum\limits_{\chi_i=HI} \frac{\Delta C_i}{T_i}$.

Proof Let us consider that the system *switches* to HI mode at time t' and that the processor idles for the first time thereafter at t'' with $t' < t''$, i.e., all jobs released prior to t'' finish executing at latest by t''. Clearly, jobs that are released after t'' are guaranteed schedulable by assumption (ii).

Let us now assume that a deadline is missed for the first time at t_{miss} by a job of any τ_i that we denote τ_{miss}. Clearly, t_{miss} must be in the interval $[t', t'']$ and the following must hold for $\delta_{miss} = t_{miss} - t'$:

$$\delta_{miss} < CO + \sum_{\chi_i=HI} \left(\left\lfloor \frac{\delta_{miss} - \phi_i - D_i}{T_i} \right\rfloor + 1 \right) C_i^{HI},$$

where $\phi_i = r_i' - t'$ is the *phase* of a τ_i at t', i.e., the release time r_i' of its first job after t' minus t'. Note that ϕ_i is in the interval $[0, T_i)$. In addition, CO denotes the *carry-over* execution demand at t'. This is the amount of execution in $[t', t_{miss}]$ by HI jobs that are released prior to t', but have not finished executing at t'.

As already discussed, it is difficult to determine CO in an accurate manner. Hence, to work around CO, let us first divide each τ_i, whose jobs have both release times and deadlines in $[t', t_{miss}]$, into two subtasks. The first subtask — denoted by τ_i^{LO} — is released for the first time at ϕ_i and requires executing C_i^{LO} within $x_i \cdot D_i$ every T_i time, i.e., this represents τ_i's execution demand in LO mode. The second subtask — denoted by τ_i^{SW} — is released for the first time at $\phi_i' = \phi_i + x_i \cdot D_i$ and requires executing $\Delta C_i = C_i^{HI} - C_i^{LO}$ within $\Delta D_i = D_i - x_i \cdot D_i$ every T_i time, i.e., this presents τ_i's increase in execution demand incurred in HI mode. Note that, in spite of this modification, the total amount of execution demand in $[t', t_{miss}]$ does not change, i.e., a deadline is still missed at t_{miss} as per assumption, and we can reshape the above inequality to:

$$\begin{aligned} \delta_{miss} < CO &+ \sum_{\chi_i=HI} \left(\left\lfloor \frac{\delta_{miss} - \phi_i - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} \\ &+ \sum_{\chi_i=HI} \left(\left\lfloor \frac{\delta_{miss} - \phi_i' - \Delta D_i}{T_i} \right\rfloor + 1 \right) \Delta C_i. \end{aligned} \tag{12}$$

Note that t_{miss} coincides with the deadline of the corresponding job of τ_{miss}^{SW}, which misses its deadline. (Recall that τ_{miss} is now divided into the subtasks τ_{miss}^{LO} and τ_{miss}^{SW}.) Now, there are two possible cases to consider in order to prove this theorem: The set of *only* subtasks τ_i^{SW} is either (1) unschedulable or (2) schedulable in isolation.

Case (1): This is a rather trivial case. If the set of only τ_i^{SW} is unschedulable in isolation, i.e., when scheduled alone on a single processor, $\mathrm{dbf}_{SW}(t) > t$ must hold for some t in $[0, \hat{t}_{SW}]$ with $\mathrm{dbf}_{SW}(t)$ given as per (10). As a result, we will

be able to detect a deadline miss in the transition between LO and HI mode by only testing the set of all τ_i^{SW}.

To this end, we need to find an upper bound on $\hat{t}_{SW}$ making $\mathrm{dbf}_{SW}(\hat{t}_{SW}) = \hat{t}_{SW}$ and removing the floor function as before:

$$\hat{t}_{SW} \leq \frac{\sum_{\chi_i=HI} (T_i - \Delta D_i)\frac{\Delta C_i}{T_i}}{1 - U_{HI}^{SW}}.$$

Here, $U_{HI}^{SW} = \sum_{\chi_i=HI} \frac{\Delta C_i}{T_i}$ is the utilization of the set of only τ_i^{SW}. Since $U_{HI}^{SW} < 1$ holds by assumption (ii), the above inequality returns a valid bound on $\hat{t}_{SW}$. This depends on $\Delta D_i = D_i - x_i \cdot D_i$ and, therefore, on the values of x_i, which we do not know in advance. However, we can make $x_i = 1$ for all i leading to the upper bound in (11).

Case (2): We show that this case leads to a contradiction. To this end, recall that we assumed that a deadline is missed for the first time at t_{miss}, hence, all previous jobs in $[t', t_{miss})$ can actually finish executing in time. Since now τ_{miss} is divided into the subtasks τ_{miss}^{LO} and τ_{miss}^{SW}, the τ_{miss}^{LO}'s job with a deadline equal to $t_{miss} - \Delta D_{miss}$ must push *carry-over* τ_i^{SW}'s jobs (i.e., those that are released prior to and have not finished executing at $t_{miss} - \Delta D_{miss}$ and that have deadlines prior to t_{miss}) by at least Δ_{miss} being Δ_{miss} the amount of the deadline miss at t_{miss}. Otherwise, no deadline can be missed at t_{miss}, since $\mathrm{dbf}_{SW}(t) \leq t$ is assumed to hold for all t. In addition, note that the processor does not idle in $[t_{miss} - \Delta D_{miss}, t_{miss}]$.

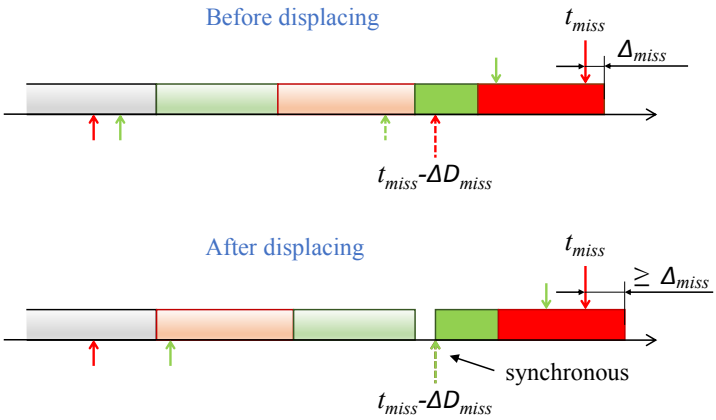

Figure 1: Illustration of Case (2.a) with $\Delta D_i \leq \Delta D_{miss}$: The carry-over τ_i^{SW}'s job is illustrated in green and τ_{miss}^{SW} is illustrated in red. The light green and red shading represent τ_i^{LO} and τ_{miss}^{LO} respectively, whereas a light gray shading represents any previous higher-priority execution (i.e., of jobs with shorter deadlines). Solid upward arrows stand for the release times of τ_i^{LO} and τ_{miss}^{LO}, while dashed upward arrows represent the release times of τ_i^{SW} and τ_{miss}^{SW} (which are also the (virtual) deadlines of τ_i^{LO} and τ_{miss}^{LO}). Solid downward arrows stand for the (real) deadlines of τ_i^{SW} and τ_{miss}^{SW}.

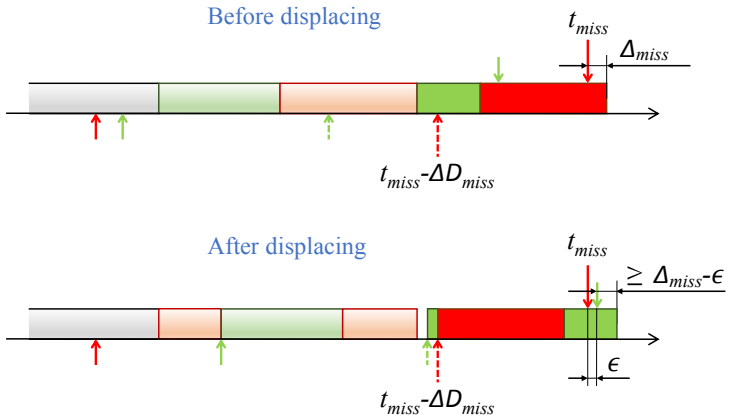

Figure 2: Illustration of Case (2.b) with $\Delta D_i > \Delta D_{miss}$: The carry-over τ_i^{SW}'s job is illustrated in green and τ_{miss}^{SW} is illustrated in red. The light green and red shading represent τ_i^{LO} and τ_{miss}^{LO} respectively, whereas a light gray shading represents any previous higher-priority execution (i.e., of jobs with shorter deadlines). Solid upward arrows stand for the release times of τ_i^{LO} and τ_{miss}^{LO}, while dashed upward arrows represent the release times of τ_i^{SW} and τ_{miss}^{SW} (which are also the (virtual) deadlines of τ_i^{LO} and τ_{miss}^{LO}). Solid downward arrows stand for the (real) deadlines of τ_i^{SW} and τ_{miss}^{SW}.

Case (2.a): Let us assume that there is only one carry-over job of an arbitrary τ_i^{SW} and that $\Delta D_i \leq \Delta D_{miss}$ holds. Note that, after moving this job forward to force its release time to coincide at $t_{miss} - \Delta D_{miss}$, τ_{miss}^{SW}'s job continues to miss its deadline by at least Δ_{miss} at t_{miss}, since the deadline of this carry-over τ_i^{SW}'s job remains within the interval $[t_{miss} - \Delta D_{miss}, t_{miss}]$. As a result, the amount of execution demand in $[t_{miss} - \Delta D_{miss}, t_{miss}]$ remains the same or even increases after this displacement — see Fig. 1.

The above analysis leads to a contradiction, since τ_{miss}^{SW}'s job and its carry-over τ_i^{SW}'s job are now released in synchrony at $t_{miss} - \Delta D_{miss}$. Consequently, τ_{miss}^{LO} cannot push any additional execution demand into $[t_{miss} - \Delta D_{miss}, t_{miss}]$ and, hence, if a deadline is missed at t_{miss}, $\mathrm{dbf}_{SW}(t) \leq t$ cannot hold for all t.

Case (2.b): Let us now assume that there is again only one carry-over job of an arbitrary τ_i^{SW}, however, $\Delta D_i > \Delta D_{miss}$ holds this time. Note that we can displace this carry-over τ_i^{SW}'s job forward until its deadline occurs at $t_{miss} + \epsilon$, where ϵ is an infinitesimally small number greater than zero. As a result, this τ_i^{SW}'s job starts missing its deadline by an amount equal to at least $\Delta_{miss} - \epsilon$, since at least the original execution demand in $[t_{miss} - \Delta D_{miss}, t_{miss}]$ is executed in $[t_{miss} - \Delta D_{miss}, t_{miss} + \epsilon]$ — see Fig. 2.

It is easy to see that we can now apply the analysis of Case (2.a) where the carry-over τ_i^{SW}'s job of this case misses its deadline at $t_{miss} + \epsilon$ (by an amount of at least $\Delta_{miss} - \epsilon$) and the τ_{miss}^{SW}'s job of this case becomes the carry-over job in Case (2.a). As a result, Case (2.b) also leads to a contradiction, i.e., if a deadline is missed at t_{miss}, $\mathrm{dbf}_{SW}(t) \leq t$ cannot hold for all t.

Clearly, there can be several carry-over jobs whose execution demands are pushed (at least partially) by τ_{miss}^{LO} into $[t_{miss} - \Delta D_{miss}, t_{miss}]$, however, the

Algorithm 1 Schedulability test for mixed-criticality EDF

Require: τ

Require: τ_{HI} /* subset of HI tasks */

1: X_{LW}=testLO(τ)
2: **if** testHI(τ_{HI}) = '*passed*' **and** $\mathrm{X}_{LW} \neq \emptyset$ **then**
3: $\quad \mathrm{X}_{UP}$ = testSW(τ_{HI})
4: $\quad$ **if** $\mathrm{X}_{UP} \neq \emptyset$ **and** $\mathrm{X}_{LW} \leq \mathbf{1} - \mathrm{X}_{UP}$ **then**
5: $\quad\quad$ Return '*passed*'
6: $\quad$ **else**
7: $\quad\quad$ Return '*not passed*'
8: $\quad$ **end if**
9: **end if**

total amount of execution demand pushed by τ_{miss}^{LO} remains Δ_{miss}. In this latter case, again, it is easy to see that we can apply the analysis of Case (2.a) and Case (2.b) to each individual carry-over job. Consequently, if a deadline is missed at t_{miss}, $\mathrm{dbf}_{SW}(t) > t$ must hold for some t. The theorem follows. □

Theorem 1 allows characterizing the *additional* execution demand in the transitions from LO to HI mode independent of carry-over jobs. Based on it, we test whether deadlines are met or not in $[t', t'']$, i.e., from the time t' of switching to HI mode to the time t'' at which the processor first idles after switching.

Finding deadline scaling factors. Now, we propose an algorithm to find valid values of x_i for each HI task in τ. Clearly, this is closely related to the technique used to *tighten* deadlines in LO mode. In this paper, however, we do not aim to improve deadline tightening. The contribution is rather a new technique for bounding demand execution, which can be combined with existing deadline tightening techniques, e.g., from [12] or [11].

The proposed algorithm shown in Alg. 1 essentially tests τ's schedulability in the LO mode (line 1), and in HI mode (line 2). If τ is schedulable in LO mode, the function testLO() returns a vector X_{LW} with the minimum values of x_i that could be found to be valid. If this vector is not empty, i.e., valid x_i values could be found, and τ is schedulable in HI mode, Alg. 1 tests schedulability at the transitions from LO to HI mode (line 3).

Further, if the set of HI tasks in τ — denoted by τ_{HI} — is schedulable at transitions from LO to HI, the function testSW() returns a vector X_{UP} with the minimum values of $1 - x_i$ that are also valid. That is, if X_{UP} is neither empty, the whole τ will be schedulable under mixed-criticality EDF provided that $\mathrm{X}_{LW} \leq \mathbf{1} - \mathrm{X}_{UP}$ holds (line 4). Here, $\mathbf{1}$ denotes a unity vector (where all elements are equal to one). That is, for each element in the vectors X_{LW} and X_{UP}, the following condition must hold:

Algorithm 2 Function testLO()

Require: τ

1: Compute $\hat{t}_{LO}$ by (7)
2: $\mathrm{X}_{LW} = \mathbf{1}$
3: **while** $t \leq \max(D_{max}, \hat{t}_{LO})$ **do**
4: **if** $\mathrm{dbf}_{LO}(t) > t$ **then**
5: **if** χ_i=*LO* **or** $\mathrm{dbf}_{LO}(t) - r_i > D_i$ **then**
6: $\mathrm{X}_{LW} = \emptyset$
7: Return
8: **end if**
9: **end if**
10: **if** χ_i=*HI* **then**
11: **if** Computed(i) = '*false*' **or** $\mathrm{X}_{LW}(i) < \frac{\mathrm{dbf}_{LO}(t) - r_i}{D_i}$ **then**
12: $\mathrm{X}_{LW}(i) = \frac{\mathrm{dbf}_{LO}(t) - r_i}{D_i}$ /* r_i = job's release time */
13: **end if**
14: **end if**
15: (t, i)=getNextDeadline()
16: **end while**
17: Return

$$\begin{aligned} \mathrm{X}_{LW}(i) &\leq x_i, \\ \mathrm{X}_{UP}(i) &\leq 1 - x_i, \\ \Longrightarrow\ x_i &\leq 1 - \mathrm{X}_{UP}(i). \end{aligned}$$

As already mentioned, the functions testLO() and testSW() — shown in Alg. 2 and Alg. 3 — test schedulability in LO mode and at the transitions from LO to HI mode. These two functions are very similar — apart from testLO() dealing with the whole τ and testSW() with the subset τ_{HI} — and return lower bounds on x_i and on $1 - x_i$ respectively. Thus, the following discussion of testLO() also applies to testSW().

Basically, testLO() computes $\mathrm{dbf}_{LO}(t)$ for all $0 \leq t \leq \hat{t}_{LO}$ starting from $x_i = 1$ for all HI tasks. However, at least the first deadline of each task should be checked, since we need to compute each x_i. That is, we need to compute $\mathrm{dbf}_{LO}(t)$ at least until $D_{max} = \max_{\forall i}(D_i)$ — see line 3 in Alg. 2. If the current t corresponds to a deadline of a HI task (lines 10 to 14), its (relative) virtual deadline $x_i \cdot D_i$ is adjusted such that its absolute deadline $r_i + x_i \cdot D_i$ is equal to $\mathrm{dbf}_{LO}(t)$ (i.e., the total execution demand at t).

Algorithm 3 Function testSW()

Require: τ_{HI}

1: Compute $\hat{t}_{SW}$ by (11)
2: $\mathrm{X}_{UP} = \mathbf{1}$
3: **while** $t \leq \max(D_{max}, \hat{t}_{SW})$ **do**
4: **if** $\mathrm{dbf}_{SW}(t) > t$ **and** $\mathrm{dbf}_{SW}(t) - r_i > D_i$ **then**
5: $\mathrm{X}_{UP} = \emptyset$
6: Return
7: **end if**
8: **if** Computed(i) = '*false*' **or** $\mathrm{X}_{UP}(i) < \frac{\mathrm{dbf}_{SW}(t)-r_i}{D_i}$ **then**
9: $\mathrm{X}_{UP}(i) = \frac{\mathrm{dbf}_{SW}(t)-r_i}{D_i}$ /* r_i = job's release time */
10: **end if**
11: (t, i)=getNextDeadline()
12: **end while**
13: Return

Note that the execution demand of jobs with prior deadlines to t is contained in $\mathrm{dbf}_{LO}(t)$. As a result, the computed x_i in line 12 can never compromise schedulability of these previous jobs. In addition, an x_i currently being computed can only replace a previously computed x_i, if either this is the first value of x_i computed for the corresponding τ_i, i.e., t coincides with the first deadline of τ_i, or this is greater than the previous value of x_i (lines 11 to 13). In other words, after initialization, the values of x_i that are selected never shorten τ_i's virtual deadline $x_i \cdot D_i$. As a result, we do not need to test τ_i's past deadlines anew. This deadline tightening technique reduces the number of possibilities for x_i, but it also keeps the algorithm simple.

Computed(i) in line 11 returns 'false', if no x_i has been computed yet for the current i, which we need to flag whether a value of x_i has been already computed for the current τ_i or not. The function getNextDeadline() in line 15 returns the point in time t at which the next deadline occurs and the index i of the task corresponding to that deadline. Clearly, this function has to take the computed values of x_i into account.

The function testLO() succeeds if it finishes testing $\mathrm{dbf}_{LO}(t)$ for $0 \leq t \leq \max(D_{max}, \hat{t}_{LO})$[4] and it could find a value of x_i in $(0, 1]$ for each HI task in τ. On the other hand, testLO() fails, if $\mathrm{dbf}_{LO}(t) > t$ holds for some t and either t corresponds to a deadline of a LO task — whose deadline cannot be adjusted by the used tightening technique — or the resulting x_i becomes greater than 1 (lines 4 to 8).

[4] As values of x_i start being known, we can update this bound in order to reduce runtime by testLO(). However, we did not implement this behavior in the current version.

Analogous to testLO(), testSW() computes $\text{dbf}_{SW}(t)$ for all deadlines in the interval $0 \leq t \leq \max(D_{max}, \hat{t}_{SW})$ starting from $1 - x_i = 1$ for all HI tasks — recall that deadlines in $\text{dbf}_{SW}(t)$ are equal to $(1-x_i)D_i$. Otherwise, as mentioned above, testLO() and testSW() are very similar and, hence, the above explanation for testLO() also applies to testSW(). Finally, the function testHI() in Alg. 1 is the known schedulability test for EDF from the literature [1] and, hence, does not require further discussion.

5 Analytical Comparison

In this section, let us first compare the proposed demand bound functions from Section 4 with those used by Ekberg and Yi in the GREEDY algorithm [12] and by Easwaran in the ECDF algorithm [11]. We show, for most cases, that the proposed ones result in tighter bounds on the execution demand than the other mentioned approaches.

5.1 The GREEDY algorithm

In LO mode, note that $\text{dbf}_{LO}(t)$ in (5) is identical to that of Ekberg and Yi — denoted by $\text{gdbf}_{LO}(t)$ in this paper. That is, $\text{dbf}_{LO}(t) = \text{gdbf}_{LO}(t)$ for all $0 < t \leq \hat{t}_{LO}$.

In HI mode, Ekberg and Yi proposed a demand bound function — denoted $\text{gdbf}_{HI}(t)$ in this paper — which is given by the following expression [12]:

$$\text{gdbf}_{HI}(t) = \sum_{\chi_i = HI} \left(\left\lfloor \frac{t - \Delta D_i}{T_i} \right\rfloor + 1 \right) C_i^{HI} - \sum_{\chi_i = HI} \text{done}_i(t), \tag{13}$$

with $\Delta D_i = D_i - x_i \cdot D_i$ and $\text{done}_i(t)$ is given by:

$$\text{done}_i(t) = \begin{cases} \max\left(0, C_i^{LO} - \text{mod}\left(\frac{t}{T_i}\right) + \Delta D_i\right), & \text{if } D_i > \text{mod}\left(\frac{t}{T_i}\right) \geq \Delta D_i, \\ 0, \quad \text{otherwise.} \end{cases}$$

Note that $\text{gdbf}_{HI}(t)$ bounds the execution demand in HI mode taking transitions into account. In our case, as discussed above, we derive different bounds on the execution demand at transitions and in stable HI mode, i.e., $\text{dbf}_{SW}(t)$ and $\text{dbf}_{HI}(t)$ respectively.

Now, since $\text{done}_i(t) \leq C_i^{LO}$ holds for all valid values of t and x_i, $\text{dbf}_{SW}(t) \leq \text{gdbf}_{HI}(t)$ also holds for all t. In other words, if the transition to HI mode is safe by $\text{gdbf}_{HI}(t)$, it will also be safe by the proposed $\text{dbf}_{SW}(t)$. However, this does not hold the other way around, i.e., $\text{dbf}_{SW}(t)$ results in a tighter bound on the execution demand at transitions from LO to HI mode than $\text{gdbf}_{HI}(t)$.

On the other hand, if transitions are safe, it is guaranteed that no deadlines are missed after switching to HI mode at a t' and until the processor first idles at a t''. From t'' onwards, it is easy to see that our proposed $\text{dbf}_{HI}(t)$ is sufficient and necessary. That is, if $\text{dbf}_{HI}(t) \leq t$ does not hold for some t with $0 \leq t \leq \hat{t}_{HI}$, then the system is not feasible, i.e., it will be neither be feasible by $\text{gdbf}_{HI}(t)$.

5.2 The ECDF algorithm

Similar to the case of the GREEDY algorithm, note that $\mathrm{dbf}_{LO}(t)$ in (5) is identical to that used in the ECDF algorithm in LO mode. We denote this latter by $\mathrm{edbf}_{LO}(t)$ in this paper. That is, $\mathrm{dbf}_{LO}(t) = \mathrm{edbf}_{LO}(t)$ for all $0 < t \leq \hat{t}_{LO}$.

In HI mode, the ECDF algorithm uses a demand bound function — denoted $\mathrm{edbf}_{HI}(t)$ in this paper — which is given by the following expression [11]:

$$\begin{aligned}\mathrm{edbf}_{HI}(t_1, t_2) = \min &\left(t_1, \sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2) \right) \\ &+ \sum_{\substack{\chi_i = HI \\ \text{and case 2 or 3}}} \mathrm{dbf}_i^{HI}(t_1, t_2) \\ &+ \sum_{\substack{\chi_i = HI \\ \text{and case 2}}} \left(CO(t_2 - t_1) + \Delta C_i \right), \end{aligned} \tag{14}$$

where ΔC_i is defined as $C_i^{HI} - C_i^{LO}$. In addition, $0 \leq t_2 \leq \hat{t}_{HI}$ and $0 \leq t_1 \leq t_2 - \min_{\chi_i = HI} (\Delta D_i)$ hold with again $\Delta D_i = D_i - x_i \cdot D_i$.

Here t_1 represents the point in time at which the system switches to HI mode (i.e., $t_1 = t'$ in our notation) and t_2 is the point in time at which a deadline is potentially missed (i.e., $t_2 = t_{miss} \leq t''$ in this paper). Note that $\mathrm{dbf}_i^{HI}(\cdot)$ is a τ_i's contribution to $\mathrm{dbf}_{HI}(\cdot)$ shown in (8). HI tasks in (14) are classified into three cases: case 1 which plays a role in computing $\mathrm{dbf}_{L_j}(\cdot)$, case 2, and case 3. For details on how to compute $\mathrm{dbf}_{L_j}(\cdot)$ for $1 \leq j \leq 3$ and how to compute $CO(\cdot)$ we refer to [11].

The system is schedulable in HI mode, if $\mathrm{edbf}_{HI}(t_1, t_2) \leq t_2$ holds. (Here again no distinction is made between transition and stable HI mode.) Let us now consider that t_1 is less than or equal to $\sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2)$, such that the schedulability condition by ECDF now becomes:

$$\sum_{\substack{\chi_i = HI \\ \text{and case 2 or 3}}} \mathrm{dbf}_i^{HI}(t_1, t_2) + \sum_{\substack{\chi_i = HI \\ \text{and case 2}}} \left(CO(t_2 - t_1) + \Delta C_i \right) \leq t_2 - t_1. \tag{15}$$

Easwaran proved that the left-hand side of the above condition is equal to $\mathrm{gdbf}_{HI}(t_2 - t_1)$ [11]. As a consequence, the proposed $\mathrm{dbf}_{SW}(t)$ results in a tighter bound than (15), since $\mathrm{dbf}_{SW}(t) \leq \mathrm{gdbf}_{HI}(t)$ holds for all t — see again the above Section 5.1.

In the case where t_1 is greater than $\sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2)$, the analytical comparison between $\mathrm{edbf}_{HI}(\cdot)$ and $\mathrm{dbf}_{SW}(\cdot)$ becomes difficult. This is the case where some amount of the execution demand given by $\mathrm{edbf}_{HI}(\cdot)$ starts being executed before t_1, at $t_1 - \sum_{j=1}^{3} \mathrm{dbf}_{L_j}(t_1, t_2)$ to be more precise. Whether a proof of dominance exists (in either way) remains an open problem.

At least, from the above discussion, we can assert that the proposed $\mathrm{dbf}_{SW}(\cdot)$ is tighter for the more stringent case, i.e., when no execution demand by $\mathrm{edbf}_{HI}(\cdot)$ can be executed before t_1. Our experiments in the following section present evidence that this also holds on average, i.e., $\mathrm{dbf}_{SW}(\cdot)$ usually results in tighter bounds, particularly, when the number of HI task increases.

6 Applying Approximation Techniques

In this section, we apply approximation techniques, particularly, Devi's test [10] to derive two variants of the proposed approach trading off accuracy for the sake of lesser complexity/runtime. The first variant computes one deadline scaling factor per HI task, similar to the proposed approach, but with less accuracy to reduce complexity. In contrast to it, the second variant computes only one deadline scaling factor for all HI tasks requiring, in principle, less computation. However, as discussed later in detail, this does not necessarily lead to a lower complexity.

6.1 Revisiting Devi's test

Let us consider a set $\bar{\tau}$ of non-MC tasks that are independent, preemptable and sporadic. Similar to tasks in τ defined in Section 3, each individual τ_i in $\bar{\tau}$ is defined by its minimum inter-release time T_i and its relative deadline D_i being $D_i \leq T_i$. However, in contrast to tasks in τ, tasks in $\bar{\tau}$ are characterized by only one WCET parameter denoted by C_i.

In addition, let us assume that tasks in $\bar{\tau}$ are sorted in the order of non-decreasing deadlines, i.e., $D_i \leq D_j$ holds, if $i < j$ holds where i and j are indices identifying tasks. Devi's test states that $\bar{\tau}$ is feasible on one processor under preemptive EDF scheduling, if the following condition holds for $1 \leq k \leq |\bar{\tau}|$ (where $|\bar{\tau}|$ denotes the number of tasks in $\bar{\tau}$) [10]:

$$\sum_{i=1}^{k} \frac{C_i}{T_i} + \frac{1}{D_k} \left(\sum_{i=1}^{k} \frac{T_i - D_i}{T_i} C_i \right) \leq 1, \tag{16}$$

resulting in a sufficient, but not necessary test with polynomial complexity. As opposed to it, an exact test for $\bar{\tau}$ — based on the concept of demand bound function — leads to a higher, pseudo-polynomial complexity [1].

6.2 Per-task deadline scaling

We can now derive the first approximated variant of our proposed approach computing a scaling factor for each individual HI task in τ. To this end, we use Devi's test to analyze each mode and the transition between them.

Schedulability in LO mode. When applying Devi's test in LO mode, the demand bound function in (5) reduces to the following condition:

$$\sum_{i=1}^{k} \frac{C_i^{LO}}{T_i} + \frac{1}{x_k \cdot D_k} \left(\sum_{\substack{i=1 \\ \chi_i = LO}}^{k} \frac{T_i - D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - x_i \cdot D_i}{T_i} C_i^{LO} \right) \leq 1, \tag{17}$$

where $1 \leq k \leq |\tau|$ and $|\tau|$ denotes the total number of tasks in τ. Tasks in (17) need to be sorted in order of non-decreasing real deadlines D_i for $\chi_i = LO$ or virtual deadlines $x_i \cdot D_i$ for $\chi_i = HI$. Note that the order of tasks might change depending on the values of x_i. As a result, (17) might need to be recomputed for the corresponding tasks, if their relative order changes.

Further, in (17), we have considered that the current τ_k (i.e., for the current value of k) is a HI task. As a result, a deadline scaling factor x_k needs to be computed. If τ_k is a LO task, the second term of (17) is divided by D_k instead of $x_k \cdot D_k$, since LO tasks are scheduled within their real deadlines. In this case, no x_k needs to be computed.

Schedulability in stable HI mode. Now, when applying Devi's test in HI mode, the demand bound function in (8) reduces to the following condition:

$$\sum_{\substack{i=1 \\ \chi_i=HI}}^{k} \frac{C_i^{HI}}{T_i} + \frac{1}{D_k} \left(\sum_{\substack{i=1 \\ \chi_i=HI}}^{k} \frac{T_i - D_i}{T_i} C_i^{HI} \right) \leq 1, \tag{18}$$

where again $1 \leq k \leq |\tau|$ holds. Note that only HI tasks are considered in (18), which need to be sorted in order of non-decreasing D_i.

Schedulability in the transition from LO to HI mode. The demand bound function in (10) reduces to the following condition after applying Devi's test:

$$\sum_{\substack{i=1 \\ \chi_i=HI}}^{k} \frac{\Delta C_i}{T_i} + \frac{1}{(1-x_k)D_k} \left(\sum_{\substack{i=1 \\ \chi_i=HI}}^{k} \frac{T_i - \Delta D_i}{T_i} \Delta C_i \right) \leq 1, \tag{19}$$

with $1 \leq k \leq |\tau|$ as before, $\Delta C_i = C_i^{HI} - C_i^{LO}$, and $\Delta D_i = (1 - x_i)D_i$. Similar to stable HI mode, only HI tasks are considered in (19). However, this time, tasks are sorted in the order of non-decreasing ΔD_i instead. Similar to the LO mode, the order of tasks might change depending on the values of x_i. In this case, (19) needs to be recomputed for all tasks whose relative order changes for a newly computed x_k.

Finding deadline scaling factors. As already discussed, (17) needs to hold for all k in $1 \leq k \leq |\tau|$, which is tested in an iterative manner. Clearly, x_k only needs to be computed for $\chi_k = HI$. To this end, let us first reshape (17) to the following:

$$\sum_{i=1}^{k} \frac{C_i^{LO}}{T_i} + \frac{1}{x_k \cdot D_k} \left(\sum_{\substack{i=1 \\ \chi_i=LO}}^{k} \frac{T_i - D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1 \\ \chi_i=HI}}^{k-1} \frac{T_i - x_i \cdot D_i}{T_i} C_i^{LO} + \frac{T_k - x_k \cdot D_k}{T_k} C_k^{LO} \right) \leq 1,$$

which we can then solve for x_k leading to:

$$\frac{\sum_{\substack{i=1\\ \chi_i=LO}}^{k} \frac{T_i-D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i-x_i\cdot D_i}{T_i} C_i^{LO} + C_k^{LO}}{D_k\left(1-\sum_{i=1}^{k}\frac{C_i^{LO}}{T_i}\right) + D_k\frac{C_k^{LO}}{T_k}} \leq x_k.$$

Note that we can change the upper limit of the first summation in the numerator to $k-1$, since $\chi_k = HI$ holds. Further, reshaping the denominator, we finally obtain:

$$\frac{\sum_{\substack{i=1\\ \chi_i=LO}}^{k-1} \frac{T_i-D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i-x_i\cdot D_i}{T_i} C_i^{LO} + C_k^{LO}}{D_k\left(1-\sum_{i=1}^{k-1}\frac{C_i^{LO}}{T_i}\right)} \leq x_k. \tag{20}$$

As already mentioned, we might need to recompute (17) for all tasks whose relative order changes depending on the value of x_k. To avoid this complication, we derive the following lower bound for x_k:

$$\frac{D_{k-1}^{LO}}{D_k} \leq x_k, \tag{21}$$

where $D_{k-1}^{LO} = D_{k-1}$ or $D_{k-1}^{LO} = x_{k-1}\cdot D_{k-1}$ depending on whether $\chi_{k-1} = LO$ or $\chi_{k-1} = HI$ respectively. In words, (21) guarantees that the selected x_k does not change the order of tasks in LO mode to avoid having to recompute (17), clearly, at the cost of a lesser accuracy.

Now, we can obtain an upper bound on x_k reshaping (19) to:

$$\sum_{\substack{i=1\\ \chi_i=HI}}^{k} \frac{\Delta C_i}{T_i} + \frac{1}{(1-x_k)D_k}\left(\sum_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i-\Delta D_i}{T_i}\Delta C_i + \frac{T_k-(1-x_k)D_k}{T_k}\Delta C_k\right) \leq 1.$$

Solving for $1-x_k$ and reshaping as before, we obtain:

$$1 - \frac{\sum_{\substack{i=1\\ \chi_i=HI}}^{k-1} \frac{T_i-\Delta D_i}{T_i}\Delta C_i + \Delta C_k}{D_k\left(1-\sum_{i=1}^{k-1}\frac{\Delta C_i}{T_i}\right)} \geq x_k. \tag{22}$$

Similar to before, to avoid having to recompute (19), we need to prevent that the order of tasks changes, for which we derive the following upper bound on x_k:

$$1 - \frac{(1-x_{k-1})D_{k-1}}{D_k} \geq x_k. \tag{23}$$

Finally, a system is feasible under mixed-criticality EDF, if (17) holds for all k and (18) holds for all k with $\chi_k = HI$. In addition, we need to find a valid x_k for every τ_k with $\chi_k = HI$. That is, x_k must be greater than or equal to the maximum between (20) and (21). Simultaneously, x_k must be less than or equal to the minimum between (22) and (23).

6.3 Uniform deadline scaling

We can also apply Devi's test to derive a second approximated variant of our proposed approach, computing only one deadline scaling factor for all HI tasks in τ. In principle, this requires less computation than our first approximated variant. However, in contrast to what is expected, this second variant does not result in a lower complexity, since it cannot prevent the order of tasks from changing as discussed below.

Schedulability in LO mode. We again apply Devi's test to the demand bound function in (5), but considering this time a uniform deadline scaling factor x:

$$\sum_{i=1}^{k} \frac{C_i^{LO}}{T_i} + \frac{1}{x \cdot D_k} \left(\sum_{\substack{i=1 \\ \chi_i = LO}}^{k} \frac{T_i - D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - x \cdot D_i}{T_i} C_i^{LO} \right) \leq 1, \tag{24}$$

where $1 \leq k \leq |\tau|$ and $|\tau|$ denote the total number of tasks in τ. Just as before, tasks in (24) need to be sorted in order of non-decreasing real deadlines D_i for $\chi_i = LO$ or virtual deadlines $x \cdot D_i$ for $\chi_i = HI$. Although the relative order of HI tasks (among themselves) never changes, the order of LO tasks with respect to HI tasks might still change for some x. If that happens, (24) needs to be recomputed for the corresponding tasks.

In (24), note that the current τ_k is considered to be a HI task and, hence, x needs to be computed. If τ_k is a LO task, the second term of (24) is divided by D_k instead of $x \cdot D_k$, since LO tasks are scheduled within their real deadline. In this case, no x is computed; however, (24) still needs to hold for the previously selected x.

Schedulability in stable HI mode. When considering a uniform deadline scaling factor x, we still obtain (18) as a result of applying Devi's test to (9). Hence, this case requires no further discussion.

Schedulability in the transition from LO to HI mode. The demand bound function in (11) reduces to the following condition after applying Devi's test for a uniform deadline scaling factor x:

$$\sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{\Delta C_i}{T_i} + \frac{1}{(1-x)D_k} \left(\sum_{\substack{i=1 \\ \chi_i = HI}}^{k} \frac{T_i - (1-x)D_i}{T_i} \Delta C_i \right) \leq 1, \tag{25}$$

with $1 \leq k \leq |\tau|$, and ΔC_i is defined as before. Only HI task are considered in (25) and tasks are sorted in the order of non-decreasing $(1-x)D_i$. This time, however, the order of tasks cannot change for different values of x, since x affects all deadlines the same.

Finding a uniform deadline scaling factor. Proceeding as before, we can obtain a lower bound on x from (24):

$$\frac{\sum_{\substack{i=1 \\ \chi_i=LO}}^{k} \frac{T_i-D_i}{T_i} C_i^{LO} + \sum_{\substack{i=1 \\ \chi_i=HI}}^{k} C_i^{LO}}{D_k \left(1 - \sum_{i=1}^{k} \frac{C_i^{LO}}{T_i}\right) + \sum_{\substack{i=1 \\ \chi_i=HI}}^{k} D_i \frac{C_i^{LO}}{T_i}} \leq x. \tag{26}$$

Note that the order between some HI and LO tasks might change for a given x, requiring us to recompute (24). As mentioned above, in contrast to the per-task deadline scaling, it becomes difficult to derive an additional lower bound on x that prevents this from happening. We can, of course, select an x for the current τ_k such that $D_{k-1}^{LO} \leq x \cdot D_k$ holds, where D_{k-1}^{LO} represents τ_{k-1}'s deadline in LO mode (independent of whether this is a LO or HI task). However, this is not sufficient, since a new value of x also affects all previously tested HI tasks, whose deadlines might become smaller than some deadline of a LO task.

An upper bound on x can be obtained from (25), which leads to:

$$x \leq 1 - \frac{\sum_{\substack{i=1 \\ \chi_i=HI}}^{k} \Delta C_i}{D_k \left(1 - \sum_{\substack{i=1 \\ \chi_i=HI}}^{k} \frac{\Delta C_i}{T_i}\right) + \sum_{\substack{i=1 \\ \chi_i=HI}}^{k} D_i \frac{\Delta C_i}{T_i}}. \tag{27}$$

According to this second variant of our proposed approach, the system is feasible under mixed-criticality EDF, if (24) holds for all k and (18) holds for all k with $\chi_k = HI$. In addition, none of the values of x obtained with (27) should be less than any value obtained by (26) for $1 \leq k \leq |\tau|$ with $\chi_k = HI$.

6.4 Complexity

Similar to GREEDY [12] and ECDF [11], the proposed approach of Section 4 is based on computing demand bound functions and, hence, has a pseudo-polynomial complexity $\mathcal{O}(K_n \cdot n)$ for task sets with a total utilization that is strictly less than 1 [1]. Note that n represents the number of tasks in the given task set and K_n is a *factor* that depends on task parameters and, therefore, on n. In contrast to this, EDF-VD [4] has a linear complexity $\mathcal{O}(n)$.

The approximated variant in Section 6.2, based on computing per-task deadline scaling factors, has a polynomial complexity $\mathcal{O}(n \cdot \log n)$, when the bounds in (21) and (23) are considered. As explained above, these prevent that the order of tasks changes for a newly computed deadline scaling factor. If (21) and (23) are not considered, the order of tasks might change requiring us to retest some of the (previously tested) tasks and, hence, leading to a quadratic complexity $\mathcal{O}(n^2)$ instead.

On the other hand, the complexity of our second approximated variant of Section 6.3 is quadratic $\mathcal{O}(n^2)$, since, in the general case, i.e., with at least one LO

task in the system, it cannot guarantee that the order of tasks does not change. As a result, retesting cannot be avoided.

7 Experimental Evaluation

In this section, we present evaluation results based on synthetic data. The intention is to show how the different algorithms behave with respect to each other and not to provide any absolute performance metrics.

In particular, we compare the proposed approach of Section 4 with EDF-VD [4], with the GREEDY algorithm by Ekberg and Yi [12], and with ECDF by Easwaran [11]. Note that we had to modify EDF-VD to consider the deadlines D_i of tasks instead of their inter-arrival times or periods T_i, i.e., to consider the tasks' densities instead of their utilizations, to account for the case of constrained deadlines $D_i \leq T_i$ and be compared with the other algorithms in a meaningful manner.

In addition, we include our first approximated variant from Section 6.2 in this comparison. As discussed in Section 6.4, this is the one with the lowest complexity and helps illustrating how much performance can be attained with the proposed technique at the least possible cost.

Schedulability curves. Fig. 3 shows schedulability curves, i.e., the percentage of feasible task sets by the above algorithms, versus LO utilization. For every increase in LO utilization, a total number of 1000 different sets of 20 tasks each were randomly generated — 10,000 task sets in total. We made use of UUniFast [7] to generate individual task utilizations.

Further, we used the log-uniform distribution proposed by Emberson *et al.* [14] to create the task periods T_i in the range of $1ms$ to $1000ms$. The log-uniform distribution guarantees that task periods are equally spread into the time bands $1-10ms$, $10-100ms$, etc.

With T_i and the task utilization, we obtained the values of C_i^{LO}. We assumed that 30% of the tasks are HI tasks, i.e., 6 out of 20 tasks. Further, for each HI task, we randomly selected an increase in HI execution demand of at most 50% of $\frac{C_i^{LO}}{T_i}$. With this, we then obtained the values of C_i^{HI}. Deadlines D_i are constrained and chosen from a uniform distribution in the range $[C_i^{HI}, T_i]$ for HI tasks and in $[C_i^{LO}, T_i]$ for LO tasks.

As depicted in Fig. 3, expectedly, the percentage of schedulable task sets decreases with an increasing LO utilization. On the other hand, whereas all algorithms perform similarly for a LO utilization below 50%, they exhibit different behaviors for higher LO utilizations. In particular, the proposed approach outperforms ECDF by around 10% to 20% more accepted task sets in the range of 60% to 100% LO utilization. Interestingly, in spite of having a lesser complexity, our approximated variant shows a similar performance to the GREEDY algorithm.

Weighted schedulability. Next, we make use of the concept of *weighted schedulability* [6][9] to analyze the performance by the above algorithms. That is, for a schedulability test A whose accuracy on testing a task set τ is a function of parameter p, its weighted schedulability $W_A(p)$ is given by:

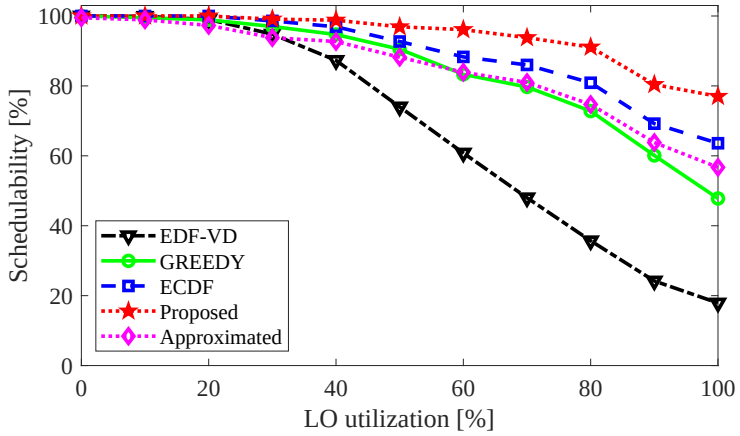

Figure 3: Schedulability vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

$$\mathrm{W}_A(p) = \frac{\sum\limits_{\forall\tau} U(\tau) \cdot \mathrm{S}_A(\tau, p)}{\sum\limits_{\forall\tau} U(\tau)}, \tag{28}$$

where $\mathrm{U}(\tau)$ is the utilization of a given τ and $S_A(\tau, p)$ is A's binary result (1 if schedulable and 0 if not) for a task set τ with parameter value p. In other words, individual schedulability results by A are weighted according to the utilization of the task sets tested, putting more emphasis on higher-utilization ones.

We created weighted schedulability curves varying following parameters: (i) total number of tasks, (ii) percentage of HI tasks, (iii) increase of HI execution demand and (iv) range of task periods. Every time we varied one of these parameters, we generated 1000 different task sets for each LO utilization value between 0 and 100% at steps of 10%, i.e., a total of 10,000 task sets per marker on the shown curves. To this end, we proceeded as described previously to obtain task parameters.

Fig. 4 shows weighted schedulability curves for a varying total number of tasks where we selected the number of HI tasks to be equal to 30% of the total (i.e., it also varies proportionally) and the increase of HI execution demand to be 50% of the LO execution demand for each HI task. The proposed approach outperforms all other algorithms by around 10% to 20% depending on the total number of tasks. In general, the more the tasks, the better the proposed algorithm performs with respect to the others. It is interesting to note that ECDF performs better than GREEDY up to 30 tasks per set, after which GREEDY performs better. Further, ECDF seems to stagnate at around 80% weighted schedulability, in contrast to GREEDY and the proposed approach.

Our approximated variant has a good performance up to around 40 tasks, after which it decays pronouncedly, even becoming worse than EDF-VD from 80 tasks onwards. The reason is that this algorithm is based on Devi's test, which requires tasks to be sorted by deadline. Since the order of tasks may change with every new deadline scaling factor, some tasks may need to be retested as explained

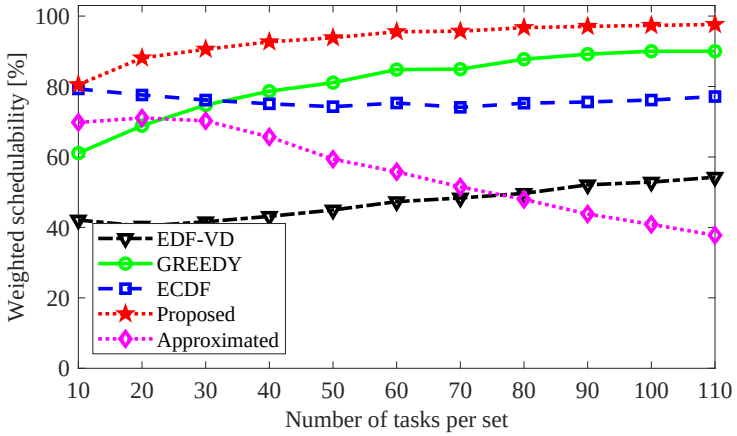

Figure 4: Weighted schedulability vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand

in Section 6.2. Clearly, the more tasks there are, the more likely it is that their order changes (when scaling one deadline). To avoid this and reduce complexity, we introduced conditions (21) and (23), which truncate the valid range of deadline scaling factors. This has the downside, however, that the number of wrongly rejected task sets increases disproportionally as the total number of tasks grows.

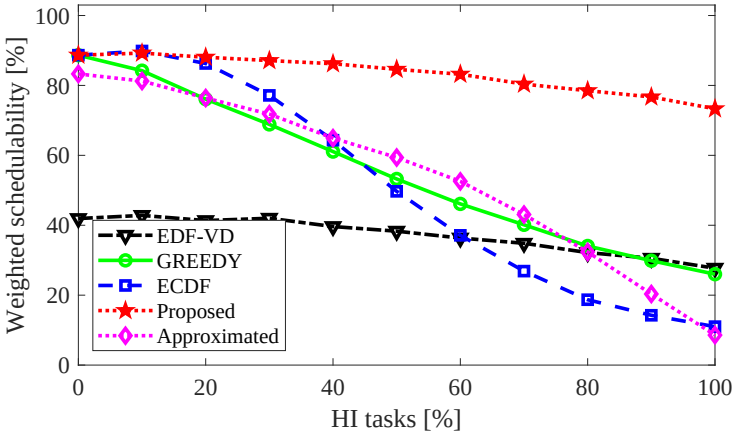

Figure 5: Weighted schedulability vs. percentage of HI tasks for $|\tau| = 20$ and 50% increase of HI execution demand

Weighted schedulability curves for a varying percentage of HI tasks are shown in Fig. 5. This time, we selected the total number of tasks to be 20, whereas the increase of HI execution demand continues to be 50% as in the previous case. We can see that the performance of all approaches decreases with an increasing number of HI tasks. Up to around 20% HI tasks (i.e., 4 out of 20), the proposed approach

and ECDF behave similarly. However, ECDF's performance then decreases rapidly, becoming worse than GREEDY and even EDF-VD at 50% and 60% HI tasks respectively.

At around 80% HI tasks (16 out of 20 tasks), the proposed approach still allows for around 80% schedulable task sets independent of the LO utilization, whereas all other algorithms are at or below 40% schedulable task sets. This evidences the effectiveness of the proposed approach over GREEDY and ECDF for general cases. In particular, GREEDY and ECDF are based on estimating the worst-case contributions by carry-over jobs at the moment of switching from LO to HI mode. This inevitably becomes pessimistic as the number of carry-over jobs grows, which directly depends on the number of HI tasks.

In the case of our approximated variant, again, conditions (21) and (23) start dominating in Fig. 5. Even though the total number of tasks remains constant, these two conditions are evaluated for each HI task. As a result, if the number of HI tasks increases, they start playing a bigger role and, hence, accentuating the decrease in performance.

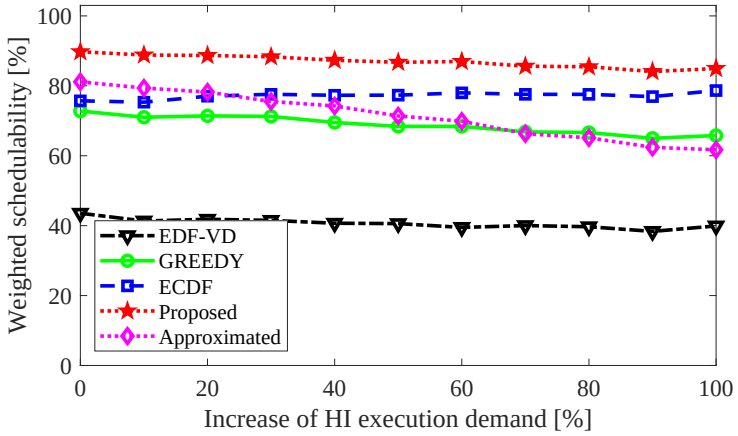

Figure 6: Weighted schedulability vs. increase of HI execution demand for $|\tau| = 20$ and 30% HI tasks

In Fig. 6, we further present weighted schedulability curves for a varying increase of HI execution demand. We again selected the total number of tasks to be 20 and the number of HI tasks is set to 30%. In this case, the behavior of algorithms slightly worsens for a growing HI execution demand with exception of ECDF, whose behavior slightly improves. In spite of this, the proposed algorithm outperforms all others by around 10% more schedulable task sets in the range of 10% to 80% increase in HI execution demand. Interestingly, our approximated variant also shows a good performance in this range, which is even better than that of the GREEDY algorithm up to 60% increase in HI execution demand.

Last, Fig. 7 shows weighted schedulability curves for a varying range of task periods with the total number of tasks being again 20, out of which 30% are HI tasks with a 50% increase in HI execution demand. In this case, the performance of all algorithms rapidly goes down for an increasing range of task periods. The

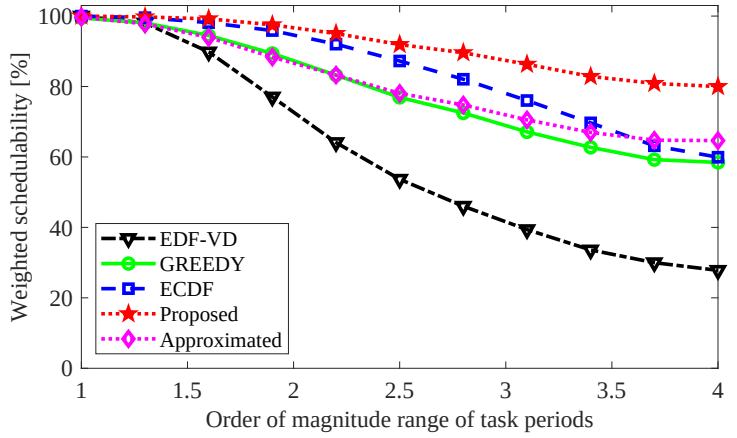

Figure 7: Weighted schedulability vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

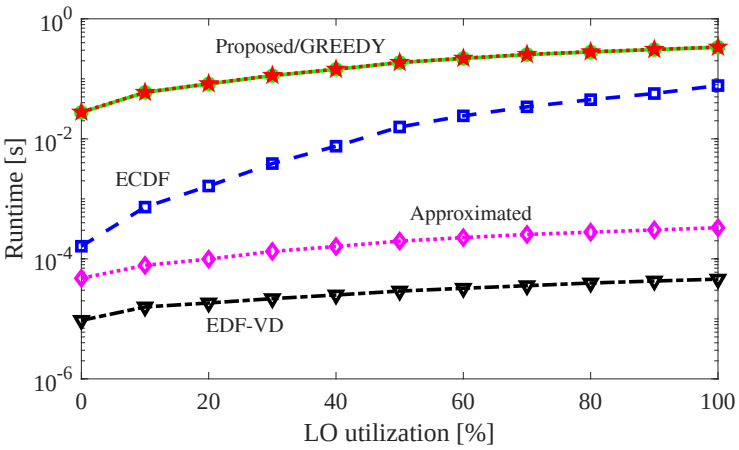

Figure 8: Runtime vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

proposed approach outperforms ECDF by 10% to 20% more schedulable task sets when the minimum and the maximum task period are 3 to 4 orders of magnitude apart. Note that our approximated variant outperforms GREEDY for period ranges of 2.5 orders of magnitude upwards and has a comparable performance to that of ECDF between 3.5 to 4 orders of magnitude.

Runtime comparison. Fig. 8, Fig. 9 to Fig. 10 show a comparison of runtime versus LO utilization, total number of tasks and range of task periods respectively. However, note that we have implemented the different algorithms in Matlab and, hence, they can be further optimized, potentially changing their behavior with respect to runtime.

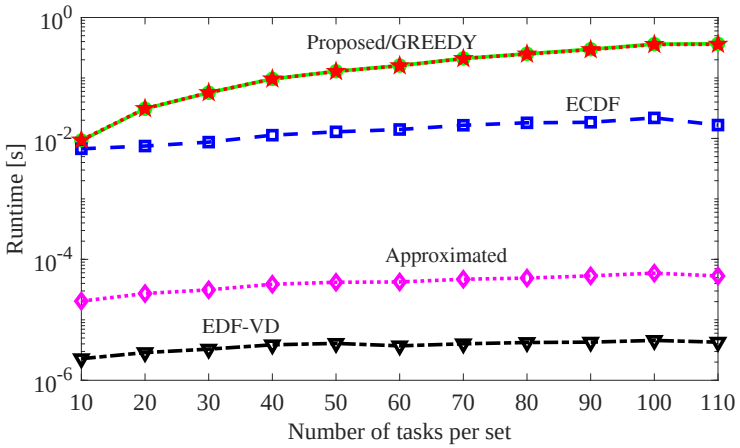

Figure 9: Runtime vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand

Now, ECDF is around one to two orders of magnitude faster than GREEDY and the proposed approach depending on LO utilization as shown in Fig. 8. Our approximated variant is around one order of magnitude slower than EDF-VD and around one to two orders of magnitude faster than ECDF. This behavior remains almost unchanged as the number of tasks increases towards 100 tasks per set — see Fig. 9. Here we maintained the percentage of HI tasks and the increase of HI execution demand equal to 30% and 50% respectively. Only for 10 tasks per set, GREEDY and the proposed algorithm are as fast as ECDF.

Fig. 10, shows runtime curves for an increasing range of task periods. We again kept the total number of tasks at 20, the percentage of HI tasks at 30% and the increase of HI execution demand at 50%. As expected, our approximated variant and EDF-VD have a constant runtime for an increasing range of periods, since both have polynomial complexity. All other algorithms experience an increasing runtime for greater period ranges. ECDF continues to be around one order of magnitude faster than GREEDY and the proposed algorithm. However, this difference reduces as the range of task periods grows. At 4 orders of magnitude between the minimum and the maximum period, all these algorithms show the same runtime.

It should be noted that Fig. 9 and Fig. 10 are independent of LO utilization. For each marker on these curves, we generated 1000 different task sets for each LO utilization value between 0 and 100% at steps of 10%, i.e., a total of 10,000 task sets per marker.

8 Concluding Remarks

In this paper, we studied the problem of mixed-criticality scheduling under EDF, where a set of low-criticality (LO) and high-criticality (HI) tasks share the processor. Similar to the literature, we characterize execution demand in a mixed-criticality task set by deriving demand bound functions in the different operation

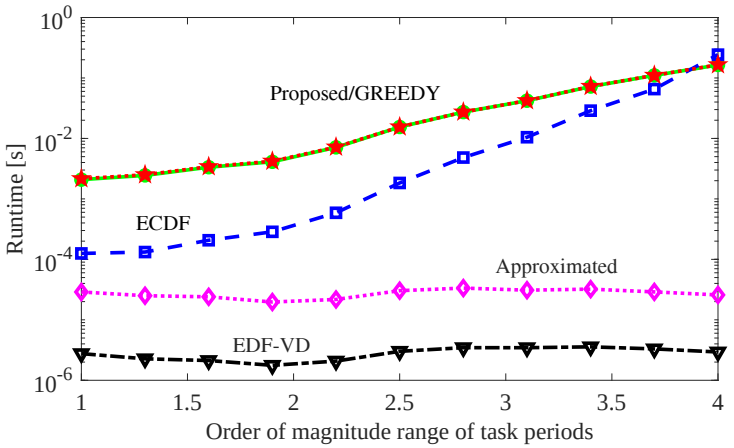

Figure 10: Runtime vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand

modes, i.e., HI and LO mode. Particularly, we handle transitions from LO to HI mode separately from the stable HI mode, which allows us to work around carry-over jobs and, therefore, to reduce pessimism in estimating execution demand under mixed-criticality EDF.

It is interesting to notice that the proposed approach reduces the problem of testing schedulability under mixed-criticality EDF to testing schedulability of three *almost* unrelated task sets: the one in LO mode, the one in HI mode and the equivalent task set for transitions between LO and HI mode. This allows applying well-known approximation techniques that trade off accuracy for the sake of a lesser complexity/runtime as illustrated for the case of Devi's test.

Further, we conducted a large set of experiments on synthetic data showing the effectiveness of the proposed approach and of its approximated variant in terms of (weighted) schedulability and runtime respectively. This is particularly notable as the number of HI tasks increases.

Which algorithm should be used depends very much on the context. If we are testing offline, we believe, there is no harm in using all three algorithms (GREEDY, ECDF and the proposed test). If tests are to be performed online, it is probably better to used the approximated variant of the proposed test — with a $\mathcal{O}(n \cdot \log n)$ rather than pseudo-polynomial complexity.

Finally, in the appendix, we show how to extend the proposed approach to more than two levels of criticality considering ordered (i.e., where criticality levels cannot be skipped) and unordered modes switches.

Acknowledgements Mitra Mahdiani was funded by the German Academic Exchange Service (DAAD). The authors would like to thank reviewers for the valuable comments and suggestions.

References

1. Baruah S, Mok A, Rosier L (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proc. of Real-Time Systems Symposium (RTSS)
2. Baruah S, Bonifaci V, D'Angelo G, Marchetti-Spaccamela A, Van Der Ster S, Stougie L (2011) Mixed-criticality scheduling of sporadic task systems. In: Proc. of European Symposium on Algorithms (ESA)
3. Baruah S, Burns A, Davis RI (2011) Response-time analysis for mixed criticality systems. In: Proc. of Real-Time Systems Symposium (RTSS)
4. Baruah S, Bonifaci V, D'Angelo G, Li H, Marchetti-Spaccamela A, van der Ster S, Stougie L (2012) The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In: Proc. of Euromicro Conference on Real-Time Systems (ECRTS)
5. Baruah S, Bonifaci V, D'angelo G, Li H, Marchetti-Spaccamela A, Van Der Ster S, Stougie L (2015) Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. Journal of the ACM (JACM) 62(2)
6. Bastoni a, Brandenburg B, Anderson J (2010) Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In: Proc. of Workshop on Operating Systems Platforms for Embedded Real-Time Applications
7. Bini E, Buttazzo G (2005) Measuring the performance of schedulability tests. Real-Time Systems (RTS) 30(1-2)
8. Burns A, Davis RI (2018) A survey of research into mixed criticality systems. ACM Computing Surveys (CSUR) 50(6)
9. Davis RI (2016) On the evaluation of schedulability tests for real-time scheduling algorithms. In: Proc. of Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)
10. Devi UC (2003) An improved schedulability test for uniprocessor periodic task systems. In: Proc. of Euromicro Conference on Real-Time Systems (ECRTS)
11. Easwaran A (2013) Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In: Proc. of Real-Time Systems Symposium (RTSS)
12. Ekberg P, Yi W (2012) Bounding and shaping the demand of mixed-criticality sporadic tasks. In: Proc. of Euromicro Conference on Real-Time Systems (ECRTS)
13. Ekberg P, Yi W (2014) Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. Real-Time Systems (RTS) 50(1)
14. Emberson P, Stafford R, Davis RI (2010) Techniques for the synthesis of multiprocessor tasksets. In: Proc. of Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)
15. Kuo T, Mok AK (1991) Load adjustment in adaptive real-time systems. In: Proc. of Real-Time Systems Symposium (RTSS)
16. Mahdiani M, Masrur A (2018) On bounding execution demand under mixed-criticality EDF. In: Proc. of Conference on Real-Time Networks and Systems
17. Masrur A, Müller D, Werner M (2015) Bi-level deadline scaling for admission control in mixed-criticality systems. In: Proc. of Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)
18. Müller D (2016) SS01 — explicit per-task deadline scaling for uniprocessor scheduling of job-level static mixed-criticality systems. In: Proc. of Symposium

on Industrial Embedded Systems (SIES)
19. Su H, Zhu D (2013) An elastic mixed-criticality task model and its scheduling algorithm. In: Proc. of Design, Automation and Test in Europe (DATE)
20. Vestal S (2007) Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proc. of Real-Time Systems Symposium (RTSS)
21. Wang Y, Saksena M (1999) Scheduling fixed-priority tasks with preemption threshold. In: Proc. of Real-Time Computing Systems and Applications (RTCSA)
22. Zhao Q, Gu Z, Zeng H (2013) PT-AMC: Integrating Preemption Thresholds into Mixed-Criticality Scheduling. In: Proc. of Design, Automation and Test in Europe (DATE)

A Multiple Levels of Criticality

In practice, usually more than two levels of criticality are common, e.g., four different *automotive safety integrity levels* (ASIL) are defined in the ISO 26262 standards. To account for this, we illustrate how to apply the proposed approach to add a third level of criticality between LO and HI: the mid-criticality (MI) level. Note that additional levels of criticality can also be added in an straightforward manner based on the presented analysis.

Just as before, the system implements a mode of operation per criticality level resulting now in three modes: LO, MI, and HI mode. Further, tasks are classified according to their criticality χ_i into LO, MI and HI tasks. LO tasks only run in LO mode and are discarded in MI and HI mode. MI tasks run in LO and MI mode, but are discarded in HI mode, whereas HI tasks run in all modes.

All tasks are defined by their minimum inter-release times T_i and their relative deadlines D_i. LO task are characterized by their WCET parameter C_i^{LO}, whereas MI tasks have a C_i^{LO} and a C_i^{MI} parameter, which denote their WCET in LO and MI mode respectively. HI tasks now have three WCET parameters, i.e., C_i^{LO}, C_i^{MI}, and C_i^{HI} where $C_i^{LO} < C_i^{MI} < C_i^{HI} \leq D_i \leq T_i$ holds.

A.1 Ordered mode switches

We first consider *ordered* mode switches. That is, the system switches from LO to MI, if either a MI or a HI task runs for more than its C_i^{LO} in LO mode, and from MI to HI mode, if a HI task runs for more than its C_i^{MI} in MI mode. Note that there is no direct transition from LO and HI mode, i.e., the system first switches to MI and then to HI mode. The necessary extensions for *unordered* mode switches, i.e., when the system switches from LO directly to HI mode, are discussed below. Next, for ease of exposition, we first analyze schedulability in MI mode, then in HI mode, and last in LO mode.

Schedulability in stable MI mode. In MI mode, LO tasks are discarded and MI tasks are scheduled together with HI tasks. MI tasks are scheduled within their real deadline, however, HI tasks are assigned virtual deadlines $y_i \cdot D_i$. Here, $y_i \in (0, 1]$ denotes the per-task scaling factor in MI mode. Both MI and HI run for their corresponding C_i^{MI} leading to the following demand bound function — which resembles (5):

$$\begin{aligned} \mathrm{dbf}_{MI}(t) = & \sum_{\chi_i = MI} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{MI} \\ & + \sum_{\chi_i = HI} \left(\left\lfloor \frac{t - y_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{MI}. \end{aligned} \tag{29}$$

Now, proceeding as before, we obtain an upper bound on $\hat{t}_{MI}$, i.e., the point in time until which we need to check feasibility, i.e., that $\text{dbf}_{MI}(t) < t$ holds. With $U_{MI}^{MI} = \sum_{\chi_i=MI} \frac{C_i^{MI}}{T_i}$, $U_{HI}^{MI} = \sum_{\chi_i=HI} \frac{C_i^{MI}}{T_i}$ and letting y_i tend to 0, we obtain:

$$\hat{t}_{MI} \leq \frac{\sum_{\chi_i=MI} (T_i - D_i)\frac{C_i^{MI}}{T_i}}{1 - U_{MI}^{MI} - U_{HI}^{MI}} + \frac{\sum_{\chi_i=HI} C_i^{MI}}{1 - U_{MI}^{MI} - U_{HI}^{MI}}. \tag{30}$$

Schedulability in stable HI mode. Only HI tasks are allowed to run and they are scheduled within their real deadlines in HI mode. Hence, $\text{dbf}_{HI}(t)$ is given by (8) and the upper bound on $\hat{t}_{HI}$ is given by (9), which requires no further discussion.

Schedulability in the transition from MI to HI mode. We can apply Theorem 1 to obtain the demand bound function at transitions from MI to HI mode. Note that this also resembles (10):

$$\text{dbf}_{SW1}(t) = \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - \Delta D_i^{SW1}}{T_i} \right\rfloor + 1 \right) \Delta C_i^{SW1}, \tag{31}$$

with $\Delta D_i^{SW1} = D_i - y_i \cdot D_i$, and $\Delta C_i^{SW1} = C_i^{HI} - C_i^{MI}$. Similarly, we proceed to obtain an upper bound on $\hat{t}_{SW1}$, i.e., the point in time until which $\text{dbf}_{SW1}(t) < t$ needs to be checked. The resulting expression resembles (11) with U_{HI}^{SW1} given by $\sum_{\chi_i=HI} \frac{\Delta C_i^{SW1}}{T_i}$:

$$\hat{t}_{SW1} \leq \frac{\sum_{\chi_i=HI} \Delta C_i^{SW1}}{1 - U_{HI}^{SW1}}. \tag{32}$$

Schedulability in LO mode. In LO mode, LO tasks need to be scheduled together with MI and HI tasks. MI tasks are assigned virtual deadlines $x_i \cdot D_i$, while HI tasks are assigned virtual deadlines $x_i \cdot y_i \cdot D_i$. That is, their virtual deadlines in MI mode (i.e., $y_i \cdot D_i$) are again scaled by $x_i \in (0, 1]$. As a consequence, the resulting demand bound function $\text{dbf}_{LO}(t)$ in LO mode is given by:

$$\begin{aligned} \text{dbf}_{LO}(t) = & \sum_{\chi_i=LO} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} \\ & + \sum_{\chi_i=MI} \left(\left\lfloor \frac{t - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}, \\ & + \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - x_i \cdot y_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}. \end{aligned} \tag{33}$$

We can proceed as before to obtain an upper bound on $\hat{t}_{LO}$. That is, the point in time until which we need to check that $\text{dbf}_{LO}(t) < t$ holds. In addition to U_{LO}^{LO} and U_{HI}^{LO}, considering $U_{MI}^{LO} = \sum_{\chi_i=MI} \frac{C_i^{LO}}{T_i}$ and letting x_i tend to 0, we obtain:

$$\hat{t}_{LO} \leq \frac{\sum_{\chi_i=LO} (T_i - D_i)\frac{C_i^{LO}}{T_i}}{1 - U_{LO}^{LO} - U_{MI}^{LO} - U_{HI}^{LO}} + \frac{\sum_{\chi_i=MI \vee HI} C_i^{LO}}{1 - U_{LO}^{LO} - U_{MI}^{LO} - U_{HI}^{LO}}, \tag{34}$$

where it should be noted that the second summation on the right-hand side applies to both MI and HI tasks in the system.

Schedulability in the transition from LO to MI mode. We can again apply Theorem 1 to obtain the demand bound function for transitions from LO to MI mode:

$$\begin{aligned}\mathrm{dbf}_{SW2}(t) = &\sum_{\chi_i=MI} \left(\left\lfloor \frac{t - \Delta D_i^{SW2}}{T_i} \right\rfloor + 1 \right) \Delta C_i^{SW2} \\ &+ \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - \hat{\Delta} D_i^{SW2}}{T_i} \right\rfloor + 1 \right) \Delta C_i^{SW2}, \end{aligned} \tag{35}$$

with $\Delta D_i^{SW2} = D_i - x_i \cdot D_i$, $\hat{\Delta} D_i^{SW2} = D_i - x_i \cdot y_i \cdot D_i$, and $\Delta C_i^{SW2} = C_i^{MI} - C_i^{LO}$. Similarly, we proceed to obtain an upper bound on $\hat{t}_{SW2}$, i.e., the point in time until which $\mathrm{dbf}_{SW2}(t) < t$ needs to be checked:

$$\hat{t}_{SW2} \leq \frac{\sum_{\chi_i=MI \vee HI} \Delta C_i^{SW2}}{1 - U_{MI}^{SW2} - U_{HI}^{SW2}}, \tag{36}$$

where U_{MI}^{SW2} is given by $\sum_{\chi_i=MI} \frac{\Delta C_i^{SW2}}{T_i}$ and U_{HI}^{SW2} is given by $\sum_{\chi_i=HI} \frac{\Delta C_i^{SW2}}{T_i}$.

Finding deadline scaling factors. In contrast to the case of two levels of criticality, we now have to compute two deadline scaling factors y_i and x_i. We can still use the proposed approach from Section 4, but in an iterative manner. That is, we first use the proposed approach to obtain y_i, i.e., the deadline scaling factor in MI mode. Once we have the values of y_i, we can apply this approach again to find x_i, i.e., the deadline scaling factor in LO mode.

A.2 Unordered mode switches

If we were to allow for *unordered* mode switches, i.e., from LO directly to HI mode in the above setting with three levels of criticality, we need to consider it separately. To this end, we assume that a subset of the HI tasks cause a direct transition to HI mode (instead of MI mode as assumed so far) when running for more than C_i^{LO} in LO mode.[5] As a result, in LO mode, we now have:

$$\begin{aligned}\mathrm{dbf}_{LO}(t) = &\sum_{\chi_i=LO} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO} \\ &+ \sum_{\chi_i=MI} \left(\left\lfloor \frac{t - x_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}, \\ &+ \sum_{\chi_i=HI} \left(\left\lfloor \frac{t - z_i \cdot D_i}{T_i} \right\rfloor + 1 \right) C_i^{LO}, \end{aligned} \tag{37}$$

where z_i is a deadline scaling factor that guarantees schedulability for the direct transition from LO to HI mode. In HI mode, again, $\mathrm{dbf}_{HI}(t)$ given by (8) continues to be valid. As a result, with all x_i obtained as discussed for the case of ordered mode switches, we can compute each z_i in (37) also based on the approach from Section 4.

Finally, to guarantee safety independent of whether the system switches to MI or HI mode, HI task will now have to be scheduled in LO mode using the minimum between $x_i \cdot y_i$ that covers ordered transitions and z_i that accounts for the unordered case. For more than three levels of criticality, note that all possible unordered mode switches will have to be analyzed as shown here to determine suitable deadline scaling factors for the tasks involved.

[5] In principle, any HI task can be allowed to switch either to MI or to HI mode too. However, in this case, we will need to extend our task model such that mode switches can be triggered independent of the tasks' execution budgets.

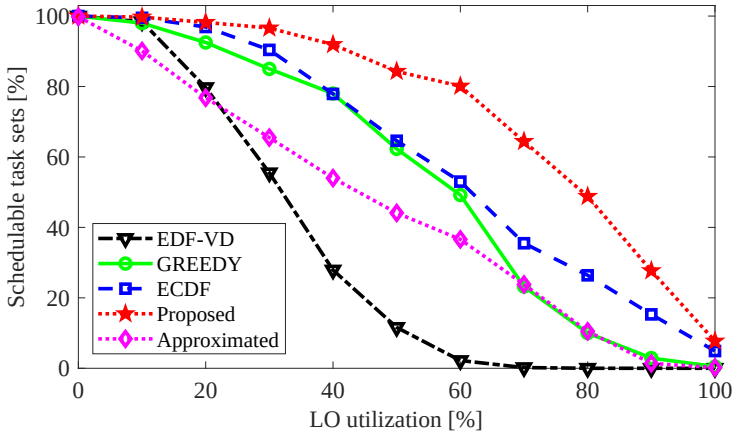

Figure 11: Schedulability vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

B Uniform Distribution for Task Periods

In Section 7, we used the log-uniform distribution proposed by Emberson *et al.*[14] to generate task periods in $[1, T_{max}]$, where T_{max} was set to 1000 in the default case. The log-uniform distribution equally spreads task periods into the time bands $1-10$, $10-100$, etc. and, hence, the resulting task sets have an equal number of tasks in each such bands.

In contrast to this, a uniform distribution tends to concentrate task periods in the middle of $[1, T_{max}]$, resulting in task sets where most tasks have periods of the same order of magnitude around 500 for $T_{max} = 1000$. Task sets generated this way lead to different performance by algorithms as shown below in Fig. 11 for schedulability and in Fig. 12 to Fig. 15 for weighted schedulability. In particular, the algorithms' behavior changes with respect to runtime as shown in Fig. 16 to Fig. 18.

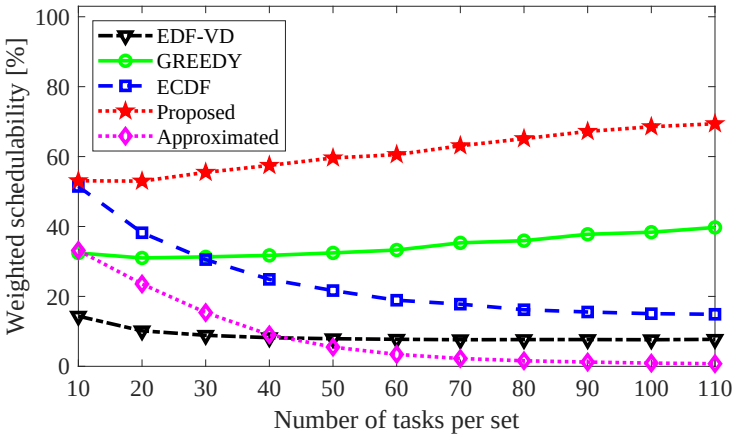

Figure 12: Weighted schedulability vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

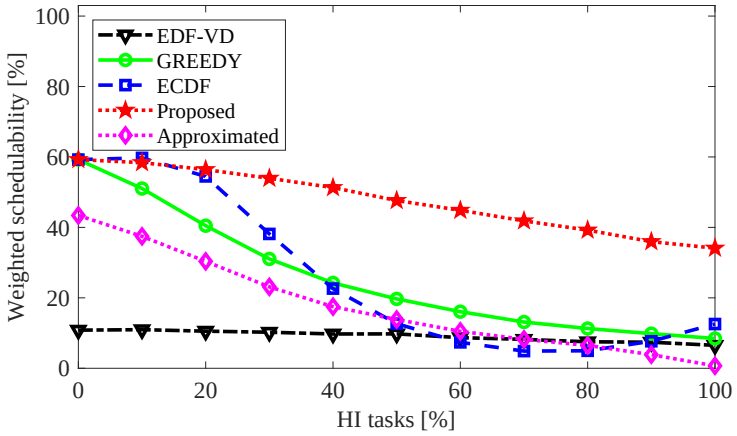

Figure 13: Weighted schedulability vs. percentage of HI tasks for $|\tau| = 20$ and 50% increase of HI execution demand — uniform distribution of task periods

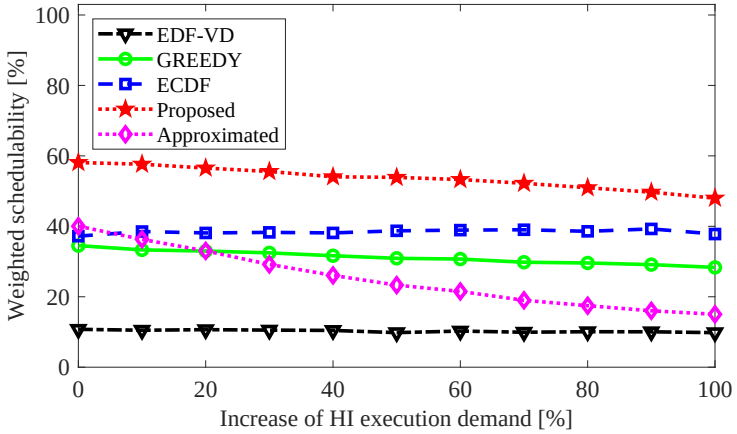

Figure 14: Weighted schedulability vs. increase of HI execution demand for $|\tau| = 20$ and 30% HI tasks — uniform distribution of task periods

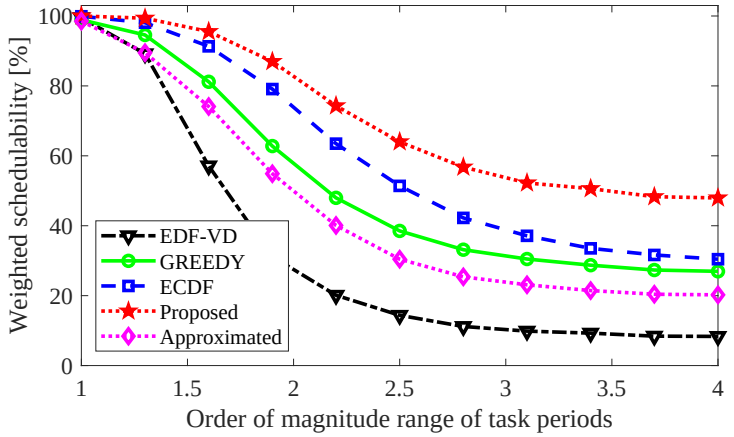

Figure 15: Weighted schedulability vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

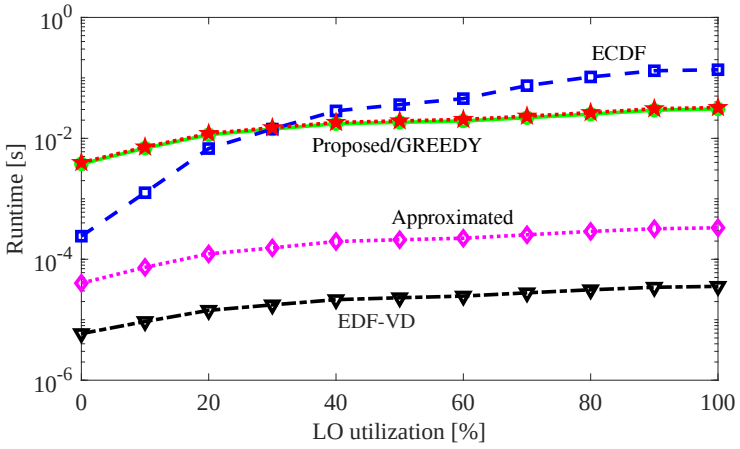

Figure 16: Runtime vs. LO utilization for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

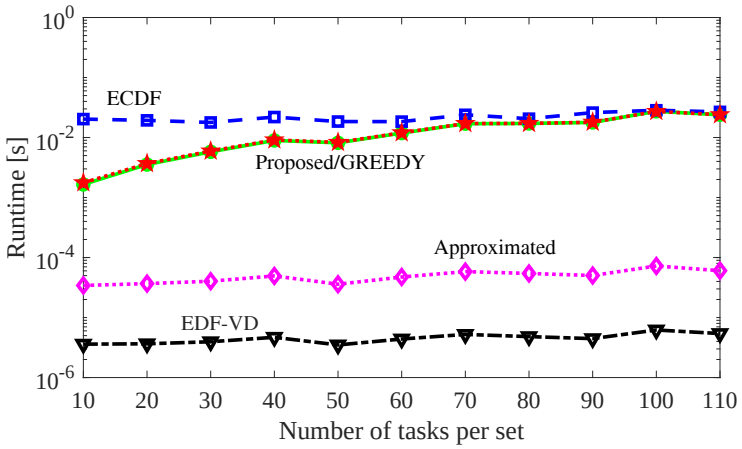

Figure 17: Runtime vs. total number of tasks for 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods

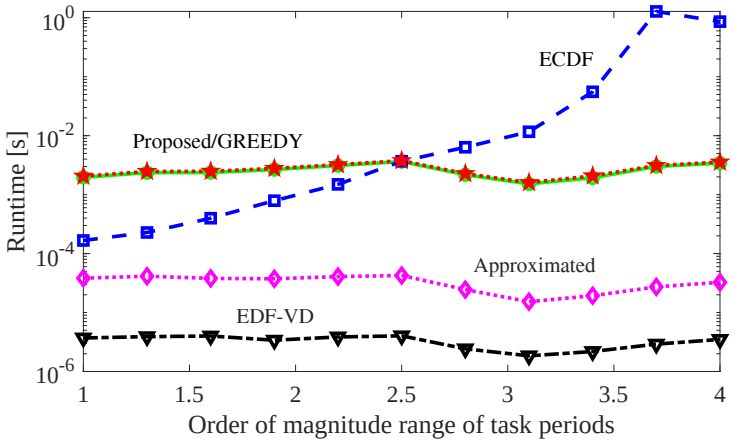

Figure 18: Runtime vs. range of task periods for $|\tau| = 20$, 30% HI tasks and 50% increase of HI execution demand — uniform distribution of task periods