The Safe and Effective Use of Optimistic Period Predictions

Sanjoy Baruah
Washington University
United States
baruah@wustl.edu

Pontus Ekberg
Uppsala University
SE
pontus.ekberg@it.uu.se

Alexander Lindermayr
University of Bremen
Germany
linderal@uni-bremen.de

Alberto Marchetti-Spaccamela
La Sapienza
Italy
alberto@diag.uniroma1.it

Nicole Megow
University of Bremen
Germany
nicole.megow@uni-bremen.de

Leen Stougie
CWI
Netherlands
leen.stougie@cwi.nl

Abstract

Parameters characterizing safety critical systems are generally assigned very conservative values for reasons of safety assurance. Provisioning computing resources on the basis of such conservatively assigned parameter values can lead to system implementations that make inefficient use of platform resources during run time. We address the problem of achieving more efficient implementations of sporadic task systems where, in addition to a conservatively assigned value for the period parameter of each task, we also have a more optimistic (i.e., larger), but perhaps incorrect, *prediction* of this value. We devise an algorithm that executes the system more efficiently during runtime if the prediction is correct, without compromising safety if it turns out to be incorrect.

CCS Concepts

• **Computer systems organization** → **Real-time systems**; • **Software and its engineering** → *Real-time schedulability*.

Keywords

Algorithms using predictions; sporadic task systems; uniprocessor EDF schedulability analysis

ACM Reference Format:

Sanjoy Baruah, Pontus Ekberg, Alexander Lindermayr, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. 2024. The Safe and Effective Use of Optimistic Period Predictions. In *The 32nd International Conference on Real-Time Networks and Systems (RTNS 2024), November 06–08, 2024, Porto, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3696355.3696356

1 Introduction

A *sporadic task* [10, 20, 23] typically models the timing aspects of code execution triggered by external events. The task, denoted as τ_i, is defined by two parameters: the *worst-case execution time* (WCET) C_i and the *period* T_i. WCET represents the maximum duration for code completion, while the period is the minimum time between successive triggerings of task τ_i.

Estimating the minimum duration between triggering events in an accurate manner can be challenging. Safety-critical systems address this challenge by assigning a small, safe lower bound value to the period parameter T_i. This conservative approach, aimed at ensuring safety, often leads to platform resource under-utilization during runtime when jobs are released much further apart than T_i. The algorithms community has recently begun studying how to make use of lower-assurance information in a safe and effective manner. Such lower-assurance information, called *predictions*, may be obtained from a variety of sources including measurements, human intuition, or machine learning. The *Algorithms using Predictions* framework [17, 21, 22] (also known as *learning-augmented algorithms*) outlines a systematic approach to safely and effectively use predictions, increasing efficiency when correct, without compromising correctness or causing excessive degradation when predictions are incorrect. (see [1] for an introduction to this topic that is targeted to the real-time computing community).

In this work we assume access to a prediction P_i for each task τ_i's period parameter. While P_i is a more realistic estimate than T_i, there is no complete assurance that successive jobs of τ_i won't be released sooner than P_i time units apart. In other words, consecutive jobs being released less than T_i time units apart represent a runtime *fault* that would likely trigger fault-tolerance mechanisms. However, successive jobs being released less than P_i time units apart, although believed to be unlikely, do not constitute a system fault and all deadlines must still be guaranteed.

§. The problem considered here. We assume that we are given a real-time system comprising a collection of several independent sporadic tasks, with each task τ_i characterized by the 3-tuple (C_i, T_i, P_i) as discussed above, that are to execute upon a shared processor that has a specified maximum speed or computing capacity. We propose to schedule this system using a **run-time scheduling algorithm** that

(1) starts out with the processor running at some speed s that is smaller than the maximum processor speed (i.e., $s < 1$);
(2) monitors job-release time in order to determine whether successive jobs of any task τ_i have been released sooner than P_i time units apart; if so
(3) increases the processor speed up to its maximum (i.e., to speed 1), and remains at this maximum speed until an idle instant occurs in the schedule at which point in time the processor speed is again returned to s (and we are back in Step (1) above of the run-time scheduling algorithm).

The Algorithms using Predictions framework assumes that the prediction is highly likely to be accurate (although as stated above, there is no absolute guarantee). Consequently, we aim to assign a small value to s so as to achieve energy efficiency, a reduction in heat-dissipation costs, etc., during runtime. Simultaneously, we must ensure that deadlines are consistently met even if the predictions happen to be inaccurate. In essence, we seek to answer the question: *What is the minimum value of s that ensures the runtime algorithm always meets all job deadlines?* Our main **contribution** in this paper is an algorithm that computes this minimum speed near-optimally (with the “nearness" to optimality characterized precisely – Lemma 4) in time pseudo-polynomial in the representation of the provided task system.

§. Organization. The remainder of this paper is organized in the following manner. In Section 2 we provide some background information that is needed in the remainder of this paper. We formally state the problem that we will be solving in Section 3, and develop an algorithm for doing so in Section 4. In Section 5 we analyze the performance of this algorithm in terms of both its asymptotic runtime complexity and its distance from optimality. We conclude in Section 6 with a summary of our findings, and brief mention of some straightforward generalizations.

2 Background and Related Work

We start out providing the necessary background on algorithms using predictions in Section 2.1, by briefly and non-exhaustively reviewing prior work on this subject. In Section 2.2 we review some well-known results from real-time scheduling theory, concerning the exact and approximate schedulability analysis of sporadic task systems that are EDF-scheduled upon preemptive uniprocessor platforms. In Section 2.3 we discuss some prior work that is related to the research we are presenting here.

2.1 Algorithms Using Predictions

Safety-critical systems should have their correctness properties verified prior to deployment; such verification is currently typically done via some form of worst-case analysis. Worst-case analysis tends to lead to very conservative system designs that make inefficient use of computing resources almost all of the time. One approach to overcoming such conservatism is to go “Beyond Worst-Case Analysis" [25] by using *predictions* to guide an algorithm. Such predictions may be drawn from a variety of sources, such as via measurements based upon empirical observations (that, despite perhaps being quite extensive and thorough, would not qualify as high-assurance); being assigned by human experts based on their expertise and intuition; or through the use of machine-learning techniques. Since such predictions are often of uncertain provenance, system design and analysis algorithms should not trust them entirely. Informally speaking, an algorithm that uses predictions to make decisions should be designed in such a manner that it achieves the best of both worlds: providing improved performance when the prediction is accurate, without suffering too much of a performance degradation, in comparison with algorithms that are developed using traditional worst-case methods, when the prediction is inaccurate. The algorithmic framework of *algorithms using predictions* (see [21] for a comprehensive introduction) offers a systematic approach to doing so. Algorithms designed according to this framework are characterized according to the following properties:

(1) Consistency: When the predictions are accurate, the performance of the algorithm is excellent, often near-optimal.
(2) Robustness: When the predictions are inaccurate, the performance of the algorithm is not much worse than that of an algorithm that does not use predictions.
(3) Smoothness: The performance of the algorithm does not fall off drastically when the predictions have small errors: “the algorithm interpolates gracefully between the robust and consistent settings" [21].
(4) Learnability: Good values of the predicted quantity can be learnt over time.

In other words, the *consistency* of an algorithm that uses predictions characterizes its performance when the predictor is perfectly accurate, while *robustness* characterizes its performance guarantee regardless of the quality of the predictions. (In this paper we focus exclusively on obtaining algorithms that are capable of achieving consistency and robustness, leaving consideration of smoothness and learnability for future work.)

Predictions have proven to be a powerful tool for breaking pessimistic bounds in various scheduling problems with non-periodic jobs. While the majority of research addresses uncertainties related to unknown processing requirements or runtime behavior [5, 6, 8, 14, 16, 18, 24, 30], few works investigate predictions regarding the online job arrival or deadlines [4, 15] or the processor speed [7, 19]. Only recently, the concept has been introduced to the real-time systems community [1].

Notably, to our knowledge, there is no prior work exploring predictions on task periods.

This paper aims implicitly at energy minimization via speed scaling which has been considered for other prediction models and simple jobs in [4, 8].

2.2 Three-Parameter Sporadic Task Systems

We now briefly review some well-known prior results on real-time scheduling (without period predictions – i.e., on task models that only assumed guaranteed bounds on both the WCET and the period parameter), that we will be using in the remainder of this paper.

We have thus far talked of sporadic task systems in which each task τ_i is characterized by a WCET C_i and a period parameter T_i, with the constraint that each job released by τ_i must complete execution prior to the release of the next job. Such task systems are called *implicit-deadline* sporadic task systems; in *3-parameter* sporadic task systems, task τ_i is additionally characterized by a *relative-deadline* parameter D_i, and the constraint is that each job released by τ_i must complete execution within D_i time units of its release time. In this section we restrict attention to *constrained-deadline* 3-parameter sporadic systems Γ, which satisfy the additional restriction that $D_i \le T_i$ for all tasks $\tau_i \in \Gamma$. We further assume that $\sum_{\tau_i \in \Gamma} (C_i/T_i) \le 1$, and examine the EDF-schedulability of Γ upon a unit-speed preemptive processor. It has been shown [10] that a necessary and sufficient condition for Γ to be EDF-schedulable is that no deadline is missed in the (simulated) EDF scheduling of the behavior of Γ in which each $\tau_i \in \Gamma$ generates a job at time-instant 0, and subsequent jobs as soon as legally permitted to do so (i.e., at time-instants $k \cdot T_i$ for all $k \in \mathbb{N}$) — such a behavior is commonly referred to as the *synchronous arrival sequence (SAS)* for Γ. It was further shown that this simulation may be terminated at the *hyperperiod* (the least common multiple of the period parameters of the tasks in Γ) — if no deadlines are missed by then, it is not possible that a deadline miss will occur.

In practice, the idea contained in the paragraph above is usually implemented via an abstraction called the *demand bound function* (dbf): for any sporadic task $\tau_i = (C_i, D_i, T_i)$ and any interval-duration $t \ge 0$, $\text{dbf}_i(t)$ denotes the maximum possible cumulative execution requirement by jobs of task τ_i that both arrive in, and have their deadlines within, any contiguous interval of duration t. The following formula for computing $\text{dbf}_i(t)$ was derived in [10]:

$$\text{dbf}_i(t) = \max\left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0\right) \times C_i \tag{1}$$

and it was shown that a necessary and sufficient condition for a constrained-deadline 3-parameter sporadic task system Γ to be EDF-schedulable upon a preemptive unit-speed processor is that the following condition should hold for all t that correspond to deadlines of jobs in the SAS that are no larger than the hyperperiod:

$$\sum_{\tau_i \in \Gamma} \text{dbf}_i(t) \le t. \tag{2}$$

For *bounded-utilization* sporadic task systems —systems Γ satisfying the additional condition that $\left(\sum_{\tau_i \in \Gamma} (C_i/T_i)\right) \le c$ for some pre-defined constant c strictly smaller than 1— that are not EDF-schedulable upon a preemptive unit-speed processor, however, it is known [9, Theorem (3.1))] that Condition 2 is violated for some t that lies within the first *busy interval* of the EDF schedule of the SAS, and that the duration of this busy interval is upper-bounded by

$$\left(\frac{c}{1-c}\right) \times \max_{\tau_i \in \Gamma}\{T_i - D_i\}. \tag{3}$$

(We point out that this upper bound is pseudo-polynomial in the representation of Γ.)

2.2.1 The Albers-Slomka Approximation.

Since EDF schedulability verification is known to be coNP-hard [13], we should not expect to be able to develop polynomial-time algorithms for doing EDF schedulability-verification exactly — Condition 2 must in general be checked for exponentially many distinct values of t. However, polynomial-time sufficient (rather than exact) EDF schedulability verification algorithms are known; many of the best ones are based upon an approximation proposed by Albers and Slomka [3] to the demand bound function. In this approximation, one fixes an integer value for a parameter $\kappa \in \mathbb{N}$ and defines the approximation, $\text{dbf}_i^{\langle\kappa\rangle}$, as follows:

$$\text{dbf}_i^{\langle\kappa\rangle}(t) = \begin{cases} \text{dbf}_i(t), & \text{if } t \le \kappa \times T_i + D_i \\ C_i + \left(\frac{C_i}{T_i}\right) \cdot (t - D_i), & \text{otherwise} \end{cases} \tag{4}$$

(A quick glance at Figure 3 (a) may be helpful to the reader unfamiliar with this approximation.)

A **testing set** $\mathcal{T}(\Gamma)$ is defined, comprising the deadlines of the first $(\kappa+1)$ jobs in the SAS that are released by each task. It was shown [3] that task system Γ is EDF schedulable upon a unit-speed processor if the following analog of Condition 2:

$$\sum_{\tau_i \in \Gamma} \text{dbf}_i^{\langle\kappa\rangle}(t) \le t \tag{5}$$

is satisfied for all $t \in \mathcal{T}(\Gamma)$; since $|\mathcal{T}(\Gamma)| \le (\kappa+1) \times |\Gamma|$, this immediately yields a polynomial-time sufficient EDF-schedulability test.

It has been shown [3] that

$$\text{dbf}_i(t) \le \text{dbf}_i^{\langle\kappa\rangle}(t) < \text{dbf}_i(t) + C_i\ . \tag{6}$$

It follows from the definition of $\text{dbf}_i^{\langle\kappa\rangle}$ in Eq. 4 and the second inequality in Eq. 6 above that

$$\text{dbf}_i^{\langle\kappa\rangle}(t) < \left(1 + \frac{1}{\kappa}\right) \times \text{dbf}_i(t).$$

Combined with the exact test in Eq. 2 it is easily concluded that *any sporadic task system that is deemed to not be EDF schedulable using the polynomial-time schedulability test in Eq. 5 is not EDF schedulable upon a speed-$(\frac{\kappa}{\kappa+1})$-processor.*

Choosing a value for κ. Since the running time of the schedulability test depends on the size of the testing set, it is evident that the smaller the value assigned to κ, the more efficient this test is. On the other hand, the larger the value of κ, the more accurate the test in the following sense: if the test deems a task system to not be schedulable on unit-speed processors, it is guaranteed to actually not be so on processors that are closer in speed to one for larger values of κ. Albers and Slomka [3] point out that the sufficient test described above can in fact be turned into an FPTAS for approximating the required processor speed.

2.3 Mixed-Criticality Scheduling

In this paper, we are assuming that each periodic task's period parameter is given two values: a conservative one that is guaranteed to be safe, and a more optimistic one that is very likely to be safe (but is not guaranteed to be so). This is similar in spirit to much work on *mixed-criticality scheduling* [29] (see, e.g., [12] for a review) in which tasks are characterized by multiple WCET parameter values that are guaranteed to be accurate at different assurance levels.

Much of mixed-criticality scheduling theory deals with a decision problem (is a given system schedulable upon a particular processing platform?) rather than an optimization problem such as the one we are addressing (what is the minimum initial processor speed that guarantees to never miss any deadlines?) in this paper. The standard model of mixed-criticality scheduling is also to react to incorrect assumptions (or mispredictions, in the terminology of this paper) by reducing service, usually entirely, to some tasks that are considered less critical. This is different from the model considered here where all tasks are considered equally important and all deadlines must always be met.

An exception is the work on the mixed-criticality *precise scheduling* model [11, 26–28]; in this work, the same question is asked as the one we are posing here (i.e., determining the minimum initial speed that guarantees that all jobs of all tasks will always meet their deadlines in all low-criticality and high-criticality behaviors), but for the standard mixed-criticality task model with each task characterized by two WCET values.

3 System Model

We assume that we are given a sporadic task system

$$\Gamma = \bigcup_{i=1}^{n} \{\tau_i = (C_i, T_i, P_i)\}$$

where C_i and T_i are the WCET and (guaranteed safe) period respectively of task τ_i, and $P_i \geq T_i$ is a *prediction* of its period. Each task $\tau_i \in \Gamma$ releases a sequence of jobs which must be executed; it is required that a job of τ_i must have completed execution before the next job of τ_i is released. Observe that Γ may generate different sets of jobs each time it is executed; we refer to each execution as a **behavior** of the system. In *consistent* behaviors, successive jobs of each task τ_i arrive $\geq P_i$ time units apart. Any behavior in which a pair of successive jobs of any $\tau_i \in \Gamma$ arrive sooner than P_i (but $\geq T_i$) time units apart is not consistent, whereas behaviors in which successive jobs of any $\tau_i \in \Gamma$ arrive sooner than T_i time units apart are said to be *faulty*. We do not discuss faulty behaviors any further in this paper, but assume that they are handled by a separate fault-recovery mechanism that is invoked whenever a fault is detected at run-time.

We seek to schedule Γ upon a single preemptive processor with maximum speed or computing capacity one: the processor can complete one unit of execution in one time-unit.

Run-Time Algorithm. As discussed in Section 1, we will start out running the processor with its speed set to $s < 1$. Since all that can be guaranteed is that successive jobs of τ_i will be released no sooner than T_i time-units apart (in non-faulty behaviors), we must ensure that each job of τ_i completes its execution within T_i time units of its release. Hence the system must initially be modeled as a *constrained-deadline* sporadic task system comprising $|\Gamma|$ tasks, in which the i'th task has WCET C_i, relative deadline T_i, and period P_i. If any prediction violation is detected during run-time (i.e., successive jobs of some task τ_i are released sooner than P_i time-units apart), we immediately increase the processor speed to 1.

Optimization criterion: As stated in Section 1, the implicit assumption in the Algorithms using Predictions framework is that predictions are very likely to be correct, in which case prediction violations will never occur and the processor will always run at its initially-set speed of s. Our objective is therefore to optimize for consistency and find the *smallest* value of s for which the robustness guarantee holds that no deadline misses will occur (regardless of whether predictions hold or not).

Some additional **terminology**: we define the *scheduling window* of a job to denote the interval within which it must be scheduled in order to guarantee that its deadline will be met under all possible circumstances. Suppose that a job is released by τ_i at some time-instant t_a; since all we know for certain is that its next job will not be released prior to time-instant $t_a + T_i$, its scheduling window is equal to the time interval $[t_a, t_a + T_i)$.

4 An Algorithm for Determining the Initial Processor Speed

Recall our task model from Section 3: we have an implicit-deadline sporadic task system with guaranteed and predicted period estimates

$$\Gamma = \bigcup_{i=1}^{n} \{\tau_i = (C_i, T_i, P_i)\}$$

where C_i and T_i are the WCET and (guaranteed safe) period respectively of τ_i, and $P_i \geq T_i$ is a *prediction* of its period, that we propose to schedule using the following **run-time scheduling algorithm** using EDF.

- We will start out running the processor at some speed that is smaller than the maximum processor speed.
- We will monitor job-release times, in order to determine whether successive jobs of any task τ_i have been released sooner than P_i time units apart. If this happens, we say that a *prediction failure* has occurred; we will occasionally refer to this task as the *triggering task*, and the instant at which the sooner-than-expected job of the triggering task arrives as the *triggering instant*.
- At the triggering instant, we immediately begin running the processor at its maximum speed, and remain at this maximum speed until an idle instant occurs in the schedule. When this happens the processor speed is again returned to the initial slower speed.

(We point out that a nice feature of this run-time algorithm is its *simplicity*, which allows for very efficient implementation with minimal run-time overhead. Notice that the scheduling deadlines assigned to already-arrived jobs do not change at the triggering instant, and hence no re-ordering of the run-time queue is needed upon detection of a prediction failure.)

In the remainder of this section we will describe how to determine, prior to run-time, the speed at which the processor is to initially be run. We start out with a **high-level overview**: we will first derive a necessary condition for a deadline miss for a given initial

Algorithm 1: Computing the initial processor speed

```
1  Input: Task system $\Gamma$, with each $\tau_i \in \Gamma$ characterized by a three-tuple: $\tau_i = (C_i, T_i, P_i)$
2  Output: The speed $s_o$, $s_o < 1$, at which the processor should initially be run
3  $s_o \leftarrow$ an initial value for the speed that ensures that fully-consistent behaviors are schedulable (see Section 5.2)
4  $H \leftarrow$ a pseudo-polynomial upper bound on the triggering instants that must be considered to discover a deadline miss (see Section 5.2)
5  for each $t_f \in \{1, 2, \ldots, H\}$ do // (Assumption: integer job arrivals)
6      Suppose that $t_f$ is the triggering instant
7      for each $\tau_\ell \in \Gamma$ do
8          Suppose that $\tau_\ell$ is the triggering task
9          Compute a safe set $K$ of possible values for $t_d$, such that a deadline miss must occur for one of these values of $t_d$ if any
             deadline miss is to occur at all
10         for each $t_d \in K$ do
11             Update $s_o$ to be the larger of its current value, and the value determined according to Equation 7:
                   $s_o \leftarrow \max\left(s_o, \frac{\left(\sum_{\tau_i \in \Gamma} \delta_i(t_f, t_d)\right) - (t_d - t_f)}{t_f}\right)$
12         end
13     end
14 end
15 return $s_o$
```

processor speed s_o. By then negating this condition, we will obtain a formula for assigning s_o a value that guarantees no deadline miss. (Later in Section 5, we will evaluate the effectiveness of this means of assigning the initial processor speed by quantifying how far removed it is from the lowest possible value.)

To derive a necessary condition for a deadline miss, let us suppose that we start out at speed s_o, and let t_d denote the earliest time-instant at which a deadline miss can possibly occur when Γ is executed using the run-time algorithm described above. Consider some collection of jobs $\mathcal{J}$ of Γ upon which this deadline miss at t_d occurs. Let $t_f < t_d$ denote the triggering instant – the (earliest) time-instant at which a prediction failure occurred.[1] Let $\tau_\ell \in \Gamma$ denote the triggering task: a job of τ_ℓ was released at time-instant t_f despite less than P_ℓ time having passed since the prior release of a job of τ_ℓ.

We point out that there are no idle instants in the EDF schedule of $\mathcal{J}$ when executed upon a processor of speed s_o over $[0, t_f)$ and speed 1 over $[t_f, t_d)$; else it is easily shown that the jobs arriving after the idle instant would constitute a collection of jobs on which an earlier deadline miss occurs.

Let $\delta_i(t_f, t_d)$ denote a (tight) upper bound on the cumulative execution requirement by jobs that are in $\mathcal{J}$ that were generated by task τ_i —we will describe how $\delta_i(t_f, t_d)$ is computed in Section 4.1, and how it may be approximated in Section 4.2. Since the processor runs at speed s_o over $[0, t_f)$ and speed 1 over $[t_f, t_d)$, it must be the case that

$$\sum_{\tau_i \in \Gamma} \delta_i(t_f, t_d) > s_o \times t_f + 1 \times (t_d - t_f)$$

in order for the deadline miss to occur. Hence, assigning s_o a value satisfying

$$s_o \geq \frac{\left(\sum_{\tau_i \in \Gamma} \delta_i(t_f, t_d)\right) - (t_d - t_f)}{t_f} \qquad (7)$$

for all t_f, t_d values is sufficient to ensure that no deadline miss can occur. Additionally, the smallest such value of s_o is a lower bound on the speed at which the processor must initially be run in order to ensure that no deadline will be missed in the event of a prediction failure. Algorithm 1 depicts, in pseudocode form, how we compute a value for s_o satisfying Expression 7 for all possible choices of triggering task τ_ℓ and all relevant pairs of t_f, t_d values. Line 3 of this pseudocode is discussed below. In Section 4.1 we show that $\delta_i(t_f, t_d)$ can be computed in constant time for given τ_i, t_f, and t_d. In Section 5.2 we will see that the value assigned to H in Line 4 is pseudo-polynomial in the representation of Γ, and the testing set K computed in Line 9 contains at most polynomially many distinct values, and that these facts together imply that our overall algorithm has pseudo-polynomial running time.

Initializing s_o. The value of s_o we will have computed as described above ensures no deadline miss in the event of a prediction failure. We must also consider the possibility that a deadline miss may occur even without a prediction failure – some consistent behavior of Γ may be unschedulable. To rule this possibility out, we initialize s_o (Line 3 of Algorithm 1) such that the 3-parameter constrained deadline sporadic task system that models all possible consistent behaviors of Γ is EDF-schedulable upon a speed-s_o processor; prior algorithms, e.g., [3, 10], are known that can accomplish this exactly or approximately to any desired degree of accuracy. (This is discussed briefly in Section 5.2).

[1] It is also possible that some behavior of Γ misses a deadline upon a speed-s_o processor even without a prediction failure occurring; we explain how we account for this possibility a bit later in this section.

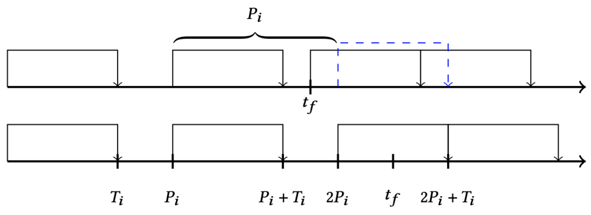

Figure 1: For Lemma 1. Jobs of τ_i are initially released P_i time units apart. If t_f lies outside the scheduling window of any such job, then a job is released at t_f (top); else, a job is released immediately upon the end of the scheduling window within which it lies (bottom).

4.1 Computing $\delta_i(t_f, t_d)$

As outlined above, our overall strategy is centered on identifying conditions that must hold for a deadline miss to occur at some time-instant t_d due to some triggering task τ_ℓ experiencing a prediction failure at some earlier triggering instant t_f, and then negating these conditions to ensure that a deadline miss can never occur. This strategy requires us to repeatedly compute $\delta_i(t_f, t_d)$ values, in order to repeatedly evaluate Eqn 7; we now discuss how to do so, in constant time for given τ_i, t_f, and t_d.

Lemmas 1 and 2 below characterize the system behaviors from which we can compute the desired upper bounds on $\delta_i(t_f, t_d)$ for given t_f, t_d.

Lemma 1. *For each task $\tau_i \in \Gamma$ other than the triggering task (i.e., for all $\tau_i \neq \tau_\ell$), the value of $\delta_i(t_f, t_d)$ is maximized when each job of τ_i is released as soon as legally permitted to do so. I.e.,*

- *the first job is released at time-instant 0;*
- *subsequently jobs are released P_i time-units apart over $[0, t_f)$;*
- *the first job released after time-instant t_f is released at the later of t_f and T_i plus the release time of the preceding job of τ_i; and*
- *subsequent jobs are released T_i time-units apart over $[t_f, t_d)$.*

(This is illustrated in Fig 1.)

PROOF SKETCH. Consider the collection of jobs $\mathcal{J}$ discussed in the overview of our strategy above, for which a triggering instant at t_f causes a deadline miss at t_d. It is evident that moving the release of each job earlier cannot reduce the total amount of execution that needs to be completed; hence, the total amount of execution that must be completed by time-instant t_d cannot decrease. And since τ_i is, by assumption, not the triggering task, changing the instants at which its jobs are released cannot *in*crease the amount of available computing capacity (by speeding up the processor at an earlier instant in time). □

Lemma 2. *The triggering task τ_ℓ must release a job at time-instant t_f, and $\delta_\ell(t_f, t_d)$ is maximized when*

- *its first job is released at time-instant 0;*

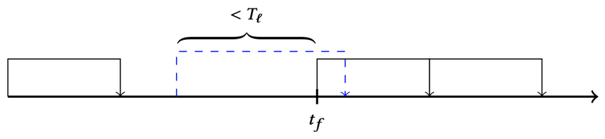

Figure 2: For Lemma 2. No job is released by the triggering job τ_ℓ within the interval $[t_f - T_\ell, t_f)$ (i.e., the dotted blue job does not get released despite P_ℓ time having elapsed since the prior release).

- *subsequent jobs are released each P_ℓ time-units apart, over the time-interval $[0, t_f - T_\ell)$;*
- *a job is released at time-instant t_f; and*
- *subsequent jobs are released T_ℓ time-units apart over $[t_f, t_d)$.*

(This scenario is illustrated in Fig 2.)

PROOF SKETCH. This is essentially the same proof as the one for Lemma 1, with the added restriction that since τ_ℓ is the triggering task, it *must* release a job at the triggering instant t_f. □

How many jobs are released? Let $\text{cnt}_i(t_f, t_d)$ denote the largest number of jobs of τ_i that can have deadlines $\leq t_d$ for given 3-tuple (τ_i, t_f, t_d). Lemmas 1 and 2 enable us to efficiently determine $\text{cnt}_i(t_f, t_d)$ as follows.

Let $\eta_i(t_f)$ denote the number of jobs of τ_i that have their entire scheduling windows prior to t_f. The following formula for computing η_i was derived in [10]:

$$\eta_i(t_f) = \max\left(0, \left\lfloor \frac{t_f - T_i}{P_i} + 1 \right\rfloor\right). \tag{8}$$

Note that the $(\eta_i(t_f)+1)$'th job is released at time-instant $\eta_i(t_f) \times P_i$ (and its scheduling window extends to $(\eta_i(t_f) \times P_i + T_i)$, which is $> t_f$).

If

$$\eta_i(t_f) \times P_i < t_f \tag{9}$$

Then t_f lies within the scheduling window of this job of τ_i. In this case, the number of additional jobs of τ_i with deadline $\leq t_d$ equals

$$\max\left(0, \left\lfloor \frac{t_d - \left(\eta_i(t_f) \times P_i\right)}{T_i} \right\rfloor\right) \tag{10}$$

since the first such job is released at time $\left(\eta_i(t_f) \times P_i\right)$ and subsequent job releases are T_i time units apart, and each job release is the deadline of the previously-released job.

Else (i.e., Condition 9 does not hold) t_f does not lie within the scheduling window of a job of τ_i, in which case a job is released at t_f and hence the number of additional jobs of τ_i equals

$$\max\left(0, \left\lfloor \frac{t_d - t_f}{T_i} \right\rfloor\right). \tag{11}$$

Summarizing for tasks other than the triggering task,

$$\text{cnt}_i(t_f, t_d) = \eta_i(t_f) + \begin{cases} \textit{Expression } 10 & \text{if Condition 9 holds} \\ \textit{Expression } 11 & \text{otherwise} \end{cases} \tag{12}$$

For the triggering task, we have seen (Figure 2) that no jobs are released over the interval $[t_f - T_\ell, t_f)$. The number of jobs released prior to time-instant $(t_f - T_\ell)$ is equal to $\left(\lfloor (t_f - T_\ell)/P_\ell \rfloor + 1\right)$; hence, the total number of jobs is given by

$$\text{cnt}_\ell(t_f, t_d) = \left(\left\lfloor \frac{t_f - T_\ell}{P_\ell} \right\rfloor + 1\right) + \max\left(0, \left\lfloor \frac{t_d - t_f}{T_\ell} \right\rfloor\right). \tag{13}$$

Since Expressions 12 and 13 can clearly be evaluated in constant time and $\delta_i(t_f, t_d) = C_i \times \text{cnt}_i(t_f, t_d)$, it follows that $\delta_i(t_f, t_d)$ can be determined in constant time for given τ_i, t_f, and t_d.

4.2 Approximating $\delta_i(t_f, t_d)$

In this section we will, in the spirit of the Albers-Slomka approximation [3] (see Section 2.2.1), derive an upper bound $\delta_i^{\langle\kappa\rangle}(t_f, t_d)$ on $\delta_i(t_f, t_d)$. It helps to take a closer look at the Albers-Slomka approximation in order to better understand our new approximation. Recall the Albers-Slomka approximation (Expression 4, reproduced below) to the demand bound function dbf_i of a constrained-deadline sporadic task that is characterized by a WCET parameter C_i, a relative deadline D_i, and a period T_i:

$$\text{dbf}_i^{\langle\kappa\rangle}(t) = \begin{cases} \text{dbf}_i(t), & \text{if } t \le \kappa \times T_i + D_i \\ C_i + \left(\frac{C_i}{T_i}\right) \times (t - D_i), & \text{otherwise.} \end{cases}$$

Figure 3 (a) provides a visual representation of $\text{dbf}_i^{\langle\kappa\rangle}(t)$ as a function of t. The blue step function denotes the exact demand bound function $\text{dbf}_i(t)$. The red line tracks the demand bound function over $[0, D_i)$; after that, it is a straight line with slope C_i/T_i. For a given value of κ, $\text{dbf}_i^{\langle\kappa\rangle}(t)$ traces the blue step function for the first $(\kappa + 1)$ steps (i.e., for $t \le (\kappa \times T_i + D_i)$), and the red line for larger values of t.

Figures 3 (b) and (c) mimic the spirit of [3] (and hence the graph in Figure 3 (a)) upon the $\delta_i(t_f, t_d)$ function for a given value of t_f. Figure 3 (b) corresponds to the situation when all jobs are released as soon as possible. Recall that the triggering task τ_ℓ is required to release a job at time-instant t_f and may therefore postpone the release of a job that is eligible to be released during the time interval $(t_f - T_\ell, t_f)$; Figure 3 (c) represents this possibility.

- The blue line denotes the maximum cumulative execution requirement by jobs of τ_i that have their deadline $\le t$.
- The red line is piece-wise linear with slope C_i/P_i for $t \in [0, t_f)$, and slope C_i/T_i thereafter.

For a given value of κ, our approximation $\delta_i^{\langle\kappa\rangle}(t_f, t_d)$ to the $\delta_i(t_f, t_d)$ function traces the blue line for the first κ steps, and the red line thereafter. Since we can easily compute (see Equation 8) how many jobs have deadline before t_f, and hence how many steps of the blue line occur $\le (t_f - T_i)$, we can easily compute $\delta_i^{\langle\kappa\rangle}(t_f, t_d)$. Algorithm 2 provides the details in pseudo-code form.

5 Analysis

Recall that in Algorithm 1, we repeatedly compute $\delta_i(t_f, t_d)$ for different combinations of (τ_i, t_f, t_d) values. Rather than using exact values here, let us instead replace $\delta_i(t_f, t_d)$ in Line 11 of Algorithm 1 with the κ-approximations $\delta_i^{\langle\kappa\rangle}(t_f, t_d)$. In Section 5.1 below, we examine the implications of using this approximation, rather than the exact $\delta_i(t_f, t_d)$ values, on the accuracy of our algorithm. Then in Section 5.2 we show that the worst-case running time of Algorithm 1 (using the approximation rather than exact values for $\delta_i(t_f, t_d)$ in Line 11) can be bounded by a pseudo-polynomial in the representation of the task system Γ that is being scheduled.

5.1 A speedup bound

We now quantify, via the speedup factor metric, the consequence of approximating $\delta_i(t_f, t_d)$ in Algorithm 1 with the κ-approximation $\delta_i^{\langle\kappa\rangle}(t_f, t_d)$. Lemma 3 below shows that $\delta_i^{\langle\kappa\rangle}(t_f, t_d)$ is always an over-approximation of $\delta_i(t_f, t_d)$, and bounds from above the maximum amount by which it can exceed the value of $\delta_i(t_f, t_d)$.

Lemma 3. *For all τ_i, t_f, and t_d,*

$$\delta_i(t_f, t_d) \le \delta_i^{\langle\kappa\rangle}(t_f, t_d) \tag{14}$$

$$\delta_i^{\langle\kappa\rangle}(t_f, td) \begin{cases} = \delta_i(t_f, t_d) & \textit{if } \delta_i(t_f, t_d) \le \kappa\, C_i \\ < \delta_i(t_f, t_d) + 2\, C_i & \textit{otherwise.} \end{cases} \tag{15}$$

PROOF SKETCH. Let us first examine Inequality 14. Observe that the slope of the red line in Figure 3 (b) increases from (C_i/P_i) to (C_i/T_i) at time-instant t_f. Hence the red line is an upper bound on the cumulative demand of jobs of τ_i in all scenarios in which a job of τ_i arrives at time-instant t_f — this covers both the top scenario in Figure 1 and the sole scenario in Figure 2. It is evident that the cumulative demand in the remaining scenario – the bottom

Algorithm 2: The κ-approximation (Assume: $t \ge t_f$)

```
Input: $\tau_i, t_f, \kappa$, and $t$
Output: The approximation $\delta_i^{\langle\kappa\rangle}(t_f, t)$
Compute $\eta_i$, the number of jobs with deadlines $\le t_f$, as per
  Equation 8:
    $\eta_i = \max\left(0, \left\lfloor \frac{t_f - T_i}{P_i} + 1 \right\rfloor\right)$
if ($\kappa \le \eta_i$) then // switch to the red line before $t_f$
    return $\left(C_i + (t_f - T_i) \times \frac{C_i}{P_i} + (t - t_f) \times \frac{C_i}{T_i}\right)$
end
else // ($\kappa > \eta_i$: switch to the red line after $t_f$
    if $\text{cnt}_i(t_f, t) \le \kappa$ then // Exact: the blue line
        return $\left(C_i \times \text{cnt}_i(t_f, t)\right)$
    end
    else // Approximate: the red line
        return $\left(C_i \times \eta_i + (t - t_f) \cdot \frac{C_i}{T_i}\right)$
    end
end
```

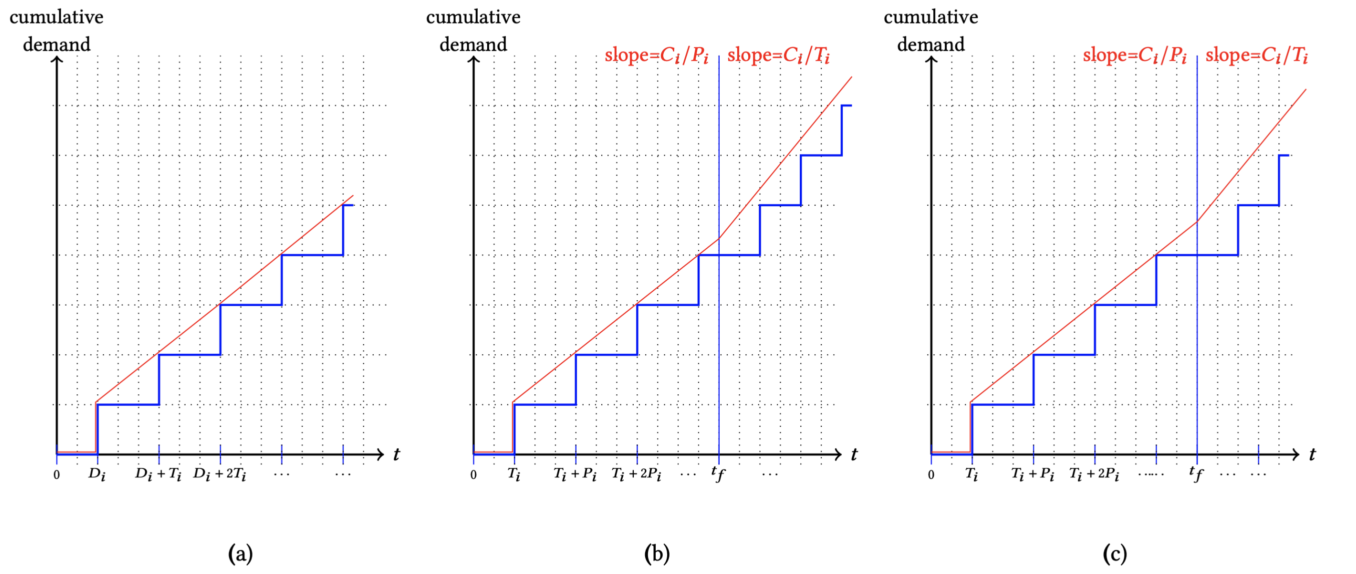

Figure 3: Depicting the approximations from (a):– Albers and Slomka [3]; (b) and (c):– this paper.

scenario in Figure 1 – cannot exceed the cumulative demand of the the top scenario in Figure 1, since the jobs that are released more frequently (i.e., T_i, rather than P_i, time units apart) begin arriving later.

We now turn our attention to Inequality 15. The proof for this upper bound essentially mirrors the proof in [3] for the second inequality in Expression 6 ("$\text{dbf}_i^{\langle\kappa\rangle}(t) < \text{dbf}_i(t) + C_i$"), with an additional "$+C_i$", which is only needed for the scenario depicted in Figure 3 (c), to account for the job release that may have been postponed during the time interval $[t_f - T_\ell, t_f)$. □

The speedup bound of Lemma 4 below follows from Lemma 3.

Lemma 4. *If Algorithm 1 determines that the initial speed of the processor is $\widehat{s_o}$, then*

$$\left(\frac{\kappa}{\kappa+2}\right) \times \widehat{s_o} \tag{16}$$

is a lower bound on the initial minimum speed at which the processor can be run and still guarantee to always meet all deadlines.

PROOF SKETCH. We first observe that

$$\left(\frac{\kappa}{\kappa+2}\right) \times \delta_i^{\langle\kappa\rangle}(t_f, t_d) < \delta_i(t_f, t_d) \tag{17}$$

for all τ_i, t_f, and t_d. This follows from Lemma 3, since $\delta_i^{\langle\kappa\rangle}(t_f, t_d) = \delta_i(t_f, t_d)$ for all $\delta_i(t_f, t_d) \le \kappa C_i$, while for $\left(\delta_i(t_f, t_d) > \kappa C_i\right)$ we have

$$\begin{aligned}
& \delta_i^{\langle\kappa\rangle}(t_f, t_d) && < && \delta_i(t_f, t_d) + 2C_i \text{ (by Eq. 15)} \\
\equiv\ & \frac{\delta_i^{\langle\kappa\rangle}(t_f, t_d)}{\delta_i(t_f, t_d)} && < && 1 + 2 \times \left(\frac{C_i}{\delta_i(t_f, t_d)}\right) \\
\Rightarrow\ & \frac{\delta_i^{\langle\kappa\rangle}(t_f, t_d)}{\delta_i(t_f, t_d)} && < && 1 + 2 \times \frac{1}{\kappa} \\
\equiv\ & \delta_i^{\langle\kappa\rangle}(t_f, t_d) && < && \left(1 + \frac{2}{\kappa}\right) \times \delta_i(t_f, t_d) \\
\equiv\ & \left(\frac{\kappa}{\kappa+2}\right) \cdot \delta_i^{\langle\kappa\rangle}(t_f, t_d) && < && \delta_i(t_f, t_d)
\end{aligned} \tag{18}$$

as stated in Inequality 17.

Now, it may be verified that Algorithm 1 essentially returns the smallest s_o for which Expression 7, with each $\delta_i(t_f, t_d)$ replaced by $\delta_i^{\langle\kappa\rangle}(t_f, t_d)$, is satisfied, for all combinations of t_f, t_d values. If $\widehat{s_o}$ is the value returned by Algorithm 1, it therefore follows from Inequality 17 above that $\left(\frac{\kappa}{\kappa+2}\right) \cdot \widehat{s_o}$ is a lower bound on the value of s_o for which the original Expression 7 is satisfied, for all combinations of t_f, t_d values. And as argued in Section 4, this is the smallest value at which the processor must initially be run in order to not miss any deadlines even in the event of prediction failures. Lemma 4 follows. □

5.2 Runtime Complexity

We will show below that Algorithm 1, Line 4, can safely assign the variable H a value that is pseudo-polynomial in the representation of the task system Γ under analysis, and that each set K that is computed in Algorithm 1, Line 9 comprises no more than $\kappa \times |\Gamma|$

elements. The overall worst-case running time of Algorithm 1 can then be written as

$$\overbrace{H}^{①} \times \overbrace{|\Gamma|}^{②} \times \overbrace{(\kappa \times |\Gamma|)}^{③} \times \overbrace{|\Gamma|}^{④} \tag{19}$$

where

① accounts for the choice of triggering instant t_f;
② counts the choice of triggering task τ_ℓ;
③ bounds the number of deadlines that must be considered explicitly; and
④ denotes the cost of updating speed (Equation 7)

Since the terms ②, ③, ④ are each polynomial, this is clearly pseudo-polynomial in the representation of the task system Γ under analysis.

It remains to explain why the values of H and $|K|$ may be upper-bounded as claimed above. We will also show how the initial value of s_o on Line 3 of Algorithm 1 can be calculated.

Calculating the initial value of s_o and H. First, we see from Lemma 4 that we are, in effect, willing to run the processor with an initial speed s_o that is up to $(1 + 2/\kappa)$ times as large as the lowest possible speed with which deadline misses can be avoided. From this perspective, therefore, there is no downside in having s_o be assigned (in Line 3 of Algorithm 1) a starting value as large as

$$s_{\min} \overset{\text{def}}{=} \left(1 + \frac{2}{\kappa}\right) \times \sum_{\tau_i \in \Gamma} \left(\frac{C_i}{P_i}\right)$$

since $\sum_{\tau_i \in \Gamma} \left(\frac{C_i}{P_i}\right)$ is clearly a lower bound on s_o.

Furthermore, the approximation algorithm of Albers and Slomka [3] (see Expression 5) can easily be applied to determine, in polynomial time and to within an approximation factor of $(1 + 2/\kappa)$, the minimum speed of a processor upon which the constrained-deadline sporadic task system

$$\bigcup_{\tau_i \in \Gamma} \{(C_i, T_i, P_i)\},$$

representing all possible consistent behaviors of Γ, is guaranteed to meet all deadlines. We will therefore initialize s_o in Line 3 of Algorithm 1 to be the larger of this speed and $s_{\min}$. The value of s_o is subsequently only ever *in*creased by Algorithm 1 (Line 11). From this we get two desirable properties: (1) no deadlines can be missed in consistent behaviors or in inconsistent behaviors prior to a misprediction, in line with our previous assumptions; and (2) each consistent behavior of Γ is that of a sporadic task system with a utilization that is $\leq (\kappa/(\kappa + 2))$ relative to the speed of the processor, effectively making consistent behaviors (or inconsistent behaviors up to a mis-prediction) that of a bounded-utilization task system. Recall from Section 2.2 that the duration of the initial busy interval for a bounded-utilization sporadic task system is bounded by a pseudo-polynomial; hence the value of H is pseudo-polynomial[2] in the representation of Γ.

Bounding $|K|$: Since we are only approximating each $\delta_i(t_f, t_d)$ to be exact for the first κ steps, it follows, using arguments virtually identical to the ones that explain the Albers-Slomka approximation [3], that K need only include the first $(\kappa + 1)$ deadlines of each task, and hence $|K|$ is no larger than

$$|\Gamma| \times (\kappa + 1). \tag{20}$$

In fact, since $t_d > t_f$ only those of these first $(\kappa + 1)$ deadlines of each task that are $> t_f$ need to be in K

6 Conclusions

We have studied the problem of achieving more efficient implementations of systems of implicit-deadline sporadic tasks upon preemptive unit-speed processors, where each task $\tau_i = (C_i, T_i)$ is additionally characterized by a prediction P_i of its period parameter that is more optimistic (i.e., larger) than the value that is conservatively assigned to T_i and guaranteed to always be correct. We have proposed a formalization of this problem within the Algorithms using Predictions framework. We have developed a pseudo-polynomial time algorithm that determines an initial speed $s_o < 1$ at which the processor should be run such that all deadlines will always be met by (i) running the processor at speed s_o so long as the predictions hold; and (ii) immediately increasing the processor speed to 1 upon detecting a prediction failure. We have shown that this speed that is determined by our algorithm is within a $(1 + 2/\kappa)$ factor of the minimum possible value, where κ is a tuning parameter: the larger the value of κ, the closer the computed speed is to the optimal one (at the cost of greater, although still pseudo-polynomially bounded, worst-case running time for the algorithm that determines this initial speed).

Although we have restricted consideration to implicit-deadline sporadic task systems, we point out that all our results extend in a straightforward manner to constrained-deadline task systems in which each task τ_i is characterized by the three-tuple (C_i, D_i, T_i) (as discussed in Section 2.2), $\underline{\text{plus}}$ a prediction P_i on the value of T_i.

In closing, we point out that our goal here has been to develop a pseudo-polynomial time algorithm for computing an initial processor speed s_o that can, by an appropriate choice of the tuning parameter κ be made arbitrarily close to the minimum such speed. We have not attempted to obtain the most efficient pseudo-polynomial time algorithm, occasionally avoiding discussion of possible optimizations that may further speed up the algorithm (although it would remain pseudo-polynomial) for ease of presentation/comprehension. Similarly, our proofs have not been aimed at identifying the tightest speedup bound in Lemma 4 – it is possible that the $\left(\frac{\kappa}{\kappa+2}\right)$ term in Expression 16 could be made larger with more careful analysis. We also point out that this work has not considered *smoothness* or *learnability*, important concepts for algorithms with predictions (as described in Section 2.1).

[2]In fact, the value of H is pseudo-*linear* [2, Def. 2] in the representation of Γ —it depends in a linear fashion upon the magnitude of the largest integer in the representation of Γ— making the complexity of Algorithm 1 as a whole pseudo-linear as well. On the other hand, we note that Algorithm 1 is not *robust* [2, Def. 3] due to the for-loop on Line 5.

Acknowledgments

This research was supported in part by the US National Science Foundation (Grants CNS-2141256 and CPS-2229290), the Swedish Research Council (Grants 2018-04446 and 2023-04586), the German Research Foundation (DFG, grant 547924951), and the Dutch Research Council (NWO, grants OCENW.GROOT.2019.015 and NETWORKS-024.002.003).

References

[1] Kunal Agrawal, Sanjoy Baruah, Michael A. Bender, and Alberto Marchetti-Spaccamela. 2023. The Safe and Effective Use of Low-Assurance Predictions in Safety-Critical Systems. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 262)*, Alessandro V. Papadopoulos (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 3:1–3:19. https://doi.org/10.4230/LIPIcs.ECRTS.2023.3

[2] Kunal Agrawal, Sanjoy K. Baruah, and Pontus Ekberg. 2023. Rethinking Tractability for Schedulability Analysis. In *IEEE Real-Time Systems Symposium, RTSS 2023, Taipei, Taiwan, December 2023*. IEEE.

[3] K. Albers and F. Slomka. 2004. An Event Stream Driven Approximation for the Analysis of Real-Time Systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*. IEEE Computer Society Press, Catania, Sicily, 187–195.

[4] Antonios Antoniadis, Peyman Jabbarzade Ganje, and Golnoosh Shahkarami. 2022. A Novel Prediction Setup for Online Speed-Scaling. In *SWAT (LIPIcs, Vol. 227)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:20.

[5] Yossi Azar, Stefano Leonardi, and Noam Touitou. 2021. Flow time scheduling with uncertain processing time. In *STOC*. ACM, 1070–1080.

[6] Yossi Azar, Stefano Leonardi, and Noam Touitou. 2022. Distortion-Oblivious Algorithms for Minimizing Flow Time. In *SODA*. SIAM, 252–274.

[7] Eric Balkanski, Tingting Ou, Clifford Stein, and Hao-Ting Wei. 2023. Scheduling with Speed Predictions. In *WAOA (Lecture Notes in Computer Science, Vol. 14297)*. Springer, 74–89.

[8] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. 2020. Learning Augmented Energy Minimization via Speed Scaling. In *NeurIPS*. 15350–15359.

[9] S. Baruah, R. Howell, and L. Rosier. 1990. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems: The International Journal of Time-Critical Computing* 2 (1990), 301–324.

[10] S. Baruah, A. Mok, and L. Rosier. 1990. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11th Real-Time Systems Symposium*. IEEE Computer Society Press, Orlando, Florida, 182–190.

[11] Ashikahmed Bhuiyan, Sai Sruti, Zhishan Guo, and Kecheng Yang. 2019. Precise scheduling of mixed-criticality tasks by varying processor speed. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems, RTNS 2019, Toulouse, France, November 06-08, 2019*, Jérôme Ermont, Ye-Qiong Song, and Christopher D. Gill (Eds.). ACM, 123–132. https://doi.org/10.1145/3356401.3356410

[12] Alan Burns and Robert I. Davis. 2018. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.* 50, 6 (2018), 82:1–82:37. https://doi.org/10.1145/3131347

[13] Friedrich Eisenbrand and Thomas Rothvoß. 2010. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*.

[14] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. 2021. Non-Clairvoyant Scheduling with Predictions. In *SPAA*. ACM, 285–294.

[15] Alexandra Anna Lassota, Alexander Lindermayr, Nicole Megow, and Jens Schlöter. 2023. Minimalistic Predictions to Schedule Jobs with Online Precedence Constraints. In *ICML (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 18563–18583.

[16] Shi Li and Jiayi Xian. 2021. Online Unrelated Machine Load Balancing with Predictions Revisited. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 6523–6532.

[17] Alexander Lindermayr and Nicole Megow. [n. d.]. Open source project: Algorithms With Predictions. https://algorithms-with-predictions.github.io. Accessed: 2023-08-02.

[18] Alexander Lindermayr and Nicole Megow. 2022. Permutation Predictions for Non-Clairvoyant Scheduling. In *SPAA*. ACM, 357–368.

[19] Alexander Lindermayr, Nicole Megow, and Martin Rapp. 2023. Speed-Oblivious Online Scheduling: Knowing (Precise) Speeds is not Necessary. *IJCAI*.

[20] C. Liu and J. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *J. ACM* 20, 1 (1973), 46–61.

[21] Michael Mitzenmacher and Sergei Vassilvitskii. 2021. Algorithms with Predictions. In *Beyond the Worst-Case Analysis of Algorithms*, Tim Roughgarden (Ed.). Cambridge University Press, 646–662. https://doi.org/10.1017/9781108637435.037

[22] Michael Mitzenmacher and Sergei Vassilvitskii. 2022. Algorithms with Predictions. *Commun. ACM* 65, 7 (jun 2022), 33?35. https://doi.org/10.1145/3528087

[23] Aloysius Mok. 1983. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. Ph. D. Dissertation. Laboratory for Computer Science, Massachusetts Institute of Technology. Available as Technical Report No. MIT/LCS/TR-297.

[24] Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving Online Algorithms via ML Predictions. In *NeurIPS*. 9684–9693.

[25] Tim Roughgarden (Ed.). 2020. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press. https://doi.org/10.1017/9781108637435

[26] Tianning She, Zhishan Guo, and Kecheng Yang. 2022. Scheduling Constrained-Deadline Tasks in Precise Mixed-Criticality Systems on a Varying-Speed Processor. In *RTNS 2022: The 30th International Conference on Real-Time Networks and Systems, Paris, France, June 7 - 8, 2022*, Yasmina Abdeddaïm, Liliana Cucu-Grosjean, Geoffrey Nelissen, and Laurent Pautet (Eds.). ACM, 94–102. https://doi.org/10.1145/3534879.3534897

[27] Sai Sruti, Ashikahmed Bhuiyan, and Zhishan Guo. 2018. Work-in-Progress: Precise Scheduling of Mixed-Criticality Tasks by Varying Processor Speed. In *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*. IEEE Computer Society, 173–176. https://doi.org/10.1109/RTSS.2018.00033

[28] Sudharsan Vaidhun, Tianning She, Qijun Gu, Sajal K. Das, Kecheng Yang, and Zhishan Guo. 2023. Precise Mixed-Criticality Scheduling on Varying-Speed Multiprocessors. *IEEE Trans. Computers* 72, 1 (2023), 43–54. https://doi.org/10.1109/TC.2022.3197078

[29] Steve Vestal. 2007. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proceedings of the Real-Time Systems Symposium*. IEEE Computer Society Press, Tucson, AZ, 239–243.

[30] Tianming Zhao, Wei Li, and Albert Y. Zomaya. 2022. Real-Time Scheduling with Predictions. In *RTSS*. IEEE, 331–343.