Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks

Pontus Ekberg and Wang Yi
Uppsala University, Sweden
Email: {pontus.ekberg | yi}@it.uu.se

***Abstract*—We derive 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 from high- to low-criticality mode by tuning the relative deadlines. This allows us to shape the demand characteristics of each task. We propose an efficient algorithm 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 significantly more powerful than previous approaches to mixed-criticality scheduling. This new approach has the added benefit of supporting hierarchical scheduling frameworks.**

I. 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 (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 [1] 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. In the mixed-criticality setting, each task has one demand-bound function per criticality mode. We show that the functions for the different criticality modes are inherently connected, and that we can shift demand from one function to another by tuning the parameters of the tasks, specifically the relative deadlines. In light of this, EDF is extended to the mixed-criticality setting by allowing it to use different relative deadlines for tasks depending on the current criticality mode.

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 [2]. We present an efficient algorithm that automatically shapes the demand of a task set in this manner. An experimental evaluation indicates that the acceptance ratio of randomly generated task sets is significantly higher with this 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 new approach directly enables the use of mixed-criticality scheduling within common hierarchical scheduling frameworks.

A. Related Work

Vestal extended fixed-priority response-time analysis of sporadic tasks to the mixed-criticality setting [3]. 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. [4]

A number of papers have considered the more restricted problem of scheduling a finite set of mixed-criticality jobs (e.g., [5], [6]). Baruah et al. have shown 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 [6]. 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 scheduling strategies developed for scheduling a finite set of mixed-criticality jobs is the *own criticality based priority* (OCBP) scheduling strategy by Baruah et al. [5] It assigns priorities to the individual jobs using a variant of the so called Audsley approach [7]. This scheduling strategy was later extended by Li and Baruah to systems of mixed-criticality sporadic tasks, where priorities are calculated and assigned to all jobs in a busy period [8]. 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., who presented an OCBP-based scheduler for sporadic task sets where runtime decisions are of at most polynomial complexity [9].

Baruah et al. have proposed an EDF-based approach called EDF-VD [10] for scheduling implicit-deadline mixed-criticality sporadic task sets. EDF-VD splits the period of each high-criticality task into two parts. In essence, during the schedulability analysis one part is treated as the period in low-criticality mode, and the other part as the period in high-criticality mode. The task set is then considered schedulable if the utilization of the (modified) task set is sufficiently low. In EDF-VD, different deadlines are used in different criticality modes, similar to how EDF is used in this paper.

II. PRELIMINARIES

A. System Model and Notation

We use the same system model as in previous work on the scheduling of mixed-criticality tasks [8], [9], [4], [3], [10]. This is a straightforward extension of the classic sporadic task model [11] to a mixed-criticality setting (previous work differs in the assumption of implicit, constrained or arbitrary deadlines). Formally, each task τ_i in a mixed-criticality sporadic task set $\tau = \{\tau_1, \ldots, \tau_m\}$ 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 times 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.

The techniques presented in this paper may be generalized to an arbitrary number of criticality levels, but we restrict the model to two levels here for the sake of brevity. 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 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

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

For compactness of presentation we use the notation $[\![\cdot]\!]_k$ to constrain an expression, such that $[\![A]\!]_k \stackrel{\text{def}}{=} \max(A, k)$.

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

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

The time interval $[r(J), d(J)]$ is called the scheduling window of job J. If any job executes for its low-criticality worst-case execution time without signaling that it has finished, the system will immediately switch to high-criticality mode. If the system has switched to high-criticality mode, it will never switch back to low-criticality.[1] After the switch we are not required to meet any deadlines for low-criticality jobs, but high-criticality jobs may instead execute for up to their high-criticality worst-case execution times. 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.

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 II.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 [8], [9] or EDF-VD [10]. However, we will see that its demand characteristics can be tuned using the techniques presented in this paper until it is schedulable by EDF.

B. Demand-Bound Functions

A successful approach to analyzing the schedulability of real-time workloads is to use demand-bound functions [1]. A demand-bound function captures the maximum execution demand of a task in any time interval of a given size.

Definition II.2 (Demand-bound function)**.** *A demand-bound function $\text{dbf}(\tau_i, \ell)$ 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 [1].

A similar concept is the supply-bound function $\text{sbf}(\ell)$ [2], 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 $\text{sbf}(\ell) = \ell$. Other platforms, such as virtual servers used in hierarchical scheduling, have their own particular supply-bound functions (e.g., [2], [12]).

[1] One could find a time point where it is safe to switch back, but it is out of scope of this paper.

We say that a supply-bound function sbf is of *at most unit speed* if

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

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 II.3 (See e.g., [12])**.** *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*

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

III. 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 II.3 is extended in the straightforward way:

Proposition III.1. *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:*

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

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

Conditions A and B 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 $\mathrm{sbf}_{\mathrm{LO}}$ is of at most unit speed. This can always be achieved by simply scaling the parameters of the task set together with $\mathrm{sbf}_{\mathrm{LO}}$ and $\mathrm{sbf}_{\mathrm{HI}}$. Note that $\mathrm{sbf}_{\mathrm{LO}}$ and $\mathrm{sbf}_{\mathrm{HI}}$ 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 $\mathrm{dbf}_{\mathrm{LO}}$ it is simple. In low-criticality mode, each task τ_i behaves as a normal sporadic task, and all of its jobs are guaranteed to execute for at most $C_i(\mathrm{LO})$ 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 [1].

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

Definition III.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.

A. 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 taken into account in each high-criticality task's $\mathrm{dbf}_{\mathrm{HI}}$. 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 already for some time before the switch, its execution demand may also be lower than that of other jobs.

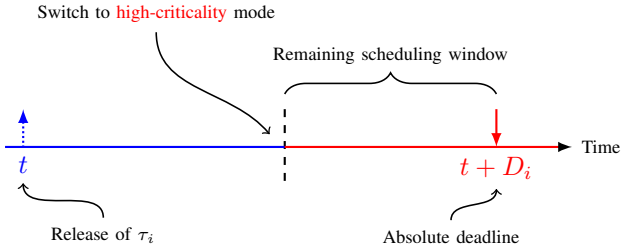

Fig. 1. After a switch to high-criticality mode, the remaining execution demand of a carry-over job must be finished in its remaining scheduling window.

For the sake of bounding the demand in high-criticality mode (in order to meet Condition B), we can assume that the demand is met in low-criticality mode (Condition A), or the task set would be deemed unschedulable anyway. In other words, we seek to show $\mathrm{A} \wedge \mathrm{B}$ by showing $\mathrm{A} \wedge (\mathrm{A} \rightarrow \mathrm{B})$. 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 $n \geqslant 0$ time units left until its deadline. The remaining scheduling window of this job is therefore of length n. Since this job would have met its deadline in low-criticality mode if the switch had *not* happened, there can be at most n time units left of its low-criticality execution demand $C_i(\mathrm{LO})$ at the time of the switch (this follows directly from the assumption that $\mathrm{sbf}_{\mathrm{LO}}$ is of at most unit speed). The job must therefore have executed for at least $C_i(\mathrm{LO}) - n$ time units before the switch. Since the system has switched to high-criticality mode, the job may now execute for up to $C_i(\mathrm{HI})$ time units in total. The total execution demand

remaining for the carry-over job after the switch is therefore at most $C_i(\text{HI}) - (C_i(\text{LO}) - n)$. Unfortunately, as n 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 B. Next we will show how this problem can be mitigated.

B. 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.

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 III.3 (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 while a job from τ_i has n time units left until its true deadline, as illustrated in Fig. 2, then the following hold:*

1) *If $n < D_i(\text{HI}) - D_i(\text{LO})$, the job has already finished before the switch.*
2) *If $n \geqslant D_i(\text{HI}) - D_i(\text{LO})$, the job may be a carry-over job, and at least $[\![C_i(\text{LO}) - n + D_i(\text{HI}) - D_i(\text{LO})]\!]_0$ 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 $n - (D_i(\text{HI}) - D_i(\text{LO}))$ 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 $n - (D_i(\text{HI}) - D_i(\text{LO}))$ time units left of the job's low-criticality execution demand. At least $[\![C_i(\text{LO}) - n + D_i(\text{HI}) - D_i(\text{LO})]\!]_0$ time units of the job's work must therefore have been finished already by the time of the switch. ■

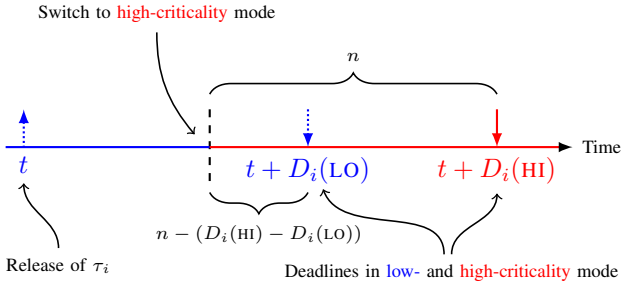

Fig. 2. A carry-over job of τ_i has a remaining scheduling window of length n after the switch to high-criticality mode. Here the switch happens before the job's low-criticality deadline.

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 IV.

C. 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 [1]:

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

The demand-bound function for task τ_i in high-criticality mode, $\text{dbf}_{\text{HI}}(\tau_i, \ell)$, must upper-bound 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 III.3 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

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

The function $\text{full}(\tau_i, \ell)$ is disregarding that a carry-over job may have finished some execution in low-criticality mode (i.e.,

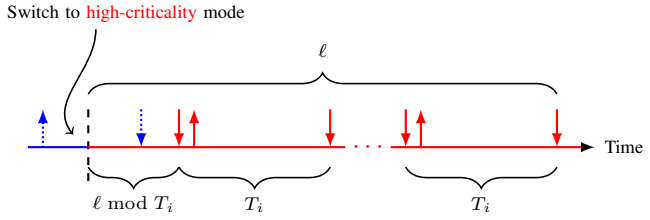

Fig. 3. After fitting a number of full jobs into an interval of length ℓ, there are $\ell \bmod T_i$ time units left for either another full job, a carry-over job, or no job at all. In this figure it is enough for a carry-over job.

it is counting $C_i(\text{HI})$ for all jobs). We can check whether all jobs that contributed execution demand to $\text{full}(\tau_i, \ell)$ 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 III.3.

As shown in Fig. 3, for a time interval of length ℓ, there are at most $n = \ell \bmod T_i$ time units left for the "first" job (which may be a carry-over job). If $n \geqslant D_i(\text{HI})$, it is enough for the scheduling window of a full job, and we cannot subtract anything from $\text{full}(\tau_i, \ell)$. If $n < D_i(\text{HI}) - D_i(\text{LO})$, all jobs that contributed to $\text{full}(\tau_i, \ell)$ can fit their entire periods inside the interval, so there is again nothing to subtract. Otherwise, we use Lemma III.3 to quantify the amount of work that must have been finished in low-criticality mode:

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

where $n = \ell \bmod T_i$. 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:

$$\text{dbf}_{\text{HI}}(\tau_i, \ell) \stackrel{\text{def}}{=} \text{full}(\tau_i, \ell) - \text{done}(\tau_i, \ell) \tag{2}$$

Example III.4. *Consider task τ_3 from Example II.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.*

IV. Efficiently 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 A and B of Proposition III.1 hold.

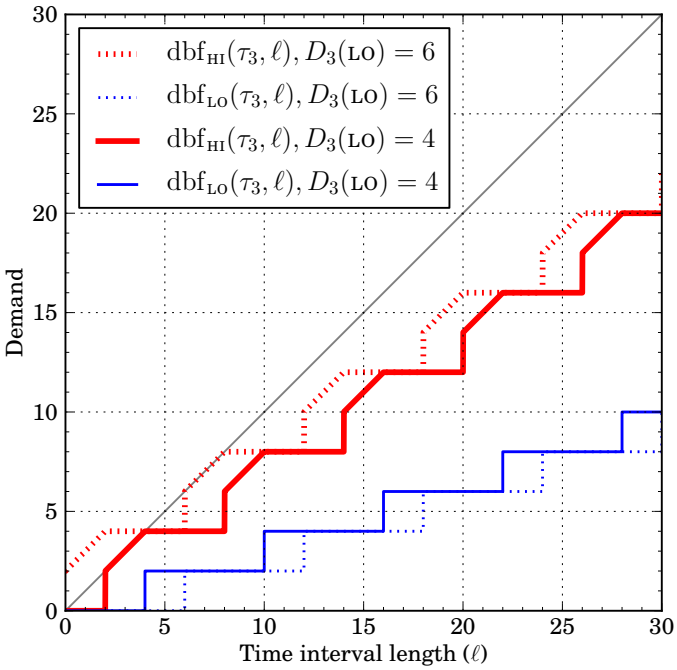

Fig. 4. Demand-bound functions for task τ_3 from Example II.1 with two different values for $D_3(\text{LO})$.

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.[2]

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

Lemma IV.1 (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, 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

[2]Our prototype implementation typically terminates in the order of minutes for task sets of 100 tasks, with utilization of about 90% and parameter values ranging up to a few thousands (assuming $\text{sbf}_{\text{LO}}(\ell) = \text{sbf}_{\text{HI}}(\ell) = \ell$). For smaller task sets of about 20 tasks, it typically terminates in the order of seconds.

total demand of the task set is met by the supply in both low- and high-criticality mode.

The algorithm in Fig. 5 tunes the demand of a task set in a somewhat greedy fashion. Let $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$ be predicates corresponding to the inequalities in Conditions A and B, respectively:

$$\mathcal{A}(\ell): \quad \sum_{\tau_i \in \tau} \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \ell) \leqslant \mathrm{sbf}_{\mathrm{LO}}(\ell)$$

$$\mathcal{B}(\ell): \quad \sum_{\tau_i \in \mathrm{HI}(\tau)} \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \ell) \leqslant \mathrm{sbf}_{\mathrm{HI}}(\ell)$$

The general idea is to check $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$ for increasing time interval lengths ℓ (from 0 up to an upper bound $\ell_{\max}$ described in Section IV-A). 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 $\mathcal{B}(\ell)$ fails, the low-criticality relative deadline of one task is decreased by 1. It picks the task τ_i which would decrease the demand $\mathrm{dbf}_{\mathrm{HI}}(\tau_i, \ell)$ the most when $D_i(\mathrm{LO})$ is decreased by 1 (ties broken arbitrarily).
- If $\mathcal{A}(\ell)$ 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.

The algorithm terminates with SUCCESS if and only if it has found low-criticality relative deadlines with which $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$ hold for all $\ell \in \{0, 1, \ldots, \ell_{\max}\}$. This implies that both Conditions A and B hold, as will be shown in Section IV-A. Therefore, the algorithm terminates with SUCCESS if and only if the task set is schedulable according to Proposition III.1.

Example IV.2. *Consider how the algorithm in Fig. 5 assigns values to $D_2(\mathrm{LO})$ and $D_3(\mathrm{LO})$ for the two high criticality tasks τ_2 and τ_3 in the task set from Example II.1. We assume a dedicated uniprocessor platform ($\mathrm{sbf}_{\mathrm{LO}}(\ell) = \mathrm{sbf}_{\mathrm{HI}}(\ell) = \ell$). Fig. 6 shows the demand bound functions for this task set with unmodified relative deadlines. In the first iteration, $\mathcal{B}(0)$ fails, and $D_3(\mathrm{LO})$ is decreased by 1. In the second iteration, $\mathcal{B}(0)$ fails again, but this time $D_2(\mathrm{LO})$ is decreased by 1. In the third iteration, $\mathcal{B}(1)$ fails and $D_3(\mathrm{LO})$ is decreased by 1 again. This is then repeated two more times where $\mathcal{B}(\ell)$ fails at $\ell = 2$ and $\ell = 3$, respectively, and $D_3(\mathrm{LO})$ is lowered two more times. Both $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$ then hold for all $\ell \in \{0, 1, \ldots, \ell_{\max}\}$, and the algorithm terminates with $D_2(\mathrm{LO}) = 5$ and $D_3(\mathrm{LO}) = 2$, resulting in the demand-bound functions shown in Fig. 7.*

A. Complexity and Correctness of the Algorithm

For the complexity of the algorithm in Fig. 5, note that each $\tau_i \in \mathrm{HI}(\tau)$ will have its deadline $D_i(\mathrm{LO})$ changed at most $D_i - C_i(\mathrm{LO}) + 1$ times. In every iteration of the outer loop some low-criticality relative deadline is changed, or the algorithm terminates, so the 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

```
candidates ← {i | τ_i ∈ HI(τ)}
mod ← ⊥
ℓ_max ← upper bound for ℓ in Conditions A and B
loop
  final ← true
  for ℓ = 0, 1, ..., ℓ_max do
    if ¬A(ℓ) then
      if mod = ⊥ then
        return FAILURE
      end if
      D_mod(LO) ← D_mod(LO) + 1
      candidates ← candidates \ {mod}
      mod ← ⊥
      final ← false
      break
    else if ¬B(ℓ) then
      if candidates = ∅ then
        return FAILURE
      end if
      mod ← arg max_{i ∈ candidates} (dbf_HI(τ_i, ℓ) − dbf_HI(τ_i, ℓ − 1))
      D_mod(LO) ← D_mod(LO) − 1
      if D_mod(LO) = C_mod(LO) then
        candidates ← candidates \ {mod}
      end if
      final ← false
      break
    end if
  end for
  if final then
    return SUCCESS
  end if
end loop
```

Fig. 5. Greedy algorithm for tuning low-criticality relative deadlines.

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 if and only if it has found relative deadlines with which both $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$ hold for all $\ell \in \{0, 1, \ldots, \ell_{\max}\}$. However, in Proposition III.1, the inequalities $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$ should hold for all $\ell \geqslant 0$. We will show here that $\ell_{\max}$ can be found such that if $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$ 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 $\mathcal{A}(\ell)$ and $\mathcal{B}(\ell)$). It follows directly that if $\mathcal{A}(\ell)$ or $\mathcal{B}(\ell)$ 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-

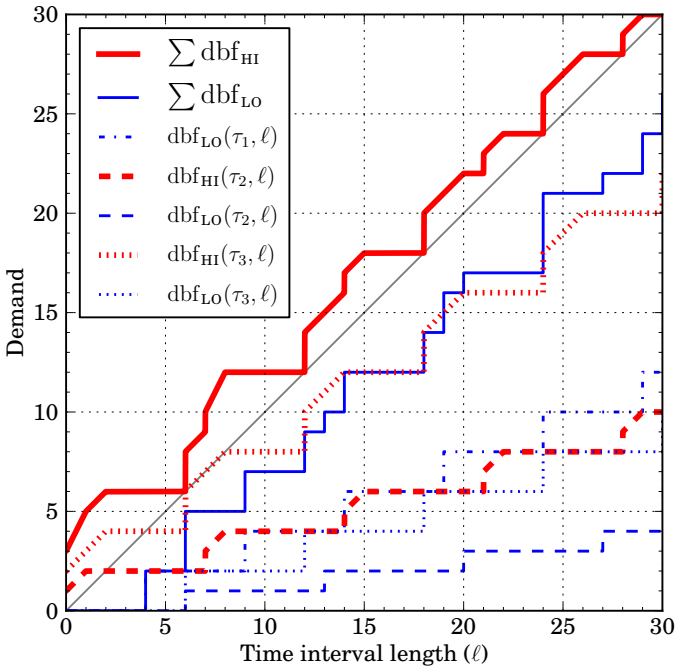

Fig. 6. Demand-bound functions for the tasks from Example II.1 with unmodified low-criticality relative deadlines ($D_i(\mathrm{LO}) = D_i(\mathrm{HI}) = D_i$).

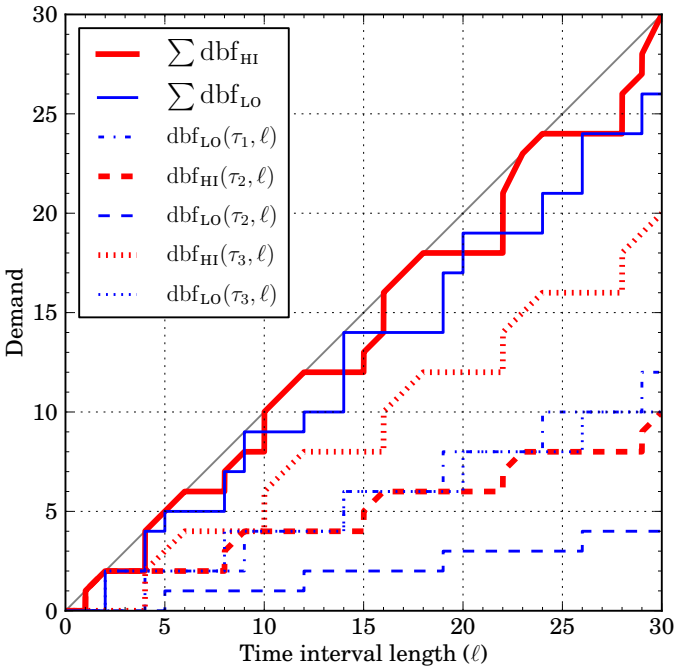

Fig. 7. Demand-bound functions for the tasks from Example II.1 after having low-criticality relative deadlines tuned by the algorithm in Fig. 5.

bound functions used. It is always possible to use the hyperperiod as the bound $\ell_{\max}$. However, for a dedicated uniprocessor ($\mathrm{sbf}_{\mathrm{LO}}(\ell) = \mathrm{sbf}_{\mathrm{HI}}(\ell) = \ell$) we can use established methods [1] to calculate a pseudo-polynomial $\ell_{\max}$ as long as $U_{\mathrm{LO}}(\tau)$ and $U_{\mathrm{HI}}(\tau)$ 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 non-mixed criticality sporadic tasks (C, D, T) in the following way:

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

Note that using the standard demand-bound function dbf for non-mixed criticality sporadic tasks [1], $\mathrm{dbf}(f_{\mathrm{LO}}(\tau_i), \ell) = \mathrm{dbf}_{\mathrm{LO}}(\tau_i, \ell)$ and $\mathrm{dbf}(f_{\mathrm{HI}}(\tau_i), \ell) = \mathrm{full}(\tau_i, \ell) \geqslant \mathrm{dbf}_{\mathrm{HI}}(\tau_i, \ell)$. Also, $U(f_{\mathrm{LO}}(\tau_i)) = U_{\mathrm{LO}}(\tau_i)$ and $U(f_{\mathrm{HI}}(\tau_i)) = U_{\mathrm{HI}}(\tau_i)$, where U gives the utilization of a non-mixed criticality task.

From [1] we know how to construct a pseudo-polynomial bound such that the inequality in Proposition II.3 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 $\ell^{\mathrm{LO}}_{\max}$ for the task set $\{f_{\mathrm{LO}}(\tau_i) \mid \tau_i \in \tau\}$, it is also valid for Condition A in Proposition III.1. Similarly, such a bound $\ell^{\mathrm{HI}}_{\max}$ for the task set $\{f_{\mathrm{HI}}(\tau_i) \mid \tau_i \in \mathrm{HI}(\tau)\}$ is valid for Condition B of Proposition III.1. We can therefore use $\ell_{\max} = \max(\ell^{\mathrm{LO}}_{\max}, \ell^{\mathrm{HI}}_{\max})$ for the algorithm in Fig. 5.[3]

[3]A small technical issue is that the bound from [1] is dependent on the relative deadlines of tasks, which are changed by the algorithm in Fig. 5. The issue is easily avoided by using the largest bound generated with any of the possible relative deadlines that may be assigned, or to use an alternative bound that is independent of relative deadlines, e.g., from [13].

V. Experimental Evaluation

In this section we evaluate the effectiveness of characterizing mixed-criticality task sets using demand-bound functions. In particular, we study the effectiveness of this approach when low-criticality relative deadlines are tuned using the algorithm in Fig. 5. We also evaluate previous approaches to mixed-criticality scheduling from the literature, and compare the acceptance ratios of their corresponding schedulability tests. The other approaches only support dedicated (uniprocessor) platforms, and some assume implicit deadline sporadic tasks. We use this setting to be able to compare all approaches. The compared approaches are:

- Greedy: The test in Proposition III.1 using the demand-bound functions in Equations (1) and (2). Relative deadlines are tuned using the algorithm in Fig. 5.
- OCBP-prio: The test for OCBP-based scheduling from [9], which is based on whether a priority ordering can be found for all jobs in a busy period.
- AMC-max: Schedulability based on the most powerful response-time calculation for fixed-priority scheduling from [4], called AMC-max. Priorities are assigned using Audsley's algorithm, as described in [4].
- Vestal: Schedulability based on the response-time calculation for fixed-priority scheduling from [3], using 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 from [4].
- EDF-VD: The test for the EDF-VD scheduling algorithm from [10].

OCBP-load: The test from [8] for OCBP-based scheduling, which measures the *load* of the task set.

Naive: A test based on simply flattening the mixed-criticality sporadic task set into a standard sporadic task set using resource reservation, and 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 standard (implicit deadline) sporadic task with worst-case execution time $C_i(L_i)$ and period T_i. This simple test is included as a baseline for the more sophisticated approaches.

A. 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 four parameters: the probability P_{HI} of being of high-criticality, the maximum ratio R_{HI} between high- and low-criticality execution time, the maximum low-criticality execution time $C_{\text{LO}}^{\max}$ and the maximum period $T^{\max}$. 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 $\{1, 2, \ldots, C_{\text{LO}}^{\max}\}$.
- $C_i(\text{HI})$ is drawn from the uniform distribution over $\{C_i(\text{LO}), C_i(\text{LO}) + 1, \ldots, R_{\text{HI}} \cdot C_i(\text{LO})\}$ if $L_i = \text{HI}$. Otherwise, $C_i(\text{HI}) = C_i(\text{LO})$.
- T_i is drawn from the uniform distribution over $\{C_i(L_i), C_i(L_i) + 1, \ldots, T^{\max}\}$.
- $D_i = T_i$ since deadlines are implicit.

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 integer parameter tasks, we allow the task set's average utilization to fall within in the small interval between $U_{\min}^* = U^* - 0.005$ and $U_{\max}^* = U^* + 0.005$.

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 also discarded.

B. Results

Fig. 8 shows the acceptance ratio (fraction of schedulable task sets) as a function of (target) average utilization for task sets generated using parameters $P_{\text{HI}} = 0.5$, $R_{\text{HI}} = 4$, $C_{\text{LO}}^{\max} = 10$ and $T^{\max} = 200$. Each data point is based on 10,000 randomly generated task sets.

Next we study the effects of varying the parameters P_{HI} and R_{HI}. We plot the *weighted acceptance ratio* (or weighted schedulability measure) [14] 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 more importance is given to the acceptance ratio for a larger utilization value, as these are the cases we are generally interested in.

In Fig. 9 and 10 we have plotted the weighted acceptance ratio as a function of P_{HI} and R_{HI}, respectively. The legends are omitted in these plots in order to avoid covering the lines, they are the same as in Fig. 8. The set $\mathfrak{U}$ of average utilization values are the same 30 values as used in Fig. 8 ($\mathfrak{U} = \{(1/30) \cdot (x + 1/2) \mid x \in \{0, \ldots, 29\}\}$). Except for the varied parameter (P_{HI} or R_{HI}), the parameters are also the same as for Fig. 8. Each data point is based on 30,000 random task sets.

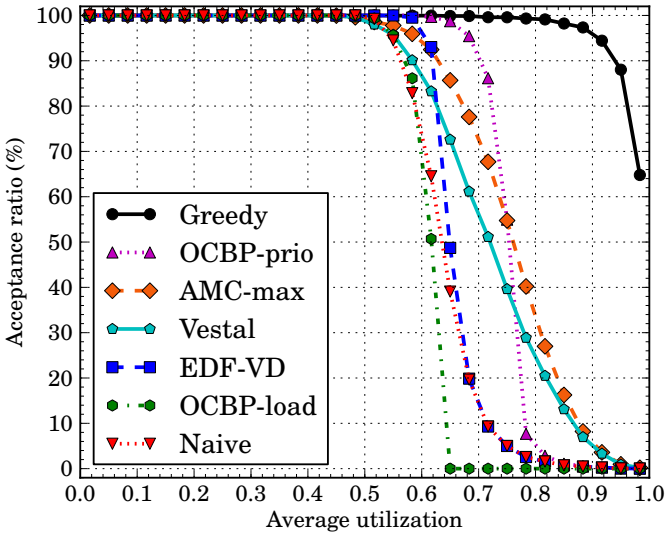

Fig. 8. $P_{\text{HI}} = 0.5$, $R_{\text{HI}} = 4$, $C_{\text{LO}}^{\max} = 10$ and $T^{\max} = 200$

C. Discussion

Evidently, there is a large gap between the acceptance ratios of the new 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_{HI}). The results can of course differ if we vary other parameters or the task set generation procedure, but the gap 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 mixed-criticality sporadic task sets.

Among previous approaches, OCBP-prio [9] and AMC-max [4] seem to perform best. Of these, AMC-max is probably the best choice in practice as it has a significantly lower run-time overhead. 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).

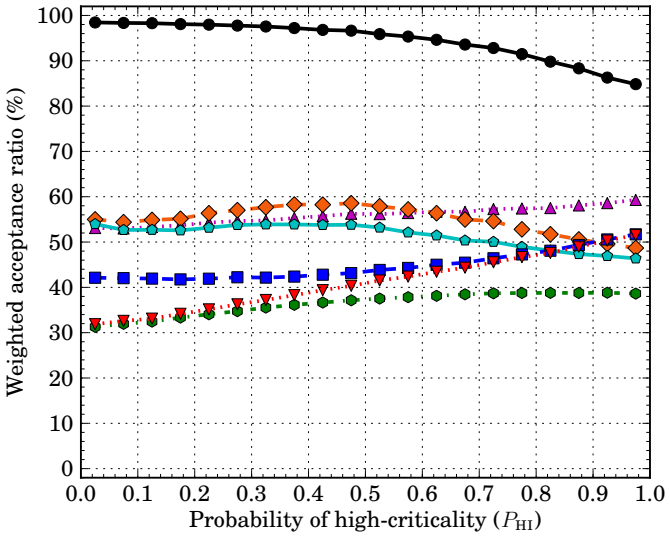

Fig. 9. P_{HI} varying, $R_{\text{HI}} = 4$, $C_{\text{LO}}^{\max} = 10$ and $T^{\max} = 200$

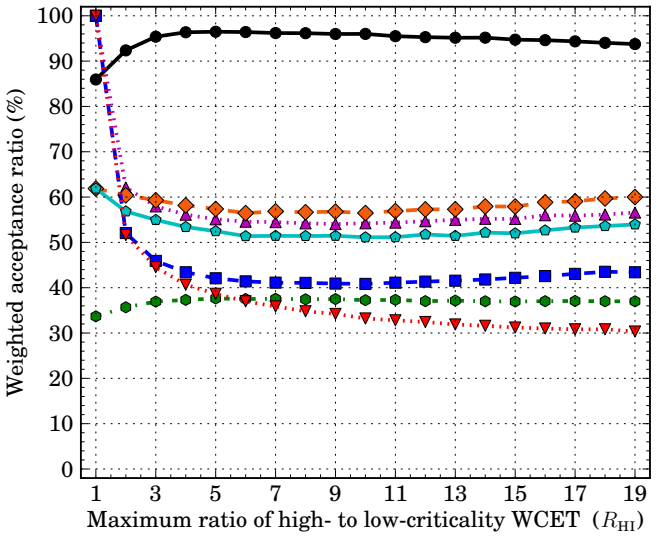

Fig. 10. $P_{\text{HI}} = 0.5$, R_{HI} varying, $C_{\text{LO}}^{\max} = 10$ and $T^{\max} = 200$

The weighted acceptance ratios of all approaches remain relatively steady when varying P_{HI} and R_{HI}. The reasons for the slow trends that can be seen remain mostly unclear to us. An exception is $R_{\text{HI}} = 1$, 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).

VI. Conclusions

We have presented a way of characterizing the demand of mixed-criticality sporadic tasks using demand-bound functions. This characterization is based on the idea that we can use different relative deadlines for the tasks depending on the criticality mode of the system. We described an algorithm that tunes the relative deadlines of tasks, and thereby shapes the demand characteristics of those tasks to the available supply of the platform. Experimental evaluation indicates that this approach is successful in practice.

These results show that EDF-based scheduling can significantly outperform 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. 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 and which tasks must continue to function, and the scheduler makes sure that the system behaves as specified while utilizing platform resources as efficiently as possible.

We mentioned in Section II that the presented methods may be generalized to an arbitrary number of criticality levels. By introducing one relative deadline per criticality level, demand-bound functions for higher levels can be constructed in the same way as dbf_{HI} in Equation (2), depending only on the parameters of the level directly below. Instead of having Conditions A and B in Proposition III.1, we would have Conditions A, B, C, ..., and their conjunction would be established with a chain of implications: $\text{A} \wedge (\text{A} \rightarrow \text{B}) \wedge (\text{B} \rightarrow \text{C}) \wedge \ldots$ As there are more relative deadlines with more criticality levels, the problem of tuning them becomes more challenging. It may turn out that different heuristics than those used in Section IV are more suitable when the number of levels increases.

Like previous work from the literature, we have considered execution-time focused mixed-criticality systems. There are certainly other points of view that can be considered. For example, we might want a system to enter high-criticality mode if external events arrive too quickly, if some subsystem breaks down, or, in general, if something unexpected happens in the environment of the system. We might also want greater flexibility in how the system should change in such an event. Perhaps we want to change other parameters of the tasks (e.g., deadline and period) or even add new tasks to handle the new situation. As future work we would like to address more general mixed-criticality systems. We would also like to consider task models with resource sharing or more complex job release patterns. We believe that the techniques in this paper will generalize to these cases.

References

[1] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *RTSS*, 1990, pp. 182–190.

[2] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *RTAS*, 2001, pp. 75 –84.

[3] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, 2007, pp. 239–243.

[4] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *RTSS*, 2011, pp. 34 –43.

[5] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *RTAS*, 2010, pp. 13 –22.

[6] S. Baruah, K., V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Transactions on Computers*, Jul. 2011.
[7] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," University of York, England, Tech. Rep., 1991.
[8] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *RTSS*, 2010, pp. 183 –192.
[9] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *RTSS*, 2011, pp. 13–23.
[10] S. Baruah, K., V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Ster, Van Der, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *ESA*, 2011, pp. 555–566.
[11] A. Mok, "Fundamental design problems of distributed systems for the hard real-time environment," Cambridge, MA, USA, Tech. Rep., 1983.
[12] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS*, 2003, pp. 2 – 13.
[13] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *RTAS*, 2011, pp. 71–80.
[14] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," in *OSPERT*, 2010, pp. 33–44.

APPENDIX
PROOF OF LEMMA IV.1

Before proving Lemma IV.1, we reformulate the function $\mathrm{done}(\tau_i, \ell)$ from Section III-C as

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

and show that it is an equivalent definition:

Lemma A.1.

$$\mathrm{done}^*(\tau_i, \ell) = \mathrm{done}(\tau_i, \ell)$$

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

$$\begin{aligned}&(\ell \bmod T_i) - (D_i(\text{HI}) - D_i(\text{LO}))\\ &\quad = (\ell - (D_i(\text{HI}) - D_i(\text{LO}))) \bmod T_i. \qquad (3)\end{aligned}$$

With (3) we can rewrite $\mathrm{done}^*(\tau_i, \ell)$ as

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

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

From $D_i(\text{HI}) \leqslant \ell \bmod T_i$ the equality (3) follows again. We can rewrite $\mathrm{done}^*(\tau_i, \ell)$:

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

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

$$\mathrm{done}^*(\tau_i, \ell) = 0 = \mathrm{done}(\tau_i, \ell).$$

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[4]

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

Therefore,

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

and

$$\mathrm{done}^*(\tau_i, \ell) = 0 = \mathrm{done}(\tau_i, \ell).$$

■

We can now prove Lemma IV.1.

Proof of Lemma IV.1: The lemma follows from straightforward substitutions, first:

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

To show the dbf_{HI} part, we consider full and done separately:

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

We use Lemma A.1 for $\mathrm{done}(\tau_i, \ell) = \mathrm{done}^*(\tau_i, \ell)$:

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

The dbf_{HI} part follows directly:

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

■

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