Graceful Degradation in Semi-Clairvoyant Scheduling

Sanjoy Baruah
Washington University in Saint Louis, USA

Pontus Ekberg
Uppsala University, Sweden

Abstract

In the Vestal model of mixed-criticality systems, jobs are characterized by multiple different estimates of their actual, but unknown, worst-case execution time (WCET) parameters. Some recent research has focused upon a *semi-clairvoyant* model for mixed-criticality systems in which it is assumed that each job reveals upon arrival which of its WCET parameters it will respect. We study the problem of scheduling such semi-clairvoyant systems to ensure graceful degradation of service to less critical jobs in the event that the systems exhibit high-criticality behavior. We propose multiple different interpretations of graceful degradation in such systems, and derive efficient scheduling algorithms that are capable of ensuring graceful degradation under these different interpretations.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Software and its engineering → Real-time schedulability

Keywords and phrases Mixed criticality, semi-clairvoyance, graceful degradation

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2021.9

Funding *Sanjoy Baruah*: National Science Foundation Grants CNS-1814739 and CPS-1932530.
Pontus Ekberg: Swedish Research Council grant 2018-04446.

1 Introduction

A model for mixed-criticality workloads was proposed by Vestal [30] as a means of achieving timing predictability upon modern processors. In this model, individual pieces of real-time code are represented as jobs with associated deadlines that are characterized by multiple worst-case execution time (WCET) parameters. These different WCET parameters represent different estimates, made at differing levels of assurance, of the actual unknown WCET of the code. Each job is also assigned a criticality — in the two-criticality level model considered in this paper (and much of the mixed-criticality scheduling theory literature), these are called HI and LO, denoting greater and lesser criticality respectively. The two WCET parameters are determined for each job, one at a level of assurance consistent with HI criticality and a second at a level of assurance consistent with LO criticality. The correctness criterion in the Vestal mixed-criticality model is that if each job completes execution within a duration not exceeding its LO-criticality WCET estimate then all the jobs should complete execution by their respective deadlines, whereas if some jobs do not complete execution within their LO-criticality WCET estimates (but all jobs would complete execution if allowed to execute for as much as their HI-criticality WCET), then all the HI-criticality jobs should complete execution by their respective deadlines although the LO-criticality jobs may fail to do so.

Run-time monitoring. The methodology introduced by Vestal [30] was initially applied for the verification of timing correctness of *criticality-agnostic* scheduling algorithms: algorithms that do not seek to determine during run-time whether or not LO-criticality WCET estimates have been exceeded. It was later observed [12] that if system state were *monitored* during run-time to determine when jobs execute in excess of their LO-criticality WCETs, then

custom-designed mixed-criticality scheduling algorithms could be developed that explicitly exploit such run-time information. Several such mixed-criticality algorithms were developed — OCBP [10], MC-EDF [29], EDF-VD [5], AMC [7], MC-Fluid [26], etc. — that determine during run-time if and when the system "mode" transitions from LO-criticality mode (no job has executed beyond its LO-criticality WCET) to HI-criticality mode (some job has executed for more than its LO-criticality WCET), and adjusts its scheduling decisions accordingly.

Forms of Clairvoyance. The notion of *clairvoyance*, which has previously been used to quantify the effectiveness of on-line algorithms (see, e.g., [25]), forms the basis of the speedup factor metric that is widely used for quantitatively characterizing these mixed-criticality scheduling algorithms. In the context of mixed-criticality scheduling, a clairvoyant algorithm is one that knows prior to run-time whether any job is going to exceed its LO-criticality WCET or not. Generally speaking, a clairvoyant scheduling algorithm is an idealized abstraction against which to compare the performance of actual scheduling algorithms.

More recently, the concept of *semi-clairvoyance* was introduced for mixed-criticality scheduling [1] (also see [14]). A semi-clairvoyant scheduling algorithm is one that knows, at the instant of a job's arrival, whether it will complete execution within its LO-criticality WCET. Unlike [full] clairvoyance, which is a purely conceptual abstraction that is not realisable in practice, it is persuasively argued in both [1] and [14] that semi-clairvoyance is a realistic and practically useful model for certain circumstances. For instance, a system developer may provide two separate implementations of a job: upon arrival, the system determines which implementation it is appropriate to execute given the current circumstances – i.e., the current mode. (E.g., one implementation may be intended for execution under regular conditions, and another for execution under unexpected – i.e., HI criticality – conditions: the HI-criticality implementation including code to perform crisis-mitigation functionalities). As stated in [14], semi-clairvoyance is particularly applicable "when the execution time [...] depends on the state of the system at the time the job arrives, rather than on some internal property that emerges as it executes."

Graceful degradation and mixed-criticality scheduling. Despite its very significant impact on the real-time scheduling theory literature, Vestal's mixed-criticality workload model [30] has been criticized [20, 19, 31] for not matching systems developers' expectations in some important aspects. Our focus here is upon one such aspect: that LO-criticality jobs ought to be guaranteed some amount of execution prior to their deadlines even in HI-criticality modes. Modifications to the Vestal model have been proposed (e.g., in [13] and [18]) that allows for the specification of some degraded service for LO-criticality jobs even in HI-criticality system behaviors – we will describe such a model in Section 2. Many algorithms have been proposed (e.g., [8, 23, 24, 22]) that seek to ensure such graceful degradation for systems.

This research. In this paper, we study graceful degradation for semi-clairvoyant algorithms upon preemptive uniprocessors – to our knowledge, this is the first piece of research that integrates consideration of these two concepts. We consider three different notions of graceful degradation, characterized here as three different *correctness criteria*, by placing different requirements as to which LO-criticality jobs it is acceptable to provide degraded service to, upon some HI-criticality job's arrival signalling HI-criticality mode:

CC-1. all LO-criticality jobs that *have deadlines* after this instant;

CC-2. all LO-criticality jobs that *begin execution* after this instant; or

CC-3. all LO-criticality jobs that *arrive* after this instant.

Although the three correctness criteria appear very similar, differing only in the treatment accorded to LO-criticality jobs that span a mode-change instant, we will see that the associated schedulability analysis problems are very different. We will show that determining whether a collection of independent jobs can be scheduled correctly upon a preemptive uniprocessor can be done in polynomial time via reduction to a linear program for correctness criterion CC-1, is NP-complete in the strong sense for CC-2, and reduces to a variant of EDF-schedulability analysis for CC-3 that is also solvable in polynomial time.[1]

Correctness criterion CC-1 is consistent with standard interpretations of mixed-criticality scheduling (and most prior work). It can be a sensible criterion when LO-criticality jobs, unlike HI-criticality jobs, do not have multiple available implementations, but their execution times are instead enforced by *budgeting*. The reduced execution time of LO-criticality jobs in HI-criticality mode is then simply achieved by lowering their execution-time budgets. This can be done even to started jobs, and jobs that do not finish within their budgets are suspended or discarded.

However, there are circumstances where the other two criteria are more appropriate. Under the scenario in which multiple implementations of each job are available and the run-time system chooses which to execute based upon current system state, it is reasonable to expect, as in CC-2, that once an implementation is chosen and its execution has commenced, it must be completed.

For systems observing "*commitment upon job arrival*" semantics [16], the choice as to which available implementation of a job to execute must be made upon the job's arrival. It is reasonable to expect that for jobs arriving before the mode-change instant, this choice will favor the LO-criticality implementations — this is what CC-3 requires.

Contributions. We have seen above that there are very good reasons for studying all three correctness criteria, which is what we do in this research. Specifically

- We formally define our three correctness criteria in Section 2, within the framework of a workload model integrating graceful degradation and mixed criticality considerations.
- In Section 3 we present a table-based run-time scheduling algorithm for scheduling collections of independent jobs, and a fluid-based algorithm for scheduling implicit-deadline sporadic task systems, under correctness criterion CC-1. Exact schedulability tests that run in polynomial time are presented for both cases.
- In Section 4 we prove that it is NP-hard in the strong sense to determine schedulability of a collection of independent jobs under correctness criterion CC-2. We also provide a mixed integer linear program (MILP) representation of this schedulability problem: solving this MILP allows one to construct a table-based run-time scheduling algorithm.
- In Section 5 we show that EDF is an optimal scheduling algorithm under CC-3 and present exact schedulability tests for scheduling both collections of independent jobs and three-parameter sporadic task systems with bounded utilization under this correctness criterion. These tests run in polynomial and pseudo-polynomial time, respectively.
- We place all these above results within a broader context in Section 6, explaining how they fit together and how they suggest some guidelines for implementation and analysis of semi-clairvoyant systems; while the benefits of these guidelines will be quite evident, we will additionally provide a quantitative evaluation of their cost.

[1] By comparison, schedulability analysis for *non-clairvoyant* scheduling of mixed-criticality jobs under the ordinary mixed-criticality semantics (corresponding to CC-1) without graceful degradation — this is the default mixed-criticality setting seen in most previous work — is NP-hard in the strong sense [4].

2 Workload Model

As mentioned in Section 1 we will restrict our attention here to *dual*-criticality systems: systems with two distinct criticality levels denoted LO and HI, that are to execute upon a single preemptive processor. We consider both collections of independent jobs and recurrent (sporadic) tasks.

For **jobs**, an instance is a collection of n dual-criticality jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$. Each job J_i is characterized by a tuple of parameters: $J_i = (\chi_i, a_i, [c_i(\text{LO}), c_i(\text{HI})], d_i)$ where $\chi_i \in \{\text{LO}, \text{HI}\}$ and the remaining parameters are non-negative integers, with the interpretation that

- χ_i denotes the criticality of the job;
- a_i denotes its release time;
- d_i denotes its deadline; and
- $c_i(\text{LO})$ and $c_i(\text{HI})$ denote LO-criticality and HI-criticality specifications of the job's worst-case execution time (WCET) parameter respectively. We require that the WCETs satisfy the constraint that

  $$\big(c_i(\text{HI}) \leq c_i(\text{LO}),\ \text{if } \chi_i = \text{LO}\big) \ \text{ and } \ \big(c_i(\text{HI}) \geq c_i(\text{LO}),\ \text{if } \chi_i = \text{HI}\big)$$

  (I.e., a LO-criticality job receives *degraded* service whereas a HI-criticality job receives *enhanced* service in HI-criticality mode.)

For **sporadic tasks**, an instance (or system) τ is a collection of n tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each τ_i is characterized by the parameters $(\chi_i, [C_i(\text{LO}), C_i(\text{HI})], D_i, T_i)$, where $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes its criticality, $C_i(\text{LO})$ and $C_i(\text{HI})$ its LO and HI criticality WCETs, $D_i \in \mathbb{N}$ its relative deadline parameter, and $T_i \in \mathbb{N}$ its period. Analogously to jobs, we require that the WCETs satisfy the constraint

$$\big(C_i(\text{HI}) \leq C_i(\text{LO}),\ \text{if } \chi_i = \text{LO}\big) \ \text{ and } \ \big(C_i(\text{HI}) \geq C_i(\text{LO}),\ \text{if } \chi_i = \text{HI}\big)$$

Some additional notation: Let $\tau^{(\text{LO})}$ and $\tau^{(\text{HI})}$ denote the subsets of tasks τ_i with $\chi_i = \text{LO}$ and $\chi_i = \text{HI}$, respectively. Let $U_i(\text{LO}) = C_i(\text{LO})/T_i$ and $U_i(\text{HI}) = C_i(\text{HI})/T_i$. Finally, let $U_{\text{LO}} = \sum_{\tau_i \in \tau} U_i(\text{LO})$ and $U_{\text{HI}} = \sum_{\tau_i \in \tau} U_i(\text{HI})$.

Correctness criteria. Any task system or collection of jobs is assumed to begin execution in LO-criticality mode, with each job requiring an amount of execution that is no greater than its LO-criticality WCET. Any HI-criticality job may signal a transition to HI-criticality mode upon its arrival. If this happens at some time-instant t_{switch}, then each HI-criticality job that arrives at or after t_{switch} may require up to its (potentially larger) HI-criticality WCET. Likewise, each LO-criticality job with deadline at or before t_{switch} requires up to its LO-criticality WCET, while each LO-criticality job that arrives at or after t_{switch} can only require up to its (potentially smaller) HI-criticality WCET. As mentioned in Section 1, we consider three different notions of correctness for LO-criticality jobs that are *active* at time point t_{switch}, and we will see that this choice has significant consequences for how to schedule and analyse such systems. These three different correctness criteria are defined as follows.

CC-1. Any LO-criticality job that has its deadline after time t_{switch} may only require its smaller HI-criticality WCET.

CC-2. Any LO-criticality job that is active and has already started executing at time t_{switch} is permitted to require its larger LO-criticality WCET.

CC-3. Any LO-criticality job that is active at time t_{switch} is permitted to require its larger LO-criticality WCET.

Here we can note that if we restrict all LO-criticality jobs/tasks to have zero-valued WCETs at HI-criticality (i.e., $c_i(\text{HI})$ and $C_i(\text{HI})$ equal zero for all LO-criticality jobs/tasks), then correctness criterion CC-1 reduces to the model that was studied in previous work on semi-clairvoyant scheduling [1, 14].

3 Correctness Criterion CC-1

In this section we devise algorithms for scheduling dual-criticality instances of independent jobs (Section 3.1), and implicit-deadline sporadic task systems (Section 3.2) under the requirement that all LO-criticality jobs *completing* after the arrival of a HI-criticality job that signals HI-criticality mode receive an amount of execution at least equal to their HI-criticality WCETs. The following example illustrates some of the challenges in determining whether such an instance can be correctly scheduled under semi-clairvoyant scheduling.

▶ **Example 1.** Consider an instance comprising the following three jobs:

	χ_i	a_i	$c_i(\text{LO})$	$c_i(\text{HI})$	d_i
J_1	LO	0	1	0	2
J_2	LO	0	2	1	3
J_3	HI	1	0	2	3

First, we point out that this instance is clairvoyant schedulable:

- in LO-criticality mode one could execute J_1 over the interval $[0, 1]$, and J_2 over $[1, 3]$;
- in HI-criticality mode one could execute J_2 over $[0, 1]$, and J_3 over $[1, 3]$.

Observe that a different job is scheduled over $[0, 1]$ in the two modes. However under semi-clairvoyant scheduling the mode only becomes known at time-instant 1 (i.e., upon J_3's arrival). An EDF schedule would choose J_1 over the interval $[0, 1]$; if job J_3 were to signal HI-criticality mode upon arrival, that would require that an execution amount equal to $c_2(\text{HI}) + c_3(\text{HI}) = (1 + 2) = \mathbf{3}$ units be completed over the next two time units. Hence EDF is <u>not</u> able to schedule this instance correctly; we leave it to the reader to verify that the following on-line scheduling strategy is correct:

Execute J_2 over $[0, 1]$
if J_3 signals LO-criticality mode upon its arrival
  execute J_1 over $[1, 2]$, and J_2 over $[2, 3]$
else // (i.e., J_3 signals HI-criticality mode upon its arrival)
  execute J_3 over $[1, 3]$ ◀

In the remainder of this section we describe how such strategies may be determined for collections of jobs and implicit-deadline task systems. Our results for CC-1 are closely inspired by those of [1] (which effectively targets CC-1 without graceful degradation), though in order to enable graceful degradation we take a very different approach to the formulation of the linear program that we will see next.

3.1 Jobs

We partition the time-line by the release-dates and the deadlines (the a_i and d_i parameters) of the jobs. I.e., the time-line from the first release date to the last deadline is divided into subintervals by dividing it at every a_i and d_i. Let $I_1, I_2, \ldots$ denote these subintervals and note that there are at most $2n - 1$ of them. Let $t_1, t_2, \ldots$ denote the distinct time instants

at which HI-criticality jobs arrive, in order (i.e., $t_k < t_{k+1}$). There are at most n such time instants t_k.

In the following we let i range over jobs J_i, let j range over subintervals I_j, and let k range over the time points t_k where a HI-criticality job arrives.

Variables. Our LP uses the following **variables**:

1. The variable x_{ij} represents the amount of execution assigned to the i'th job J_i in the j'th interval I_j, in a schedule in which no HI-criticality job signals the transition to HI-criticality mode. There are $O(n^2)$ such variables.
2. For each k, the variable $y_{ij}^{(k)}$ represents the amount of execution assigned to job J_i in the interval I_j, in a schedule in which HI-criticality mode is signalled at time-instant t_k. There are $O(n^3)$ such variables – $O(n^2)$ for each value of k.

For the example instance of Example 1 there are $3 \times 3 = 9$ x_{ij} variables, and the same number of $y_{ij}^{(1)}$ variables.

Constructing Scheduling Tables. A solution to our LP will assign values to these variables. We will use these assigned values to construct several scheduling tables prior to run-time:

- Scheduling table S_o will schedule job J_i for a duration x_{ij} during the interval I_j.
- For each k, there will be a scheduling table S_k that schedules job J_i for a duration $y_{ij}^{(k)}$ during the interval I_j.

Our run-time scheduling strategy is then to start out making scheduling decisions according to S_o, and to switch to S_k if HI-criticality mode is first signalled by the arrival of a HI-criticality job at t_k. The Constraints in Eq. 1 below enforce that $y_{ij}^{(k)} = x_{ij}$ for all intervals I_j before t_k; hence in the event that the arrival of a HI-criticality job at time t_k signals HI-criticality mode we are effectively following scheduling table S_k from the very beginning.

▶ **Example 2.** For the 3-job example instance in Example 1, there are three subintervals $I_1 = [0, 1]$, $I_2 = [1, 2]$, and $I_3 = [2, 3]$. Since there is only one HI-criticality job, we have $t_1 = 1$ as the only instant at which HI-criticality jobs arrive. Below is a possible assignment of values for x_{ij} and $y_{ij}^{(1)}$ for this example instance (these happen to be the x_{ij} and $y_{ij}^{(1)}$ values corresponding to the correct scheduling strategy described in Example 1):

	$j = 1$	$j = 2$	$j = 3$
$i = 1$	$x_{11} = 0, y_{11}^{(1)} = 0$	$x_{12} = 1, y_{12}^{(1)} = 0$	$x_{13} = 0, y_{13}^{(1)} = 0$
$i = 2$	$x_{21} = 1, y_{21}^{(1)} = 1$	$x_{22} = 0, y_{22}^{(1)} = 0$	$x_{23} = 1, y_{23}^{(1)} = 0$
$i = 3$	$x_{31} = 0, y_{31}^{(1)} = 0$	$x_{32} = 0, y_{32}^{(1)} = 1$	$x_{33} = 0, y_{33}^{(1)} = 1$

The following scheduling tables are constructed from these $x_{i,j}$ and $y_{i,j}$ values:

1. S_o schedules J_2 over $I_1 = [0, 1]$, J_1 over $I_2 = [1, 2]$, and J_2 over $I_3 = [2, 3]$.
2. S_1 schedules J_2 over I_1, and J_3 over both I_2 and I_3.

During run-time we start out scheduling according to S_o; if job J_3 signals HI-criticality mode upon arrival at time-instant 1 then we subsequently switch to schedule S_1. ◀

Constraints. We will now describe the **constraints** added to our LP in order to ensure that the variables defined above have their intended interpretations.

- (Non-clairvoyance.) For each k, for all i, and for all j such that the interval I_j completes no later than time-instant t_k, we have

$$y_{ij}^{(k)} = x_{ij} \tag{1}$$

There are $O(n^3)$ such constraints.

- (Correctness in LO criticality.) For each i

$$\sum_{I_j \subseteq [r_i, d_i]} x_{ij} \geq c_i(\text{LO}) \tag{2}$$

There are $O(n)$ such constraints – one per job.

- (Correctness in HI criticality.) For each k (HI-criticality signalled at t_k)
  - For each i for which $\chi_i = \text{HI}$

$$\sum_{I_j \subseteq [r_i, d_i]} y_{ij}^{(k)} \geq \begin{cases} c_i(\text{LO}) & \text{if } r_i < t_k \\ c_i(\text{HI}) & \text{if } r_i \geq t_k \end{cases} \tag{3}$$

There are $O(n^2)$ such constraints.

  - For each i for which $\chi_i = \text{LO}$

$$\sum_{I_j \subseteq [r_i, d_i]} y_{ij}^{(k)} \geq \begin{cases} c_i(\text{LO}) & \text{if } d_i \leq t_k \\ c_i(\text{HI}) & \text{if } d_i > t_k \end{cases} \tag{4}$$

There are $O(n^2)$ such constraints.

Notice the difference between Expression 3 and Expression 4: the first case in Expression 3 applies to all jobs that *arrive* before t_k; in Expression 4, to all jobs that *have deadlines* no later than t_k.

- (Adequate computing capacity to construct scheduling table S_o.) For each j

$$\sum_i x_{ij} \leq |I_j|, \tag{5}$$

where $|I|$ denotes the length of an interval I. There are $O(n)$ such constraints.

- (Adequate computing capacity to construct scheduling table S_k.) For each k, for each j

$$\sum_i y_{ij}^{(k)} \leq |I_j| \tag{6}$$

There are $O(n^2)$ such constraints.

3.2 Tasks

We now give an optimal algorithm for scheduling systems of *implicit-deadline* sporadic tasks[2] – task systems in which the relative deadline parameter D_i of each task τ_i is equal to its period parameter T_i. Our algorithm is based on the fluid scheduling paradigm. Such algorithms are allowed to assign individual tasks a fraction ≤ 1 of a processor (rather than an entire processor, or none) at each instant in time. The MC-Fluid non-clairvoyant scheduling algorithm [26, 9] was designed for scheduling dual-criticality implicit-deadline sporadic task systems upon identical multiprocessor platforms. Prior to run-time, MC-Fluid computes LO-criticality and HI-criticality *execution rates* $\theta_i(\text{LO})$ and $\theta_i(\text{HI})$ for each task $\tau_i \in \tau$. Each task τ_i is initially scheduled at a rate $\theta_i(\text{LO})$; if any job does not complete despite having

[2] We do not yet have an algorithm for scheduling task systems in which the D_i and T_i parameters of individual tasks may differ—to our knowledge, there are no non-trivial speedup-competitive prior results known for semi-clairvoyant scheduling of task systems that are not implicit-deadline. In Section 6 we will describe how a correct but non-optimal algorithm may be obtained for scheduling such systems.

executed for its LO-criticality WCET, all LO-criticality tasks are immediately discarded and each HI-criticality task τ_i henceforth executes at a rate $\theta_i(\text{HI})$. An algorithm for computing suitable values for the $\theta_i(\text{LO})$ and $\theta_i(\text{HI})$ parameters is presented in [26], and a somewhat simpler algorithm subsequently derived in [9], and shown to be speedup-optimal[3], with speedup factor $\frac{4}{3}$.

The non-clairvoyant MC-Fluid algorithm is easily modified to form an optimal semi-clairvoyant algorithm for dual criticality implicit-deadline tasks upon uniprocessors. Observe first that for an implicit-deadline sporadic task system τ to be schedulable by <u>any</u> scheduler (including a clairvoyant one), it is necessary that the following condition hold:

$$U_{\text{LO}} \leq 1 \text{ and } U_{\text{HI}} \leq 1 \tag{7}$$

The schedulability test associated with our optimal semi-clairvoyant scheduling algorithm is straightforward: any task system τ satisfying the condition above will be correctly scheduled by our algorithm. The algorithm assigns the $\theta_i(\text{LO})$ and $\theta_i(\text{HI})$ execution rates to each task τ_i as follows:

$$\Big(\theta_i(\text{LO}) = U_i(\text{LO})\Big) \text{ and } \Big(\theta_i(\text{HI}) = U_i(\text{HI})\Big)$$

(We point out that therefore $\theta_i(\text{HI}) \leq \theta_i(\text{LO})$ for LO-criticality tasks while $\theta_i(\text{HI}) \geq \theta_i(\text{LO})$ for HI-criticality tasks.) It initially executes each task τ_i at a rate $\theta_i(\text{LO})$; if any HI-criticality job signals a transition to HI-criticality mode upon arrival, the algorithm subsequently executes each task τ_i at a rate $\theta_i(\text{HI})$. It is evident that this algorithm is feasible upon a uniprocessor since the rates of all the tasks both before and after such a transition sum to ≤ 1.

Proof of Correctness. We show that any task system τ satisfying the necessary schedulability conditions in Eq. 7 is scheduled correctly as per correctness criterion CC-1.

It is evident that all tasks execute correctly in all LO-criticality behaviors (since each job of each task τ_i receives a total execution $\theta_i(\text{LO}) \times T_i = U_i(\text{LO}) \times T_i = C_i(\text{LO})$ units of execution). Consider now some HI-criticality behavior, and let t_{switch} denote the instant at which HI-criticality behavior is signalled. It is evident that any job that has both its arrival time and its deadline $\leq t_{\text{switch}}$, as well as any job that has both its arrival time and its deadline $\geq t_{\text{switch}}$, receives adequate execution. It remains to consider jobs that arrive before, but have deadline after, time-instant t_{switch}.

HI-**criticality tasks.** Under semi-clairvoyant scheduling, all such HI-criticality jobs, having arrived before HI-criticality mode was signalled, are guaranteed to complete upon having executed for no more than their LO-criticality WCETs. Since $\theta_i(\text{LO}) \leq \theta_i(\text{HI})$ for any HI-criticality task τ_i, any job of such a τ_i clearly receives at least $\theta_i(\text{LO}) \times T_i = C_i(\text{LO})$ units of execution by its deadline.

LO-**criticality tasks.** Symmetrically to the case above, under correctness criterion CC-1 all such LO-criticality jobs are to be assigned an amount of execution not exceeding their HI-criticality WCETs. Since $\theta_i(\text{LO}) \geq \theta_i(\text{HI})$ for any LO-criticality task τ_i, any job of such a τ_i receives at least $\theta_i(\text{HI}) \times T_i = C_i(\text{HI})$ units of execution by its deadline.

4 Correctness Criterion CC-2

LO-criticality jobs that begin executing before a mode-transition has been signalled are required to execute for their (larger) LO-criticality WCETs under correctness criterion CC-2.

[3] The reader is referred to [25, 15] for in-depth discussions about speedup factors.

It turns out that establishing the schedulability of systems under this correctness criterion is a computationally harder problem than under correctness criterion CC-1: while we saw above (Section 3.1) that schedulability analysis of collections of jobs can be done in polynomial time under CC-1, we now show that this is NP-complete in the strong sense under CC-2.

▶ **Theorem 3.** *Under correctness criterion CC-2, it is NP-hard in the strong sense to determine whether an instance of jobs can be correctly scheduled upon a preemptive uniprocessor.*

Proof. We will give a reduction from the 3-PARTITION problem, which is well known to be NP-complete in the strong sense [21]. An instance of 3-PARTITION is a set $S = \{s_1, s_2, \ldots, s_{3m}\}$ of $3m$ positive integers, such that $\sum_{s_i \in S} s_i = mk$ for some integer k. We are asked whether S can be partitioned into m disjoint subsets, such that each subset sums to k.

Let an instance $S = \{s_1, s_2, \ldots, s_{3m}\}$ of 3-PARTITION be given. We create the following set $\mathcal{J}$ of jobs.

$$J_i = (\text{LO}, 0, [2s_i, s_i], 2mk), \qquad \text{for } 1 \le i \le 3m$$
$$J_{3m+j} = (\text{HI}, 2jk, [0, k], (2j+1)k), \qquad \text{for } 1 \le j < m.$$

It is clear that this reduction can be carried out in polynomial time and that the values of the produced numerical parameters are polynomially related to those given. In the following we show that $\mathcal{J}$ is schedulable under correctness criterion CC-2 if and only if S can be partitioned into subsets $S_0, S_1, \ldots, S_{m-1}$ with the same sum. We consider the two directions separately. Let $\mathcal{J}_{\text{LO}}$ and $\mathcal{J}_{\text{HI}}$ denote the sets of LO- and HI-criticality jobs in $\mathcal{J}$, respectively.

<u>S can be partitioned $\Rightarrow$ $\mathcal{J}$ is schedulable:</u> Assume S can be partitioned into $S_0, S_1, \ldots, S_{m-1}$ such that each partition sums to k. Let $\mathcal{J}_j$ denote the set of jobs $J_i \in \mathcal{J}_{\text{LO}}$ such that $s_i \in S_j$, for $0 \le j < m$. That is, $J_i \in \mathcal{J}_j$ if and only if $s_i \in S_j$.

The following scheduling strategy will be successful: We allocate time interval $[0, 2k)$ for the jobs in $\mathcal{J}_0$ where they are executed in any order. Then for all j such that $1 \le j < m$ we allocate time interval $[2jk, 2(j+1)k)$ for first executing HI-criticality job J_{3m+j} and then the jobs in $\mathcal{J}_j$ in any order. If the jobs that are allocated to a time interval have finished early, then we idle the processor until the start of the next allocated time interval.

We note that the total execution time of the jobs in $\mathcal{J}_0$ is at most $\sum_{J_i \in \mathcal{J}_0} c_i(\text{LO}) = \sum_{s_i \in S_0} 2s_i = 2k$, and therefore they can all finish in their allocated time interval $[0, 2k)$.

We then note that for $1 \le j < m$, if neither J_{3m+j} nor any previous HI-criticality job has signaled HI-criticality behavior, then the LO-criticality jobs in $\mathcal{J}_j$ have the entire time interval $[2jk, 2(j+1)k)$ to execute in, which is enough since $\sum_{J_i \in \mathcal{J}_j} c_i(\text{LO}) = 2k$. If, on the other hand, J_{3m+j} or some previous HI-criticality job has signaled HI-criticality behavior, then the jobs in $\mathcal{J}_j$ might only have the time interval $[(2j+1)k, 2(j+1)k)$ to execute in, but this is still enough as they then only need to execute for up to $\sum_{J_i \in \mathcal{J}_j} c_i(\text{HI}) = k$.

The HI-criticality tasks J_{3m+j}, for $1 \le j < m$, all execute first in their allocated time intervals. Those time intervals start at the arrival time of the corresponding HI-criticality job, so the HI-criticality jobs will always finish no later than their deadlines.

<u>S cannot be partitioned $\Rightarrow$ $\mathcal{J}$ is unschedulable:</u> Assume that S cannot be partitioned into m subsets of equal sum. We will show that no matter what scheduling decisions are taken, there will always exist some runtime behaviors that lead to a deadline miss. In the following we consider only behaviors where each job J_i requires an execution time of either exactly $c_i(\text{LO})$ or $c_i(\text{HI})$. We then note that no job has a deadline later than $2mk$ and that $\sum_{J_i \in \mathcal{J}_{\text{LO}}} c_i(\text{LO}) = 2mk$. Therefore, no idle time can possibly be allowed in a successful schedule as long as HI-criticality behavior has not been signaled. In time interval $[0, 2k)$,

LO-criticality jobs must then be scheduled for the entire $2k$ time units. Let $\mathcal{J}_{\text{LO}}^{\text{start}}$ be the jobs that begin execution in $[0, 2k)$, and let $x = \sum_{J_i \in \mathcal{J}_{\text{LO}}^{\text{start}}} c_i(\text{LO})$. We consider two cases.

Case 1 ($x > 2k$) Suppose the HI-criticality job J_{3m+1} that arrives at time $2k$ signals HI-criticality behavior, and that this and all following HI-criticality jobs require their HI-criticality WCETs. The total execution time of the HI-criticality jobs is then $(m-1)k$, and the total time left over for the LO-criticality jobs until their deadline at $2mk$ is $2mk - 2k - (m-1)k = (m-1)k$. However, the LO-criticality jobs that started execution before time $2k$ require their larger LO-criticality WCET. The total remaining execution time requirement of the LO-criticality jobs is then

$$\begin{aligned}\sum_{J_i \in \mathcal{J}_{\text{LO}} \setminus \mathcal{J}_{\text{LO}}^{\text{start}}} c_i(\text{HI}) + \sum_{J_i \in \mathcal{J}_{\text{LO}}^{\text{start}}} c_i(\text{LO}) - 2k &= \sum_{J_i \in \mathcal{J}_{\text{LO}}} c_i(\text{HI}) + \sum_{J_i \in \mathcal{J}_{\text{LO}}^{\text{start}}} c_i(\text{HI}) - 2k \\ &= mk + \frac{x}{2} - 2k \\ &> (m-1)k,\end{aligned}$$

and they can not all finish before their deadline.

Case 2 ($x = 2k$) Since $\sum_{J_i \in \mathcal{J}_{\text{LO}}^{\text{start}}} c_i(\text{LO}) = 2k$, the jobs in $\mathcal{J}_{\text{LO}}^{\text{start}}$ correspond to a subset of S that sums to k. In this case, let the HI-criticality job J_{3m+1} that arrives at time $2k$ require zero execution time and therefore *not* signal HI-criticality behavior.

If the second case holds, we simply note that the LO-criticality jobs that were executed corresponds to a subset of S that sums to k and repeat the same argument, but starting from time $2k$ instead. If again the second case holds, we note that the rest of S again has a subset that sums to k and repeat the argument from time $4k$ and so on. Since by assumption S cannot be partitioned completely into subsets that each sum to k, any scheduler must eventually either idle the processor or behave according to Case 1 above, both of which can then lead to a deadline miss. ◀

An MILP for schedulability analysis under CC-2. Theorem 3 above establishes that we cannot analyze schedulability of collections of jobs under CC-2 even in pseudo-polynomial time (assuming P≠NP). This means that we cannot solve it with a polynomially-sized LP, but below we adapt the LP obtained in Section 3.1 for correctness criterion CC-1 to make it applicable to schedulability analysis under correctness criterion CC-2 by introducing some additional *binary integer variables*, which in effect turns the LP into a mixed-integer linear program (MILP). Since it is known [11] that determining whether an MILP has a feasible solution is in NP, the existence of this MILP also serves to show that schedulability analysis of collections of jobs under correctness criterion CC-2 is in the complexity class NP. This fact, in conjunction with Theorem 3, establishes that the problem is NP-complete.

Observe that the difference between correctness criteria CC-1 and CC-2 is in the execution time that LO-criticality jobs may require when some HI-criticality job signals HI-criticality behavior. This was captured by the constraints in Eq. 4 for CC-1; reproduced below:
For each i for which $\chi_i = \text{LO}$

$$\sum_{I_j \subseteq [r_i, d_i]} y_{ij}^{(k)} \geq \begin{cases} c_i(\text{LO}) & \text{if } d_i \leq t_k \\ c_i(\text{HI}) & \text{if } d_i > t_k \end{cases}$$

We replace the constraints in Eq. 4 by the following set of constraints, keeping the other constraints of the LP in Section 3.1 as they are.

For each LO-criticality job J_i and time instant t_k where some HI-criticality job arrives, let $b_i^{(k)}$ be a new binary (i.e., 0–1 valued) integer variable. The intended interpretation is that job J_i has started execution before t_k in schedule S_o if and only if $b_i^{(k)} = 1$. Instead of the constraints in Eq. 4 we add the following constraints.

For each k (HI-criticality signalled at t_k) and each i for which $\chi_i = \text{LO}$:

- If $d_i \leq t_k$ (i.e., the entire job must be scheduled by time-instant t_k),

$$\sum_{I_j \subseteq [r_i, d_i]} y_{ij}^{(k)} \geq c_i(\text{LO}) \tag{8}$$

- If $r_i > t_k$ (i.e., the entire job must be scheduled after time-instant t_k),

$$\sum_{I_j \subseteq [r_i, d_i]} y_{ij}^{(k)} \geq c_i(\text{HI}) \tag{9}$$

- Otherwise (i.e., the job spans t_k),

$$\sum_{I_j \subseteq [r_i, d_i]} y_{ij}^{(k)} \geq c_i(\text{LO}) \times b_i^{(k)} \tag{10}$$

$$\sum_{I_j \subseteq [t_k, d_i]} y_{ij}^{(k)} \geq c_i(\text{HI}) \times (1 - b_i^{(k)}) \tag{11}$$

$$\sum_{I_j \subseteq [r_i, t_k]} y_{ij}^{(k)} \leq M \times b_i^{(k)}, \tag{12}$$

where M is some large enough positive constant (e.g., $M = \max_i(c_i(\text{LO}))$ can be used). We note that if $b_i^{(k)} = 1$, then the constraint in Eq. 10 "forces" job J_i to be allocated at least $c_i(\text{LO})$ units of execution, while the constraints in Eqs. 11 and 12 are trivially satisfied. If instead $b_i^{(k)} = 0$, then the constraint in Eq. 10 is always satisfied, while the constraint in Eq. 11 forces J_i to be allocated at least $c_i(\text{HI})$ units of execution after t_k, and the constraint in Eq. 12 forces J_i to not have begun execution before t_k.

5 Correctness Criterion CC-3

Under correctness criterion CC-3, it is required that all LO-criticality jobs arriving before mode-change is signalled receive an amount of service equal to their (larger) LO-criticality WCETs. It is easily seen that unlike with regards to correctness criteria CC-1 and CC-2, under correctness criterion CC-3 *the WCET of each job is known upon the job's arrival* regardless of whether some future job will signal HI-criticality mode upon arrival or not. And it follows from the optimality property of the Earliest Deadline First scheduling algorithm (EDF) upon preemptive uniprocessor platforms [27, 17] that any collection of such jobs that can be scheduled to all complete by their deadlines is scheduled to all complete by their deadlines by EDF. Therefore, under correctness criterion CC-3 *EDF is an optimal run-time algorithm*. This is in contrast to correctness criteria CC-1 and CC-2, for neither of which do we have a general, efficient, run-time scheduling algorithm. In the remainder of this section we derive efficient schedulability-analysis for the EDF scheduling of dual-criticality instances of independent jobs (Section 5.1), and three-parameter sporadic task systems (Section 5.2) under this correctness criterion.

5.1 Jobs

In Sections 3.1 and 4, we had solved a linear program and an MILP respectively in order to construct scheduling tables for the run-time scheduling of collections of jobs subject to correctness criteria CC-1 and CC-2. Such an approach is not necessary for CC-3: as observed above, we know that EDF is an optimal algorithm for scheduling instances subject to correctness criterion CC-3. An associated schedulability test is easily obtained: simply simulate the EDF scheduling of the instance multiple times, once assuming LO-criticality behavior and once each under the assumption that each individual HI-criticality job is the one that signals transition to HI-criticality behavior.[4] For an instance with n jobs, $O(n)$ such simulations of EDF need to be performed; each can be done in $O(n \log n)$ time [28], yielding an overall complexity of $O(n^2 \log n)$ for the schedulability test.

5.2 Tasks

As stated above, EDF is an optimal run-time scheduling algorithm under correctness criterion CC-3. We now derive an exact EDF schedulability test for 3-parameter sporadic task systems under this correctness criterion; our test holds for "arbitrary-deadline" systems — i.e., systems in which tasks may have relative deadlines smaller than, equal to, or larger than their periods. We will show that our schedulability test has pseudo-polynomial running time upon systems in which $\max(U_{\text{LO}}, U_{\text{HI}})$ is a priori bounded by some constant $c < 1$.

Demand bound function. Let $\text{DBF}_i(t, s)$ denote the *demand bound function* [2] (see [3, Chapter 10.3] for a text-book description) of task τ_i in an interval of length t, where HI-criticality mode is first signalled s time units into the interval (possibly by some other task). That is, the function $\text{DBF}_i(t, s)$ bounds the maximum sum of execution times of jobs from τ_i that have both release times and deadlines within any such interval.

Let t and s be given. We make the following observations (illustrated in Figure 1).

- For a HI-criticality task τ_i, the execution demand is maximized when as many jobs as possible fit into the interval, and as many of those as possible are released at or after the signalling of HI-criticality mode, and therefore can have the larger WCET $C_i(\text{HI})$. This corresponds to a scenario where one job from τ_i has its deadline at the end of the interval, and the previous jobs are each released as late as possible.
- For a LO-criticality task τ_i, the execution demand is instead maximized when the maximum number of jobs from τ_i fit into the interval, but as many as possible are released before the time instant where HI-criticality mode is signalled, and therefore can have the larger WCET $C_i(\text{LO})$. This corresponds to a scenario where one job is released at the start of the interval and subsequent jobs as early as possible.

We use the above observations to express $\text{DBF}_i(t, s)$. Let $\psi_i(t)$ denote the maximum number of jobs of τ_i that both arrive in, and have their deadlines within, any contiguous interval of duration t; it is known [2] that

$$\psi_i(t) = \max\left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0\right).$$

[4] It follows from the *sustainability* property of EDF [6] that each such simulation can be done assuming that each job executes to exactly its WCET.

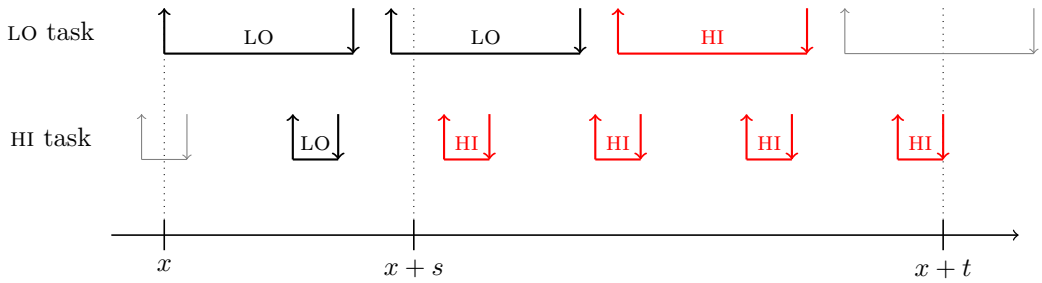

Figure 1 An interval of length t with HI-criticality mode revealed s time units into the interval. The LO-criticality task maximizes execution demand within the interval by fitting two jobs with WCET $C(\text{LO})$ and one job with WCET $C(\text{HI})$. The HI-criticality task maximizes demand by fitting four jobs with WCET $C(\text{HI})$ and one with WCET $C(\text{LO})$.

For a HI-criticality task τ_i, a total of $\psi_i(t)$ jobs can fit inside the interval, and a total of $\psi_i(t-s)$ of those jobs can have an execution time requirement of $C_i(\text{HI})$. We therefore have

$$\text{DBF}_i(t,s) \;=\; \psi_i(t) \times C_i(\text{LO}) + \psi_i(t-s) \times (C_i(\text{HI}) - C_i(\text{LO})).$$

For a LO-criticality task τ_i the number of jobs that can fit in the interval is also at most $\psi_i(t)$. No more than $\lfloor s/T_i \rfloor + 1$ of those jobs can be released before[5] the instant when HI-criticality behavior is first signaled, and therefore have the larger execution time requirement $C_i(\text{LO})$. We have

$$\text{DBF}_i(t,s) \;=\; \psi_i(t) \times C_i(\text{HI}) + \min\left(\psi_i(t), \left\lfloor \frac{s}{T_i} \right\rfloor + 1\right) \times (C_i(\text{LO}) - C_i(\text{HI})).$$

Putting the above together, we have the following expression to bound the maximum total execution time demand of jobs from task τ_i in an interval of size t, where HI-criticality behavior is first revealed s time units into the interval.

$$\text{DBF}_i(t,s) = \begin{cases} \psi_i(t) \times C_i(\text{LO}) + \psi_i(t-s) \times (C_i(\text{HI}) - C_i(\text{LO})), & \text{if } \chi_i = \text{HI} \\ \psi_i(t) \times C_i(\text{HI}) + \min\big(\psi_i(t), \lfloor \frac{s}{T_i} \rfloor + 1\big) \times (C_i(\text{LO}) - C_i(\text{HI})), & \text{if } \chi_i = \text{LO} \end{cases} \tag{13}$$

The Schedulability Test

We assume here that $\max(U_{\text{LO}}, U_{\text{HI}}) < 1$. Before establishing the schedulability test we present three lemmas. Let B and $\mathcal{S}(t)$ be defined for any task system τ as follows.

$$B \;=\; \frac{\sum_{\tau_i \in \tau} C_i(\chi_i)}{1 - \max(U_{\text{LO}}, U_{\text{HI}})},$$

$$\mathcal{S}(t) \;=\; \bigcup_{\tau_i \in \tau^{(\text{HI})}} \{t - kT_i - D_i \mid 0 \le k < \psi_i(t)\} \cup \{t\}.$$

[5] In fact, this captures the maximum number of jobs that can be released before *or at* the time point where HI-criticality is signaled. It seems as if we should replace $\lfloor s/T_i \rfloor + 1$ by $\lceil s/T_i \rceil$, but we will see later that this is in fact the suitable formulation in order to use the demand bound function for an efficient schedulability test that is both sufficient and necessary. Specifically, this formulation is required for Lemma 6, but we will see in Theorem 7 that it does not detract from the exactness of the test.

As we will see in the following, B is the upper bound for the values of t that we need to consider when using $\text{DBF}_i(t,s)$ for a schedulability test, and $\mathcal{S}(t)$ is the set of values for s that needs to be considered for each t. In our first lemma we show that the demand bound function for a task set is maximized at some value of $s \in \mathcal{S}(t)$, which corresponds to a release of a job from a HI-criticality task when its jobs are aligned as in Figure 1.

▶ **Lemma 4.** *Let t and s be given, where $t > 0$ and $s \in [0,t]$. Then there exists $s' \in \mathcal{S}(t)$ such that $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) \leq \sum_{\tau_i \in \tau} \text{DBF}_i(t,s')$.*

Proof. Let s' be the smallest $s' \in \mathcal{S}(t)$ such that $s' \geq s$.

For all $\tau_i \in \tau^{(\text{HI})}$, the set of values s' where $\psi_i(t-s')$ is discontinuous in the interval $[0,t]$ is a subset of $\mathcal{S}(t)$. As ψ_i is a right-continuous step function, we must have $\psi_i(t-s) = \psi_i(t-s')$. Hence, $\text{DBF}_i(t,s) = \text{DBF}_i(t,s')$ if $\chi_i = \text{HI}$.

If $\tau_i \in \tau^{(\text{LO})}$, then $\text{DBF}_i(t,s)$ in non-decreasing in s. Therefore $\text{DBF}_i(t,s) \leq \text{DBF}_i(t,s')$ if $\chi_i = \text{LO}$, which completes the proof. ◀

Our second lemma puts an upper bound on the values of t that need to be considered.

▶ **Lemma 5.** *If $t \geq B$, then $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) \leq t$ for all $s \in [0,t]$.*

Proof. Take any $\tau_i \in \tau$. We first note that $\psi_i(x) \leq x/T_i + 1$ and then consider two cases.

Case 1 ($\chi_i = \text{HI}$): We must have

$$\begin{aligned}
\text{DBF}_i(t,s) &\leq \left(\frac{t}{T_i}+1\right) \times C_i(\text{LO}) + \left(\frac{t-s}{T_i}+1\right) \times (C_i(\text{HI}) - C_i(\text{LO})) \\
&= \frac{C_i(\text{LO})}{T_i} \times t + C_i(\text{LO}) + \frac{C_i(\text{HI}) - C_i(\text{LO})}{T_i} \times (t-s) + C_i(\text{HI}) - C_i(\text{LO}) \\
&= U_i(\text{HI}) \times (t-s) + U_i(\text{LO}) \times s + C_i(\text{HI}).
\end{aligned}$$

Case 2 ($\chi_i = \text{LO}$): Similarly, we have

$$\begin{aligned}
\text{DBF}_i(t,s) &\leq \left(\frac{t}{T_i}+1\right) \times C_i(\text{HI}) + \left(\frac{s}{T_i}+1\right) \times (C_i(\text{LO}) - C_i(\text{HI})) \\
&= \frac{C_i(\text{HI})}{T_i} \times t + C_i(\text{HI}) + \frac{C_i(\text{LO}) - C_i(\text{HI})}{T_i} \times s + C_i(\text{LO}) - C_i(\text{HI}) \\
&= U_i(\text{HI}) \times (t-s) + U_i(\text{LO}) \times s + C_i(\text{LO}).
\end{aligned}$$

In both cases we then have

$$\text{DBF}_i(t,s) \leq U_i(\text{HI}) \times (t-s) + U_i(\text{LO}) \times s + C_i(\chi_i)$$

and therefore

$$\begin{aligned}
\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) &\leq U_{\text{HI}} \times (t-s) + U_{\text{LO}} \times s + \sum_{\tau_i \in \tau} C_i(\chi_i) \\
&\leq \max(U_{\text{LO}}, U_{\text{HI}}) \times t + \sum_{\tau_i \in \tau} C_i(\chi_i).
\end{aligned}$$

But if $t \geq B$, then

$$\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) \leq \max(U_{\text{LO}}, U_{\text{HI}}) \times t + \sum_{\tau_i \in \tau} C_i(\chi_i) \leq t,$$

which concludes the proof. ◀

Our third lemma puts the above two together to limit the values of both t and s that we must consider.

▶ **Lemma 6.** *If $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) > t$ for some $t > 0$ and $s \in [0,t]$, then there exists $t' \in \{0,1,\ldots,\lfloor B \rfloor\}$ and $s' \in \mathcal{S}(t')$ such that $\sum_{\tau_i \in \tau} \text{DBF}_i(t',s') > t'$.*

Proof. Assume that $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) > t$ for some $t > 0$ and $s \in [0,t]$. By Lemma 4 there exists $s' \in S(t)$ such that $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s') \geq \sum_{\tau_i \in \tau} \text{DBF}_i(t,s)$.

Let $t' = \lfloor t \rfloor$ and $s'' = \lfloor s' \rfloor$. We note then that for any task τ_i, we have $\psi_i(t) = \psi_i(t')$ as ψ_i is a right-continuous step function that only changes value at integers. Further, we note that by definition of $\mathcal{S}(t)$ it must be the case that the fractional parts of t and s' are the same. We must then have $t - s' = t' - s''$ and $\psi_i(t - s') = \psi_i(t' - s'')$. Also, we note that as T_i is integer we must have $\lfloor s/T_i \rfloor = \lfloor s'/T_i \rfloor$. From Eq. 13 it follows then that $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s') = \sum_{\tau_i \in \tau} \text{DBF}_i(t',s'')$.

Finally, by Lemma 4 there must exist $s''' \in \mathcal{S}(t')$ such that $\sum_{\tau_i \in \tau} \text{DBF}_i(t',s'') \leq \sum_{\tau_i \in \tau} \text{DBF}_i(t',s''')$. Putting the above together we have

$$\begin{aligned}
\sum_{\tau_i \in \tau} \text{DBF}_i(t',s''') &\geq \sum_{\tau_i \in \tau} \text{DBF}_i(t',s'') \\
&= \sum_{\tau_i \in \tau} \text{DBF}_i(t,s') \\
&\geq \sum_{\tau_i \in \tau} \text{DBF}_i(t,s) \\
&> t \\
&\geq t'
\end{aligned}$$

By Lemma 5 we must have $t < B$ and therefore $t' \in \{0,1,\ldots,\lfloor B \rfloor\}$. As $s''' \in \mathcal{S}(t')$, the lemma follows. ◀

We can now establish the schedulability test.

▶ **Theorem 7.** *Let τ be a task set of arbitrary-deadlines sporadic mixed-criticality tasks with $\max(U_{\text{LO}}, U_{\text{HI}}) < 1$. The task set τ is schedulable by EDF under correctness criterion CC-3 on a single preemptive processor if and only if*

$$\forall t \in \{0,1,2,\ldots,\lfloor B \rfloor\}, \forall s \in \mathcal{S}(t): \quad \sum_{\tau_i \in \tau} \text{DBF}_i(t,s) \leq t.$$

Proof. We separately prove the necessity and sufficiency of the schedulability test.

Test fails $\Rightarrow \tau$ is unschedulable: Assume there exists $t \in \{1,2,\ldots,\lfloor B \rfloor\}$ and $s \in \mathcal{S}(t)$ such that $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) > t$. Let τ release jobs in an interval of length $t + \epsilon$, for some $0 < \epsilon < 1$, such that all LO-criticality tasks release a job at the start of the interval and then subsequent jobs as early as possible. Let HI-criticality tasks instead release jobs such that one job has a deadline at the end of the interval, and previous jobs are released as late as possible in the interval.

By definition of $\mathcal{S}(t)$, at least one HI-criticality task must then release a job exactly $s + \epsilon$ time units into the interval. Let that job be the first to signal HI-criticality behavior, and let all other jobs require their largest allowed execution time.

Any HI-criticality task τ_i then releases $\psi_i(t+\epsilon)$ jobs with both release times and deadlines within the interval, of which $\psi_i(t + \epsilon - (s + \epsilon))$ are released at or after the time point where

HI-criticality behavior is signaled, for a total execution time requirement of

$$\begin{aligned} & \psi_i(t+\epsilon) \times C_i(\text{LO}) + \psi_i(t+\epsilon-(s+\epsilon)) \times (C_i(\text{HI}) - C_i(\text{LO})) \\ = \; & \psi_i(t) \times C_i(\text{LO}) + \psi_i(t-s) \times (C_i(\text{HI}) - C_i(\text{LO})) \\ = \; & \text{DBF}_i(t,s), \end{aligned}$$

where $\psi_i(t+\epsilon) = \psi(t)$ since t is integer and ψ_i only changes value at integers.

Similarly, any LO-criticality task τ_i will release $\psi_i(t+\epsilon)$ jobs in total, of which at most $\lceil (s+\epsilon)/T_i \rceil$ are released *before* the time point where HI-criticality behavior is signaled, for a total execution time requirement of

$$\begin{aligned} & \psi_i(t+\epsilon) \times C_i(\text{HI}) + \min\Big(\psi_i(t+\epsilon), \left\lceil \frac{s+\epsilon}{T_i} \right\rceil\Big) \times (C_i(\text{LO}) - C_i(\text{HI})) \\ = \; & \psi_i(t) \times C_i(\text{HI}) + \min\Big(\psi_i(t), \left\lfloor \frac{s}{T_i} \right\rfloor + 1\Big) \times (C_i(\text{LO}) - C_i(\text{HI})) \\ = \; & \text{DBF}_i(t,s), \end{aligned}$$

where $\lceil (s+\epsilon)/T_i \rceil = \lfloor (s+\epsilon)/T_i \rfloor + 1 = \lfloor s/T_i \rfloor + 1$ since both s and T_i are integer.

The total workload of jobs with both release time and deadline within the interval of size $t+\epsilon$ is then $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s)$. Since $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s)$ is integer-valued and since $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) > t$ by assumption, we must also have $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) > t+\epsilon$. It follows that the total workload that must be scheduled inside the interval is greater than the length of the interval, hence it is impossible to meet all deadlines on a single processor.

<u>τ is unschedulable $\Rightarrow$ test fails:</u> Assume that τ is unschedulable by EDF and let t_2 be the time point of a deadline miss. Let t_1 be the earliest time point before t_2 such that there exists at least one active job with deadline no later than t_2 at any time in the interval $[t_1, t_2]$. By definition of t_1, there are no active jobs in $[t_1, t_2]$ with deadline latest at t_2 that are also released earlier than t_1. It follows that EDF schedules jobs that both arrive no earlier than t_1 and have deadline no later than t_2 during the entirety of $[t_1, t_2]$. Still one of those jobs misses its deadline at t_2, so the total workload of those jobs must exceed $t_2 - t_1$.

We let $t = t_2 - t_1$ and consider three cases.

- HI-*criticality behavior is signaled before* t_1*:* The total workload of all jobs scheduled by EDF in $[t_1, t_2]$ can be no more than $\sum_{\tau_i \in \tau} \psi_i(t) \times C_i(\text{HI})$. By Eq. 13 we have

$$\sum_{\tau_i \in \tau} \psi_i(t) \times C_i(\text{HI}) \leq \sum_{\tau_i \in \tau} \text{DBF}_i(t,0).$$

- HI-*criticality behavior has not been signaled by* t_2*:* The total workload of all jobs scheduled by EDF in $[t_1, t_2]$ can be no more than $\sum_{\tau_i \in \tau} \psi_i(t) \times C_i(\text{LO})$. Using $\psi_i(t) \leq \lfloor t/T_i \rfloor + 1$ and Eq. 13 we get

$$\sum_{\tau_i \in \tau} \psi_i(t) \times C_i(\text{LO}) = \sum_{\tau_i \in \tau} \text{DBF}_i(t,t).$$

- HI-*criticality behavior is first signaled in* $[t_1, t_2]$*:* Let t_{signal} be the time point where HI-criticality behavior is first signaled. The total workload of all jobs scheduled by EDF in $[t_1, t_2]$ can be no more than $\sum_{\tau_i \in \tau} \text{DBF}_i(t, t_{\text{signal}} - t_1)$.

In all three cases, the total workload of the jobs scheduled by EDF in $[t_1, t_2]$ can be no more than $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s)$ for some $s \in [0,t]$. Since the total workload of those jobs must exceed t, we then have $\sum_{\tau_i \in \tau} \text{DBF}_i(t,s) > t$ for some $s \in [0,t]$.

By Lemma 6, there must then exist $t' \in \{0, 1, \ldots, \lfloor B \rfloor\}$ and $s' \in \mathcal{S}(t')$ such that $\sum_{\tau_i \in \tau} \mathrm{DBF}_i(t', s') > t'$. This demonstrates the sufficiency of the test and concludes the proof. ◀

▶ **Corollary 8.** *The schedulability test described in Theorem 7 can be implemented to run in pseudo-polynomial time if* $\max(U_{\mathrm{LO}}, U_{\mathrm{HI}}) \leq c$ *for some constant* $c < 1$.

Proof. If $\max(U_{\mathrm{LO}}, U_{\mathrm{HI}}) \leq c$, then B is clearly pseudo-polynomially bounded. It follows that $\{0, 1, \ldots, \lfloor B \rfloor\}$ is of pseudo-polynomial size, and so is $\mathcal{S}(t)$ for any $t \in \{0, 1, \ldots, \lfloor B \rfloor\}$. ◀

6 Comparison and Recommendations

The results in Sections 3–5 above establish that CC-3 is the most tractable of our three correctness criteria both from the run-time complexity perspective and in the sense that we have the most positive results regarding sporadic task systems about this criterion. We point out that *correctness criterion CC-3 is a stronger constraint than correctness criterion CC-2* — any schedule for an instance that satisfies correctness criterion CC-3 also satisfies correctness criterion CC-2 for that instance (this follows from the observation that CC-3 guarantees any LO-criticality job spanning a mode-transition instant its larger WCET, while correctness criterion CC-2 only requires this for those LO-criticality jobs that have already begun execution prior to the mode transition). In a similar vein *correctness criterion CC-2 is a stronger constraint than correctness criterion CC-1* since CC-2 guarantees some LO-criticality jobs spanning a mode-transition instant (those that began execution prior to the transition) their larger WCET, while correctness criterion CC-1 does not require this for any LO-criticality job. It therefore follows that a schedule for an instance satisfying correctness criterion CC-3 also satisfies correctness criteria CC-2 and CC-1: *correctness criterion CC-3 is a conservative over-approximation of correctness criteria CC-2 and CC-1.* Based on this observation and the additional tractability of CC-3 compared to CC-1 and CC-2, we recommend that when graceful degradation is the goal **correctness criterion CC-3 be considered the default correctness criterion for semi-clairvoyant scheduling**, and furthermore that **EDF be considered the default preferred run-time semi-clairvoyant scheduling algorithm**.

Quantifying the cost. We now quantify the cost of our recommendation: how much faster does a processor need to be in order to ensure that an instance that can be correctly scheduled under either of the weaker correctness criteria CC-1 or CC-2 can also be correctly scheduled under the more conservative correctness criterion CC-3? We formalize this metric as the *criteria loss*:

▶ **Definition 9** (Criteria Loss). *For two different correctness criteria CC-x and CC-y for* $x, y \in \{1, 2, 3\}, (x \neq y)$, *the* criteria loss *of CC-x compared to CC-y is the smallest number* ℓ *such that any collection of jobs* $\mathcal{J}$ *that is schedulable under correctness criterion CC-y on a unit-speed processor is also schedulable on a speed-*ℓ *processor under correctness criterion CC-x.*

We start with an upper bound for the criteria loss of CC-3 compared to the others.

▶ **Lemma 10.** *The criteria loss of CC-3 compared to CC-1 or CC-2 is no greater than 2.*

Proof. First, we note that a necessary condition for schedulability of a collection of jobs $\mathcal{J}$ with any algorithm under CC-1 or CC-2 is that it should meet all deadlines in the two cases where either (i) every job $J_i \in \mathcal{J}$ executes for exactly $c_i(\mathrm{LO})$ time units (i.e., HI-criticality

mode is never signaled) or (ii) every job $J_i \in \mathcal{J}$ executes for exactly $c_i(\text{HI})$ time units (i.e., HI-criticality mode is signaled at the first HI-criticality job arrival, and each LO-criticality job completes upon having executed for exactly $c_i(\text{HI})$ time units, even if it would be allowed to execute for $c_i(\text{LO})$ time units).

Second, due to the sustainability of EDF [6], we note that a sufficient condition for EDF to successfully schedule $\mathcal{J}$ under CC-3 is that it could meet all deadlines if (iii) every job $J_i \in \mathcal{J}$ executes for exactly $\max(c_i(\text{LO}), c_i(\text{HI}))$ time units. It follows directly from standard analysis of EDF on non-mixed criticality jobs that it will succeed with (iii) on a speed-2 processor if any algorithm can succeed with both (i) and (ii) on a unit-speed processor. ◀

Next we see that this bound is tight compared to CC-2.

▶ **Lemma 11.** *The criteria loss of CC-3 compared to CC-2 is at least 2.*

Proof. Consider the collection $\mathcal{J} = \{J_1 = (\text{LO}, 0, [k-1, 0], k), J_2 = (\text{HI}, 1, [0, k-1], k)\}$ of two jobs. Clearly $\mathcal{J}$ is schedulable under CC-2: simply idle the processor until the arrival of J_2 at time instant 1, after which there will be at most $(k-1)$ work to be done no matter if J_2 signals HI-criticality behavior or not.

Under CC-3, J_1, having arrived prior to J_2, can require up to $c_1(\text{LO}) = k-1$ units of execution. If J_2 then signals HI-criticality behavior, the total workload over $[0, k]$ may be as high as $2(k-1)$. As $k \to \infty$, we see that this would require a speed-2 processor. ◀

Lemmas 10 and 11 together yield the following theorem which completely characterizes the worst-case penalty of over-approximating CC-1 or CC-2 by CC-3.

▶ **Theorem 12.** *The criteria loss of CC-3 compared to either CC-1 or CC-2 is exactly 2.*

Proof. From Lemma 10 we know that the criteria loss of CC-3 compared to CC-1 or CC-2 is at most 2. From Lemma 11 we know that the criteria loss of CC-3 compared to CC-2 is at least 2. Since CC-2 is a conservative over-approximation of CC-1, the criteria loss of CC-3 compared to CC-1 must be at least 2 as well. ◀

For the sake of completeness we also present bounds on the criteria loss of CC-2 compared to CC-1.

▶ **Lemma 13.** *The criteria loss of CC-2 compared to CC-1 is in $[\varphi, 2]$, where φ is the golden ratio $\varphi = (1+\sqrt{5})/2 \approx 1.618$.*

Proof. First, observe that the upper bound of 2 clearly holds here as well. For simplicity of presentation in deriving the lower bound we use real-valued job parameters in this proof, with the observation that we can approximate those to an arbitrary level of precision with rational parameters. Rational parameters can in turn be changed to integer parameters by scaling everything with the least common multiple of the denominators without affecting the schedulability of the jobs.

Let $x = (3-\sqrt{5})/2$ and consider the collection of jobs $\mathcal{J} = \{J_1, J_2\}$, where

$$J_1 = (\text{LO}, 0, [1, 0], 1),$$
$$J_2 = (\text{HI}, x, [0, 1-x], 1).$$

Note that $\mathcal{J}$ is schedulable under CC-1 since we can simply schedule J_1 in $[0, x)$ and then see whether J_2 signals HI-criticality behavior when it arrives. If J_2 does not signal HI-criticality behavior we continue executing J_1 until it finishes, otherwise we execute J_2 until it finishes since J_1 already has received more than $c_1(\text{HI})$ execution time.

Under CC-2 we must make the choice of whether to start executing J_1 before the arrival of J_2. If we do start executing J_1 immediately and J_2 later signals HI-criticality behavior, then we need to finish a total of $1 + (1 - x)$ units of work over $[0, 1]$, and we need a speed-ℓ processor where

$$\ell \geq 2 - x = \frac{1 + \sqrt{5}}{2} = \varphi.$$

If we instead decide to idle the processor until the arrival of J_2 and J_2 arrives without signaling HI-criticality behavior, then we need to finish J_1's entire execution time $c_1(\text{LO}) = 1$ in $[x, 1]$. For this we need a speed-ℓ processor where

$$\ell \geq \frac{1}{1 - x} = \frac{2}{\sqrt{5} - 1} = \frac{1 + \sqrt{5}}{2} = \varphi,$$

which completes the proof. ◀

7 Context and Conclusions

Since the mixed-criticality model was introduced by Vestal [30], several extensions and variations have been proposed. Criticism of the original model has made it clear that some form of *graceful degradation* often is necessary for mixed-criticality scheduling to be used in practice. In this paper we have combined graceful degradation with *semi-clairvoyant scheduling*, an interesting new take on how and when information becomes available at runtime to a scheduler, and studied this under three different *correctness criteria* that we labeled CC-1, CC-2 and CC-3. Although the differences between the correctness criteria appear minor—they differ only in the treatment of LO-criticality jobs that are active when a HI-criticality job signals HI-criticality behavior—we have seen that they require wildly differing solutions. The difference in the complexity of the associated schedulability problems is also stark: schedulability for a collection of jobs is solvable in $O(n^2 \log n)$ time for CC-3, but is NP-complete in the strong sense for CC-2.

There is no single correctness criterion which is the correct one in all situations: each is a reasonable model for some types of systems. However, as CC-3 is a safe over-approximation of the other criteria it looks particularly useful as a default model. This is especially true considering that it leads to easy scheduling (plain EDF is an optimal scheduler) and that it is easy to analyze (in polynomial time for jobs, in pseudo-polynomial time for arbitrary deadline tasks if utilization is bounded).

While we have studied these problems with the added generalization of graceful degradation, it should be noted that the correctness criteria—and the results of this paper—are equally valid without graceful degradation. This is represented by simply having $c_i(\text{HI}) = 0$ or $C_i(\text{HI}) = 0$ for all LO-criticality jobs or tasks.

We also note that the correctness criteria can apply equally to systems without semi-clairvoyance: in ordinary (non-clairvoyant) mixed-criticality scheduling we can still have different correctness criteria for the LO-criticality jobs that are active when it is first discovered that the system is exhibiting HI-criticality behavior (i.e., when a job has executed for its LO-criticality WCET without signaling completion). In such systems, CC-1 would correspond to the standard semantics as studied in most previous work, but it is not necessarily the most appropriate one, or the one that is easiest to work with.

References

1. Kunal Agrawal, Sanjoy Baruah, and Alan Burns. Semi-clairvoyance in mixed-criticality scheduling. In *Proceedings of the Real-Time Systems Symposium (RTSS)*, pages 458–468, Dec 2019. doi:10.1109/RTSS46320.2019.00047.
2. S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press. doi:10.1109/REAL.1990.128746.
3. Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor Scheduling for Real-Time Systems*. Springer Publishing Company, Incorporated, 2015. doi:10.1007/978-3-319-08696-5.
4. Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 2012. doi:10.1109/TC.2011.142.
5. Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM*, 62(2):14:1–14:33, May 2015. doi:10.1145/2699435.
6. Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 159–168, Rio de Janeiro, December 2006. IEEE Computer Society Press. doi:10.1109/RTSS.2006.47.
7. Sanjoy Baruah, Alan Burns, and Robert Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press. doi:10.1109/RTSS.2011.12.
8. Sanjoy Baruah, Alan Burns, and Zhishan Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *Proceedings of the 2016 28th EuroMicro Conference on Real-Time Systems*, ECRTS '16, Toulouse (France), 2016. IEEE Computer Society Press. doi:10.1109/ECRTS.2016.12.
9. Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo. MC-Fluid: simplified and optimally quantified. In *Real-Time Systems Symposium (RTSS), 2015 IEEE*, Dec 2015. doi:10.1109/RTSS.2015.38.
10. Sanjoy Baruah, Haohan Li, and Leen Stougie. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. IEEE, April 2010. doi:10.1109/RTAS.2010.10.
11. I. Borosh and L. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55:299–304, 1976. doi:10.1090/S0002-9939-1976-0396605-3.
12. A. Burns and S. Baruah. Timing faults and mixed criticality systems. In Jones and Lloyd, editors, *Dependable and Historic Computing*, volume LNCS 6875, pages 147–166. Springer, 2011. doi:10.1007/978-3-642-24541-1_12.
13. Alan Burns and Sanjoy Baruah. Towards a more practical model for mixed criticality systems. In *Proceedings of the International Workshop on Mixed Criticality Systems (WMC)*, December 2014.
14. Alan Burns and Robert Ian Davis. Schedulability analysis for adaptive mixed criticality systems with arbitrary deadlines and semi-clairvoyance. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 2020. doi:10.1109/RTSS49844.2020.00013.
15. Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Robert I. Davis. On the Pitfalls of Resource Augmentation Factors and Utilization Bounds in Real-Time Scheduling. In Marko Bertogna, editor, *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, volume 76 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:25, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ECRTS.2017.9.

16 Lin Chen, Franziska Eberle, Nicole Megow, Kevin Schewior, and Cliff Stein. A general framework for handling commitment in online throughput maximization. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 141–154, Cham, 2019. Springer International Publishing.

17 Michael Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

18 Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-Time Systems*, 50(1):48–86, 2014. doi:10.1007/s11241-013-9187-z.

19 Rolf Ernst and Marco Di Natale. Mixed criticality systems - A history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016. doi:10.1109/MDAT.2016.2594790.

20 Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. How realistic is the mixed-criticality real-time system model? In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 139–148, New York, NY, USA, 2015. ACM. doi:10.1145/2834848.2834869.

21 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

22 G. Giannopoulou, P Huang, R Ahmed, D Bartolini, and L Thiele. Isolation scheduling on multicores: model and scheduling approaches. *Real-Time Systems: The International Journal of Time-Critical Computing*, 53:614–667, 2017. doi:10.1007/s11241-017-9277-4.

23 Xiaozhe Gu and Arvind Easwaran. Dynamic budget management with service guarantees for mixed-criticality systems. In *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*, pages 47–56, 2016. doi:10.1109/RTSS.2016.014.

24 Z. Guo, K. Yang, S. Vaidhun, S. Arefin, S. K. Das, and H. Xiong. Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 373–383, 2018. doi:10.1109/RTSS.2018.00052.

25 B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 214–223, Los Alamitos, October 1995. IEEE Computer Society Press. doi:10.1109/SFCS.1995.492478.

26 Jaewoo Lee, Kieu-My Phan, Xiaozhe Gu, Jiyeon Lee, A. Easwaran, Insik Shin, and Insup Lee. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 41–52, Dec 2014. doi:10.1109/RTSS.2014.32.

27 C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

28 A. Mok. Task management techniques for enforcing ED scheduling on a periodic task set. In *Proceedings of the 5th IEEE Workshop on Real-Time Software and Operating Systems*, pages 42–46, Washington D.C., May 1988.

29 Dario Socci, Petro Poplavko, Saddek Bensalem, and Marius Bozga. Mixed critical earliest deadline first. In *Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems*, ECRTS '13, Paris (France), 2013. IEEE Computer Society Press. doi:10.1109/ECRTS.2013.20.

30 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the Real-Time Systems Symposium (RTSS)*, pages 239–243, Tucson, AZ, December 2007. IEEE Computer Society Press. doi:10.1109/RTSS.2007.47.

31 Reinhard Wilhelm. Mixed feelings about mixed criticality (invited paper). In Florian Brandner, editor, *Proceedings of the 18th International Workshop on Worst-Case Execution Time Analysis*, pages 1:1–1:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.WCET.2018.1.