This paper is published in Real-Time Systems. The final publication is available at http://link.springer.com.

Bounding and Shaping the Demand of Generalized Mixed-Criticality Sporadic Task Systems

Pontus Ekberg · **Wang Yi**

Abstract We generalize the commonly used mixed-criticality sporadic task model to let all task parameters (execution-time, deadline and period) change between criticality modes. In addition, new tasks may be added in higher criticality modes and the modes may be arranged using any directed acyclic graph, where the nodes represent the different criticality modes and the edges the possible mode switches. We formulate demand bound functions for mixed-criticality sporadic tasks and use these to determine EDF-schedulability. Tasks have different demand bound functions for each criticality mode. We show how to shift execution demand between different criticality modes by tuning the relative deadlines. This allows us to shape the demand characteristics of each task. We propose efficient algorithms for tuning all relative deadlines of a task set in order to shape the total demand to the available supply of the computing platform. Experiments indicate that this approach is successful in practice. This new approach has the added benefit of supporting hierarchical scheduling frameworks.

Keywords Real-time · Mixed-criticality · Demand bound functions · Earliest deadline first · Schedulability analysis

1 Introduction

An increasing trend in real-time systems is to integrate functionalities of different criticality, or importance, on the same platform. Such mixed-criticality systems lead to new research challenges, not least from the scheduling point of view. The major challenge is to simultaneously guarantee temporal correctness at all different levels of assurance that are mandated by the different criticalities. Typically, at a high level of assurance, we need to guarantee correctness under very pessimistic assumptions

P. Ekberg · W. Yi
Uppsala University, Department of Information Technology, Box 337, SE-751 05 Uppsala, Sweden
E-mail: pontus.ekberg@it.uu.se

W. Yi
E-mail: yi@it.uu.se

(e.g., worst-case execution times from static analysis), but only for the most critical functionalities. At a lower level of assurance, we want to guarantee the temporal correctness of all functionalities, but under less pessimistic assumptions (e.g., measured worst-case execution times).

We adapt the concept of demand bound functions (Baruah et al. 1990) to the mixed-criticality setting, and derive such functions for mixed-criticality sporadic tasks. These functions can be used to establish whether a task set is schedulable by EDF on a uniprocessor. In the mixed-criticality setting, each task has different demand bound functions for different criticality modes. We show that a task's demand bound functions for different modes are inherently connected, and that we can shift demand from one function to another by tuning the parameters of the task, specifically the relative deadline.

We are free to tune the relative deadlines of tasks as long as they are never larger than the true relative deadlines that are specified by the system designer. By such tuning we can shape the demand characteristics of a task set to match the available supply of the computing platform, specified using supply bound functions (Mok et al. 2001). We present efficient algorithms that automatically shape the demand of a task set in this manner.

The standard mixed-criticality task model, which is used in most prior work, is generalized to allow arbitrary changes in task parameters between criticality modes. The generalized model also enables the addition of tasks in higher criticality modes (e.g., to implement hardware functionality in software in case of hardware faults). The manner in which a system can switch between different criticality modes is expressed with any directed acyclic graph, giving the system designer the tools necessary to express orthogonal criticality dimensions in a single system. To the best of our knowledge, systems with non-linearly ordered criticality modes have not been considered before. The adaptation of all results to the generalized model is the main new contribution of this paper, which extends a preliminary version (Ekberg and Yi 2012).

Experimental evaluations indicate that, for most settings, the acceptance ratio of randomly generated task sets is higher with this scheduling approach than with previous approaches from the literature.

Because we allow the supply of the computing platform to be specified with supply bound functions, this scheduling approach directly enables the use of mixed-criticality scheduling within common hierarchical scheduling frameworks that employ such abstractions.

1.1 Related Work

Vestal (2007) extended fixed-priority response-time analysis of sporadic tasks to the mixed-criticality setting. His work can be considered the first on mixed-criticality scheduling. Response-time analysis for fixed-priority scheduling has since been improved by Baruah et al. (2011b)

A number of papers have considered the more restricted problem of scheduling a finite set of mixed-criticality jobs (e.g., Baruah et al. 2010, 2011c). It has been shown

by Baruah et al. (2011c) that the problem of deciding whether a given set of jobs is schedulable by an optimal scheduling algorithm is NP-hard in the strong sense. Work on mixed-criticality scheduling has since been focused on finding scheduling strategies that, while being suboptimal, still work well in practice.

One of the strategies developed for scheduling a finite set of mixed-criticality jobs is the *own criticality based priority* (OCBP) strategy by Baruah et al. (2010). It assigns priorities to the individual jobs using a variant of the so-called Audsley approach (Audsley 2001). This scheduling strategy was later extended by Li and Baruah (2010) to systems of mixed-criticality sporadic tasks, where priorities are calculated and assigned to all jobs in a busy period. A problem with this approach is that some runtime decisions by the scheduler are computationally very demanding. This was mitigated to some degree by Guan et al. (2011), who presented an OCBP-based scheduler for sporadic task sets where runtime decisions are of at most polynomial complexity.

An EDF-based approach called EDF-VD for scheduling implicit-deadline mixed-criticality sporadic task sets was proposed by Baruah et al. (2011a). An improvement to the schedulability analysis for EDF-VD was later described by Baruah et al. (2012). In EDF-VD, smaller (virtual) relative deadlines are used in lower criticality modes to ensure schedulability across mode changes, similar to how EDF is used in this paper. There are important differences in how relative deadlines are assigned in EDF-VD and in this paper: EDF-VD applies a single scaling factor to the relative deadlines of all tasks, and we allow them to be set independently. The main difference lies, however, in the schedulability analysis: EDF-VD uses a schedulability test based on the utilization metric, while we formulate demand bound functions. We believe that schedulability analysis based on demand bound functions is typically more precise, and is easier to generalize to more complex system models. The former is supported by the evaluation in Section 8 and the latter is supported in part by the fact that we have adapted our solution to a generalized system model in this paper.

An alternative mixed-criticality system model, which lets tasks' periods change between criticality modes instead of their execution-time budgets, was proposed by Baruah (2012). He also provided a schedulability analysis for EDF-based scheduling of such tasks. This system model can be encoded as a special case of the generalized mixed-criticality system model described in this paper, as will be shown in Section 8.2.

Mixed-criticality scheduling on multiprocessors has been considered by Li and Baruah (2012), who combined results from the uniprocessor scheduling of EDF-VD with global EDF-based schedulability analysis of regular multiprocessor systems. Pathan (2012) instead combined ideas from fixed-priority response-time analysis for uniprocessor mixed-criticality scheduling with regular response-time analysis for fixed-priority multiprocessor scheduling.

2 Preliminaries

2.1 Simple System Model and Notation

In the first part of this paper we use the same system model as in most previous work on the scheduling of mixed-criticality tasks (e.g., Li and Baruah 2010; Guan et al. 2011; Baruah et al. 2011b; Vestal 2007; Baruah et al. 2011a, 2012). This is a straightforward extension of the classic sporadic task model (Mok 1983) to a mixed-criticality setting, allowing worst-case execution times to vary between criticality levels.[1] Formally, each such task τ_i in a mixed-criticality sporadic task set $\tau = \{\tau_1, \ldots, \tau_k\}$ is defined by a tuple $(C_i(\text{LO}), C_i(\text{HI}), D_i, T_i, L_i)$, where:

- $C_i(\text{LO}), C_i(\text{HI}) \in \mathbb{N}_{>0}$ are the task's worst-case execution time budgets in low- and high-criticality mode, respectively,
- $D_i \in \mathbb{N}_{>0}$ is its relative deadline,
- $T_i \in \mathbb{N}_{>0}$ is its minimum inter-release separation time (also called period),
- $L_i \in \{\text{LO}, \text{HI}\}$ is the criticality of the task.

We assume constrained deadlines and also make the standard assumptions about the relations between low- and high-criticality worst-case execution times:

$$\forall \tau_i \in \tau : C_i(\text{LO}) \leqslant C_i(\text{HI}) \leqslant D_i \leqslant T_i$$

We will generalize the above model in Section 5. In the generalized model, all task parameters, including relative deadlines and periods, can change between criticality levels. It also allows the addition of new tasks in higher criticality modes and the use of an arbitrary number of modes that are structured as any directed acyclic graph.

Let $\text{LO}(\tau) \stackrel{\text{def}}{=} \{\tau_i \in \tau \mid L_i = \text{LO}\}$ denote the subset of low-criticality tasks in τ, and $\text{HI}(\tau) \stackrel{\text{def}}{=} \{\tau_i \in \tau \mid L_i = \text{HI}\}$ the subset of high-criticality tasks. We define low- and high-criticality utilization as

$$\begin{aligned} U_{\text{LO}}(\tau_i) &\stackrel{\text{def}}{=} C_i(\text{LO})/T_i \\ U_{\text{HI}}(\tau_i) &\stackrel{\text{def}}{=} C_i(\text{HI})/T_i \\ U_{\text{LO}}(\tau) &\stackrel{\text{def}}{=} \textstyle\sum_{\tau_i \in \tau} U_{\text{LO}}(\tau_i) \\ U_{\text{HI}}(\tau) &\stackrel{\text{def}}{=} \textstyle\sum_{\tau_i \in \text{HI}(\tau)} U_{\text{HI}}(\tau_i). \end{aligned}$$

For compactness of presentation we use the notation $[\![\cdot]\!]_c$ and $[\![\cdot]\!]^c$ to constrain an expression from below or above, such that $[\![A]\!]_c \stackrel{\text{def}}{=} \max(A, c)$ and $[\![A]\!]^c \stackrel{\text{def}}{=} \min(A, c)$. Also, $[\![A]\!]_c^{c'} \stackrel{\text{def}}{=} [\![[\![A]\!]_c]\!]^{c'}$.

The semantics of the system model is as follows. The system starts in low-criticality mode, and as long as it remains there, each task $\tau_i \in \tau$ releases a (possibly infinite) sequence of jobs $\langle J_i^1, J_i^2, \ldots \rangle$ in the standard way for sporadic tasks: if $r(J), d(J) \in \mathbb{R}$ are the release time and deadline of job J, then

- $r(J_i^{k+1}) \geqslant r(J_i^k) + T_i$,
- $d(J_i^k) = r(J_i^k) + D_i$.

[1] The cited works differ in the assumption of implicit, constrained or arbitrary deadlines.

The time interval $[r(J), d(J)]$ is called the scheduling window of job J. If any job executes for its entire low-criticality worst-case execution time budget without signaling that it has finished, the system will immediately switch to high-criticality mode. This switch signifies that the system's behavior is not consistent with the assumptions made at the lower level of assurance (in particular, the worst-case execution time estimates are invalid). After the switch we are not required to meet any deadlines for low-criticality jobs, but we must still meet all deadlines for high-criticality jobs, even if they execute for up to their high-criticality worst-case execution times (i.e., the high-criticality tasks get increased execution-time budgets). In practice, the low-criticality jobs can continue to execute whenever the processor would otherwise be idle, but from the modeling perspective we simply view all low-criticality tasks in $\text{LO}(\tau)$ as being discarded along with their active jobs at the time of the switch. The tasks in $\text{HI}(\tau)$ carry on unaffected. If the system has switched to high-criticality mode, it will never switch back to low-criticality.[2]

For such a system to be successfully scheduled, all (non-discarded) jobs must always meet their deadlines. Note that the only jobs that exist in high-criticality mode are from tasks in $\text{HI}(\tau)$. Since low-criticality jobs do not run in high-criticality mode, we omit to specify high-criticality worst-case execution times for low-criticality tasks.

Example 1 *As a running example we will use the following simple task set. It consists of three tasks (τ_1, τ_2 and τ_3), one of low- and two of high-criticality:*

Task	$C(\text{LO})$	$C(\text{HI})$	D	T	L
τ_1	*2*		*4*	*5*	LO
τ_2	*1*	*2*	*6*	*7*	HI
τ_3	*2*	*4*	*6*	*6*	HI

This task set is not schedulable by any fixed-priority scheduler on a dedicated unit-speed processor, as can be verified by trying all 6 possible priority assignments. We can also see that the task set is not schedulable directly by EDF: in the scenario where all tasks release a job at the same time, EDF would execute τ_1 first, leaving τ_2 and τ_3 unable to finish on time if they need to execute for $C_2(\text{HI})$ and $C_3(\text{HI})$, respectively. Neither does the task set pass the schedulability tests for OCBP (Li and Baruah 2010; Guan et al. 2011) or EDF-VD (Baruah et al. 2011a, 2012), even if deadlines are increased to be implicit, as is required by EDF-VD. However, we will see that its demand characteristics can be tuned using the techniques presented in this paper until it is schedulable by EDF.

2.2 Demand Bound Functions

A successful approach to analyzing the schedulability of real-time workloads is to use demand bound functions (Baruah et al. 1990). A demand bound function captures the maximum execution demand of a task in any time interval of a given size.

[2] One could easily find a time point where it is safe to switch back, e.g., at any time the system is idle, but it is out of scope of this paper.

Definition 1 (Demand bound function) A demand bound function $\mathrm{dbf}(\tau_i, \ell)$ gives an upper bound on the maximum possible execution demand of task τ_i in *any* time interval of length ℓ, where demand is calculated as the total amount of required execution time of jobs with their whole scheduling windows within the time interval.

There exist methods for precisely computing the demand bound functions for many popular task models in the normal (non-mixed-criticality) setting. For example, the demand bound function for a given ℓ can be computed in constant time for a standard sporadic task (Baruah et al. 1990).

A similar concept is the supply bound function $\mathrm{sbf}(\ell)$ (Mok et al. 2001), which lower bounds the amount of supplied execution time of the platform in any time window of size ℓ. For example, a unit-speed, dedicated uniprocessor has $\mathrm{sbf}(\ell) = \ell$. Other platforms, such as virtual servers used in hierarchical scheduling, have their own particular supply bound functions (e.g., Mok et al. 2001; Shin and Lee 2003). We say that a supply bound function sbf is of *at most unit speed* if

$$\mathrm{sbf}(0) = 0 \ \wedge \ \forall \ell, k \geqslant 0 : \mathrm{sbf}(\ell + k) - \mathrm{sbf}(\ell) \leqslant k.$$

We assume that a supply bound function is linear in all intervals $[k, k+1]$ between consecutive integer points k and $k+1$. The assumption of piecewise-linear supply bound functions is a natural one, and to the best of our knowledge, all proposed virtual resource platforms in the literature have such supply bound functions.

The key insight that make demand and supply bound functions useful for the analysis of real-time systems is the following known fact.

Proposition 1 (e.g., see Shin and Lee (2003)) *A non-mixed-criticality task set τ is successfully scheduled by the* earliest deadline first *(EDF) algorithm on a (uniprocessor) platform with supply bound function* sbf *if*

$$\forall \ell \geqslant 0 : \sum_{\tau_i \in \tau} \mathrm{dbf}(\tau_i, \ell) \leqslant \mathrm{sbf}(\ell).$$

3 Demand Bound Functions for Mixed-Criticality Tasks

We extend the idea of demand bound functions to the mixed-criticality setting. For each task we will construct two demand bound functions, $\mathrm{dbf}_{\mathrm{LO}}$ and $\mathrm{dbf}_{\mathrm{HI}}$, for the low- and high-criticality modes, respectively. Proposition 1 is extended in the straightforward way:

Proposition 2 *A mixed-criticality task set τ is schedulable by EDF on a platform with supply bound function* $\mathrm{sbf}_{\mathrm{LO}}$ *in low-criticality mode and* $\mathrm{sbf}_{\mathrm{HI}}$ *in high-criticality mode if both of the following conditions hold:*

$$\textbf{\textit{Condition}}\ \mathsf{S}_{\mathbf{LO}}\textbf{\textit{:}} \quad \forall \ell \geqslant 0 : \sum_{\tau_i \in \tau} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \ell) \leqslant \mathrm{sbf}_{\mathrm{LO}}(\ell)$$

$$\textbf{\textit{Condition}}\ \mathsf{S}_{\mathbf{HI}}\textbf{\textit{:}} \quad \forall \ell \geqslant 0 : \sum_{\tau_i \in \mathrm{HI}(\tau)} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \ell) \leqslant \mathrm{sbf}_{\mathrm{HI}}(\ell)$$

Conditions S_{LO} and S_{HI} capture the schedulability of the task set in low- and high-criticality mode. While the two modes can be analyzed separately with the above conditions, we will see that the demand in high-criticality mode depends on what can happen in low-criticality mode.

We assume, without loss of generality, that sbf_{LO} is of at most unit speed. This can always be achieved by simply scaling the parameters of the task set together with sbf_{LO} and sbf_{HI}. Note that sbf_{LO} and sbf_{HI} may be different, allowing a change of processor speed or virtual server scheduling policy when switching to high-criticality mode.

How then do we construct these demand bound functions? In the case of dbf_{LO} it is simple. In low-criticality mode, each task τ_i behaves like a normal sporadic task, and all of its jobs are guaranteed to execute for at most $C_i(\text{LO})$ time units (otherwise the system, by definition, would switch to high-criticality mode). We can therefore use the standard method for computing demand bound functions for sporadic tasks (Baruah et al. 1990).

With dbf_{HI} it gets more tricky because we need to consider the high-criticality jobs that are active during the switch to high-criticality mode.

Definition 2 (Carry-over jobs) A job from a high-criticality task that is active (released, but not finished) at the time of the switch to high-criticality mode is called a *carry-over job*.

3.1 Characterizing the Demand of Carry-Over Jobs

In high-criticality mode we need to finish the remaining execution time of carry-over jobs before their respective deadlines. The demand of carry-over jobs must therefore be accounted for in each high-criticality task's dbf_{HI}. Conceptually, when analyzing the schedulability in high-criticality mode, we can think of a carry-over job as a job that is released at the time of the switch. However, the scheduling window of such a job is the remaining interval between switch and deadline (see Fig. 1), and can therefore be shorter than for other jobs of the same task. Because it might have executed for some time before the switch, its execution demand may also be lower.

For the sake of bounding the demand in high-criticality mode (in order to meet Condition S_{HI}), we can assume that the demand is met in low-criticality mode (Condition S_{LO}), or the task set would be deemed unschedulable anyway. In other words, we seek to show $\mathsf{S}_{\text{LO}} \wedge \mathsf{S}_{\text{HI}}$ by showing $\mathsf{S}_{\text{LO}} \wedge (\mathsf{S}_{\text{LO}} \rightarrow \mathsf{S}_{\text{HI}})$. For a system scheduled by EDF, we can therefore assume that all deadlines are met in low-criticality mode when we bound the demand in high-criticality mode.

Consider then what we can show about the remaining execution demand of carry-over jobs. At the time of the switch to high-criticality mode, a carry-over job from high-criticality task τ_i has x time units left until its deadline, for some $x \geqslant 0$. The remaining scheduling window of this job is therefore of length x. Since this job would have met its deadline in low-criticality mode if the switch had *not* happened, there can be at most x time units left of its low-criticality execution demand $C_i(\text{LO})$ at the time of the switch (this follows directly from the assumption that sbf_{LO} is of at most

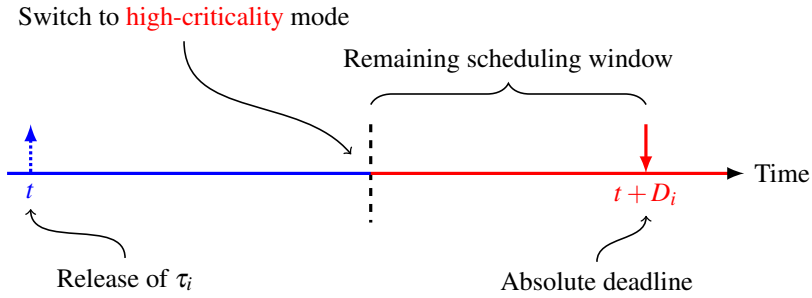

Fig. 1 After a switch to high-criticality mode, the remaining execution demand of a carry-over job must be finished in its remaining scheduling window.

unit speed). The job must therefore have executed for at least $[\![C_i(\text{LO}) - x]\!]_0$ time units before the switch. Since the system has switched to high-criticality mode, the job may now execute for up to $C_i(\text{HI})$ time units in total. The total execution demand remaining for the carry-over job after the switch is therefore at most $C_i(\text{HI}) - [\![C_i(\text{LO}) - x]\!]_0$. Unfortunately, as x becomes smaller, this demand is increasingly difficult to accommodate, and leads to $\text{dbf}_{\text{HI}}(\tau_i, 0) = C_i(\text{HI}) - C_i(\text{LO})$ in the extreme case. Clearly, with such bounds we cannot hope to satisfy Condition S_{HI}. Next we will show how this problem can be mitigated.

3.2 Adjusting the Demand of Carry-Over Jobs

The problem above stems from the fact that EDF may execute a high-criticality job quite late in low-criticality mode. When the system switches to high-criticality mode, a carry-over job can be left with a very short scheduling window in which to finish what remains of its high-criticality worst-case execution demand. In order to increase the size of the remaining scheduling window we separate the relative deadlines used in the different modes. For a task τ_i we let EDF use relative deadlines $D_i(\text{LO})$ and $D_i(\text{HI})$, such that if a job is released at time t, the priority assigned to it by EDF is based on the value $t + D_i(\text{LO})$ while in low-criticality mode and based on $t + D_i(\text{HI})$ while in high-criticality mode. This is essentially the same run-time scheduling as that of EDF-VD (Baruah et al. 2011a, 2012).

We can safely lower the relative deadline of a task because meeting the earlier deadline implies meeting the original (true) deadline. We can gain valuable extra slack time for a carry-over job from high-criticality task τ_i by lowering $D_i(\text{LO})$, albeit at the cost of a worsened demand in low-criticality mode. We therefore want $D_i(\text{LO}) = D_i$ if $L_i = \text{LO}$ and $D_i(\text{LO}) \leqslant D_i(\text{HI}) = D_i$ if $L_i = \text{HI}$. Also, $C_i(\text{LO}) \leqslant D_i(\text{LO})$ is assumed, just as with the original deadline. Note that $D_i(\text{LO})$ is not an actual relative deadline for τ_i in the sense that it does not necessarily correspond to the timing constraints specified by the system designer. However, it is motivated to call it a "deadline", because we construct each dbf_{LO} and use EDF in low-criticality mode *as*

if it was the relative deadline. With separated relative deadlines we can make stronger guarantees about the remaining execution demand of carry-over jobs:

Lemma 1 (Demand of carry-over jobs) *Assume that EDF uses relative deadlines $D_i(\text{LO})$ and $D_i(\text{HI})$ with $D_i(\text{LO}) \leqslant D_i(\text{HI}) = D_i$ for high-criticality task τ_i, and that we can guarantee that the demand is met in low-criticality mode (using $D_i(\text{LO})$). If the switch to high-criticality mode happens when a job from τ_i has a remaining scheduling window of x time units left until its true deadline, as illustrated in Fig. 2, then the following hold:*

1. *If $x < D_i(\text{HI}) - D_i(\text{LO})$, then the job has already finished before the switch.*
2. *If $x \geqslant D_i(\text{HI}) - D_i(\text{LO})$, then the job may be a carry-over job, and no less than $[\![C_i(\text{LO}) - x + D_i(\text{HI}) - D_i(\text{LO})]\!]_0$ time units of the job's work were finished before the switch.*

Proof In the first case, the switch to high-criticality mode happens after the low-criticality deadline. Since we assume that the demand is met in low-criticality mode (using relative deadline $D_i(\text{LO})$), EDF is guaranteed to finish the job by this deadline, and therefore it was finished by the time of the switch.

In the second case, there are $x - (D_i(\text{HI}) - D_i(\text{LO}))$ time units left until the low-criticality deadline. Since the demand is guaranteed to be met in low-criticality mode, and the supply of the platform is of at most unit speed, there can be at most $x - (D_i(\text{HI}) - D_i(\text{LO}))$ time units left of the job's low-criticality execution demand. At least $[\![C_i(\text{LO}) - x + D_i(\text{HI}) - D_i(\text{LO})]\!]_0$ time units of the job's work must therefore have been finished already by the time of the switch. □

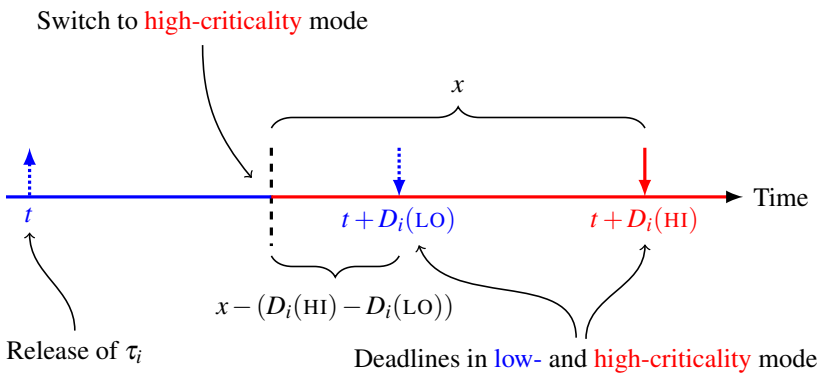

Fig. 2 A carry-over job of τ_i has a remaining scheduling window of length x after the switch to high-criticality mode. Here the switch happens before the job's low-criticality deadline.

Next we will show how to define $\text{dbf}_{\text{LO}}(\tau_i, \ell)$ and $\text{dbf}_{\text{HI}}(\tau_i, \ell)$ for a given $D_i(\text{LO})$. An algorithm for computing reasonable values for $D_i(\text{LO})$ for each task $\tau_i \in \tau$ is presented in Section 4.

3.3 Formulating the Demand Bound Functions

As described above, while the system is in low-criticality mode, each task τ_i behaves as a normal sporadic task with parameters $C_i(\text{LO})$, $D_i(\text{LO})$ and T_i. Note that it uses relative deadline $D_i(\text{LO})$, where $D_i(\text{LO}) = D_i$ if $L_i = \text{LO}$ and $D_i(\text{LO}) \leqslant D_i(\text{HI}) = D_i$ if $L_i = \text{HI}$. A tight demand bound function of such a task is known (Baruah et al. 1990):

$$\text{dbf}_{\text{LO}}(\tau_i, \ell) \stackrel{\text{def}}{=} \left[\!\!\left[\left(\left\lfloor \frac{\ell - D_i(\text{LO})}{T_i} \right\rfloor + 1 \right) \cdot C_i(\text{LO}) \right]\!\!\right]_0 \tag{1}$$

The demand bound function for task τ_i in high-criticality mode, $\text{dbf}_{\text{HI}}(\tau_i, \ell)$, must provide an upper bound on the maximum execution demand of jobs from τ_i with scheduling windows inside any interval of length ℓ. This may include one carry-over job. From Lemma 1 we know that the (remaining) scheduling window of a carry-over job from τ_i is at least $D_i(\text{HI}) - D_i(\text{LO})$ time units long. A time interval of length $D_i(\text{HI}) - D_i(\text{LO})$ is therefore the smallest in which we can fit the scheduling window of *any* job from τ_i. More generally, the smallest time interval in which we can fit the scheduling windows of k jobs is of length $(D_i(\text{HI}) - D_i(\text{LO})) + (k-1) \cdot T_i$. The execution demand of τ_i in an interval of length ℓ is therefore bounded by

$$\text{full}_{\text{HI}}(\tau_i, \ell) \stackrel{\text{def}}{=} \left[\!\!\left[\left(\left\lfloor \frac{\ell - (D_i(\text{HI}) - D_i(\text{LO}))}{T_i} \right\rfloor + 1 \right) \cdot C_i(\text{HI}) \right]\!\!\right]_0 \tag{2}$$

The function $\text{full}_{\text{HI}}(\tau_i, \ell)$ is disregarding that a carry-over job may have finished some execution in low-criticality mode (i.e., it is counting $C_i(\text{HI})$ for all jobs). We can check whether all jobs that contributed execution demand to $\text{full}_{\text{HI}}(\tau_i, \ell)$ can fit their scheduling windows into an interval of length ℓ without one of them being a carry-over job. If one must be a carry-over job, we can subtract the execution time that it must have finished before the switch according to Lemma 1.

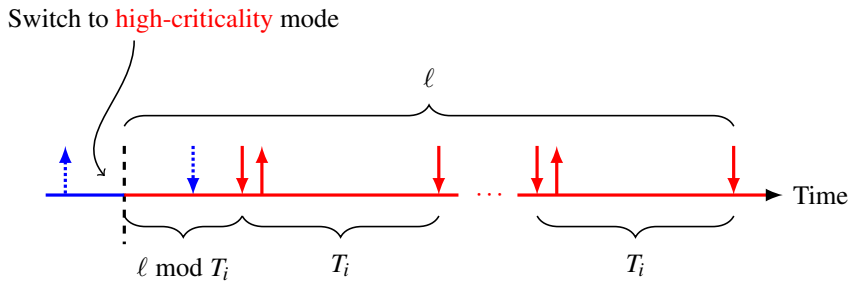

Fig. 3 After fitting a number of full jobs into an interval of length ℓ, there are ℓ mod T_i time units left for either another full job, a carry-over job, or no job at all. In this figure it is enough for a carry-over job.

As shown in Fig. 3, for a time interval of length ℓ, there are at most $x = \ell \bmod T_i$ time units left for the "first" job (which may be a carry-over job). If $x \geqslant D_i(\text{HI})$, it is enough for the scheduling window of a full job, and we cannot subtract anything from

$\text{full}_{\text{HI}}(\tau_i, \ell)$. If $x < D_i(\text{HI}) - D_i(\text{LO})$, all jobs that contributed to $\text{full}_{\text{HI}}(\tau_i, \ell)$ can fit their entire periods inside the interval, so there is again nothing to subtract. Otherwise, we use Lemma 1 to quantify the amount of work that must have been finished in low-criticality mode:

$$\text{done}_{\text{HI}}(\tau_i, \ell) \stackrel{\text{def}}{=} \begin{cases} [\![C_i(\text{LO}) - x + D_i(\text{HI}) - D_i(\text{LO})]\!]_0\,, \\ \qquad\qquad \text{if } D_i(\text{HI}) > x \geqslant D_i(\text{HI}) - D_i(\text{LO}) \\ 0, \qquad \text{otherwise,} \end{cases} \tag{3}$$

where $x = \ell \bmod T_i$. Note that by maximizing the remaining scheduling window of the carry-over job (to $\ell \bmod T_i$) we also maximize its remaining execution demand.

The two terms can now be combined to form the demand bound function in high-criticality mode:

$$\text{dbf}_{\text{HI}}(\tau_i, \ell) \stackrel{\text{def}}{=} \text{full}_{\text{HI}}(\tau_i, \ell) - \text{done}_{\text{HI}}(\tau_i, \ell) \tag{4}$$

Example 2 *Consider task τ_3 from Example 1. Part of the demand bound functions for τ_3 are shown in Fig. 4, using two different values for $D_3(\text{LO})$. Note that a smaller $D_3(\text{LO})$ leads to a lessened demand in high-criticality mode, at the cost of an increased demand in low-criticality mode.*

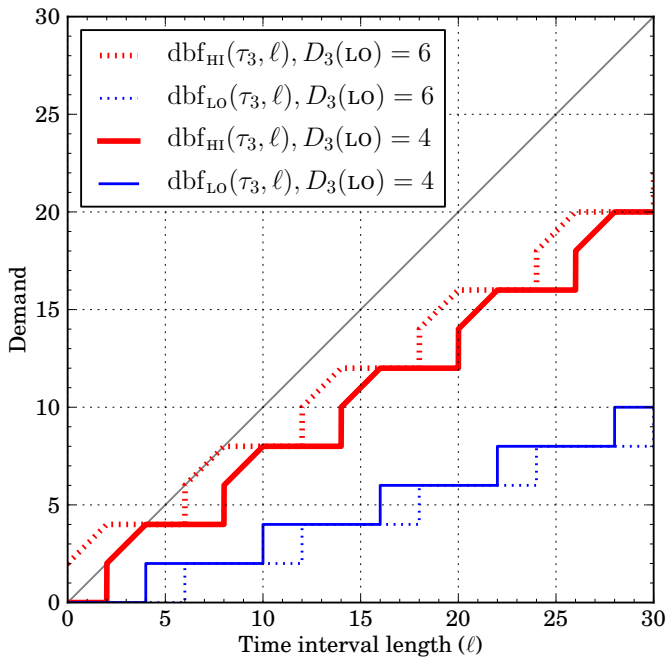

Fig. 4 Demand bound functions for task τ_3 from Example 1 with two different values for $D_3(\text{LO})$.

4 Tuning Relative Deadlines

In the previous section we constructed demand bound functions for mixed-criticality sporadic tasks, where the relative deadlines used by EDF may differ in low- and high-criticality mode for high-criticality tasks. The motivation for separating the relative deadlines used is that by artificially lowering the relative deadline $D_i(\text{LO})$ used in low-criticality mode, we can lessen τ_i's demand in high-criticality mode at the cost of increasing the demand in low-criticality mode. By choosing suitable values for $D_i(\text{LO})$ for all tasks $\tau_i \in \text{HI}(\tau)$, we are increasing our chances of fitting the total demand under the guaranteed supply in both modes, and thereby make both Conditions S_{LO} and S_{HI} of Proposition 2 hold.

We are constrained to pick a value for $D_i(\text{LO})$ such that $C_i(\text{LO}) \leqslant D_i(\text{LO}) \leqslant D_i$. This gives us

$$\prod_{\tau_i \in \text{HI}(\tau)} (D_i - C_i(\text{LO}) + 1)$$

possible combinations for the task set. The number of combinations is exponentially increasing with the number of high-criticality tasks, and it is infeasible to simply try all combinations. We instead seek a heuristic algorithm for tuning the relative deadlines of all tasks. In this section we present one such algorithm, which is of pseudo-polynomial time complexity for suitable supply bound functions.

The following lemma is a key insight for understanding the effects of changing relative deadlines. A proof is given in Appendix A.

Lemma 2 (Shifting) *If high-criticality tasks τ_i and τ_j are identical (i.e., have equal parameters), except that $D_i(\text{LO}) = D_j(\text{LO}) - \delta$ for $\delta \in \mathbb{Z}$, then*

$$\text{dbf}_{\text{LO}}(\tau_i, \ell) = \text{dbf}_{\text{LO}}(\tau_j, \ell + \delta)$$
$$\text{dbf}_{\text{HI}}(\tau_i, \ell) = \text{dbf}_{\text{HI}}(\tau_j, \ell - \delta)$$

In other words, if we consider the demand bound functions graphically as in Fig. 4, then by decreasing $D_i(\text{LO})$ by δ, we are allowed to move $\text{dbf}_{\text{HI}}(\tau_i, \ell)$ by δ steps to the right at the cost of moving $\text{dbf}_{\text{LO}}(\tau_i, \ell)$ by δ steps to the left. Informally, we can think of the problem as moving around the dbf_{LO} and dbf_{HI} of each task until we hopefully find a configuration where the total demand of the task set is met by the supply in both low- and high-criticality mode.

Algorithm 1 tunes the demand of a task set in a somewhat greedy fashion. Let $\mathscr{S}_{\text{LO}}(\ell)$ and $\mathscr{S}_{\text{HI}}(\ell)$ be predicates corresponding to the inequalities found in Conditions S_{LO} and S_{HI}, respectively:

$$\mathscr{S}_{\text{LO}}(\ell) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \text{dbf}_{\text{LO}}(\tau_i, \ell) \leqslant \text{sbf}_{\text{LO}}(\ell)$$
$$\mathscr{S}_{\text{HI}}(\ell) \stackrel{\text{def}}{=} \sum_{\tau_i \in \text{HI}(\tau)} \text{dbf}_{\text{HI}}(\tau_i, \ell) \leqslant \text{sbf}_{\text{HI}}(\ell)$$

The general idea is to check $\mathscr{S}_{\text{LO}}(\ell)$ and $\mathscr{S}_{\text{HI}}(\ell)$ for increasing time interval lengths ℓ (from 0 up to an upper bound ℓ_{max} described in Section 4.1). As soon as it finds a value for ℓ for which either condition fails, it changes one relative deadline (or terminates) and goes back to $\ell = 0$:

- If $\mathscr{S}_{\text{HI}}(\ell)$ fails, the low-criticality relative deadline of one task is decreased by 1. It picks the task τ_i which would see the largest decrease in $\text{dbf}_{\text{HI}}(\tau_i, \ell)$ when $D_i(\text{LO})$ is decreased by 1 (ties broken arbitrarily).
- If $\mathscr{S}_{\text{LO}}(\ell)$ fails, the latest deadline change is undone. If there is no change to undo, the algorithm fails. Note that it backtracks at most one step in this way.

Algorithm 1: GreedyTuning(τ)

```
1  begin
2      candidates ← {i | τ_i ∈ HI(τ)}
3      mod ← ⊥
4      ℓ_max ← upper bound for ℓ in Conditions S_LO and S_HI
5      repeat
6          changed ← false
7          for ℓ = 0, 1, ..., ℓ_max do
8              if ¬S_LO(ℓ) then
9                  if mod = ⊥ then
10                     return FAILURE
11                 D_mod(LO) ← D_mod(LO) + 1
12                 candidates ← candidates \ {mod}
13                 mod ← ⊥
14                 changed ← true
15                 break
16             else if ¬S_HI(ℓ) then
17                 if candidates = ∅ then
18                     return FAILURE
19                 mod ← arg max_{i∈candidates}(dbf_HI(τ_i, ℓ) − dbf_HI(τ_i, ℓ − 1))
20                 D_mod(LO) ← D_mod(LO) − 1
21                 if D_mod(LO) = C_mod(LO) then
22                     candidates ← candidates \ {mod}
23                 changed ← true
24                 break
25     until ¬changed
26     return SUCCESS
```

The algorithm terminates with SUCCESS only if it has found low-criticality relative deadlines with which $\mathscr{S}_{\text{LO}}(\ell)$ and $\mathscr{S}_{\text{HI}}(\ell)$ hold for all $\ell \in \{0, 1, \ldots, \ell_{\max}\}$. This implies that both Conditions S_{LO} and S_{HI} hold, as will be shown in Section 4.1. Therefore, the algorithm terminates with SUCCESS only if the task set is schedulable according to Proposition 2. If the algorithm terminates with FAILURE, it has failed to find relative deadlines with which both Conditions S_{LO} and S_{HI} hold. This does not necessarily mean that such relative deadlines can not be found in some other way.

Example 3 *Consider how Algorithm 1 assigns values to $D_2(\text{LO})$ and $D_3(\text{LO})$ for the two high criticality tasks τ_2 and τ_3 in the task set from Example 1. We assume a dedicated platform ($\text{sbf}_{\text{LO}}(\ell) = \text{sbf}_{\text{HI}}(\ell) = \ell$). Fig. 5 shows the demand bound functions for this task set with unmodified relative deadlines. In the first iteration, $\mathscr{S}_{\text{HI}}(0)$ fails, and $D_3(\text{LO})$ is decreased by 1. In the second iteration, $\mathscr{S}_{\text{HI}}(0)$ fails again, but this time $D_2(\text{LO})$ is decreased by 1. In the third iteration, $\mathscr{S}_{\text{HI}}(1)$ fails and $D_3(\text{LO})$ is decreased by 1 again. This is then repeated two more times where $\mathscr{S}_{\text{HI}}(\ell)$ fails at $\ell = 2$ and $\ell = 3$, respectively, and $D_3(\text{LO})$ is lowered two more times. Both $\mathscr{S}_{\text{LO}}(\ell)$ and $\mathscr{S}_{\text{HI}}(\ell)$ then hold for all $\ell \in \{0, 1, \ldots, \ell_{\max}\}$, and the algorithm terminates with $D_2(\text{LO}) = 5$ and $D_3(\text{LO}) = 2$, resulting in the demand bound functions shown in Fig. 6.*

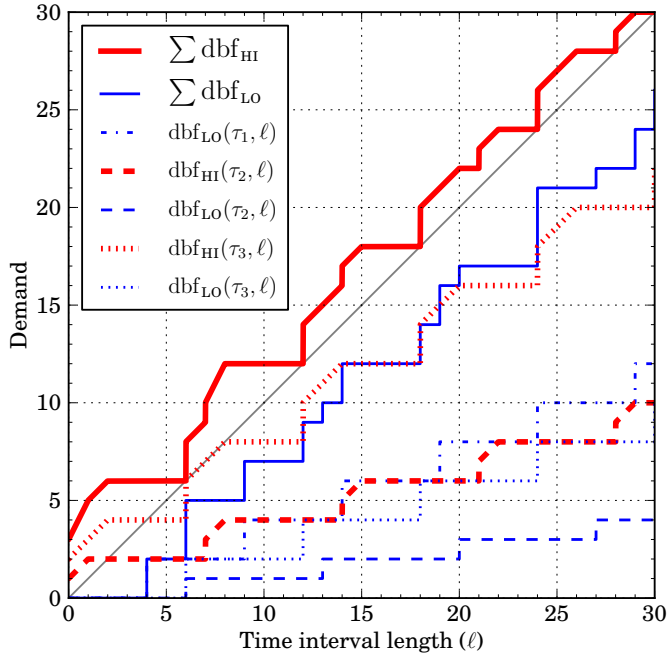

Fig. 5 Demand bound functions for the tasks from Example 1 with unmodified low-criticality relative deadlines ($D_i(\text{LO}) = D_i(\text{HI}) = D_i$).

4.1 Complexity and Correctness of the Algorithm

For the complexity of Algorithm 1, note that each $\tau_i \in \text{HI}(\tau)$ will have its deadline $D_i(\text{LO})$ changed at most $D_i - C_i(\text{LO}) + 1$ times. In every iteration of the outer loop some low-criticality relative deadline is changed, or the algorithm terminates, so the

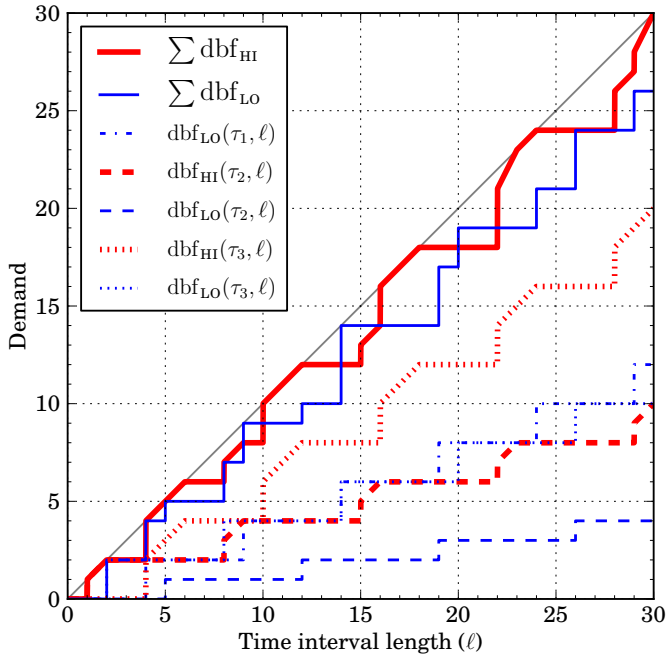

Fig. 6 Demand bound functions for the tasks from Example 1 after having low-criticality relative deadlines tuned by Algorithm 1.

outer loop is iterated at most

$$\sum_{\tau_i \in \mathrm{HI}(\tau)} (D_i - C_i(\mathrm{LO}) + 1)$$

times. The inner for-loop is iterated at most $\ell_{\max} + 1$ times for every iteration of the outer loop. The algorithm is therefore of pseudo-polynomial time complexity if $\ell_{\max}$ is pseudo-polynomial. We will see that a pseudo-polynomial $\ell_{\max}$ can be found in the common setting where the supply is from a dedicated platform.

The algorithm terminates with SUCCESS only if it has found relative deadlines with which both $\mathscr{S}_{\mathrm{LO}}(\ell)$ and $\mathscr{S}_{\mathrm{HI}}(\ell)$ hold for all $\ell \in \{0,1,\ldots,\ell_{\max}\}$. However, in Proposition 2, the inequalities $\mathscr{S}_{\mathrm{LO}}(\ell)$ and $\mathscr{S}_{\mathrm{HI}}(\ell)$ should hold for all $\ell \geqslant 0$. We will show here that $\ell_{\max}$ can be found such that if $\mathscr{S}_{\mathrm{LO}}(\ell)$ and $\mathscr{S}_{\mathrm{HI}}(\ell)$ hold for $\ell \in \{0,1,\ldots,\ell_{\max}\}$, then they hold for all $\ell \geqslant 0$.

Consider first why it is enough to check only integer-valued ℓ. Both $\mathrm{sbf}_{\mathrm{LO}}$ and $\mathrm{sbf}_{\mathrm{HI}}$ are linear in all intervals $[k,k+1]$ between consecutive integer points k and $k+1$. All $\mathrm{dbf}_{\mathrm{LO}}$ and $\mathrm{dbf}_{\mathrm{HI}}$ are non-decreasing in ℓ and also linear in all intervals $[k,k+1)$ for consecutive integers k and $k+1$ (and so are the left-hand sides of $\mathscr{S}_{\mathrm{LO}}(\ell)$ and $\mathscr{S}_{\mathrm{HI}}(\ell)$). It follows directly that if $\mathscr{S}_{\mathrm{LO}}(\ell)$ or $\mathscr{S}_{\mathrm{HI}}(\ell)$ does not hold for an $\ell \in [k,k+1]$ with $k \in \mathbb{N}_{\geqslant 0}$, then it also does not hold for either k or $k+1$.

How a bound $\ell_{\max}$ can be found depends on the supply bound functions used. It is always possible to use the hyperperiod as the bound $\ell_{\max}$. However, for a dedicated

uniprocessor ($\text{sbf}_{\text{LO}}(\ell) = \text{sbf}_{\text{HI}}(\ell) = \ell$) we can use established methods (Baruah et al. 1990) to calculate a pseudo-polynomial $\ell_{\max}$ as long as $U_{\text{LO}}(\tau)$ and $U_{\text{HI}}(\tau)$ are a priori bounded by a constant smaller than 1. To see this, we first create mappings f_{LO} and f_{HI} from mixed-criticality sporadic tasks to normal (non-mixed-criticality) sporadic tasks (C, D, T) in the following way:

$$f_{\text{LO}}(\tau_i) \stackrel{\text{def}}{=} (C_i(\text{LO}), D_i(\text{LO}), T_i)$$
$$f_{\text{HI}}(\tau_i) \stackrel{\text{def}}{=} (C_i(\text{HI}), D_i(\text{HI}) - D_i(\text{LO}), T_i)$$

Note that using the classic demand bound function dbf for normal sporadic tasks, first described by Baruah et al. (1990), we have $\text{dbf}(f_{\text{LO}}(\tau_i), \ell) = \text{dbf}_{\text{LO}}(\tau_i, \ell)$ and $\text{dbf}(f_{\text{HI}}(\tau_i), \ell) = \text{full}_{\text{HI}}(\tau_i, \ell) \geqslant \text{dbf}_{\text{HI}}(\tau_i, \ell)$. Also, if U gives the utilization of a normal sporadic task, we have $U(f_{\text{LO}}(\tau_i)) = U_{\text{LO}}(\tau_i)$ and $U(f_{\text{HI}}(\tau_i)) = U_{\text{HI}}(\tau_i)$.

Baruah et al. (1990) showed how to construct a pseudo-polynomial bound for normal sporadic task sets such that the inequality in Proposition 1 holds for all ℓ larger than the bound (using a dedicated uniprocessor), as long as the utilization of the task set is bounded by a constant smaller than 1. Clearly, if we construct such a bound $\ell^{\text{LO}}_{\max}$ for the task set $\{f_{\text{LO}}(\tau_i) \mid \tau_i \in \tau\}$, it is also valid for Condition S_{LO} in Proposition 2. Similarly, such a bound $\ell^{\text{HI}}_{\max}$ for the task set $\{f_{\text{HI}}(\tau_i) \mid \tau_i \in \text{HI}(\tau)\}$ is valid for Condition S_{HI} of Proposition 2. We can therefore use $\ell_{\max} = \max(\ell^{\text{LO}}_{\max}, \ell^{\text{HI}}_{\max})$ for Algorithm 1.[3]

5 Generalizing the Mixed-Criticality Task Model

In Section 2 we described the standard mixed-criticality sporadic task model, which is used in most previous work on mixed-criticality scheduling (e.g., Li and Baruah 2010; Guan et al. 2011; Baruah et al. 2011b; Vestal 2007; Baruah et al. 2011a, 2012). This task model is execution-time centric, as it focuses solely on differences in the worst-case execution-time parameter between criticality levels. Arguably, if one has to pick a single parameter to focus on, the execution time is a good choice because it is almost always an approximation, and its value typically varies greatly with the level of assurance that is desired. There are cases where it is desirable to vary other parameters, though. Consider, for example, a task that is triggered by external events. The period of such a task should be an under-approximation of the time interval between two consecutive trigger events. At different criticality levels, different values for the period parameter might be more suitable, depending on the required assurance that it is a safe under-approximation. Baruah (2012) introduced a task model where the period parameter differs between criticality levels, instead of the execution-time parameter.

We would like the task model to be as general as possible, without forcing an interpretation of it on the system designer. It should be up to the system designer to

[3] A small technical issue is that the bound by Baruah et al. (1990) is dependent on the relative deadlines of tasks, which are changed by Algorithm 1. The issue is easily avoided by using the largest bound generated with any of the possible relative deadlines that may be assigned (this is simply $D_i(\text{LO}) = C_i(\text{LO})$ for all $\tau_i \in \text{HI}(\tau)$). An even easier solution is to use an alternative bound that is independent of relative deadlines, e.g., the one described by Stigge et al. (2011).

decide what it means for the system to be in any one particular criticality mode, e.g., which tasks should run there; what parameters they should have; and which events trigger the system to switch to or from that criticality mode, be it an execution-time budget overrun, a hardware malfunction or anything else. Note that such generalizations bring the notion of mixed criticality closer to that of regular mode switches (e.g., see Real and Crespo 2004). We think that this is a proper development, as long as we retain the differences between mixed criticality on the one hand, and regular mode switches on the other. We argue that the most important difference between these concepts is that while a regular mode switch often is controlled, a change of criticality modes is forced upon the system by immediate and unexpected events. Such events cannot be handled by deferring task releases as is often done for controlled mode switches. Instead, the possibility of them must be prepared for in advance as is done in mixed-criticality scheduling. Still, the border between these concepts is somewhat fuzzy, and we think that it is not unlikely that some existing solutions regarding the scheduling of regular mode switching systems can be adapted for mixed-criticality scheduling. One can look at mixed-criticality systems as mode switching systems with a particular class of mode change protocols.

We will generalize the mixed-criticality sporadic task model to allow *all* task parameters to change between criticality modes. It will also be possible to add new tasks to the system when it switches to a higher criticality mode. To motivate the latter, consider as an example a system where a hardware malfunction triggers the creation of a new task that compensates for the missing functionality in software; in this system a hardware fault triggers a switch to a higher criticality mode, where some new tasks are added and possibly some old tasks are suspended or have their parameters changed. Another example is a distributed system where a node failure causes some critical tasks to be migrated to another node. From the point of view of the node receiving the tasks, there is a switch to a new criticality mode where it must accommodate the new tasks.

In addition, we will lift the restriction to only two criticality modes, and allow an arbitrary number of modes that are not necessarily linearly ordered. The ways in which criticality modes can be changed are expressed using any directed acyclic graph (DAG), as in Example 4. To the best of our knowledge, non-linearly ordered criticality modes have not been considered for mixed-criticality scheduling before.

Example 4 *Consider a system that the designer wants to have different criticality levels with different worst-case execution time budgets, in the standard manner for mixed-criticality systems. Also, the designer wants to be able to compensate for missing hardware functionality in software in the case of some specific hardware failures, and therefore wishes to add one or more tasks and possibly modify others in the face of such an event. The criticality modes of this system could be arranged as in Fig.7. The system would start running in the mode entitled* m_{NORMAL}*, which is its normal operating mode. In the event of an attempted execution-time overrun, it would switch to the mode* m_{WCET} *where some non-critical tasks may be suspended, and the remaining tasks get higher worst-case execution time budgets. In the event of a hardware fault, the system instead switches to the mode* m_{HW}*, in which some new tasks are added. In order to accommodate the new tasks, the designer may wish to suspend some old*

ones, or lower the demand of some tasks by, for example, increasing their periods or relative deadlines. In the event that both execution-time overruns and hardware faults occur, the system switches to the mode $m_{\text{WCET+HW}}$, where the designer must decide which tasks are most critical for the system in such extreme conditions.

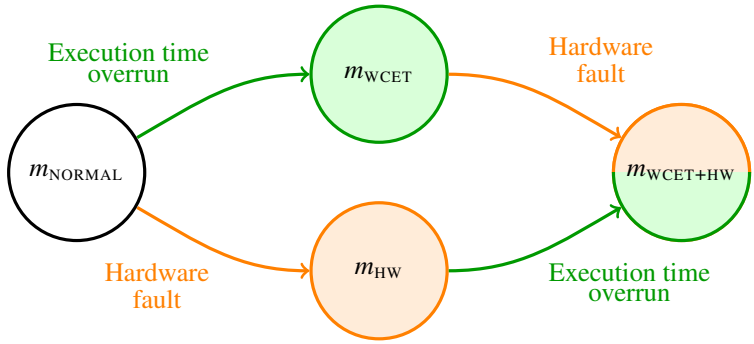

Fig. 7 An example structure of a system's criticality modes.

5.1 Formalizing the Generalized System Model

A generalized mixed-criticality sporadic task system is formally defined by a pair (τ, G), where τ is a set of tasks, $\{\tau_1, \ldots, \tau_k\}$, and G is a DAG describing the structure of the criticality modes. The vertex set $V(G)$ contains the possible criticality modes and the edge set $E(G)$ the ways in which criticality modes may change. The graph G is called the *criticality-mode structure* of the system.

Each task $\tau_i \in \tau$ is defined by a set $\mathfrak{L}_i$, and a tuple $(C_i(m), D_i(m), T_i(m))$ for each $m \in \mathfrak{L}_i$, where:

- $\mathfrak{L}_i \subseteq V(G)$ is the set of criticality modes in which τ_i is active,
- $C_i(m) \in \mathbb{N}_{>0}$ is the task's worst-case execution time in criticality mode m,
- $D_i(m) \in \mathbb{N}_{>0}$ is its relative deadline in criticality mode m,
- $T_i(m) \in \mathbb{N}_{>0}$ is its minimum inter-release separation time (also called period) in criticality mode m.

We assume that for each $\tau_i \in \tau$ and for each $m \in \mathfrak{L}_i$ that $C_i(m) \leqslant D_i(m) \leqslant T_i(m)$, similar to the assumptions about the standard task model. However, there are no restrictions on the relations between the parameters of a task in different criticality modes, i.e., all its parameters may change to arbitrary values.

Utilization is defined in the natural way:

$$U_m(\tau_i) \stackrel{\text{def}}{=} \begin{cases} C_i(m)/T_i(m), & \text{if } m \in \mathfrak{L}_i \\ 0, & \text{otherwise} \end{cases}$$

$$U_m(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_m(\tau_i)$$

The new model generalizes the standard mixed-criticality task model described in Section 2. Note that the criticality-mode structure G for the standard model would have only two vertices, $V(G) = \{\text{LO}, \text{HI}\}$, which are connected by a single edge, $E(G) = \{(\text{LO}, \text{HI})\}$.

The semantics of the generalized model is very similar to the semantics of the standard model: In criticality mode m, each task τ_i that is active in m releases jobs as if it was a normal sporadic task with parameters $(C_i(m), D_i(m), T_i(m))$. The system may switch from criticality mode m to another mode m' if $(m, m') \in E(G)$, where G is the criticality-mode structure. If the system switches from m to m', each task τ_i can be affected in different ways:

- If $m \in \mathfrak{L}_i$ and $m' \notin \mathfrak{L}_i$, the task is suspended and its active jobs discarded.
- If $m \notin \mathfrak{L}_i$ and $m' \in \mathfrak{L}_i$, the task is activated and may immediately start releasing jobs.
- If $m, m' \in \mathfrak{L}_i$, the task remains active, but its parameters are immediately changed to those at criticality level m'. This also affects any active (carry-over) job of the task, which will have its absolute deadline and execution-time budget immediately updated. If $C_i(m) > C_i(m')$ and a carry-over job has already executed for at least $C_i(m')$ before the mode switch, the job's execution-time budget in m' is considered to be spent, but not exceeded; the job must therefore be stopped or trigger another mode switch. The first new job of τ_i in m' can be released $T_i(m')$ time units after the task's last job release in previous modes.

The system may start in any criticality mode $m \in V(G)$ that has no incoming edges in $E(G)$. The set of such vertices is denoted $\text{roots}(G)$. We expect most systems to have only one possible start mode. Also, let $\text{pred}(m) \stackrel{\text{def}}{=} \{m' \mid (m', m) \in E(G)\}$ and $\text{succ}(m) \stackrel{\text{def}}{=} \{m' \mid (m, m') \in E(G)\}$.

Another aspect of the semantics that must be revisited is *when* a system should switch between criticality modes. In Section 2 we stated, in common with previous work on mixed-criticality scheduling, that a system switches to a higher criticality mode if some job has executed for its entire execution-time budget without signaling completion, i.e., if a job behaves in a manner that is not valid in the current criticality mode. Similarly, the generalized task model requires that a system must switch to a new criticality mode if any job or task fails to behave in a valid manner for the current mode.[4] However, while the system must switch modes in such a situation, it is also allowed to switch to a new criticality mode at any other point in time, for whatever reason the system designer deems relevant, e.g., because of hardware malfunctions or changes in the system's environment. In fact, the analysis presented so far for the standard mixed-criticality model is already safe in the face of such arbitrary mode switches, because nowhere does it assume that some job has depleted its execution-time budget at the time point where a mode switch occurs. There are no restrictions on how long the system stays in any particular criticality mode before some event triggers a mode switch; it may stay there indefinitely or move on to a new mode immediately.

[4] If the behavior is not valid in any criticality mode that the system can switch to either, the system is considered erroneous.

For the remainder of this paper we make one simplifying assumption to the above model: If $m, m' \in \mathfrak{L}_i$ and $(m, m') \in E(G)$, then it makes no sense to have $D_i(m) > D_i(m')$ because any job of τ_i has to be finished within $D_i(m')$ of its release also in mode m, or it would have already missed its deadline in case the system switches to m' after that time. We therefore assume that $D_i(m) \leqslant D_i(m')$ if $m, m' \in \mathfrak{L}_i$ and $(m, m') \in E(G)$. We refer to this as the *non-decreasing deadline invariant*.[5]

6 Extending the Schedulability Analysis to the Generalized Task Model

The schedulability analysis in Section 3 must be adapted to the generalized task model. This is mainly done by generalizing the demand bound functions presented previously.

Let $\mathrm{dbf}_{m,m'}(\tau_i, \ell)$ denote a demand bound function of task τ_i for a time-interval length ℓ, when the system is currently in criticality mode m' and was in criticality mode m before that. If there was no previous mode to m', i.e., if $m' \in \mathrm{roots}(G)$, the demand bound function is instead denoted $\mathrm{dbf}_{\perp,m'}(\tau_i, \ell)$. To avoid naming collisions, we assume that no criticality mode is ever denoted with the symbol $\perp$. The reason demand bound functions must be formulated with both a current and a previous criticality mode in mind is that we must know if and how a task can have carry-over jobs from the previous mode.

Note that a demand bound function $\mathrm{dbf}_{m,m'}(\tau_i, \ell)$, as defined above, must always provide a safe upper bound on the demand of τ_i in m' when reached from m, for *any* possible time interval of length ℓ. In particular, it must provide a safe bound no matter how the system earlier reached mode m, and no matter how long the system stayed in m before switching to m'. A $\mathrm{dbf}_{m,m'}(\tau_i, \ell)$ is therefore an abstraction of all concrete system traces where m' is reached from m.

With the above notation for demand bound functions, Proposition 2 has a natural extension:

Proposition 3 *A (generalized) mixed-criticality task set τ with criticality-mode structure G is schedulable by EDF if the following holds for all $m \in V(G)$:*

$$\textbf{\textit{Condition}}\ \mathsf{S}(m)\textbf{\textit{:}}\quad \forall m' \in P(m) : \quad \forall \ell \geqslant 0 : \sum_{\tau_i \in \tau} \mathrm{dbf}_{m',m}(\tau_i, \ell) \leqslant \mathrm{sbf}_m(\ell),$$

where

$$P(m) = \begin{cases} \mathrm{pred}(m), & \textit{if } \mathrm{pred}(m) \neq \emptyset, \\ \{\perp\}, & \textit{otherwise}, \end{cases}$$

and where the platform's supply in criticality mode m is characterized by supply bound function sbf_m.

[5] In practice, a preprocessing step can just set $D_i(m) \leftarrow D_i(m')$ if $D_i(m) > D_i(m')$ in such a case. The purpose of the non-decreasing deadline invariant is not to restrict the expressiveness of the task model, but to increase the conciseness of the schedulability analysis by removing cases that can trivially be seen to not lead to schedulability.

For each criticality mode $m \in V(G)$, Condition $\mathsf{S}(m)$ captures the schedulability of the system in that mode. Condition $\mathsf{S}(m)$ generalizes Conditions S_{LO} and S_{HI} from Proposition 2, and expresses that the system's execution demand never exceeds the available supply in mode m. If Condition $\mathsf{S}(m)$ holds, then m is schedulable when reached from all of m's possible predecessor modes in G, or as a start mode if m has no predecessors. If $\mathsf{S}(m)$ holds for all $m \in V(G)$, then all modes of the system are schedulable, no matter how they are reached, as is stated by Proposition 3.

When formulating the demand bound functions later in this section, we will make two assumptions:

1. Each supply bound function sbf_m is of at most unit speed if $\text{succ}(m) \neq \emptyset$, similarly to what was assumed of sbf_{LO} in Section 3. This is, again, simply a matter of scaling the parameters.
2. When formulating $\text{dbf}_{m,m'}$, where $m \neq \perp$, we assume that m is schedulable by EDF. This is analogous to what was done in Section 3, where the demand in HI was bounded under the assumption that LO is schedulable.

The second assumption clearly restricts the correctness of the demand bound functions to certain cases.[6] However, our purpose with the demand bound functions is to use them with Proposition 3 to show that a system is schedulable, and restricting them in this way does not invalidate their use in Proposition 3. In other words, if $\mathsf{S}(m)$ holds for all $m \in V(G)$, the system is schedulable despite the above assumptions made for the demand bound functions. To see this, consider a sequence $\mathfrak{T}$ that is any topological ordering $\langle m_0, m_1, \ldots \rangle$ of G. The assumptions made when bounding the demand in a mode m are that all of m's predecessors are schedulable. For m_0, the first mode in $\mathfrak{T}$, this is trivially true (as m_0 can have no predecessors), and we can conclude that the bounds are valid and therefore that m_0 is schedulable. If m_1, the next mode in $\mathfrak{T}$, has any predecessor in G, it must be m_0. We have already concluded that m_0 is schedulable, so any assumptions about the schedulability of m_1's predecessors are also true and m_1 is also schedulable. The same reasoning can then be applied, in order, to the remaining modes $\mathfrak{T}$ to see that they are all schedulable.

6.1 Formulating the Generalized Demand Bound Functions

There can be no carry-over jobs in any of the criticality modes in $\text{roots}(G)$ because there are no previous modes from which they can be carried over. The demand bound function $\text{dbf}_{\perp,m}$ therefore does not take carry-over jobs into account, and can be based on the standard demand bound function for sporadic tasks (Baruah et al. 1990), just like dbf_{LO} in (1). The only difference is that $\text{dbf}_{\perp,m}$ is defined to be equal to 0 for tasks that are not active in m.

$$\text{dbf}_{\perp,m}(\tau_i, \ell) \stackrel{\text{def}}{=} \begin{cases} \left[\!\left[\left(\left\lfloor \frac{\ell - D_i(m)}{T_i(m)} \right\rfloor + 1 \right) \cdot C_i(m) \right]\!\right]_0, & \text{if } m \in \mathfrak{L}_i \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

[6] This is not an issue that can be avoided. Some knowledge about a system's behavior prior to entering a new criticality mode must be assumed in order to provide any usable bound at all.

Similarly, dbf_{HI} from (4) can be used as the basis for the new $\text{dbf}_{m,m'}$, as it captures the demand in modes that can be switched *to* and must consider carry-over jobs for tasks that are active in both m and m'. In the same manner as dbf_{HI}, the function $\text{dbf}_{m,m'}$ provides a safe upper bound on the demand in m' under the assumption that m is schedulable. Note that as $D_i(m) \leqslant T_i(m)$ for all $\tau_i \in \tau$ and $m \in \mathfrak{L}_i$, there can be at most one active job per task at any time point (as long as no deadline is missed). This holds also at the time of a mode switch, and so we need to consider at most one carry-over job per demand bound function, like before with dbf_{HI}. Recall that dbf_{HI} was built from the two functions full_{HI} and done_{HI}, from (2) and (3), respectively. We start by generalizing these two auxiliary functions.

The challenge in extending full_{HI} and done_{HI} to the generalized task model lies in dealing with the fact that all of a task's worst-case execution time, relative deadline and period may change between m and m'. However, the actual changes needed to the functions are few. The execution-time parameter could change between criticality modes already in the standard model, although it could only increase in the new mode, and changing relative deadlines were already introduced as a technique for scheduling and analysis. The new aspects that must be considered are therefore only the possibilities for a task to get a decreased worst-case execution time, and a decreased or increased period. It turns out that changing the period parameter when switching to a new mode does not complicate the demand bound functions at all, as it does not affect the way we calculate demand for carry-over jobs or for jobs released in the new mode. Fig. 8 illustrates this. The only new thing that needs to be handled is then the case where the execution-time parameter decreases.

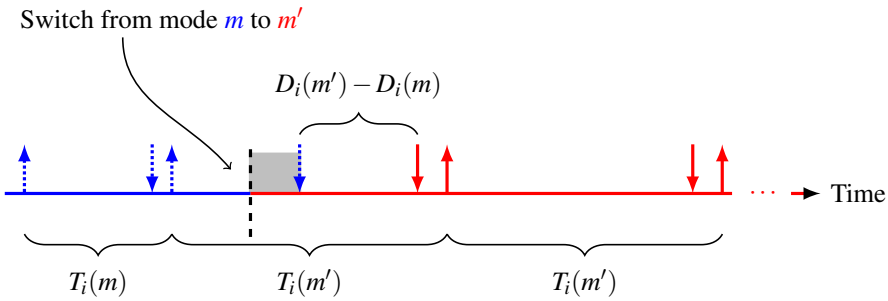

Fig. 8 The carry-over job is unaffected by the fact that the period of τ_i was changed at the switch from m to m'. The minimum (remaining) scheduling window of a carry-over job is still the difference between the relative deadlines in the new and old mode, just as before with the standard model. As before, the remaining execution time budget for the job in the old mode m can be no larger the length of the interval between switch and deadline in m (the shaded interval).

The function $\text{full}_{m,m'}(\tau_i, \ell)$ captures the demand of τ_i in the new mode m' without considering that carry-over jobs can be partly executed in m, i.e., it counts a full $C_i(m')$ also for carry-over jobs. For this it does not matter what the execution-time parameter was in m, so the function only needs to be updated to use the relevant parameters of the generalized model:

$$\mathrm{full}_{m,m'}(\tau_i,\ell) \stackrel{\text{def}}{=} \left[\!\!\left[\left(\left\lfloor \frac{\ell - (D_i(m') - D_i(m))}{T_i(m')} \right\rfloor + 1 \right) \cdot C_i(m') \right]\!\!\right]_0 \tag{6}$$

The function $\mathrm{done}_{m,m'}(\tau_i,\ell)$ quantifies the amount of work that must have been finished before switching to m' for any carry-over job included in $\mathrm{full}_{m,m'}(\tau_i,\ell)$. If $C_i(m) > C_i(m')$, it is possible that the amount of work already finished before m' exceeds $C_i(m')$. In such a case the carry-over job's execution-time budget in m' is considered to be completely spent, but not exceeded (as per the semantics in Section 5.1), and we can only subtract $C_i(m')$ from $\mathrm{full}_{m,m'}(\tau_i,\ell)$. This is achieved by simply bounding the value of $\mathrm{done}_{m,m'}(\tau_i,\ell)$ by $C_i(m')$ from above, otherwise it is a direct adaptation of $\mathrm{done}_{\mathrm{HI}}$.

$$\mathrm{done}_{m,m'}(\tau_i,\ell) \stackrel{\text{def}}{=} \begin{cases} [\![C_i(m) - x + D_i(m') - D_i(m)]\!]_0^{C_i(m')}, \\ \qquad\qquad \text{if } D_i(m') > x \geqslant D_i(m') - D_i(m) \\ 0, \qquad \text{otherwise,} \end{cases} \tag{7}$$

where $x = \ell \bmod T_i(m')$.

With these auxiliary functions generalized we can formulate $\mathrm{dbf}_{m,m'}$. For a task that is not active in m, the new mode m' is equivalent to a start mode in which the task may be activated, and its demand can be captured with $\mathrm{dbf}_{\perp,m'}$. For a task that is active in both modes, the demand is given by the difference of $\mathrm{full}_{m,m'}$ and $\mathrm{done}_{m,m'}$.

$$\mathrm{dbf}_{m,m'}(\tau_i,\ell) \stackrel{\text{def}}{=} \begin{cases} \mathrm{full}_{m,m'}(\tau_i,\ell) - \mathrm{done}_{m,m'}(\tau_i,\ell), & \text{if } m,m' \in \mathfrak{L}_i \\ \mathrm{dbf}_{\perp,m'}(\tau_i,\ell), & \text{otherwise} \end{cases} \tag{8}$$

Note that if a standard two-level mixed-criticality task set is described in the generalized syntax, we have $\mathrm{dbf}_{\mathrm{LO}} = \mathrm{dbf}_{\perp,\mathrm{LO}}$ and $\mathrm{dbf}_{\mathrm{HI}} = \mathrm{dbf}_{\mathrm{LO,HI}}$ as expected. Also note that the $\mathrm{dbf}_{m,m'}$ presented above depends only on the parameters of jobs in m and m', and not on any execution history except for the assumption of schedulability in m. It is indifferent to the origin mode of carry-over jobs, i.e., if they are released in m or in an earlier mode and carried over via m to m'.

7 Tuning Parameters for the Generalized Task Model

In Section 4 we showed how it is possible to shape the demand of a task set by tuning the relative deadlines of tasks. For the standard task model, the demand of carry-over jobs in HI was reduced by decreasing the corresponding relative deadlines in LO. A similar approach can be used to shape the demand of a generalized task system (τ, G): If $(m,m') \in E(G)$ and $m,m' \in \mathfrak{L}_i$, the demand of carry-over jobs of τ_i in mode m', when reached from m, can be reduced by decreasing $D_i(m)$.

Lemma 2 provided insights about the effects of tuning relative deadlines for the standard task model. It can be extended to the generalized model:

Lemma 3 (Generalized shifting) *If tasks τ_i and τ_j are active in mode m and are identical (i.e., have equal parameters in all modes and $\mathfrak{L}_i = \mathfrak{L}_j$), with the exception that $D_i(m) = D_j(m) - \delta$ for $\delta \in \mathbb{Z}$, then*

$$\forall m' \in P(m,\tau_i): \quad \mathrm{dbf}_{m',m}(\tau_i,\ell) = \mathrm{dbf}_{m',m}(\tau_j,\ell+\delta),$$
$$\forall m' \in S(m,\tau_i): \quad \mathrm{dbf}_{m,m'}(\tau_i,\ell) = \mathrm{dbf}_{m,m'}(\tau_j,\ell-\delta),$$

where $P(m,\tau_i) = (\mathrm{pred}(m) \cap \mathfrak{L}_i) \cup \{\bot\}$ and $S(m,\tau_i) = \mathrm{succ}(m) \cap \mathfrak{L}_i$.

A proof of the above lemma is similar to that of Lemma 2, and is therefore omitted. Note that the correctness of our schedulability analysis or demand shaping does not depend on the correctness of this lemma, it serves only as an illustration of the effects of tuning relative deadlines.

The process of finding suitable values for the relative deadlines is more challenging for the generalized model due to the presence of arbitrarily many criticality modes and their non-linear structure. Our approach to the tuning is to first adapt Algorithm 1, which tunes the relative deadlines in one mode (LO), into the slightly more general Algorithm 3 (TuneMode(m)), which tunes the deadlines of any mode m. TuneMode(m) tries to lower the deadlines in m until all modes in succ(m) are schedulable when reached from m. Algorithm 2 (TuneSystem(τ, G)) applies Algorithm 3 on all the modes of the criticality-mode structure G, starting with the terminal nodes and proceeding in a reverse topological order until it has successfully tuned all modes or failed in tuning some mode.

Algorithm 2: TuneSystem(τ, G)

```
1 begin
2     for m ∈ V(G), sorted in reverse topological order do
3         if TuneMode(m) = FAILURE then return FAILURE
4     return SUCCESS
```

TuneMode(m) decreases the relative deadlines of tasks in m to reduce the demand of those tasks in m's successor modes. Unless $m \in \mathrm{roots}(G)$, it does this without considering of the schedulability in m itself, hoping that m can later be made schedulable by decreasing deadlines in its predecessor modes. TuneMode(m) will undo changes to deadlines that would make m unschedulable only if $m \in \mathrm{roots}(G)$.

Note that TuneMode(m) will only return the value SUCCESS if it has found values for the relative deadlines in m with which all $m' \in \mathrm{succ}(m)$ are schedulable when reached from m (assuming that m itself is ultimately made schedulable). In addition, if $m \in \mathrm{roots}(G)$, it will only return SUCCESS if m is also schedulable as a start mode. It is enough to check all integer $\ell \leqslant \ell_{\max}$ to determine schedulability by the reasoning given in Section 4.1. A value for the bound $\ell_{\max}$ can be computed as $\max\{\ell^{m'}_{\max} \mid m' \in \mathrm{succ}(m) \cup \{m\}\}$, where $\ell^{m}_{\max}$ is easily defined similarly to $\ell^{\mathrm{LO}}_{\max}$ and $\ell^{\mathrm{HI}}_{\max}$ from Section 4.1 when applicable, or as the hyperperiod in m. If $\ell_{\max}$ is pseudo-polynomial, then so is the complexity of Algorithm 3. Algorithm 2 makes

$|V(G)|$ calls to Algorithm 3, and therefore scales linearly in the number of criticality modes.

Algorithm 3: TuneMode(m)

```
1  begin
2      if succ(m) = ∅ ∧ m ∉ roots(G) then return SUCCESS
3      candidates ← {i | m ∈ 𝔏_i ∧ 𝔏_i ∩ succ(m) ≠ ∅}
4      mods ← empty stack
5      ℓ_max ← upper bound for ℓ
6      for i ∈ candidates do
7          D_i(m) ← min{D_i(m') | m' ∈ (succ(m) ∩ 𝔏_i) ∪ {m}}
8          if D_i(m) = C_i(m) then candidates ← candidates \ {i}
9          if D_i(m) < C_i(m) then return FAILURE
10     repeat
11         changed ← false
12         for ℓ = 0, 1, ..., ℓ_max do
13             if m ∈ roots(G) then
14                 if Σ_{τ_i∈τ} dbf_{⊥,m}(τ_i, ℓ) > sbf_m(ℓ) then
15                     if mods is empty then return FAILURE
16                     i ← mods.pop()
17                     D_i(m) ← D_i(m) + 1
18                     candidates ← candidates \ {i}
19                     changed ← true
20                     break
21             for m' ∈ succ(m) do
22                 if Σ_{τ_i∈τ} dbf_{m,m'}(τ_i, ℓ) > sbf_{m'}(ℓ) then
23                     c ← candidates ∩ {i | m' ∈ 𝔏_i}
24                     if c = ∅ then return FAILURE
25                     i ← arg max_{j∈c}(dbf_{m,m'}(τ_j, ℓ) − dbf_{m,m'}(τ_j, ℓ − 1))
26                     D_i(m) ← D_i(m) − 1
27                     mods.push(i)
28                     if D_i(m) = C_i(m) then candidates ← candidates \ {i}
29                     changed ← true
30                     break
31             if changed then break
32     until ¬changed
33     return SUCCESS
```

Lines 6-9 and 28 in Algorithm 3 ensure that $C_i(m) \leqslant D_i(m)$ and the non-decreasing deadline invariant hold. In contrast to Algorithm 1, which can only undo the latest deadline change if it harms the schedulability in LO, Algorithm 3 stores all deadline

changes in a stack (*mods*) and can undo all of them in an effort to restore schedulability of a start mode. This does not affect the scalability much, as each deadline parameter $D_i(m)$ can only be changed $2 \cdot (D - C_i(m))$ times at most, where $D = D_i(m)$ at the start of Algorithm 3. It is of course possible to define Algorithm 3 without the stack (or Algorithm 1 with it), but we think that it is motivated to be able to undo more changes in Algorithm 3 as there can be several successor modes incurring them.

Because Algorithm 2 traverses the DAG in a reverse topological order, we know that all successor modes of modes tuned so far will be schedulable as long as we can successfully tune the remaining modes. Example 5 illustrates this process.

Example 5 *Consider a task set τ with a criticality-mode structure G. Let $V(G) = \{m_0, m_1, m_2, m_3\}$ and $E(G) = \{(m_0, m_1), (m_0, m_2), (m_1, m_3), (m_2, m_3)\}$, resulting in a shape similar to the one in Example 4. A reverse topological ordering of the criticality modes is $\langle m_3, m_1, m_2, m_0 \rangle$.* TuneSystem$(\tau, G)$ *would first call* TuneMode(m_3)*, which does nothing as m_3 has no successors, and then continue with m_1, m_2, m_0 in order. As shown in Fig. 9,* TuneMode(m_1) *and* TuneMode(m_2) *will decrease some relative deadlines in m_1 and m_2, as needed to make m_3 schedulable when reached from these modes.* TuneMode(m_0) *will decrease deadlines in m_0 to make both m_1 and m_2 schedulable (using the lowered deadlines previously set in* TuneMode(m_1) *and* TuneMode(m_2)*). Because $m_0 \in$ roots(G),* TuneMode(m_0) *will also make sure that m_0 is schedulable as a start mode, or return* FAILURE.

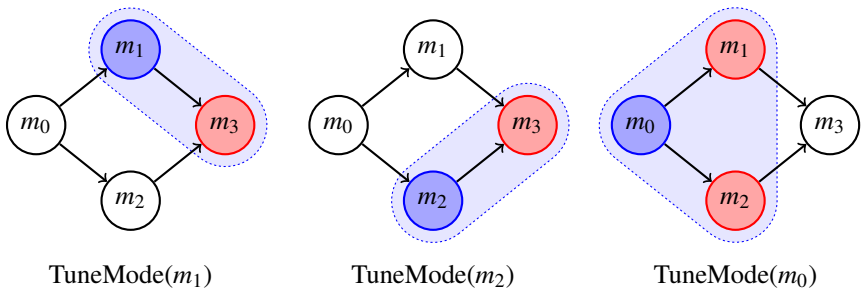

Fig. 9 The order in which modes are tuned by TuneSystem(τ, G), excluding the trivial TuneMode(m_3)

8 Experimental Evaluation

In this section we evaluate the effectiveness of characterizing mixed-criticality task sets using the demand bound functions formulated earlier. In order to make the evaluation meaningful, we compare to previous approaches to mixed-criticality scheduling from the literature and study the acceptance ratios of their corresponding schedulability tests. The previous work only supports dedicated platforms, and therefore we use that setting for the experiments. Most of the previous work assumes the standard two-level mixed-criticality sporadic task model described in Section 2, so we start by considering such task sets.

8.1 Evaluation Using the Standard Two-Level Task Model

We compare the following approaches:

GreedyTuning: The test in Proposition 2 using the demand bound functions in Equations (1) and (4). Relative deadlines are tuned using Algorithm 1.

OCBP-prio: The test for OCBP-based scheduling by Guan et al. (2011), which is based on whether a priority ordering can be found for all jobs in a busy period. This test has been shown to dominate the test for OCBP-based scheduling by Li and Baruah (2010), and is therefore the only test for OCBP included.

EDF-VD: The test for the EDF-VD scheduling algorithm by Baruah et al. (2012).[7]

AMC-max: A test based on the most powerful response-time calculation for fixed-priority scheduling by Baruah et al. (2011b), called AMC-max. Priorities are assigned using Audsley's algorithm, as suggested by Baruah et al. (2011b).

Vestal: A test based on the response-time calculation for fixed-priority scheduling by Vestal (2007) combined with Audsley's algorithm. Because we assume that low-criticality tasks are discarded in high-criticality mode, the budgets of low-criticality task's execution times are implicitly enforced. This is therefore equivalent to the algorithm SMC by Baruah et al. (2011b).

Naive: A test based on simply flattening the mixed-criticality sporadic task set into a normal sporadic task set using resource reservation, and seeing whether the constructed task set is schedulable. In the case of implicit deadline tasks this is done by simply checking if the utilization of the constructed task set is at most 1. If deadlines are not implicit, the exact test by Baruah et al. (1990) is used instead. Each mixed-criticality task $\tau_i \in \tau$ is mapped to a normal sporadic task with worst-case execution time $C_i(L_i)$, deadline D_i and period T_i. This simple test is included as a baseline for the more sophisticated approaches.

Necessary: This is not a schedulability test but an over-approximation of task set feasibility. A task set τ passes the necessary test if both of the normal sporadic task sets $\{(C_i(\text{LO}), D_i, T_i) \mid \tau_i \in \tau\}$ and $\{(C_i(\text{HI}), D_i, T_i) \mid \tau_i \in \text{HI}(\tau)\}$ are schedulable according to the exact analysis by Baruah et al. (1990). This test is helpful in providing a bound on how much improvement upon existing solutions we can ever hope to achieve. It is not clear, of course, how close the upper bound provided by this test is to true feasibility. It follows from the work of Baruah et al. (2011c) that an exact feasibility test, should one be developed, is NP-hard in the strong sense. We only use the necessary test if deadlines are not implicit, as all implicit-deadline task sets are guaranteed to pass it.

8.1.1 Task Set Generation

A random task set is generated by starting with an empty task set $\tau = \emptyset$, which random tasks are successively added to. The generation of a random task is controlled by five parameters: the probability P_{HI} of being of high-criticality; the maximum ratio R_{C} between high- and low-criticality execution time; the maximum low-criticality

[7] A similar evaluation performed in a preliminary version of this paper (Ekberg and Yi 2012) used the original test for EDF-VD (Baruah et al. 2011a). Here we use the improved test (Baruah et al. 2012).

execution time $C_{\text{LO}}^{\max}$; the maximum period $T^{\max}$; and the ratio R_{D} giving the range of possible relative deadline values. Each new task τ_i is then generated as follows:

- $L_i = \text{HI}$ with probability P_{HI}, otherwise $L_i = \text{LO}$.
- $C_i(\text{LO})$ is drawn from the uniform distribution over $\{1, \ldots, C_{\text{LO}}^{\max}\}$.
- $C_i(\text{HI})$ is drawn from the uniform distribution over $\{C_i(\text{LO}), \ldots, R_{\text{C}} \cdot C_i(\text{LO})\}$ if $L_i = \text{HI}$, otherwise, $C_i(\text{HI}) = C_i(\text{LO})$.
- T_i is drawn from the uniform distribution over $\{C_i(L_i), \ldots, T^{\max}\}$.
- D_i is drawn from the uniform distribution over $\{D^{\min}, \ldots, T_i\}$, where we have $D^{\min} = \lfloor C_i(L_i) + R_{\text{D}} \cdot (T_i - C_i(L_i)) \rfloor$.

We define the *average utilization* $U_{\text{avg}}(\tau)$ of a mixed-criticality task set τ as

$$U_{\text{avg}}(\tau) \stackrel{\text{def}}{=} \frac{U_{\text{LO}}(\tau) + U_{\text{HI}}(\tau)}{2}.$$

Each task set is generated with a *target* average utilization U^* in mind. Due to the difficulty of getting an exact utilization with random integer parameter tasks, we allow the task set's average utilization to fall within the small interval between $U_{\min}^* \stackrel{\text{def}}{=} U^* - 0.005$ and $U_{\max}^* \stackrel{\text{def}}{=} U^* + 0.005$.

As long as $U_{\text{avg}}(\tau) < U_{\min}^*$, we generate more tasks and add them to τ. If a task is added such that $U_{\text{avg}}(\tau) > U_{\max}^*$, we discard the whole task set and start with a new empty task set. If a task is added such that $U_{\min}^* \leqslant U_{\text{avg}}(\tau) \leqslant U_{\max}^*$, the task set is finished, unless all tasks in τ have the same criticality level or $U_{\text{LO}}(\tau), U_{\text{HI}}(\tau) > 0.99$, in which case the task set is instead discarded.[8]

8.1.2 Results

Fig. 10 shows the acceptance ratio (fraction of schedulable task sets) as a function of (target) average utilization for task sets generated with $P_{\text{HI}} = 0.5$, $R_{\text{C}} = 4$, $C_{\text{LO}}^{\max} = 10$, $T^{\max} = 200$ and $R_{\text{D}} = 1$ (resulting in implicit-deadline tasks). Each data point is based on 10,000 randomly generated task sets.

Next we study the effects of varying the parameters P_{HI}, R_{C} and R_{D}. We plot the *weighted acceptance ratio* (called the weighted schedulability measure by Bastoni et al. (2010)) as a function of the varied parameter. If $A(U)$ is the acceptance ratio for (target) average utilization U, then the weighted acceptance ratio of a set of target utilizations $\mathfrak{U}$ is

$$A(\mathfrak{U}) \stackrel{\text{def}}{=} \frac{\sum_{U \in \mathfrak{U}} U \cdot A(U)}{\sum_{U \in \mathfrak{U}} U}.$$

Using the weighted acceptance ratio we can reduce the number of dimensions in the plots by one. Note that this measure gives more importance to the acceptance ratios for larger utilization values, as these are the cases we are generally interested in.

In Fig. 11, 12 and 13 we have plotted the weighted acceptance ratio as a function of P_{HI}, R_{C} and R_{D}, respectively. The set $\mathfrak{U}$ of average utilization values are the same

[8] The reason we use a maximum utilization of 0.99 instead of 1 is to be able to use a pseudo-polynomial upper bound $\ell_{\max}$ for Algorithm 1 instead of the hyperperiod, as described in Section 4.1. This speeds up the run-time of the experiments significantly, which is needed as several million task sets are analyzed in total.

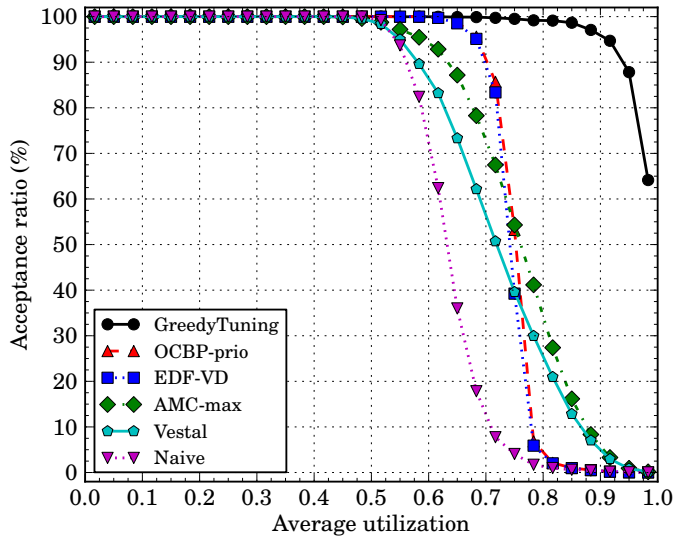

Fig. 10 $P_{\mathrm{HI}} = 0.5$, $R_{\mathrm{C}} = 4$, $C_{\mathrm{LO}}^{\max} = 10$, $T^{\max} = 200$ and $R_{\mathrm{D}} = 1$

30 values as used in Fig. 10 ($\mathfrak{U} = \{(1/30) \cdot (x + 1/2) \mid x \in \{0, \ldots, 29\}\}$). Except for the varied parameter, the parameters are also the same as for Fig. 10 to allow easy comparisons. Each data point is based on 30,000 random task sets (1000 per target utilization in $\mathfrak{U}$). Note that in Fig. 13 we have omitted EDF-VD, as it only supports implicit deadline tasks. We have instead added the Necessary test to provide an over-approximation on feasibility.

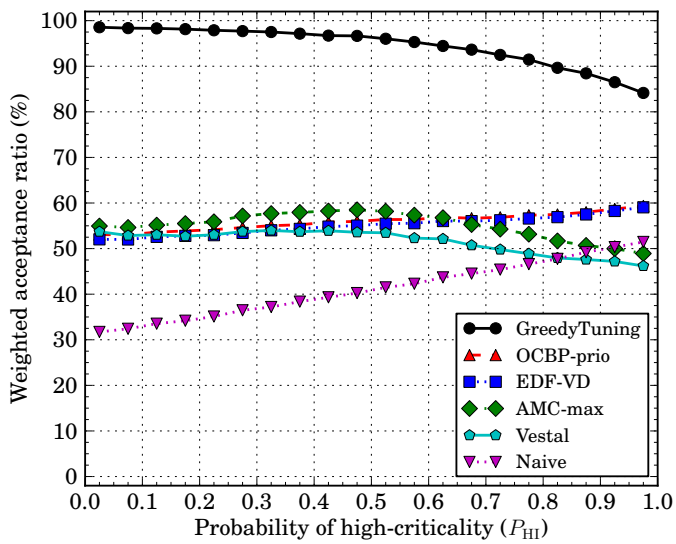

Fig. 11 P_{HI} varying, $R_{\mathrm{C}} = 4$, $C_{\mathrm{LO}}^{\max} = 10$, $T^{\max} = 200$ and $R_{\mathrm{D}} = 1$

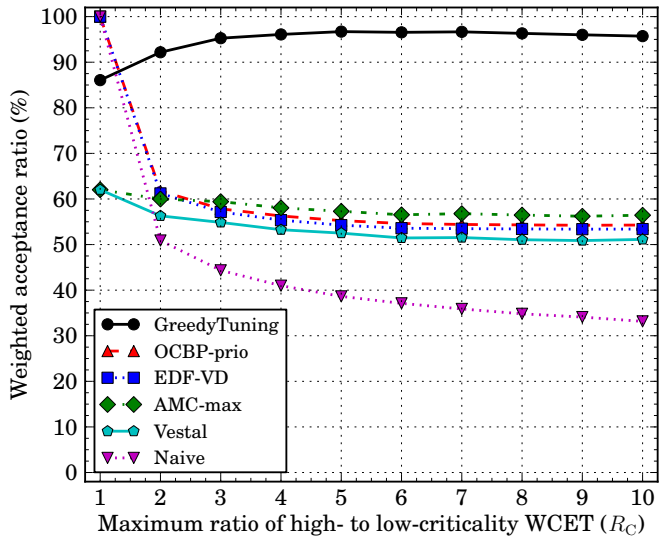

Fig. 12 $P_{HI} = 0.5$, R_C varying, $C_{LO}^{max} = 10$, $T^{max} = 200$ and $R_D = 1$

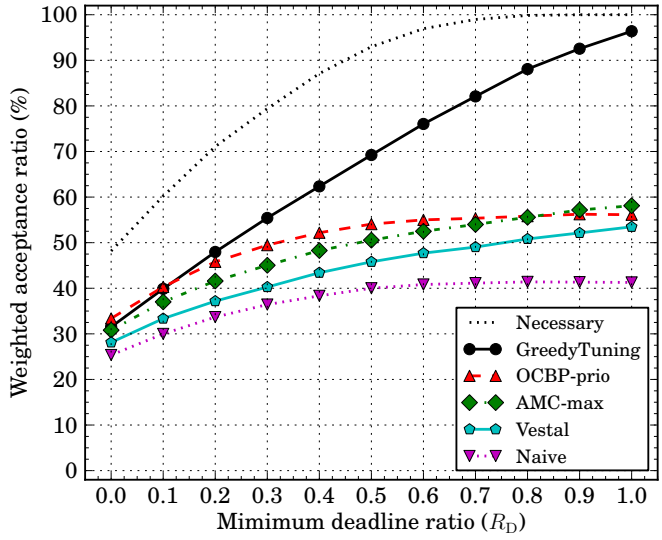

Fig. 13 $P_{HI} = 0.5$, $R_C = 4$, $C_{LO}^{max} = 10$, $T^{max} = 200$ and R_D varying

8.1.3 Discussion

Evidently, for the standard task model there is often a large gap between the acceptance ratios of the proposed approach in this paper and those of previous approaches. Moreover, this gap remains when varying the fraction of high-criticality tasks (P_{HI}) or the ratio between low- and high-criticality worst-case execution times (R_C). However,

the differences between acceptance ratios decrease when relative deadlines become shorter (R_{D} gets smaller). When relative deadlines can range from being no larger than worst-case execution times ($R_{\mathrm{D}} = 0$) all compared approaches have quite similar (weighted) acceptance ratios. The results can of course differ from those presented if we vary other parameters or the task set generation procedure, but the typical gap between acceptance ratios is large enough that we think it is safe to say that the proposed approach in this paper marks a significant improvement in the scheduling of standard mixed-criticality sporadic task sets with two criticality levels.

Among previous approaches, OCBP-prio (Guan et al. 2011), AMC-max (Baruah et al. 2011b) and EDF-VD (Baruah et al. 2012) seem to perform best. Of these, AMC-max and EDF-VD are probably the best choices in practice as they have a significantly lower run-time overhead than OCBP-based scheduling. The run-time overhead of our approach is also low because it is basically just plain EDF (potentially with a change of deadlines at single point in time), the same as EDF-VD.

The weighted acceptance ratios of all approaches remain relatively steady when varying P_{HI} and R_{C}. The reasons for the slow trends that can be seen in Fig. 11 and 12 remain mostly unclear to us. An exception is $R_{\mathrm{C}} = 1$, with which worst-case execution times do not differ between low- and high-criticality modes. Such a task set is actually equivalent to a non-mixed-criticality task set, which is why the baseline (Naive) and EDF-VD approaches have 100% acceptance ratios (both reduce in this case to checking whether the utilization is at most 1). When R_{D} decreases, the acceptance ratios of all compared approaches decrease as well, as seen in Fig. 13. This is to be expected as the timing constraints become tougher with smaller deadlines. The approach proposed in this paper sees a more marked decrease in acceptance ratio than the others, we think this is explained by its heavy dependency on being able to shift demand between criticality modes by tuning the relative deadline parameters. When these become smaller, less demand can be shifted. However, it should be noted that the upper bound on feasibility provided by the necessary test sees a decrease in acceptance ratio that is almost as sharp; it may be that the room for improvement in this regard is fairly limited.

8.2 Evaluation Using the Pessimistic Frequency Specification Model

Baruah (2012) introduced a system model that is similar to the standard mixed-criticality model, but where the periods of tasks, instead of their execution-time budgets, can change between the criticality modes LO and HI. Any such task set can be expressed as an instance (τ, G) of the generalized mixed-criticality model described in Section 5, where $V(G) = \{\mathrm{LO}, \mathrm{HI}\}$, $E(G) = \{(\mathrm{LO}, \mathrm{HI})\}$ and for each task $\tau_i \in \tau$:

- $\mathfrak{L}_i = \{\mathrm{LO}\}$ or $\mathfrak{L}_i = \{\mathrm{LO}, \mathrm{HI}\}$.
- If $\mathrm{HI} \in \mathfrak{L}_i$, then
  - $C_i(\mathrm{HI}) = C_i(\mathrm{LO})$,
  - $T_i(\mathrm{HI}) \leqslant T_i(\mathrm{LO})$,
  - $D_i(\mathrm{LO}) = D_i(\mathrm{HI}) = T_i(\mathrm{HI})$.
- Otherwise, $D_i(\mathrm{LO}) = T_i(\mathrm{LO})$.

The following approaches are compared:

TuneSystem: The test in Proposition 3 using the demand bound functions in Equations (5) and (8). Relative deadlines are tuned using Algorithm 2.

Baruah: The test by Baruah (2012) based on finding a smallest parameter $x \in (0,1)$ satisfying certain requirements, and then abstracting the workload in each criticality mode with normal sporadic tasks, specially constructed using x. It is suggested by Baruah (2012) that a smallest x satisfying the requirements can be found using bisection search. We did this and made sure that the precision was at least 0.001. We found that further increasing the precision did not noticeably affect the results.

Naive: A baseline test based on flattening the mixed-criticality sporadic task set into a normal sporadic task set using resource reservation, and then checking whether the utilization of the constructed task set is at most 1. Each mixed-criticality task $\tau_i \in \tau$ is mapped to a normal implicit-deadline sporadic task with worst-case execution time $C_i(\text{LO})$ and period $T_i(\text{HI})$ if $\text{HI} \in \mathfrak{L}_i$ and period $T_i(\text{LO})$ otherwise.

8.2.1 Task Set Generation

The generation of random task sets is controlled by four parameters: the probability P_{HI} of high criticality; the maximum execution time $C^{\max}$; the maximum period $T^{\max}$; and the minimum ratio R_{T} between the periods in high- and low-criticality mode. Each random task is generated as follows.

- $\mathfrak{L}_i = \{\text{LO}, \text{HI}\}$ with probability P_{HI}, otherwise $\mathfrak{L}_i = \{\text{LO}\}$.
- $C_i(\text{LO})$ is drawn from the uniform distribution over $\{1, \ldots, C^{\max}\}$.
- $T_i(\text{LO})$ is drawn from the uniform distribution over $\{C_i(\text{LO}), \ldots, T^{\max}\}$.
- If $\text{HI} \in \mathfrak{L}_i$, then
  - $C_i(\text{HI}) = C_i(\text{LO})$,
  - $T_i(\text{HI})$ is drawn from the uniform distribution over $\{T^{\min}, \ldots, T_i(\text{LO})\}$, where $T^{\min} = \max(C_i(\text{HI}), \lceil R_{\text{T}} \cdot T_i(\text{LO}) \rceil)$,
  - $D_i(\text{LO}) = D_i(\text{HI}) = T_i(\text{HI})$.

  If $\text{HI} \notin \mathfrak{L}_i$, then $D_i(\text{LO}) = T_i(\text{LO})$.

Average utilization for the generalized task model is defined as

$$U_{\text{avg}}(\tau) \stackrel{\text{def}}{=} \frac{\sum_{m \in V(G)} U_m(\tau)}{|V(G)|}.$$

The task sets are then generated with a target average utilization U^* in mind, in the same manner as in Section 8.1.1. Only task sets τ with $U^* - 0.005 \leqslant U_{\text{avg}}(\tau) \leqslant U^* + 0.005$ and with $\forall m \in V(G) : U_m(\tau) \leqslant 0.99$ are kept.

8.2.2 Results

Fig. 14 shows the acceptance ratios as a function of target average utilization for task sets generated with $P_{\text{HI}} = 0.5$, $R_{\text{T}} = 0.25$, $C^{\max} = 10$ and $T^{\max} = 200$. These values

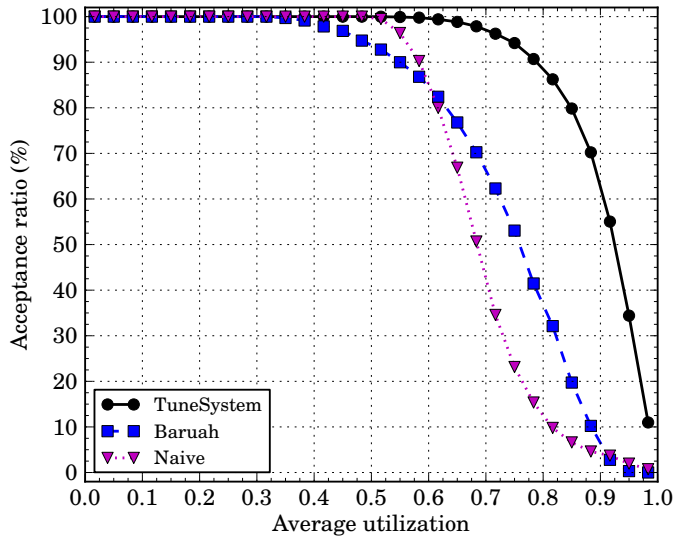

Fig. 14 $P_{\text{HI}} = 0.5$, $R_{\text{T}} = 0.25$, $C^{\max} = 10$ and $T^{\max} = 200$

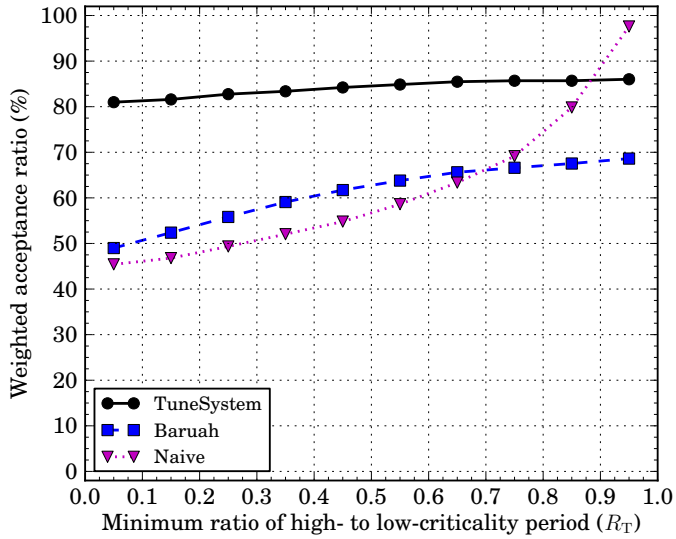

Fig. 15 $P_{\text{HI}} = 0.5$, R_{T} varying, $C^{\max} = 10$ and $T^{\max} = 200$

were chosen to mirror those used for Fig. 10, with the difference that here the periods in HI may be up to four times shorter, instead of execution times being up to four times longer. Each data point is based on 10,000 random task sets.

Next we study the effects of varying the value of R_{T}. Fig. 15 shows the weighted acceptance ratio as a function of R_{T}. The set of target utilizations used for the weighted acceptance ratio is $\mathfrak{U} = \{(1/30) \cdot (x + 1/2) \mid x \in \{0, \ldots, 29\}\}$. Each data point is based on 30,000 task sets.

8.2.3 Discussion

From Figs. 14 and 15 it seems that the demand bound functions formulated in this paper, combined with the heuristic parameter tuning, provides a scheduling approach that mostly performs better than the currently known alternatives. However, the acceptance ratio curve of our approach is generally lower for this task model than for the standard task model, with experiment settings as similar as possible (e.g., compare Figs. 14 and 10). We think that this is mostly due to the generally lower periods (and relative deadlines) of critical tasks in this model, which leave less room for parameter tuning, limiting the ability to shift demand from one criticality mode to another.

As seen in Fig. 15, all compared scheduling approaches improve with larger values for R_{T}. The naive approach gains rapidly as R_{T} approaches 1, reaching near perfect acceptance ratio with $R_{\mathrm{T}} = 0.95$. This is similar to the effect seen in Fig. 12, and is explained by the fact that these task sets are very close to being equivalent to normal sporadic tasks sets, and so the pessimism introduced by flattening them becomes relatively insignificant.

8.3 Evaluation Using Several Linearly Ordered Criticality Modes

Some of the scheduling approaches evaluated in 8.1 support an extension of that system model with $K \geqslant 2$ linearly ordered criticality modes. Any such task set can be expressed as an instance (τ, G) of the generalized system model in Section 5, where

- $V(G) = \{m_1, \ldots, m_K\}$,
- $E(G) = \{(m_1, m_2), \ldots, (m_{K-1}, m_K)\}$.
- For each task $\tau_i \in \tau$,
  - $\mathfrak{L}_i = \{m_1, \ldots, m_{L_i}\}$, for some $1 \leqslant L_i \leqslant K$,
  - $C_i(m_1) \leqslant \cdots \leqslant C_i(m_{L_i})$,
  - $D_i(m_1) = \cdots = D_i(m_{L_i})$,
  - $T_i(m_1) = \cdots = T_i(m_{L_i})$.

The approaches we compare are:

TuneSystem: The test in Proposition 3 using the demand bound functions in Equations (5) and (8). Relative deadlines are tuned using Algorithm 2.

OCBP-prio: The same test as we used for OCBP-prio in Section 8.1. It supports this setting as well.

Vestal: The same test as in Section 8.1 can also be used here.

Naive: A baseline test based on flattening the mixed-criticality sporadic task set into a normal sporadic task set using resource reservation, and checking whether the constructed task set is schedulable. Each mixed-criticality task $\tau_i \in \tau$ is mapped to a normal sporadic task with worst-case execution time $C_i(m_{L_i})$, deadline $D_i(m_1)$ and period $T_i(m_1)$.

8.3.1 Task Set Generation

Here the task set generation is controlled by four parameters: the number of criticality modes K; the maximum execution time $C_{m_1}^{\max}$ in the lowest criticality mode; the max-

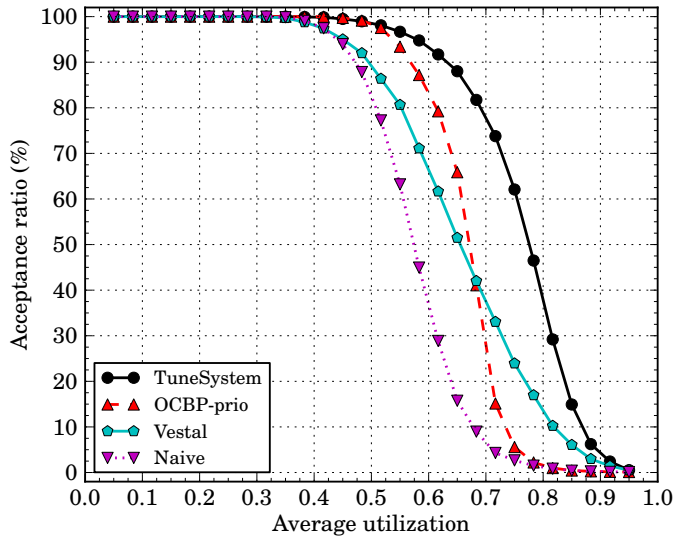

Fig. 16 $R_C = 2$, $C_{m_1}^{\max} = 10$, $T^{\max} = 200$ and $K = 3$

imum period $T^{\max}$; and the maximum ratio R_C between the execution time parameter of two consecutive criticality modes. Each task τ_i is generated as follows:

- $\mathfrak{L}_i = \{m_1, \ldots, m_{L_i}\}$ for an L_i drawn from the uniform distribution over $\{1, \ldots, K\}$,
- $C_i(m_1)$ is drawn from the uniform distribution over $\{1, \ldots, C_{m_1}^{\max}\}$,
- $C_i(m_j)$ is drawn from the uniform distribution over $\{C_i(m_{j-1}), \ldots, R_C \cdot C_i(m_{j-1})\}$ for $1 < j \leqslant L_i$,
- $D_i(m_1) = \cdots = D_i(m_{L_i}) = T_i(m_1) = \cdots = T_i(m_{L_i})$ are drawn from the uniform distribution over $\{C_i(m_{L_i}), \ldots, T^{\max}\}$,

Note that we generate tasks with implicit deadlines. This is to enable easier comparisons with previous experiments, which are mostly with implicit deadlines as well. Task sets are generated for a target average utilization in exactly the same way as in Section 8.2.1.

8.3.2 Results

Fig. 16 shows acceptance ratio as a function of average utilization for task sets generated with $K = 3$, $C_{m_1}^{\max} = 10$, $T^{\max} = 200$ and $R_C = 2$. There are 10,000 task sets per data point. Note that there are only 28 different target utilizations used, instead of the 30 used in previous experiments. We have skipped the first and the last point because it is difficult to generate random task sets fulfilling the criteria and having a very low (or very high) average utilization. If we extrapolate the acceptance ratio curves in Fig. 16, it would not seem as if the two skipped points are interesting.

The effects of having more criticality levels are shown in Fig. 17, where we vary the value of K. We have not included OCBP-prio in Fig. 17 because its schedulability

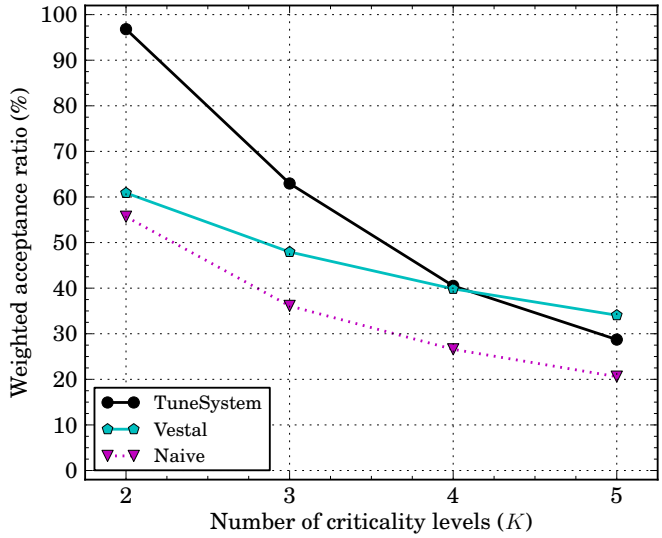

Fig. 17 $R_C = 2$, $C_{m_1}^{\max} = 10$, $T^{\max} = 200$ and K varying

test becomes prohibitively expensive for larger values of K combined with high average utilization. The set of target utilizations used for the weighted acceptance ratio is $\mathfrak{U} = \{(1/30) \cdot (x + 1/2) \mid x \in \{1, \ldots, 28\}\}$. Each data point is based on 30,000 task sets.

8.3.3 Discussion

As seen in Fig. 16, the scheduling approach taken in this paper performs better than alternative approaches also for standard mixed-criticality task sets extended to three criticality levels. The difference in performance, however, is much smaller than for the case with two levels (e.g., see Fig. 10).

Increasing the number of criticality levels further leads to quickly deteriorating performance for all evaluated approaches, as shown in Fig. 17. Vestal's fixed-priority response-time analysis deteriorates most gracefully though, and for $K > 4$ it has the best performance. This is not so surprising as the response-time analysis used for it suffers comparatively little extra pessimism for increasing K. In contrast, the scheduling approach proposed in this paper would require the tuning to produce a differentiation of the relative deadlines of a task τ_i between each adjacent criticality mode, such that if $\mathfrak{L}_i = \{m_1, \ldots, m_{L_i}\}$, we have $D_i(m_1) \leqslant \cdots \leqslant D_i(m_{L_i})$. Clearly, when a task is active in a long "chain" of criticality modes, the room available for differentiating the relative deadlines between adjacent modes becomes smaller. It may pay off to shorten such chains, if possible, when modeling the system, i.e., to make a trade-off between modeling pessimism and analysis pessimism.

9 Conclusions

We have characterized the demand of mixed-criticality sporadic tasks using demand bound functions. They are based on the observation that upper bounds on the demand of carry-over jobs can be formulated by assuming that the previous criticality mode is schedulable. As the goal of schedulability analysis in this context is to show that all criticality modes are schedulable, such assumptions are sound as long as the base cases (i.e., the starting criticality modes) can be shown to be schedulable.

We have also generalized the standard mixed-criticality sporadic task model used in most prior work. The generalized task model allows arbitrary changes of task parameters and active tasks between criticality modes. We allow the criticality modes to be expressed using a DAG, thus enabling the specification of mixed-criticality aspects in several dimensions. We would like to have a general model to cover a broad range of mixed-criticality aspects and leave the interpretations to the system designer.

For the demand bound functions that have been formulated, we have showed that demand of a task can be shifted between any adjacent criticality modes by tuning the relative deadlines of the task in different modes. This results in a constraint satisfaction problem where the challenge is to find valid values for all relative deadlines such that the whole system becomes schedulable. The tuning of parameter values can be carried out in any manner, for example with general-purpose constraint solvers. We have also presented heuristic algorithms designed specifically for this problem. Note that the values of relative deadlines only can decrease with respect to their original values, meaning that the temporal specification made by the system designer holds also after the parameter tuning.

Experimental evaluation and comparisons between this and prior work show that our approach is successful in practice for a wide range of situations. The results show that EDF-based scheduling in many cases significantly outperforms fixed-priority scheduling for mixed-criticality systems, mirroring the case for non-mixed-criticality systems. We think that this is important because it allows us to utilize the performance of EDF without sacrificing robustness in case of overloads. Often, EDF is quoted as being too unpredictable in case of overloads since it is practically impossible to predict which jobs will suffer the extra delays (see Buttazzo (2005) for a more comprehensive treatment of this subject). This is not the case for mixed-criticality systems. In a mixed-criticality system, the designer can specify exactly which tasks are more important in an overload situation. We believe that this is an appropriate separation of concerns: The system designer specifies what constitutes an overload situation, or other critical event, and which tasks must continue to function. The scheduler makes sure that the system behaves as specified while utilizing platform resources as efficiently as possible.

As future work we plan to address resource sharing or tasks with more complex job release patterns in the context of mixed-criticality systems. We believe that the techniques in this paper will generalize to these cases.

Acknowledgements This work was supported in part by the Swedish Research Council within the UPMARC Linnaeus centre of Excellence. We would like to thank the anonymous reviewers for their helpful and insightful comments, due to which the quality of this paper has been greatly improved.

References

Audsley, N. C. (2001). On priority asignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1):39–44.

Baruah, S. (2012). Certification-cognizant scheduling of tasks with pessimistic frequency specification. In *SIES*, pages 31–38.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2012). The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS*, pages 145–154.

Baruah, S., Bonifaci, V., D'Angelo, G., Marchetti-Spaccamela, A., van der Ster, S., and Stougie, L. (2011a). Mixed-criticality scheduling of sporadic task systems. In *ESA*, pages 555–566.

Baruah, S., Burns, A., and Davis, R. (2011b). Response-time analysis for mixed criticality systems. In *RTSS*, pages 34 –43.

Baruah, S., Li, H., and Stougie, L. (2010). Towards the design of certifiable mixed-criticality systems. In *RTAS*, pages 13 –22.

Baruah, S., Mok, A., and Rosier, L. (1990). Preemptively scheduling hard-real-time sporadic tasks on one processor. In *RTSS*, pages 182–190.

Baruah, K., S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L. (2011c). Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*.

Bastoni, A., Brandenburg, B., and Anderson, J. (2010). Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *OSPERT*, pages 33–44.

Buttazzo, G. C. (2005). Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1):5–26.

Ekberg, P. and Yi, W. (2012). Bounding and shaping the demand of mixed-criticality sporadic tasks. In *ECRTS*, pages 135–144.

Guan, N., Ekberg, P., Stigge, M., and Yi, W. (2011). Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *RTSS*, pages 13–23.

Li, H. and Baruah, S. (2010). An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *RTSS*, pages 183 –192.

Li, H. and Baruah, S. (2012). Global mixed-criticality scheduling on multiprocessors. In *ECRTS*, pages 166 –175.

Mok, A. (1983). Fundamental design problems of distributed systems for the hard real-time environment. Technical report, Cambridge, MA, USA.

Mok, A., Feng, X., and Chen, D. (2001). Resource partition for real-time systems. In *RTAS*, pages 75 –84.

Pathan, R. (2012). Schedulability analysis of mixed-criticality systems on multiprocessors. In *ECRTS*, pages 309 –320.

Real, J. and Crespo, A. (2004). Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197.

Shin, I. and Lee, I. (2003). Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2 – 13.

Stigge, M., Ekberg, P., Guan, N., and Yi, W. (2011). The digraph real-time task model. In *RTAS*, pages 71–80.

Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, pages 239–243.

A Proof of Lemma 2

Before proving Lemma 2, we reformulate the function $\text{done}_{\text{HI}}(\tau_i, \ell)$ from Section 3.3 as

$$\text{done}^*(\tau_i, \ell) \stackrel{\text{def}}{=} [\![C_i(\text{LO}) - ((\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \bmod T_i)]\!]_0$$

and show that it is an equivalent definition:

Lemma 4

$$\text{done}^*(\tau_i, \ell) = \text{done}_{\text{HI}}(\tau_i, \ell)$$

Proof We split the proof into three cases.

First case: $D_i(\text{HI}) > \ell \bmod T_i \geqslant D_i(\text{HI}) - D_i(\text{LO})$.

From $\ell \bmod T_i \geqslant D_i(\text{HI}) - D_i(\text{LO})$ we know that

$$(\ell \bmod T_i) - (D_i(\text{HI}) - D_i(\text{LO})) = (\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \bmod T_i. \quad (9)$$

With (9) we can rewrite $\text{done}^*(\tau_i, \ell)$ as

$$\begin{aligned}\text{done}^*(\tau_i, \ell) &= [\![C_i(\text{LO}) - ((\ell \bmod T_i) - (D_i(\text{HI}) - D_i(\text{LO})))]\!]_0 \\ &= [\![C_i(\text{LO}) - (\ell \bmod T_i) + D_i(\text{HI}) - D_i(\text{LO})]\!]_0 \\ &= \text{done}_{\text{HI}}(\tau_i, \ell).\end{aligned}$$

Second case: $D_i(\text{HI}) \leqslant \ell \bmod T_i$.

From $D_i(\text{HI}) \leqslant \ell \bmod T_i$ the equality (9) follows again. We can rewrite $\text{done}^*(\tau_i, \ell)$:

$$\begin{aligned}\text{done}^*(\tau_i, \ell) &= [\![C_i(\text{LO}) - (\ell \bmod T_i) + D_i(\text{HI}) - D_i(\text{LO})]\!]_0 \\ &= [\![(C_i(\text{LO}) - D_i(\text{LO})) + (D_i(\text{HI}) - \ell \bmod T_i))]\!]_0\end{aligned}$$

We have $C_i(\text{LO}) \leqslant D_i(\text{LO})$ and $D_i(\text{HI}) \leqslant \ell \bmod T_i$. Therefore,

$$(C_i(\text{LO}) - D_i(\text{LO})) + (D_i(\text{HI}) - \ell \bmod T_i)) \leqslant 0$$

and

$$\text{done}^*(\tau_i, \ell) = 0 = \text{done}_{\text{HI}}(\tau_i, \ell).$$

Third case: $\ell \bmod T_i < D_i(\text{HI}) - D_i(\text{LO})$.

From $\ell \bmod T_i < D_i(\text{HI}) - D_i(\text{LO})$ and from $C_i(\text{LO}) \leqslant D_i(\text{LO}) \leqslant D_i(\text{HI}) \leqslant T_i$ we have[9]

$$\begin{aligned}&(\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \bmod T_i \\ &\quad = T_i - (D_i(\text{HI}) - D_i(\text{LO})) + (\ell \bmod T_i) \\ &\quad \geqslant T_i - (D_i(\text{HI}) - D_i(\text{LO})) \\ &\quad \geqslant D_i(\text{LO}) \\ &\quad \geqslant C_i(\text{LO}).\end{aligned}$$

Therefore,

$$C_i(\text{LO}) - ((\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \bmod T_i) \leqslant 0$$

and

$$\text{done}^*(\tau_i, \ell) = 0 = \text{done}_{\text{HI}}(\tau_i, \ell).$$

□

We can now prove Lemma 2.

[9] Note that we interpret mod as positive remainder: $a \bmod b = a - \lfloor \frac{a}{b} \rfloor \cdot b$.

Proof (Proof of Lemma 2) The lemma follows from straightforward substitutions, first:

$$\begin{aligned}\mathrm{dbf}_{\mathrm{LO}}(\tau_i,\ell) &= \left[\!\!\left[\left(\left\lfloor\frac{\ell - D_i(\mathrm{LO})}{T_i}\right\rfloor + 1\right)\cdot C_i(\mathrm{LO})\right]\!\!\right]_0 \\ &= \left[\!\!\left[\left(\left\lfloor\frac{\ell - (D_j(\mathrm{LO}) - \delta)}{T_j}\right\rfloor + 1\right)\cdot C_j(\mathrm{LO})\right]\!\!\right]_0 \\ &= \mathrm{dbf}_{\mathrm{LO}}(\tau_j,\ell+\delta)\end{aligned}$$

To show the $\mathrm{dbf}_{\mathrm{HI}}$ part, we consider $\mathrm{full}_{\mathrm{HI}}$ and $\mathrm{done}_{\mathrm{HI}}$ separately:

$$\begin{aligned}\mathrm{full}_{\mathrm{HI}}(\tau_i,\ell) &= \left[\!\!\left[\left(\left\lfloor\frac{\ell - (D_i(\mathrm{HI}) - D_i(\mathrm{LO}))}{T_i}\right\rfloor + 1\right)\cdot C_i(\mathrm{HI})\right]\!\!\right]_0 \\ &= \left[\!\!\left[\left(\left\lfloor\frac{\ell - (D_j(\mathrm{HI}) - (D_j(\mathrm{LO}) - \delta))}{T_j}\right\rfloor + 1\right)\cdot C_j(\mathrm{HI})\right]\!\!\right]_0 \\ &= \mathrm{full}_{\mathrm{HI}}(\tau_j,\ell-\delta)\end{aligned}$$

We use Lemma 4 for $\mathrm{done}_{\mathrm{HI}}(\tau_i,\ell) = \mathrm{done}^*(\tau_i,\ell)$:

$$\begin{aligned}\mathrm{done}_{\mathrm{HI}}(\tau_i,\ell) &= \mathrm{done}^*(\tau_i,\ell) \\ &= [\![C_i(\mathrm{LO}) - ((\ell - (D_i(\mathrm{HI}) - D_i(\mathrm{LO}))) \bmod T_i)]\!]_0 \\ &= [\![C_j(\mathrm{LO}) - ((\ell - (D_j(\mathrm{HI}) - (D_j(\mathrm{LO}) - \delta))) \bmod T_j)]\!]_0 \\ &= \mathrm{done}^*(\tau_j,\ell-\delta) \\ &= \mathrm{done}_{\mathrm{HI}}(\tau_j,\ell-\delta)\end{aligned}$$

The $\mathrm{dbf}_{\mathrm{HI}}$ part follows directly:

$$\begin{aligned}\mathrm{dbf}_{\mathrm{HI}}(\tau_i,\ell) &= \mathrm{full}_{\mathrm{HI}}(\tau_i,\ell) - \mathrm{done}_{\mathrm{HI}}(\tau_i,\ell) \\ &= \mathrm{full}_{\mathrm{HI}}(\tau_j,\ell-\delta) - \mathrm{done}_{\mathrm{HI}}(\tau_j,\ell-\delta) \\ &= \mathrm{dbf}_{\mathrm{HI}}(\tau_j,\ell-\delta)\end{aligned}$$

□