Preemptive Uniprocessor Scheduling of Mixed-Criticality Sporadic Task Systems

Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo d'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne van Der Ster, Leen Stougie

Preemptive Uniprocessor Scheduling of Mixed-Criticality Sporadic Task Systems

SANJOY BARUAH, University of North Carolina
VINCENZO BONIFACI, IASI – Consiglio Nazionale delle Ricerche
GIANLORENZO D'ANGELO, Gran Sasso Science Institute (GSSI)
HAOHAN LI, University of North Carolina
ALBERTO MARCHETTI-SPACCAMELA, Sapienza Università di Roma
SUZANNE VAN DER STER, Vrije Universiteit Amsterdam
LEEN STOUGIE, Vrije Universiteit Amsterdam & CWI

Systems in many safety-critical application domains are subject to certification requirements. For any given system, however, it may be the case that only a subset of its functionality is safety-critical and hence subject to certification; the rest of the functionality is non-safety-critical and does not need to be certified, or is certified to lower levels of assurance. The certification-cognizant runtime scheduling of such mixed-criticality systems is considered. An algorithm called EDF-VD (for Earliest Deadline First with Virtual Deadlines) is presented: this algorithm can schedule systems for which any number of criticality levels are defined. Efficient implementations of EDF-VD, as well as associated schedulability tests for determining whether a task system can be correctly scheduled using EDF-VD, are presented. For up to 13 criticality levels, analyses of EDF-VD, based on metrics such as processor speedup factor and utilization bounds, are derived, and conditions under which EDF-VD is optimal with respect to these metrics are identified. Finally, two extensions of EDF-VD are discussed that enhance its applicability. The extensions are aimed at scheduling a wider range of task sets, while preserving the favorable worst-case resource usage guarantees of the basic algorithm.

General Terms: Algorithms

Additional Key Words and Phrases: Mixed criticality, sporadic task system, preemptive scheduling

---

Some of the results in this article have previously appeared in preliminary form in *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, IEEE, pp. 145–154, and in *Proceedings of the 19th European Symposium on Algorithms*, Springer, pp. 555–566.
Baruah's research was supported in part by NSF grants CNS 1016954, CNS 1115284, CNS 1218693, and CNS 1409175; and ARO grant W911NF-09-1-0535.
Authors' addresses: S. Baruah, The University of North Carolina, CB 3175, Department of Computer Science, Sitterson Hall, Chapel Hill, NC 2759-3175; V. Bonifaci, Istituto di Analisi del Sistemi ed Informatica "Antonio Ruberti", Consiglio Nazionale delle Ricerche, Via dei Taurini, 19, 00185 Roma, Italy; G. D'Angelo, Gran Sasso Science Institute (GSSI), Viale Francesco Crispi 7, 67100, L' Aquila, Italy; H. Li, Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; A. Marchetti-Spaccamela, Dipartimento di Ingegneria Informatica Automatica e Gestionale "Antonio Ruberti", Università di Roma "La Sapienza", Via Ariosto25, 00185, Rome, Italy; S. van der Ster and L. Stougie, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands. Correspondence email: baruah@cs.unc.edu.

1. INTRODUCTION

In implementing safety-critical embedded systems, there is an increasing trend towards integrated computing environments, in which multiple functionalities are implemented on a shared computing platform; this trend is evident in industry-driven initiatives such as Integrated Modular Avionics (IMA) [Prisaznuk 1992] in aerospace and AUTOSAR (AUTomotive Open System ARchitecture – see `www.autosar.org`) in the automotive industry. It is often the case that not all functionalities implemented in this manner upon a shared platform are equally important (or critical) to the overall system; such systems are called *mixed-criticality* (MC) systems.

In some application domains, critical functionalities are subject to mandatory certification by statutory *certification authorities* (CAs). In conjunction with the increasing trend towards computerized control of functionalities, including non-safety-critical ones such as entertainment and comfort features, the trend towards integration of multiple functionalities upon a shared platform means that even in highly safety-critical systems, typically only a relatively small fraction of the overall system is actually of highly critical functionality and subject to certification; the rest of the system does not need to be certified, or is certified to lower levels of assurance. For instance, the RTCA DO-178B software standard specifies several different criticality levels, with the system designer expected to assign one of these criticality levels to each task — Figure 1 lists the criticality levels, and intended interpretations, that are specified in this standard.

In order to certify a system as being correct, the CA must make certain assumptions about the worst-case behavior of the system during run time. CAs tend to be very conservative and require that the safety-critical functionalities be shown to be correct at a very high level of assurance; the remaining (non-safety-critical) functionalities are usually validated correct by the system designer/ integrator at lower levels of assurance.

Worst-Case Execution Time. The worst-case execution time (WCET) abstraction plays a central role in the analysis of real-time systems. For a specific piece of code and a particular platform upon which this code is to execute, the WCET of the code denotes the maximum duration of time the code would take to execute upon the platform. Determining the exact WCET of an arbitrary piece of code is clearly a provably undecidable problem. However, even when severe restrictions are placed upon the structure of the code (e.g., loops bounds must be known at compile time), sophisticated features that are found upon the powerful processors used in embedded systems today (such as multi-level cache, deep pipelining, speculative out-of-order execution, etc.) are hard to analyze and make it extremely difficult to determine WCETs precisely. Devising analytical techniques for obtaining tight upper bounds on WCETs is currently a very active and thriving area of research, and sophisticated tools incorporating the latest results of such research have been developed (see Wilhelm et al. [2008] for an excellent, if slightly dated, survey).

One consequence of the different levels of assurance of correctness sought by the CA and the system designer is that the same piece of code may be characterized by different WCET parameters for the purposes of certification, and for design validation. This is because different tools for determining WCET bounds may be more or less conservative than one another: a more conservative tool determines an upper bound on the actual WCET of a piece of code at a higher level of assurance than a less conservative tool. The upper bound determined by the more conservative tool is *larger* – sometimes by several

Level	Failure Condition	Interpretation
A	Catastrophic	Failure may cause a crash
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers
C	Major	Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries)
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change)
E	No Effect	Failure has no impact on safety, aircraft operation, or crew workload.

Fig. 1. DO-178B is a software development process standard, *Software Considerations in Airborne Systems and Equipment Certification*, published by RTCA, Incorporated. The United States Federal Aviation Authority (FAA) accepts the use of DO-178B as a means of certifying software in avionics applications. RTCA DO-178B assigns criticality levels to tasks, categorized by effects on commercial aircraft.

orders of magnitude – than the one determined by the less conservative tool. Although it may be necessary (i.e., mandated by a statutory Certification Authority) to use a very conservative tool for validating the correctness of safety-critical functionalities, less conservative tools should suffice for validating the correctness of less critical functionalities. Based on this observation, Vestal [2007] proposed that multiple different WCET values be specified, with the different values being determined at different levels of assurance. These different values are obtained by using different execution-time analysis tools; we expect that the tool used by the CA is more conservative than the one used by the system engineer, and hence the CA's WCET estimates are larger than the estimates used during the design process.

Multiple Criticality Levels. We have considered two criticality levels – needing certification, and not needing certification – in the previous discussion. However, in many safety-critical application domains, more than two criticality levels are specified; for instance, the DO-178B standard that is widely used in the avionics domain specifies five different criticality levels (A:-catastrophic/ B:-hazardous/ C:-major/ D:-minor/ E:-no effect – the adjectives denote the potential consequences of failure at the corresponding level) and mandates that each functionality be assigned one of these levels. Functionalities at higher criticality levels (A is the highest level, and E the lowest) are then subject to more rigorous validation requirements.

Context and Related Work. In traditional (e.g., not mixed-criticality) real-time systems, a sporadic task [Mok 1983; Leung and Whitehead 1982] τ_i is characterized by a WCET c_i, a relative deadline d_i, and a period p_i; such a task generates an unbounded sequence of jobs, with successive jobs arriving at least p_i time units apart, and each job needing up to c_i units of execution by a deadline that occurs d_i time units after the job's arrival. As already mentioned, Vestal [2007] proposed generalizing the model to mixed-criticality systems by allowing for several WCETs to be specified for each task, and studied the fixed-priority scheduling of such mixed-criticality sporadic task systems on a preemptive uniprocessor.

The preemptive uniprocessor scheduling of collections of mixed-criticality *independent jobs* was studied in Baruah et al. [2010a, 2010b, 2012]. An efficient scheduling algorithm and associated polynomial-time schedulability test was proposed that makes the following guarantee: any 2-level mixed-criticality system that can be scheduled by a clairvoyant exact algorithm on a given processor can be scheduled by this algorithm

on a processor that is $(1+\sqrt{5})/2 \approx 1.618$ times as fast. In Li and Baruah [2010], this result was extended to mixed-criticality sporadic task systems: a scheduling algorithm and associated pseudopolynomial-time schedulability test was proposed that gives the same guarantee. Guan et al. [2011] subsequently proposed an algorithm called PLRS that only has quadratic runtime complexity, able to schedule a wider range of instances.

Earlier work of the authors introduced the scheduling algorithm EDF-VD, that will be presented in Section 3. The idea is that higher-criticality tasks have their deadlines reduced as long as the system is in lower criticality levels, to ensure schedulability across a criticality change. Roughly, if there are two criticality levels, then high-criticality tasks have two deadlines. One deadline is defining the real deadline of the task, the other is a virtual earlier deadline that is used to increase the likelihood that EDF schedules high-criticality tasks before low-criticality ones.

Subsequently, virtual deadlines have been studied by other authors. Ekberg and Yi [2012, 2014] also scale relative deadlines, but their approach is somewhat different. They extend the concept of demand bound function to the mixed-criticality setting and provide a schedulability test. Namely, in the case of two levels, they consider to what extent the deadline of each high-criticality task can be lowered without violating the schedulability condition. Since there are exponentially many possibilities, a pseudopolynomial-time heuristic is proposed that is essentially greedy. The approach has then been extended [Ekberg and Yi 2014] assuming that criticality levels are represented by a directed acyclic graph. Easwaran [2013] introduces a different technique for determining the virtual deadlines, which also tries to decrease the deadlines of the high-criticality tasks separately. This technique, combined with the new schedulability test, seems to be able to schedule a larger fraction of randomly generated instances than the algorithm of Ekberg and Yi [2014]. As observed by Burns and Davis [2013], it is unclear whether the approach of Easwaran [2013] scales to more than two criticality levels. The downside of Easwaran [2013] and Ekberg and Yi [2012, 2014] is that the proposed preprocessing algorithms are not polynomial-time, no speedup bounds are provided and the authors' claim that their tests outperform EDF-VD is only supported by results on synthetic task sets.

Su and Zhu [2013] consider two criticality levels and a model in which low-criticality tasks can be released early. In this model they propose to exploit *elastic scheduling*, in which the period of a task can change. They propose a minimum service requirement for low-criticality tasks, expressed by a maximum period. The system is schedulable if the high-criticality tasks have a high-criticality execution time and for the low-criticality tasks the maximum period is considered. The intuition is to exploit the slack left by high-criticality tasks to execute the low-criticality tasks more frequently. Simulation results again show that (for certain parameters settings) an improvement over EDF-VD can be obtained. The downside of the model is that it relaxes the model of Vestal [2007] by assuming early release. We also observe that the proposal has higher overhead and it is unclear whether the approach can be effectively extended to more criticality levels.

We refer to Burns and Davis [2013] for an extensive survey of the research that has been conducted within the real-time scheduling community on mixed-criticality scheduling problems.

This Research. We study the scheduling of mixed-criticality sporadic task systems upon a single preemptive processor. An important special case of sporadic task systems are task systems in which each task τ_i satisfies the property that $d_i = p_i$ — such systems are called *implicit-deadline* or *Liu & Layland* task systems, to distinguish them from general, *arbitrary-deadline* task systems. We derive results for both implicit-deadline and arbitrary-deadline mixed-criticality sporadic task systems. Specifically, we propose a scheduling algorithm called EDF-VD, and establish the following.

(1) We show that any 2-level (respectively, 3-level) *implicit-deadline* task system that can be scheduled by a clairvoyant exact algorithm on a given processor, can be scheduled by EDF-VD on a processor that is $4/3$ (respectively, 2) times as fast.
(2) We show that any *K-level* implicit-deadline task system that can be scheduled by a clairvoyant exact algorithm on a given processor can be scheduled by EDF-VD on a processor that is f_K times as fast, where f_K is the solution of a nonlinear optimization problem. Using a global nonlinear continuous optimization solver, we compute the speedup bound f_K for up to 13 criticality levels.
(3) We also show that no non-clairvoyant algorithm can guarantee to always meet all deadlines on a processor that is less than $4/3$ times as fast as the processor available to the clairvoyant exact algorithm, thereby proving that EDF-VD is an optimal non-clairvoyant algorithm for 2-level implicit-deadline systems from the perspective of the processor speedup metric.
(4) For 2-level *arbitrary-deadline* task systems, we prove a speedup bound of $1+\sqrt{3}/2 \approx 1.866$. That is, we show that any 2-level arbitrary-deadline task system that can be scheduled by a clairvoyant exact algorithm on a given processor can be scheduled by EDF-VD on a processor that is $1 + \sqrt{3}/2$ times as fast.
(5) We provide a schedulability test for EDF-VD having polynomial-time complexity, and also show that the time complexity per scheduling decision is logarithmic in the number of tasks. Based on these runtime properties, it is evident that EDF-VD, in contrast to the algorithms in Li and Baruah [2010] and Guan et al. [2011], can be considered suitable for implementation in actual systems.
(6) Algorithm EDF-VD sets virtual deadlines in a uniform way for all tasks. In the case of a 2-level task system with implicit deadlines, the results in (1) and (3) show that EDF-VD has speedup $4/3$ and that in the worst case a speedup of $4/3$ is necessary for any algorithm. Despite this optimality result, it is of interest to investigate whether defining virtual deadlines nonuniformly might allow to schedule task sets that are not deemed schedulable by EDF-VD. We show that this is the case and present an algorithm, EDF-NUVD, that also allows scaling the deadlines nonuniformly by considering a different schedulability condition than the one used in (1). We show that, by optimizing the scaling parameters in EDF-NUVD, the resulting schedulability test recognizes a superset of the task sets schedulable by EDF-VD.

Organization. The remainder of this article is organized as follows. In Section 2, we formally describe the mixed-criticality model that we will be using in the remainder of this article. In Section 3, we consider implicit-deadline tasks; we provide a high-level description of EDF-VD (Section 3.1) and a formal analysis of its properties and behavior for implicit-deadline task sets (Sections 3.2–3.4). Then, in Section 4, we move on to analyze EDF-VD for arbitrary-deadline tasks. The efficient implementation of the runtime dispatching procedure is discussed in Section 5, while Section 6 discusses more optimistic runtime dispatching rules and analyzes how to extend EDF-VD to consider nonuniform scaling factors. We summarize our findings and conclude in Section 7.

2. MODEL AND DEFINITIONS

MC Task Systems. Let $K \geq 1$ be an integer. A K-level MC sporadic task system τ consists of a finite collection $(\tau_1, \ldots, \tau_n)$ of MC sporadic tasks. In the following, let $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$.

MC Tasks. An MC sporadic task τ_i of a K-level system is characterized by a *criticality level* $\chi_i \in [K]$ and a triple $(c_i, d_i, p_i) \in \mathbb{Q}_+^{\chi_i} \times \mathbb{Q}_+ \times \mathbb{Q}_+$, where

—$c_i = (c_i(1), c_i(2), \ldots, c_i(\chi_i))$ is a vector of *worst-case execution times* (WCET), one for each criticality level less than or equal to χ_i. We assume that $c_i(1) \leq c_i(2) \leq \cdots \leq c_i(\chi_i)$. It will be convenient to extend the definition of c_i to a K-dimensional vector, by letting $c_i(k) = c_i(\chi_i)$ when $\chi_i < k \leq K$;
—d_i is the *relative deadline* of the jobs of τ_i;
—p_i is the *minimum interarrival time* (or *period*) between two jobs of task τ_i.

MC Jobs. Task τ_i generates a sequence of jobs $(J_{i1}, J_{i2}, \ldots)$. An MC job J_{ij} of task τ_i is characterized by two parameters: $J_{ij} = (a_{ij}, \gamma_{ij})$, where

—$a_{ij} \in \mathbb{R}_+$ is the *arrival time* of the job;
—$\gamma_{ij} \in (0, c_i(\chi_i)]$ is the *execution requirement* of the job;
—the *(absolute) deadline* of job J_{ij} is $d_{ij} \stackrel{\text{def}}{=} a_{ij} + d_i$.

It is important to notice that neither the arrival times nor the execution requirements are known in advance. A collection $I = (a_{ij}, \gamma_{ij})_{i \in [n], j \geq 1}$ of arrival times and execution requirements is called a *scenario* for the task system.

The notation is summarized in Table I.

Semantics. The MC task model has the following semantics. Task τ_i generates an unbounded sequence of jobs $(J_{i1}, J_{i2}, \ldots)$, with successive jobs being released at least p_i time units apart. Job J_{ij} arrives at time a_{ij}, has a deadline at time d_{ij}, and needs to execute for some amount of time γ_{ij}. The value γ_{ij} is not known in advance; it is discovered by executing the job until it *signals* that it has completed execution.

The values of γ_{ij} for a given scenario of the system define the kind of behavior exhibited by the system under that scenario. The *criticality level* (or simply *level*) *of a scenario* is defined as the smallest integer k $(1 \leq k \leq K)$ such that $\gamma_{ij} \leq c_i(k)$ for all jobs J_{ij}. Note that such an integer always exists, since γ_{ij} is assumed to be at most $c_i(\chi_i)$.

The level of a scenario is always between 1 and K. In a scenario of level k, only the jobs from tasks of criticality at least k are required to be completed before their deadlines; all other jobs can be ignored and discarded.

In this article, we assume that there is only one processor to execute the jobs. We assume the processor to be *preemptive*: executing jobs may have their execution interrupted at any instant in time and resumed later, at no additional cost.

Definition 2.1. A schedule for a scenario of level k is *feasible* if every job J_{ij} with $\chi_i \geq k$ receives execution time γ_{ij} during its time window $[a_{ij}, d_{ij})$. A task system τ is *(clairvoyantly) feasible* if for every scenario of τ there exists a feasible schedule.

An *online* (or *non-clairvoyant*) scheduling policy for an MC task system τ discovers the criticality level of the scenario only by executing jobs. At each time instant, scheduling decisions can be based only on the partial information revealed thus far.

Definition 2.2. An online scheduling policy A is *correct* for a feasible task system τ if for any scenario of τ the policy generates a feasible schedule. A task system τ is *A-schedulable* if A is a correct online scheduling policy for τ.

Notice that it is always safe to ignore jobs whose criticality is less than k whenever the scenario has already exhibited a level of at least k. The level exhibited by a scenario can only increase, or remain constant, over time.

3. IMPLICIT-DEADLINE TASKS

In this section, we deal with *implicit-deadline* task systems. This means that the period of each task equals its relative deadline: $p_i = d_i$ for all $i \in [n]$. We define the *utilization*

Table I. Notation

Symbol	Meaning
K	number of criticality levels
n	number of tasks
$[n]$	$\{1, 2, \ldots, n\}$
i	task index
j	job index
k, l	level indices
τ	task set
τ_i	ith task of τ
χ_i	criticality of τ_i
$c_i(l)$	lth level WCET of τ_i
d_i	relative deadline of τ_i
p_i	period of τ_i
J_{ij}	jth job of τ_i
a_{ij}	arrival time of J_{ij}
d_{ij}	absolute deadline of J_{ij}
γ_{ij}	execution requirement of J_{ij}

of task τ_i at level k as

$$u_i(k) \stackrel{\text{def}}{=} \frac{c_i(k)}{p_i} \qquad i = 1, \ldots, n,\ k = 1, \ldots, \chi_i.$$

Then, the total utilization at level k of tasks that are of criticality level l is

$$U_l(k) \stackrel{\text{def}}{=} \sum_{i \in [n]:\, \chi_i = l} u_i(k) \qquad l = 1, \ldots, K,\ k = 1, \ldots, l.$$

For implicit-deadline systems, it is well known that, in the case of a single criticality level (i.e., $K = 1$), a task system is feasible on a speed-σ uniprocessor if and only if $U_1(1) \leq \sigma$ [Liu and Layland 1973]. This yields the following necessary condition for feasibility in mixed-criticality systems.

PROPOSITION 3.1. *If τ is feasible on a unit-speed processor, then*

$$\max_{k=1,\ldots,K} \sum_{l=k}^{K} U_l(k) \leq 1. \tag{1}$$

PROOF. For each $k = 1, \ldots, K$, consider a scenario where each task τ_i with $\chi_i \geq k$ releases jobs with execution requirement $c_i(k)$. □

In the sequel, we call a job *active* if it has been released but not yet completed. In the case of a single criticality level and a single processor, it is well known that the Earliest Deadline First (EDF) algorithm, which schedules the active job with earliest (absolute) deadline, is optimal [Liu and Layland 1973; Dertouzos 1974].

PROPOSITION 3.2. *If $K = 1$, then the Earliest Deadline First algorithm is a correct scheduling policy for a processor of speed σ if and only if $U_1(1) \leq \sigma$.*

In the presence of multiple criticality levels, EDF does not necessarily produce a feasible schedule, even if the total utilization at each level is less than 1. Consider the following example.

Example 3.3. Consider a task system $\tau = (\tau_1, \tau_2)$ with the following parameters.

τ_i	χ_i	$c_i(1)$	$c_i(2)$	p_i
τ_1	1	2	2	4
τ_2	2	1	5	6

Note that the total utilization at level 1 is $U_1(1) + U_2(1) = 2/3$, while at level 2 it is $U_2(2) = 5/6$. However, EDF may fail to meet deadlines, as follows: Assume jobs are released as early as possible. At time 0, EDF schedules the first job of τ_1. At time 2, the job finishes and EDF starts running the first job of τ_2. However, if it turns out that this job exhibits level-2 behavior, it will execute for five time units and miss its deadline at time 6. If we had started the other way around, either the first job of τ_2 would have finished after one time unit and there would have been enough time to schedule the job from τ_1, or the scenario would have exhibited level-2 behavior and we could have discarded the job from τ_1.

The scheduling algorithm that we propose is called EDF with Virtual Deadlines (EDF-VD) and is an adaptation of EDF that handles the problem sketched previously, while maintaining some of the desirable properties of EDF. In the remainder of this section, we first introduce the algorithm EDF-VD and give a corresponding schedulability condition for a system of K criticality levels. Then, we assess the quality of the algorithm using the notion of speedup factor.

3.1. Overview of EDF-VD for Implicit-Deadline Tasks

Let τ denote the MC implicit-deadline sporadic task system that is to be scheduled on a unit-speed preemptive processor. Algorithm EDF-VD consists of an *offline preprocessing* phase and a *runtime scheduling* phase. The first phase is performed prior to runtime and executes a schedulability test to determine whether τ can be successfully scheduled by EDF-VD or not. If τ is deemed schedulable, this phase also provides two output values that will serve as input for the runtime scheduling algorithm: an integer parameter k (with $1 \leq k \leq K$); and, for each task τ_i of τ, a parameter $\hat{d}_i$, called *virtual deadline*, which is never larger than d_i. The second phase performs the actual runtime scheduling and consists of K variants, called EDF-VD(1), EDF-VD(2), ..., EDF-VD(K). Each of these is related to a different value of the parameter k that was provided by the first phase; that is, at runtime, the variant EDF-VD(k) is applied. Intuitively, if the scenario is exhibiting a level smaller than or equal to k, then jobs are scheduled according to EDF with respect to the virtual deadlines $(\hat{d}_i)_{i=1}^n$. As soon as the scenario exhibits a level greater than k, jobs are scheduled according to EDF with respect to the original deadlines $(d_i)_{i=1}^n$. In the following, we give more details on the two phases.

Offline Preprocessing Phase. The algorithm for the first phase is given in pseudocode form in Algorithm 1. If $\sum_{l=1}^{K} U_l(l) \leq 1$, then all jobs can be scheduled for their worst-case execution times at their own criticality level. Therefore, in this case, the virtual deadlines of all the tasks are set to be equal to the original deadlines (see lines 1–5). Otherwise, at line 7, we test whether there exists a $1 \leq k \leq K$ satisfying condition (3), which is given in Section 3.2 along with its correctness proof. If no such k exists, then τ is deemed unschedulable (line 9). If there exists a k satisfying condition (3), then the virtual deadlines of all tasks having criticality level $1, \ldots, k$ are set to be equal to the original deadlines (line 14), while those of tasks having criticality level $k+1, \ldots, K$ are scaled by a factor x (line 16). The value of x is specified at line 11 and will be explained later in Section 3.2. Note that if condition (3) is satisfied, then x is well defined. Moreover, since $\sum_{l=1}^{K} U_l(l) > 1$, then $x < 1$.

ALGORITHM 1: EDF with Virtual Deadlines (EDF-VD) – Offline preprocessing phase (for implicit-deadline task systems)

Input: task system $\tau = (\tau_1, \ldots, \tau_n)$ to be scheduled on a unit-speed preemptive processor

1: **if** $\sum_{l=1}^{K} U_l(l) \leq 1$ **then**
2: $k \leftarrow K$
3: **for** $i = 1, 2, \ldots, n$ **do**
4: $\hat{d}_i \leftarrow d_i$
5: **end for**
6: **else**
7: Let k ($1 \leq k < K$) be such that (3) holds
8: **if** no such k exists **then**
9: **return** unschedulable
10: **else**
11: Let $x \in \left[\frac{\sum_{l=k+1}^{K} U_l(k)}{1-\sum_{l=1}^{k} U_l(l)}, \frac{1-\sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)} \right]$
12: **for** $i = 1, 2, \ldots, n$ **do**
13: **if** $\chi_i \leq k$ **then**
14: $\hat{d}_i \leftarrow d_i$
15: **else**
16: $\hat{d}_i \leftarrow x\, d_i$
17: **end if**
18: **end for**
19: **end if**
20: **end if**
21: **return** (schedulable, k, $(\hat{d}_i)_{i=1}^{n}$)

Runtime Scheduling Phase. A pseudocode describing the runtime phase is given in Algorithm 2. The function `current_level()` returns the level exhibited by the scenario so far, that is,

$$\texttt{current\_level()} = \min\{l \in [K] \,:\, \tilde{\gamma}_{ij} \leq c_i(l) \text{ for all jobs } J_{ij} \text{ (partially) executed so far}\},$$

where $\tilde{\gamma}_{ij}$ is the part of γ_{ij} that has been observed thus far. (When `current_level()` $= l$, we also say that *the system is in level* l.)

Algorithm 2 takes as input the integer k and the virtual (relative) deadlines $\hat{d}_i$ computed during the preprocessing phase. As long as the system is in level $1, 2, \ldots, k$, the tasks having level greater than or equal to `current_level()` are scheduled according to EDF with respect to these virtual deadlines, while tasks of level smaller than `current_level()` are discarded (lines 1–5). When the system is observed to reach a level greater than k, all jobs from tasks of criticality k or below are discarded and the original deadlines of the tasks of criticality $k+1$ and higher are restored. The priority ordering of the jobs that are active at the time when the system turns to a level greater than k is computed according to the original deadlines at lines 6–7 and the jobs are scheduled accordingly. From this moment onwards, EDF is applied to all the tasks having level greater than or equal to `current_level()` (lines 9–14). Note that if $k = K$, then Algorithm 2 is simply EDF, as no scaling of the deadlines occurs in Algorithm 1.

The efficient implementation of Algorithm 2 is discussed in Section 5.

Note that once the parameter k is fixed, when applying EDF-VD(k), the criticality levels are divided into two sets, where the levels in the same set are treated homogeneously, *as if* there were only two levels. The parameter k determines the boundary between these two sets.

ALGORITHM 2: EDF with Virtual Deadlines (EDF-VD) – Runtime scheduling

Input: task system $\tau = (\tau_1, \ldots, \tau_n)$, integer k $(1 \leq k \leq K)$, virtual deadlines $(\hat{d}_i)_{i=1}^n$

1: **loop**
2: **on job arrival:**
3: if a job of task τ_i arrives at time t, assign it a *virtual absolute deadline* equal to $t + \hat{d}_i$
4: **on job arrival/completion:**
5: schedule the active job, among the tasks τ_i such that $\chi_i \geq$ `current_level()`, having earliest virtual absolute deadline (ties broken arbitrarily);
6: **on** `current_level()` $> k$:
7: schedule the active job, among the tasks τ_i such that $\chi_i > k$, having earliest *absolute deadline* (ties broken arbitrarily); then break from the loop
8: **end loop**

9: **loop**
10: **on job arrival:**
11: if a job of task τ_i arrives at time t, assign it an *absolute deadline* equal to $t + d_i$
12: **on job arrival/completion:**
13: schedule the active job, among the tasks τ_i such that $\chi_i \geq$ `current_level()`, having earliest absolute deadline (ties broken arbitrarily)
14: **end loop**

3.2. Schedulability Conditions

THEOREM 3.4. *Given an implicit-deadline task system τ, if either*

$$\sum_{l=1}^{K} U_l(l) \leq 1 \tag{2}$$

or, for some k $(1 \leq k < K)$, the following condition holds:

$$1 - \sum_{l=1}^{k} U_l(l) > 0 \quad \text{and} \quad \frac{\sum_{l=k+1}^{K} U_l(k)}{1 - \sum_{l=1}^{k} U_l(l)} \leq \frac{1 - \sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)}, \tag{3}$$

then τ can be correctly scheduled by EDF-VD.

PROOF. If condition (2) holds, then all jobs can be scheduled for their worst-case execution times at their own criticality level. Hence, EDF-VD(K), that is, EDF without deadline scaling, will yield a correct schedule. Therefore, from here on, the proof focuses on the case that (2) does not hold and (3) holds for some $k < K$.

The proof consists of two steps. In the first step, we show that if there is an $x \leq 1$ such that the following two inequalities hold:

$$\sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} \frac{U_l(k)}{x} \leq 1 \tag{4}$$

$$x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(l) \leq 1, \tag{5}$$

then EDF-VD(k) is a correct scheduling policy for τ.

In the second step, we show that if (3) holds for some $k < K$, then there exists $x \leq 1$ such that (4) and (5) hold. We discuss later in this section how the scaling parameter x is determined . For the first step , we determine an upper bound on the "virtual utilization"; the utilization of the system as long as the virtual deadlines are applied, if EDF-VD(k) is

the scheduling algorithm we apply. We will slightly abuse notation and write $\hat{p}_i$ for the "virtual period" that equals the virtual deadline $\hat{d}_i$ for all tasks τ_i, in order to define the "virtual utilization." Assume for all tasks τ_i with χ_i at most k, that all jobs are executed at their own-criticality execution requirement. For all tasks τ_i with $\chi_i > k$, the jobs are executed at criticality level k. This clearly gives an upper bound on the utilization for as long as the system is in criticality levels $1, 2, \ldots, k$. Also note that by this assumption, no complications can occur at criticality changes before level $k+1$ is reached. The "virtual utilization" of this task system in levels $1, 2, \ldots, k$ is at most

$$\begin{aligned}\hat{U}(k) &:= \sum_{i\in[n]} \frac{c_i(k)}{\hat{p}_i} = \sum_{i\in[n]} \frac{c_i(k)}{\hat{d}_i} \\ &= \sum_{i\in[n]:\,\chi_i\le k} \frac{c_i(\chi_i)}{d_i} + \sum_{i\in[n]:\,\chi_i>k} \frac{c_i(k)}{x d_i} \\ &= \sum_{i\in[n]:\,\chi_i\le k} \frac{c_i(\chi_i)}{p_i} + \sum_{i\in[n]:\,\chi_i>k} \frac{c_i(k)}{x p_i} \\ &= \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} \frac{U_l(k)}{x}.\end{aligned}$$

Note that a sufficient condition for correctness in levels $1, 2, \ldots, k$ is that $\hat{U}(k) \le 1$. Hence, we find that (4) ensures correctness in levels $1, 2, \ldots, k$.

For any level $l > k$, assume by contradiction that a deadline miss occurs in some scenario. We bound the execution requirement of all tasks until the time of the first deadline miss, which is denoted by t_f. Let I denote a minimal collection of jobs released by τ on which a deadline is missed (by *minimal*, we mean that EDF-VD(k) would meet all deadlines if scheduling any proper subset of I). We assume that (4) and (5) hold and, without loss of generality, that the first job arrival is at time zero. Hence, the job that misses a deadline is of level $l > k$, because at time t_f the system is at least in level $k+1$. Let t^* denote the time where behavior of level $k+1$ is first exhibited.

CLAIM 3.5. *All jobs receiving execution in $[t^*, t_f)$ have deadlines at most t_f.*

PROOF. Suppose there is a job that has deadline larger than t_f and receives some execution between t^* and t_f, say in the interval $[t_1, t_2)$. This means that during $[t_1, t_2)$ there are no jobs pending with deadline at most t_f. Then, the set of jobs obtained by considering only jobs with arrival time at least t_2 will also miss a deadline at t_f, which contradicts the assumed minimality of I. □

We define the quantity $\eta_i(t)$ as the cumulative execution requirement of jobs of task τ_i until time t and derive upper bounds for it, for all tasks. Among all jobs executing in $[t^*, t_f)$, let J_0 be the job with the earliest arrival time. Denote by a_0 its arrival time and by d_0 its absolute deadline.

CLAIM 3.6. *For any task τ_i having $\chi_i \le k$, it holds that*

$$\eta_i(t_f) \le (a_0 + x(t_f - a_0))u_i(\chi_i).$$

PROOF. Note that no job of τ_i will receive execution after t^*. If such a job executes after a_0, it must have a deadline no larger than the virtual deadline of J_0, which is $a_0 + x(d_0 - a_0)$. Since $t_f \ge d_0$ by Claim 3.5, this means that no job of task τ_i with deadline greater than $a_0 + x(t_f - a_0)$ will execute after a_0.

Suppose now that a job with $\chi_i \le k$ and deadline larger than $a_0 + x(t_f - a_0)$ was executed for some time before a_0. Let t_l denote the latest instant at which any such job

executes. This means that at this instant, there were no jobs with absolute deadline at most $a_0 + x(t_f - a_0)$ awaiting execution. Hence, the set of jobs obtained by considering only those jobs in I that have arrival time at least t_l also misses a deadline. This contradicts the assumed minimality of I.

This implies that there are at most $(a_0 + x(t_f - a_0))/p_i$ jobs of τ_i until time t_f; each of them requires an execution time of at most $c_i(\chi_i)$. Therefore, the total execution requirement of τ_i is bounded by $(a_0 + x(t_f - a_0))u_i(\chi_i)$. □

CLAIM 3.7. *Any task τ_i with $\chi_i > k$ has*

$$\eta_i(t_f) \leq \frac{a_0}{x} u_i(k) + (t_f - a_0)u_i(\chi_i).$$

PROOF. We distinguish two cases.

Case 1. *Task τ_i Does not Release a Job at or after a_0.* Each job of τ_i has a virtual deadline of at most $a_0 + x(t_f - a_0)$. Otherwise, consider a job with a larger virtual deadline and let t_l denote the latest time instant at which this job executes. The job sequence consisting of only those jobs with an arrival time larger than t_l also misses a deadline and this is in contradiction with the assumed minimality of I. Hence, each job of τ_i has an actual deadline of at most $a_0/x + t_f - a_0$ and there are at most $\frac{a_0/x + t_f - a_0}{p_i}$ of them. Since these jobs do not execute in $[t^*, t_f)$ (else, J_0 would not be the one with earliest release), the execution requirement per job is at most $c_i(k)$. Combining these observations, we bound the execution requirement of jobs from task τ_i by

$$\begin{aligned}\left(\frac{a_0}{x} + t_f - a_0\right)\frac{c_i(k)}{p_i} &= \frac{a_0}{x} u_i(k) + (t_f - a_0)u_i(k) \\ &\leq \frac{a_0}{x} u_i(k) + (t_f - a_0)u_i(\chi_i).\end{aligned}$$

Case 2. *Task τ_i Releases One or More Jobs at or after a_0.* Let a_i denote the first release of a job from τ_i greater than or equal to a_0. The previously released job of τ_i did not execute in $[t^*, t_f)$, by definition of a_i and a_0. Therefore, it is safe to assume that until a_i the execution requirement per job from τ_i was bounded by $c_i(k)$ and after that it is bounded by $c_i(\chi_i)$. Hence, the cumulative requirement of all jobs from τ_i is bounded by

$$\begin{aligned}a_i u_i(k) + (t_f - a_i)u_i(\chi_i) &\leq a_0 u_i(k) + (t_f - a_0)u_i(\chi_i) \\ &\leq \frac{a_0}{x} u_i(k) + (t_f - a_0)u_i(\chi_i),\end{aligned}$$

where the first inequality comes from the facts that $a_0 \leq a_i$ and $u_i(k) \leq u_i(\chi_i)$ and the second one from $x \leq 1$. □

Summing the cumulative requirements over all tasks gives

$$\begin{aligned}&\sum_{i:\chi_i \leq k} \eta_i(t_f) + \sum_{i:\chi_i > k} \eta_i(t_f) \\ &\quad\leq \sum_{i:\chi_i \leq k} (a_0 + x(t_f - a_0))u_i(\chi_i) + \sum_{i:\chi_i > k} \left(\frac{a_0}{x} u_i(k) + (t_f - a_0)u_i(\chi_i)\right) \\ &\quad= a_0 \left(\sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} \frac{U_l(k)}{x}\right) + (t_f - a_0)\left(x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(l)\right) \\ &\quad\leq a_0 + (t_f - a_0)\left(x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(l)\right),\end{aligned}$$

where the last inequality comes from the assumption that (4) holds. The assumed deadline miss implies

$$\begin{aligned} & a_0 + (t_f - a_0)\left(x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(l)\right) > t_f \\ \Leftrightarrow \quad & (t_f - a_0)\left(x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(l)\right) > t_f - a_0 \\ \Leftrightarrow \quad & x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(l) > 1. \end{aligned}$$

But this directly contradicts (5).

Now that we have shown that (4) and (5) are sufficient for the correctness of EDF-VD(k), it remains to show that if (3) holds, then (4) and (5) also hold. Rewriting (4) gives

$$\begin{aligned} & \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} \frac{U_l(k)}{x} \leq 1 \\ \Leftrightarrow \quad & x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(k) \leq x \\ \Leftrightarrow \quad & \sum_{l=k+1}^{K} U_l(k) \leq x \left(1 - \sum_{l=1}^{k} U_l(l)\right) \\ \Leftrightarrow \quad & \frac{\sum_{l=k+1}^{K} U_l(k)}{1 - \sum_{l=1}^{k} U_l(l)} \leq x, \end{aligned} \tag{6}$$

where in the last step we used that $1 - \sum_{l=1}^{k} U_l(l) > 0$. Rewriting (5) gives

$$\begin{aligned} & x \sum_{l=1}^{k} U_l(l) + \sum_{l=k+1}^{K} U_l(l) \leq 1 \\ \Leftrightarrow \quad & x \sum_{l=1}^{k} U_l(l) \leq 1 - \sum_{l=k+1}^{K} U_l(l) \\ \Leftrightarrow \quad & x \leq \frac{1 - \sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)}. \end{aligned} \tag{7}$$

Hence, if (3) holds, there must exist an x that satisfies both (6) and (7). This concludes the proof of Theorem 3.4. Note that any x satisfying (6) and (7) suffices as a scaling parameter. □

3.3. Speedup Bounds

Between the sufficient condition for schedulability given by (3) and the necessary condition (1), there is a gap. To analyze this gap, we consider a task system satisfying the necessary condition given in (1) and determine how much faster the processor should be to correctly schedule the task set by EDF-VD.

The *speedup factor* of a scheduling algorithm A is the smallest real number f such that any task system τ that is feasible on a unit-speed processor (in the sense of Definition 2.1) is correctly scheduled by A on a speed-f processor. The speedup factor is a convenient metric for comparing the worst-case behavior of different algorithms for solving the same problem: the smaller the speedup factor, the closer the behavior of the algorithm to that of a clairvoyant exact algorithm.

We show how to derive tight speedup bounds for EDF-VD for any number K of criticality levels. When K equals 2 or 3, we analytically derive closed-form expressions for the speedup bounds. When $K > 3$, the computation of the speedup bound is an analytically challenging problem; we solve it numerically for all $K \leq 13$.

3.3.1. Two Levels

THEOREM 3.8. *If a 2-level task system τ satisfies*

$$\max\{U_1(1) + U_2(1), U_2(2)\} \leq 1,$$

then EDF-VD *correctly schedules τ on a processor of speed 4/3. In particular, if τ is schedulable on a unit-speed processor, then it is* EDF-VD*-schedulable on a processor of speed 4/3.*

PROOF. For a 2-level system, the schedulability condition (3) is

$$\frac{U_2(1)}{1 - U_1(1)} \leq \frac{1 - U_2(2)}{U_1(1)}, \tag{8}$$

and $1 - U_1(1) > 0$. To prove the claim, we find the largest q such that, if

$$\begin{aligned} U_1(1) + U_2(1) &\leq q \\ U_2(2) &\leq q, \end{aligned} \tag{9}$$

then the sufficient condition (8) still holds. The required speedup then equals $1/q$. Note that since $U_2(2)$ only appears with a negative sign on the right-hand side of (8) and $U_2(1)$ only appears on the left-hand side, the worst case in terms of tightness of the condition is that $U_2(2) = q$ and $U_2(1) = q - U_1(1)$. Henceforth, this assumption is made. Further, we define $y \stackrel{\text{def}}{=} 1 - q$. Then, sufficient condition (8) becomes

$$\frac{1 - y - U_1(1)}{1 - U_1(1)} \leq \frac{y}{U_1(1)}. \tag{10}$$

This condition is satisfied if and only if $U_1(1)^2 - U_1(1) + y \geq 0$ (here we used $1 - U_1(1) > 0$). This inequality holds for all $U_1(1) \in [0, 1)$ if and only if $y \geq 1/4$. Hence, the largest q such that (9) implies (8) is $q^* = 3/4$, and the required speedup is $1/q^* = 4/3$.

The second part of the statement follows from the first and Proposition 3.1. □

3.3.2. Three Levels

THEOREM 3.9. *If a 3-level task system τ satisfies*

$$\max_{k=1,2,3} \sum_{l=k}^{K} U_l(k) \leq 1, \tag{11}$$

then EDF-VD *correctly schedules τ on a processor of speed 2. In particular, if τ is feasible on a unit-speed processor, then it is* EDF-VD*-schedulable on a processor of speed 2.*

PROOF. We recall the schedulability conditions for a 3-level system from (3) for $i = 1, 2$:

$$\frac{U_2(1) + U_3(1)}{1 - U_1(1)} \leq \frac{1 - U_2(2) - U_3(3)}{U_1(1)} \tag{12}$$

$$\frac{U_3(2)}{1 - U_1(1) - U_2(2)} \leq \frac{1 - U_3(3)}{U_1(1) + U_2(2)}, \tag{13}$$

where for (12) we also assume that $1 - U_1(1) > 0$ and for (13) that $1 - U_1(1) - U_2(2) > 0$. We find the speedup in a similar way as in Section 3.3.1. That is, we search for the largest q, such that if

$$\begin{aligned} U_1(1) + U_2(1) + U_3(1) &\leq q \\ U_2(2) + U_3(2) &\leq q \\ U_3(3) &\leq q \end{aligned}$$

hold, then at least one of the inequalities in (12) and (13) holds. Note that the worst-case speedup is obtained when the previous inequalities hold with equality. To see this, we reason as follows.

The term $U_3(3)$ only appears in the right-hand side of (12) and (13) and the expressions are monotonically decreasing in $U_3(3)$. Hence, the worst case (lowest right-hand side value) is when $U_3(3)$ is largest, that is, $U_3(3) = q$.

Since $U_3(2)$ appears only in the left-hand side of (13) and this side is monotonically increasing in $U_3(2)$, the worst-case is if $U_3(2) = q - U_2(2)$.

Since $U_2(1)$ and $U_3(1)$ appear only in the left-hand side of (12) and this side is monotonically increasing in $U_2(1) + U_3(1)$, the worst-case is if $U_2(1) + U_3(1) = q - U_1(1)$.

Now we substitute these expressions in the inequalities (12) and (13) to obtain the following two conditions

$$\begin{aligned} \frac{q - U_1(1)}{1 - U_1(1)} &\leq \frac{1 - q - U_2(2)}{U_1(1)} \\ \frac{q - U_2(2)}{1 - U_1(1) - U_2(2)} &\leq \frac{1 - q}{U_1(1) + U_2(2)}. \end{aligned}$$

In these two equations, we substitute $y \stackrel{\text{def}}{=} 1 - q$, and after rewriting we find

$$\begin{aligned} (1 - U_1(1))(U_1(1) + U_2(2)) &\leq y \\ (1 - U_2(2))(U_1(1) + U_2(2)) &\leq y. \end{aligned}$$

It is easily verified that the minimum of these two left-hand sides attains its maximum value when $U_1(1) = U_2(2)$. After substituting, we obtain

$$2U_1(1)^2 - 2U_1(1) + y \geq 0.$$

This inequality holds for all $U_1(1) \in [0, 1)$ if and only if $y \geq 1/2$, which gives $q^* = 1/2$; the resulting speedup bound is therefore 2.

Again, the second part of the claim follows from the first and Proposition 3.1. □

3.3.3. Arbitrary Number of Levels. The problem of determining the minimum speedup factor f_K such that any K-level task system that is feasible on an unit-speed processor is correctly scheduled by EDF-VD on a f_K-speed processor can be formulated as follows: Find the *largest* q ($q \leq 1$) such that the following implication holds for all $U_l(k)$, $k = 1, 2, \ldots, K$, $l = k, k + 1, \ldots, K$:

$$\sum_{l=k}^{K} U_l(k) \le q \qquad \forall k = 1, 2, \ldots, K \quad \Rightarrow$$

$$\text{either} \quad \sum_{l=1}^{K} U_l(l) \le 1 \quad \text{or} \quad \exists k \in \{1, 2, \ldots, K-1\} \quad \text{s.t.} \begin{cases} 1 - \sum_{l=1}^{k} U_l(l) > 0 \quad \text{and} \\ \frac{\sum_{l=k+1}^{K} U_l(k)}{1 - \sum_{l=1}^{k} U_l(l)} \le \frac{1 - \sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)}. \end{cases}$$

If the largest such value of q is q^*, the speedup factor is then $f_K = 1/q^*$. Equivalently, we want to find the *smallest* q such that this implication does not hold, that is, the premise is true but the conclusion is false; in other words, the largest value of the speedup for which one can still construct a counterexample. This leads to the following formulation:

$$\begin{aligned} & \min q && \text{(NLP)} \\ \text{s.t.} \quad & \sum_{l=k}^{K} U_l(k) \le q && \forall k = 1, 2, \ldots, K; \\ & \sum_{l=1}^{K} U_l(l) > 1 \\ \neg \left(1 - \sum_{l=1}^{k} U_l(l) > 0 \quad \wedge \quad \frac{\sum_{l=k+1}^{K} U_l(k)}{1 - \sum_{l=1}^{k} U_l(l)} \le \frac{1 - \sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)} \right) & \quad \forall k = 1, 2, \ldots, K-1. \end{aligned}$$

Formulation (NLP) involves disjunctions (the negated conjunctions in the last set of constraints), which are typically disallowed by numerical solvers. We prove that solving (NLP) is equivalent to finding $q^* \stackrel{\text{def}}{=} \min_{j=1,2,\ldots,K-1} q_j^*$, where each q_j^* is the solution to the following nonlinear program:

$$\begin{aligned} & q_j^* = \min q_j && (\text{NLP}_j) \\ \text{s.t.} \quad & \sum_{l=k}^{K} U_l(k) \le q_j && \forall k = 1, 2, \ldots, K; \\ & \sum_{l=1}^{K} U_l(l) > 1 \\ & \frac{\sum_{l=k+1}^{K} U_l(k)}{1 - \sum_{l=1}^{k} U_l(l)} > \frac{1 - \sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)} && \forall k = 1, 2, \ldots, j; \\ & 1 - \sum_{l=1}^{j} U_l(l) > 0 \\ & 1 - \sum_{l=1}^{j+1} U_l(l) \le 0. \end{aligned}$$

We denote the constraints of these nonlinear programs as follows:

$$C \stackrel{\text{def}}{=} \left\{ \sum_{l=k}^{K} U_l(k) \le q \;\; \forall k = 1, 2, \ldots, K \;\wedge\; \sum_{l=1}^{K} U_l(l) > 1 \right\},$$

$$A_k \stackrel{\text{def}}{=} \left\{ \frac{\sum_{l=k+1}^{K} U_l(k)}{1-\sum_{l=1}^{k} U_l(l)} \leq \frac{1-\sum_{l=k+1}^{K} U_l(l)}{\sum_{l=1}^{k} U_l(l)} \right\},$$

$$B_k \stackrel{\text{def}}{=} \left\{ 1 - \sum_{l=1}^{k} U_l(l) > 0 \right\},$$

for each $k = 1, 2, \ldots, K-1$. Then, the constraints of program (NLP) can be written as

$$C \wedge \bigwedge_{k=1}^{K-1} \neg (A_k \wedge B_k),$$

while those of program (NLP$_j$) can be written as

$$C \wedge \left(\bigwedge_{k=1}^{j} \neg A_k \right) \wedge B_j \wedge \neg B_{j+1}.$$

The equivalence is shown in the next lemma.

LEMMA 3.10. *For any K-level feasible task system, the set of values $U_l(k)$, $k = 1, 2, \ldots, K$, $l = k, k+1, \ldots, K$ satisfies*

$$C \wedge \bigwedge_{k=1}^{K-1} \neg (A_k \wedge B_k) \tag{14}$$

if and only if it satisfies

$$\bigvee_{j=1}^{K-1} \left(C \wedge \left(\bigwedge_{k=1}^{j} \neg A_k \right) \wedge B_j \wedge \neg B_{j+1} \right). \tag{15}$$

PROOF. First, observe that condition B_j implies condition $B_{j'}$ for any $j' < j$ and that condition C implies condition $\neg B_K$. Moreover, condition B_1 holds since the task system is assumed feasible. Therefore, for $K = 2$, conditions (14) and (15) are both equivalent to $C \wedge \neg A_1$.

By induction, let us assume that conditions (14) and (15) are equivalent for task systems of $K-1$ levels. We rewrite conditions (14) and (15) for K-level systems as follows, respectively:

$$\neg (A_{K-1} \wedge B_{K-1}) \wedge C \wedge \bigwedge_{k=1}^{K-2} \neg (A_k \wedge B_k) \tag{14'}$$

and

$$\left(C \wedge \left(\bigwedge_{k=1}^{K-1} \neg A_k \right) \wedge B_{K-1} \wedge \neg B_K \right) \vee \bigvee_{j=1}^{K-2} \left(C \wedge \left(\bigwedge_{k=1}^{j} \neg A_k \right) \wedge B_j \wedge \neg B_{j+1} \right)$$
$$= \left(C \wedge \left(\bigwedge_{k=1}^{K-1} \neg A_k \right) \wedge B_{K-1} \right) \vee \bigvee_{j=1}^{K-2} \left(C \wedge \left(\bigwedge_{k=1}^{j} \neg A_k \right) \wedge B_j \wedge \neg B_{j+1} \right). \tag{15'}$$

We distinguish the cases that B_{K-1} does and does not hold. If B_{K-1} holds, then B_j holds for each $j = 1, 2, \ldots, K-1$. It follows that (14') and (15') are equivalent to

$$C \wedge \left(\bigwedge_{k=1}^{K-1} \neg A_k \right).$$

Table II. Minimum Speedup Factor for $K \leq 13$ Levels

Number of levels K	Speedup factor f_K
2	1.3333
3	2.0000
4	2.6180
5	3.0811
6	3.7321
7	4.2361
8	4.7913
9	5.3723
10	5.8551
11	6.4641
12	6.9487
13	7.5311

If B_{K-1} does not hold, the two conditions are equivalent to

$$C \wedge \bigwedge_{k=1}^{K-2} \neg (A_k \wedge B_k)$$

and

$$\bigvee_{j=1}^{K-2} \left(C \wedge \left(\bigwedge_{k=1}^{j} \neg A_k \right) \wedge B_j \wedge \neg B_{j+1} \right),$$

respectively, and the lemma holds by the inductive hypothesis. □

We solve program (NLP$_j$), for each $j = 1, 2, \ldots, K-1$, to find the minimum speedup factor $f_K = 1/q^*$, where $q^* = \min_{j=1,2,\ldots,K-1} q_j^*$. After clearing the denominators in program (NLP$_j$), we obtain a system of multivariate polynomial inequalities in the variables $U_l(k)$ and q_j. As such, it can be solved by a (numerical) global nonlinear continuous optimization solver. In this case, we used COUENNE [Belotti et al. 2009]. COUENNE was able to find the optimum for any $K \leq 13$. The resulting speedup factors are reported in Table II.

THEOREM 3.11. *Let τ be a K-level task system with $2 \leq K \leq 13$. If τ is feasible on a unit-speed processor, then it is* EDF-VD*-schedulable on a processor of speed f_K, where f_K ($\pm 10^{-4}$) is as in Table II.*

3.4. Optimality of EDF-VD for Two Levels

We now show that – at least in the case of 2-level implicit-deadline systems – EDF-VD is an optimal algorithm in terms of the speedup factor metric.

THEOREM 3.12. *No non-clairvoyant algorithm for scheduling 2-level implicit-deadline sporadic task systems can have a speedup bound better than* 4/3.

PROOF. Consider the example task system $\tau = (\tau_1, \tau_2)$, with the following parameters, where ϵ is an arbitrary small positive number.

τ_i	χ_i	$c_i(1)$	$c_i(2)$	p_i
τ_1	1	$1+\epsilon$	$1+\epsilon$	2
τ_2	2	$1+\epsilon$	3	4

This system is feasible according to Definition 2.1: EDF would meet all deadlines in scenarios of level 1 (since $U_1(1) + U_2(1) \leq 1$), while only jobs of τ_2 would get to execute in scenarios of level 2 (and $U_2(2) \leq 1$).

To see that τ cannot be scheduled correctly on a unit-speed processor by any online scheduler, suppose both tasks were to generate jobs simultaneously. It need not be revealed prior to one of the jobs receiving $(1 + \epsilon)$ units of execution, whether the level of the scenario is 1 or 2. We consider two cases.

(1) τ_1's job receives $(1 + \epsilon)$ units of execution before τ_2's job does. In this case, the scenario is revealed to be of level 2. But now there is not enough time remaining for τ_2's job to complete by its deadline at time instant 4.
(2) τ_2's job receives $(1 + \epsilon)$ units of execution before τ_1's job does. In this case, the scenario is revealed to be of level 1, in that τ_2's job signals that it has completed execution. But then, there is not enough time remaining for τ_1's job to complete by its deadline at time 2.

We have thus shown that no non-clairvoyant algorithm can correctly schedule τ. The theorem follows, based on the observation that $\max(U_1(1) + U_2(1), U_2(2))$ exceeds 3/4 by an arbitrarily small amount. □

The optimality of EDF-VD for 2-level implicit-deadline task systems follows by combining the analysis in this example with Theorem 3.8. A speedup bound of 4/3 holds for EDF-VD and, by the previous argument, a lower speedup is not possible for *any* non-clairvoyant algorithm.

4. ARBITRARY-DEADLINE TASKS

We now study the case in which deadlines do not necessarily equal periods of the corresponding tasks (*arbitrary deadlines*). In this section, we only consider task systems that consist of two criticality levels.

4.1. Overview of EDF-VD for Arbitrary-Deadline Tasks

In the case that deadlines are independent of the periods, the notion of utilization is insufficient to characterize schedulability. In conventional real-time scheduling theory, the notion of *load* takes this role. The load of a collection of jobs denotes the maximum, over all time intervals, of the cumulative execution requirement by jobs of the scenario due in the interval, divided by the interval length. Informally speaking, the load of a scenario represents the minimum speed of any processor that can meet all deadlines in that scenario.

Recall that a sporadic task system τ can generate infinitely many different scenarios. Each scenario I can be seen as a collection of independent jobs, where the arrival time of job J_{ij} of task τ_i is a_{ij} and its absolute deadline is $d_{ij} = a_{ij} + d_i$. Then, for a scenario I of a mixed-criticality system, we can define three notions of load, analogous to the aforementioned concept.

Definition 4.1. The *load* $\lambda(I)$, the *level-1 load* $\lambda_1(I)$ and the *level-2 load* $\lambda_2(I)$ of a mixed-criticality scenario I with two criticality levels are defined according to the following three formulas:

$$\lambda(I) \stackrel{\text{def}}{=} \max_{0 \leq t_1 < t_2} \frac{\sum_{J_{ij} \in I : t_1 \leq a_{ij} \wedge d_{ij} \leq t_2} c_i(\chi_i)}{t_2 - t_1}$$

$$\lambda_1(I) \stackrel{\text{def}}{=} \max_{0 \le t_1 < t_2} \frac{\sum_{J_{ij} \in I : t_1 \le a_{ij} \wedge d_{ij} \le t_2} c_i(1)}{t_2 - t_1}$$

$$\lambda_2(I) \stackrel{\text{def}}{=} \max_{0 \le t_1 < t_2} \frac{\sum_{J_{ij} \in I : \chi_i = 2 \wedge t_1 \le a_{ij} \wedge d_{ij} \le t_2} c_i(2)}{t_2 - t_1}.$$

Informally, $\lambda(I)$ bounds the load of the scenario in the absence of any prior information; $\lambda_1(I)$ bounds the load when the level of the scenario is a priori known to be 1 (i.e., no job exceeds its level 1 WCET); and $\lambda_2(I)$ bounds the load when the level of the scenario is a priori known to be 2. In particular, $\lambda_2(I)$ disregards the tasks with $\chi_i = 1$.

The *load* $\lambda(\tau)$ *of a task system* τ is defined to be the largest value that $\lambda(I)$ can have, over any scenario I generated by τ. We define $\lambda_1(\tau)$ and $\lambda_2(\tau)$ similarly. For any task system τ, the values $\lambda(\tau)$, $\lambda_1(\tau)$ and $\lambda_2(\tau)$ can be computed by determining the loads of "regular" (i.e., non-MC) sporadic task systems. Namely, $\lambda(\tau)$, $\lambda_1(\tau)$ and $\lambda_2(\tau)$ are, respectively, the loads of regular sporadic task systems $\{(c_i(\chi_i), d_i, p_i) | \tau_i \in \tau\}$, $\{(c_i(1), d_i, p_i) | \tau_i \in \tau\}$ and $\{(c_i(2), d_i, p_i) | \tau_i \in \tau \wedge \chi_i = 2\}$.

The values $\lambda(\tau)$, $\lambda_1(\tau)$, $\lambda_2(\tau)$ can either be computed exactly, or they can be efficiently approximated with arbitrarily small error, using well-known techniques (see, e.g., Baruah et al. [1993] and Albers and Slomka [2004]). In the sequel, it will be convenient to abbreviate $\lambda(\tau)$, $\lambda_1(\tau)$ and $\lambda_2(\tau)$ to λ, λ_1 and λ_2, respectively.

The *synchronous arrival sequence* of τ is the job sequence in which each task τ_i generates its jth job at time $a_{ij} = (j-1)p_i$. It is well known [Baruah et al. 1993] that, on a preemptive uniprocessor, the largest demand of a non-MC sporadic task set over any time interval is achieved by the synchronous arrival sequence; this is one of the key facts used when computing the load.

The following immediate observation gives a necessary condition for any scheduling algorithm to be correct for a 2-level task system.

PROPOSITION 4.2. *Let τ be a 2-level task system. If $\lambda_1 > 1$ or $\lambda_2 > 1$, then τ is not feasible.*

A relationship between the three loads is given in the next observation, which follows immediately from the definitions.

PROPOSITION 4.3. *Let τ be a 2-level task system. Then $\lambda \le \lambda_1 + \lambda_2$.*

We consider the EDF-VD algorithm for scheduling a sporadic MC system with arbitrary deadlines. As in the case of implicit-deadline task systems, EDF-VD consists of two phases: an offline preprocessing phase and a runtime scheduling phase. The runtime scheduling is exactly as in Algorithm 2 (with $K = 2$). The offline preprocessing phase test is, however, somewhat modified as it is based on a different schedulability test.

We give here an overview of the preprocessing phase, which is described exactly in Algorithm 3, whereas details and justifications are given in Section 4.2. We start by computing λ, λ_1, λ_2 at line 1 – either exactly, as in Baruah et al. [1993], or more efficiently but approximately using, for example, the approach in Albers and Slomka [2004]. To facilitate the exposition, we work with the exact values of λ, λ_1 and λ_2, keeping in mind that we may have only approximate values for them – we revisit this issue in Section 4.4.

Since there are only two criticality levels, there are only two different cases. If $\lambda \le 1$, then the deadlines are not scaled and the runtime algorithm will coincide with EDF (lines 2–6). Otherwise, a certain condition involving λ_1 and λ_2 is checked at line 8; if the condition does not hold, the schedulability test reports a failure (line 9). If the condition holds, modified deadlines are computed based on an appropriately chosen

ALGORITHM 3: EDF with Virtual Deadlines (EDF-VD) – Offline preprocessing phase (for arbitrary-deadline task systems)

Input: task system $\tau = (\tau_1, \ldots, \tau_n)$ to be scheduled on a unit-speed preemptive processor

```
1: Compute λ, λ1, λ2
2: if λ ≤ 1 then
3:    k ← 2
4:    for i = 1, 2, ..., n do
5:       d̂_i ← d_i
6:    end for
7: else
8:    if λ1 + λ2/2 > 1 or λ1 + λ2 − λ1λ2/4 > 1 then
9:       return unschedulable
10:   else
11:      k ← 1
12:      x ← 1 − λ2/2
13:      for i = 1, 2, ..., n do
14:         if χ_i ≤ k then
15:            d̂_i ← d_i
16:         else
17:            d̂_i ← x d_i
18:         end if
19:      end for
20:   end if
21: end if
22: return (schedulable, k, (d̂_i)_{i=1}^n)
```

scaling parameter x (lines 10–20) and used as before for assigning virtual deadlines to jobs, as long as the scenario exhibits level 1. As before, once the scenario exhibits level 2, the original deadlines are restored.

4.2. Schedulability Conditions

We now prove the correctness of Algorithm 3. At a high level, the proof is very much in line with the proof of Theorem 3.4, but details are more complicated and the results are therefore presented as three separate lemmata.

LEMMA 4.4. *If a 2-level task system τ satisfies $\lambda \leq 1$, then τ is schedulable by* EDF-VD.

PROOF. Notice that it is always possible to regard τ as an ordinary, non-MC task system where the WCET of task τ_i is $c_i(\chi_i)$ (this is the "worst-case reservations" approach). The load λ then coincides with the standard notion of load for ordinary task systems. Since (for ordinary task systems) EDF is an optimal scheduling strategy on a preemptive uniprocessor, the task system is EDF-schedulable if $\lambda \leq 1$. □

LEMMA 4.5. *If a 2-level task system τ satisfies $\lambda_1 \leq 1$, then a scaling parameter x such that $\lambda_1 \leq x \leq 1$ guarantees that no deadline is missed in a level-1 scenario.*

PROOF. To prove that there is no deadline miss, we assume the opposite, which is that a job J' misses its deadline in a level-1 scenario. Let I denote a minimal collection of jobs released by τ on which a deadline is missed. The deadline of job J' is denoted by d'. Now, due to the minimality of I, criticality-1 jobs have deadlines no greater than d', and criticality-2 jobs have virtual deadlines no greater than d' (otherwise, the removal of such a job would yield a smaller collection I on which a deadline is

missed). Notice that all virtual deadlines of criticality-2 jobs can be represented as $\hat{d}_{ij} = a_{ij} + xd_i \geq x(a_{ij} + d_i) = xd_{ij}$. This together with $\hat{d}_{ij} \leq d'$ implies that all criticality-2 jobs have actual absolute deadline no later than d'/x. Therefore, the cumulative execution requirement of the system until time d' is at most $\lambda_1 \cdot d'/x$ in a level-1 scenario. Because J' misses its deadline d', it must hold that

$$\begin{aligned} & \frac{d'}{x}\lambda_1 > d' \\ \Leftrightarrow \qquad & \lambda_1 > x. \end{aligned} \tag{16}$$

Since we assumed that job J' misses its deadline in a level-1 scenario, the contrapositive of (16),

$$x \geq \lambda_1, \tag{17}$$

yields a sufficient condition for schedulability of level-1 scenarios. □

LEMMA 4.6. *If a 2-level task system τ satisfies $\lambda_1 + \lambda_2/2 \leq 1$ and $\lambda_1 + \lambda_2 - \lambda_1\lambda_2/4 \leq 1$, then setting $x = 1 - \lambda_2/2$ as the scaling parameter in* EDF-VD *guarantees that no deadline is missed in a level-2 scenario.*

PROOF. To prove that there is no deadline miss, we argue by contradiction and assume that a job J misses its deadline at time t_f. Let I denote a minimal collection of jobs for which a deadline miss occurs. Therefore, by Lemma 4.5, job J must be of criticality level 2, since the condition $\lambda_1 + \lambda_2/2 \leq 1$ (checked in Algorithm 3) ensures that $x = 1 - \lambda_2/2 \geq \lambda_1$, so no deadline miss can occur while the scenario exhibits level-1 behavior. Denote by time t^* the time instant at which the scenario first exhibits level-2 behavior. The following claim is equivalent to Claim 3.5 and so is its proof.

CLAIM 4.7. *All jobs receiving execution in $[t^*, t_f)$ have deadlines at most t_f.*

We see from Claim 4.7 that all jobs that receive execution after t^* have priority higher than or equal to the priority of J after t^*. Let S^* be the set of jobs which receive execution after t^*. Then $J \in S^*$ and all jobs in S^* are of criticality level 2.

Let J_0 be the job in S^* that is released earliest. Let a_0 and d_0 be its arrival time and absolute deadline, respectively. Thus, $\hat{d}_0 = a_0 + x(d_0 - a_0)$ is the virtual deadline of J_0.

CLAIM 4.8. *Any job J_{ij} that is in the minimal job sequence I and only receives execution before t^* must have an absolute deadline $d_{ij} \leq a_0 + x(t_f - a_0)$ if it is a level-1 job, and absolute virtual deadline $\hat{d}_{ij} \leq a_0 + x(t_f - a_0)$ if it is a level-2 job.*

PROOF. Note that no criticality-1 job J_{ij} with $d_{ij} > \hat{d}_0$ will execute at or after a_0; since J_{ij} will only receive execution time before t^*, this would imply that J_0 should be finished before t^* (contradicting the definition of J_0).

Suppose there is a criticality-1 job J_{ij} with $d_{ij} > \hat{d}_0$ that executes somewhere before a_0. Denote by t_l the latest moment in time where it executes. At this point there are no available jobs with deadline at most $\hat{d}_0$ and the job sequence obtained by only considering jobs released after t_l also misses a deadline. That contradicts the assumed minimality of I.

So, any criticality-1 job J_{ij} in I must have a deadline $d_{ij} \leq \hat{d}_0 = a_0 + x(d_0 - a_0) \leq a_0 + x(t_f - a_0)$, using Claim 4.7 saying that $d_0 \leq t_f$.

If a level-2 job $J_{ij} \in I$ does not receive execution after t^*, we have the following possible cases.

(1) $a_{ij} \geq a_0$. Then $\hat{d}_{ij} \leq \hat{d}_0$, since otherwise J_{ij} would not receive any execution after a_0, implying that it does not receive any execution before t^*, which combined with

J_{ij} not receiving any execution after t^*, would make that the instance I without J_{ij} would also see a deadline miss of J at t_f, contradicting the minimality of I.

(2) $a_{ij} < a_0$. If $d_{ij} \le t_f$, then $\hat{d}_{ij} = a_{ij} + x(d_{ij} - a_{ij}) \le a_0 + x(t_f - a_0)$. Otherwise, if $d_{ij} > t_f$, then again $\hat{d}_{ij} \le \hat{d}_0$ must hold, since otherwise we can remove J_{ij} from I without affecting the deadline miss of J, thus contradicting the minimality of I.

The proof is completed by noting that $\hat{d}_{ij} \le \hat{d}_0 = a_0 + x(d_0 - a_0) \le a_0 + x(t_f - a_0)$. □

Following Claim 4.8, we define S_1 to be the set of criticality-1 jobs of I, and S_2 to be the set of criticality-2 jobs of I that only receive execution before t^*. Therefore, the collection of jobs I is the disjoint union of S_1, S_2 and S^*.

Let C_1 be the sum of execution requirements of jobs in S_1; C_2 the sum of execution requirements of jobs in S_2; and C^* the sum of execution requirements of jobs in S^*. Let $\delta \stackrel{\text{def}}{=} t_f - a_0$, the length of the time interval in which there are pending jobs from S^*.

Then, Claim 4.8 yields the following two inequalities:

$$\begin{cases} C_1 & \le \lambda_1(a_0 + x\delta) \\ C_1 + C_2 & \le \lambda_1(a_0/x + \delta). \end{cases} \tag{18}$$

The second inequality holds because $a_{ij} + xd_i = \hat{d}_{ij} \le a_0 + x\delta$ implies that $d_{ij} \le (a_0 + x\delta)/x$, similar to the conclusion we had in the proof of Lemma 4.5.

By definition of λ_2, we also obtain another set of inequalities

$$\begin{cases} C^* & \le \lambda_2\delta \\ C^* + C_2 & \le \lambda_2(a_0/x + \delta). \end{cases} \tag{19}$$

Since J misses its deadline $t_f = a_0 + \delta$, as we assumed, we have $C_1 + C_2 + C^* > t_f$, which is equivalent to $C_1 + C_2 + C^* - a_0 > \delta$. Without loss of generality, we assume that δ is constant (since all other values can be scaled proportionally). Then, the maximum value of $C_1 + C_2 + C^* - a_0$ over all possible a_0 will be greater than δ.

In order to find this maximum value, we replace a_0 by t and study the function $o(t) \stackrel{\text{def}}{=} C_1 + C_2 + C^* - t$. From inequalities (18) and (19), we get

$$\begin{cases} o(t) & \le f(t) \stackrel{\text{def}}{=} \lambda_1(t + x\delta) + \lambda_2(t/x + \delta) - t \\ o(t) & \le g(t) \stackrel{\text{def}}{=} \lambda_1(t/x + \delta) + \lambda_2\delta - t. \end{cases} \tag{20}$$

Note that since $\lambda_1 + \lambda_2 > 1$ (otherwise, it must be the case that $\lambda \le \lambda_1 + \lambda_2 \le 1$, according to Proposition 4.3) and $\lambda_1/x \le 1$ (because of Lemma 4.5), we obtain the following bounds on the derivatives of f and g

$$\frac{\partial f(t)}{\partial t} = \lambda_1 + \lambda_2/x - 1 \ge \lambda_1 + \lambda_2 - 1 > 0$$
$$\frac{\partial g(t)}{\partial t} = \lambda_1/x - 1 \le 1 - 1 = 0.$$

Hence, when t increases, $f(t)$ increases and $g(t)$ does not increase, for $t \in (0, +\infty)$. Thus, $\max_t o(t)$ cannot exceed the value of f and g at the point where they cross. We denote this point by γ, that is, $f(\gamma) = g(\gamma)$.

Straightforward calculations yield

$$\gamma = \frac{\lambda_1(1-x)x\delta}{\lambda_2 - \lambda_1(1-x)}, \tag{21}$$

and therefore

$$\max_t o(t) \leq f(\gamma) = g(\gamma) = \frac{\lambda_2^2 + \lambda_1\lambda_2 x - \lambda_1 x + \lambda_1 x^2}{\lambda_2 - \lambda_1 + \lambda_1 x}\delta. \tag{22}$$

Hence, if J misses its deadline as we assumed, we must have $o(\gamma) > \delta$, that is,

$$\lambda_1 x^2 + (\lambda_1\lambda_2 - 2\lambda_1)x + \lambda_1 - \lambda_2 + \lambda_2^2 > 0. \tag{23}$$

Inserting our choice of $x = 1 - \lambda_2/2 = -(\lambda_1\lambda_2 - 2\lambda_1)/2\lambda_1$ in (23) leads to

$$-(\lambda_1\lambda_2 - 2\lambda_1)^2 + 4\lambda_1\left(\lambda_1 - \lambda_2 + \lambda_2^2\right) > 0,$$

which simplifies to

$$\lambda_1 + \lambda_2 - \frac{\lambda_1\lambda_2}{4} - 1 > 0, \tag{24}$$

contradicting the assumption of the theorem. □

COROLLARY 4.9. *If for a 2-level task system τ, $\lambda_1 + \lambda_2/2 \leq 1$ and $\lambda_1 + \lambda_2 - \lambda_1\lambda_2/4 \leq 1$, then τ is correctly scheduled by* EDF-VD *with scaling parameter $x = 1 - \lambda_2/2$.*

PROOF. By Lemma 4.6, we can select $x = 1 - \lambda_2/2$ to guarantee no deadline miss while the system is in level 2. Since $x = 1 - \lambda_2/2 \geq \lambda_1$, Lemma 4.5 guarantees that there is no deadline miss while the system is in level 1. □

4.3. Speedup Bound

THEOREM 4.10. *If a 2-level task system τ is feasible on a unit-speed processor, then it is schedulable by* EDF-VD *on a processor of speed $1 + \sqrt{3}/2 \approx 1.866$.*

PROOF. If a 2-level task system τ is feasible on a unit-speed processor, by Proposition 4.2, $\lambda_1(\tau)$ and $\lambda_2(\tau)$ are both no greater than 1. Thus, on a $(1 + \sqrt{3}/2)$-speed processor, the load-to-speed ratios $\lambda_1/(1+\sqrt{3}/2)$ and $\lambda_2/(1+\sqrt{3}/2)$ are no greater than $1/(1 + \sqrt{3}/2) = 4 - 2\sqrt{3}$. We complete the proof by showing that any task set is schedulable by EDF-VD on a unit-speed processor whenever $\max(\lambda_1, \lambda_2) \leq 4 - 2\sqrt{3}$.

Define $L(\lambda_1, \lambda_2) = \lambda_1 + \lambda_2 - \lambda_1\lambda_2/4 - 1$. Basic arithmetic shows that choosing $\lambda_1 = \lambda_2 = 4 - 2\sqrt{3}$ yields $L(\lambda_1, \lambda_2) = 0$ and also $\lambda_1 + \lambda_2/2 \leq 4 - 2\sqrt{3} + 2 - \sqrt{3} = 6 - 3\sqrt{3} \leq 1$. Thus, Corollary 4.9 asserts that such a system is EDF-VD schedulable, with scaling parameter $x = 1 - \lambda_2/2$. For values of λ_1 and λ_2 satisfying $\lambda_1 + \lambda_2 \geq 1$, $\lambda_1 \geq 0$ and $\lambda_2 \geq 0$, we have

$$\frac{\partial L(\lambda_1, \lambda_2)}{\partial \lambda_1} = 1 - \frac{\lambda_2}{4} > 0$$
$$\frac{\partial L(\lambda_1, \lambda_2)}{\partial \lambda_2} = 1 - \frac{\lambda_1}{4} > 0,$$

which implies the schedulability of any system with $\max(\lambda_1, \lambda_2) \leq 4 - 2\sqrt{3}$. □

This analysis of EDF-VD is tight for arbitrary-deadline systems, as shown by the following.

THEOREM 4.11. *Let $\epsilon > 0$. There exists a 2-level task system τ with $\lambda_1 = \lambda_2 = 4 - 2\sqrt{3} + \epsilon$ that cannot be scheduled by* EDF-VD *on a unit-speed processor.*

PROOF. We define $\alpha \stackrel{\text{def}}{=} 4 - 2\sqrt{3}$, and let T, N and M be large integers with $T \gg M \gg N$; for concreteness, say $T = M^2 = N^4$. Then, we choose N large enough, such that

$N(\sqrt{\alpha}-\alpha) \geq 2$ and $(\sqrt{\alpha}-\alpha)/N = \nu < \epsilon$. We define a task system $\tau = (\tau_1, \tau_2, \tau_{3,1}, \ldots, \tau_{3,M})$ as follows.

τ_i	χ_i	$c_i(1)$	$c_i(2)$	d_i	p_i
τ_1	1	α	α	$\alpha/(\alpha+\epsilon)$	T
τ_2	2	0	$\alpha+\epsilon$	1	T
$\tau_{3,1}$	2	$(\sqrt{\alpha}-\alpha)/N$	$(\sqrt{\alpha}-\alpha)/N$	$1+(1/\sqrt{\alpha}-1)/N$	T
$\tau_{3,2}$	2	$(\sqrt{\alpha}-\alpha)/N$	$(\sqrt{\alpha}-\alpha)/N$	$1+2(1/\sqrt{\alpha}-1)/N$	T
$\vdots$			$\vdots$		$\vdots$
$\tau_{3,i}$	2	$(\sqrt{\alpha}-\alpha)/N$	$(\sqrt{\alpha}-\alpha)/N$	$1+i(1/\sqrt{\alpha}-1)/N$	T
$\vdots$			$\vdots$		$\vdots$
$\tau_{3,N}$	2	$(\sqrt{\alpha}-\alpha)/N$	$(\sqrt{\alpha}-\alpha)/N$	$1/\sqrt{\alpha}$	T
$\vdots$			$\vdots$		$\vdots$
$\tau_{3,i}$	2	$(\sqrt{\alpha}-\alpha)/N$	$(\sqrt{\alpha}-\alpha)/N$	$1+M(1/\sqrt{\alpha}-1)/N$	T

In the sequel, we will frequently use that $\sqrt{\alpha}-\alpha = \alpha(1/\sqrt{\alpha}-1)$. We verify that λ_1 and λ_2 of τ are both $\alpha+\epsilon$, by considering the synchronous arrival sequences of τ for level 1 and level 2 separately. Let $\lambda_1(t)$ and $\lambda_2(t)$ be the load at level 1 and 2, respectively, over the interval $[0, t)$:

$$\lambda_1(t) \stackrel{\text{def}}{=} \frac{1}{t}\sum_{\tau_i} \left\lfloor \frac{t}{p_i} \right\rfloor c_i(1),$$

$$\lambda_2(t) \stackrel{\text{def}}{=} \frac{1}{t}\sum_{\tau_i:\chi_i=2} \left\lfloor \frac{t}{p_i} \right\rfloor c_i(2).$$

As argued before, λ_1 and λ_2 are determined by $\max_t \lambda_1(t)$ and $\max_t \lambda_2(t)$. Clearly, $\lambda_1(\alpha/(\alpha+\epsilon)) = \alpha+\epsilon$ and $\lambda_2(1) = \alpha+\epsilon$. Since T is very large, we only need to further consider $\lambda_1(1+i(1/\sqrt{\alpha}-1)/N)$, for $i = 1, \ldots, M$:

$$\lambda_1(1+i(1/\sqrt{\alpha}-1)/N) = \frac{\alpha+i(\sqrt{\alpha}-\alpha)/N}{1+i(1/\sqrt{\alpha}-1)/N} = \alpha \leq \alpha+\epsilon.$$

Similarly, $\lambda_2(1+i(1/\sqrt{\alpha}-1)/N) \leq \alpha+\epsilon$. Hence, λ_1 and λ_2 are both equal to $\alpha+\epsilon$.

Now let us assume that a scaling factor x has been selected. We would like to show that the system is not schedulable by EDF-VD with this (arbitrary) x.

Given x, assume that τ_2 releases its first job at time $1-x$, and all other tasks release their first job at time 0. Then, the actual deadline of the first job of τ_2 (denoted as J_2) is $d_2 = 1-x+1 = 2-x$, and the virtual deadline will be $\hat{d}_2 = (1-x)+x\times 1 = 1$. Because the virtual deadline is greater than the deadline of the first job of τ_1 (denoted as J_1), which is $d_1 = 1-\epsilon$, J_2 will wait until J_1 finishes.

If x is so small that $x(1+M(1/\sqrt{\alpha}-1)/N) < 1$, then all first jobs from $\tau_{3,1}, \ldots, \tau_{3,M}$ have priority over J_2. Their total execution requirement is $M(\sqrt{\alpha}-\alpha)/N = N(\sqrt{\alpha}-\alpha) \geq 2$, by the choice of M and N.

Otherwise, there exists a largest number i such that $x(1+i(1/\sqrt{\alpha}-1)/N) \leq 1$. Let this number be i_x. Then all first jobs of $\tau_{3,1}, \ldots, \tau_{3,i_x}$ have priority over J_2. Their total

worst-case execution time is

$$\begin{aligned} i_x(\sqrt{\alpha}-\alpha)/N &= (i_x+1)(\sqrt{\alpha}-\alpha)/N - (\sqrt{\alpha}-\alpha)/N \\ &= (i_x+1)\alpha(1/\sqrt{\alpha}-1)/N - \nu \\ &\geq \alpha\left(\frac{1}{x}-1\right) - \nu. \end{aligned}$$

This, together with the WCET of J_1 and the level-2 WCET of J_2 itself, yields a total execution requirement of

$$\alpha + \alpha + \epsilon + \frac{\alpha}{x} - \alpha - \nu = \alpha + \frac{\alpha}{x} + \epsilon - \nu.$$

Now, by the choice of N we have $\epsilon' \stackrel{\text{def}}{=} \epsilon - \nu > 0$ and

$$\begin{aligned} \alpha + \alpha/x + \epsilon' - (2-x) &= \alpha + \alpha/x + x - 2 + \epsilon' \geq \alpha + 2\sqrt{\alpha} - 2 \\ &= 4 - 2\sqrt{3} + 2(\sqrt{3}-1) - 2 + \epsilon' = \epsilon' > 0. \end{aligned}$$

Hence, for this arbitrarily given x, we can always find a scenario in which J_2 misses its deadline, implying that it is impossible to schedule τ using EDF-VD. □

4.4. Effect of the Approximation Error when Computing the Loads

Theorem 4.10 presumes that the load λ, the level-1 load λ_1, and the level-2 load λ_2 are computed exactly in Algorithm 3 (line 1). However, for improved efficiency – for example, if a polynomial-time schedulability test is desired – it may be necessary to resort to approximations of such loads. The approach of Albers and Slomka [2004], for example, guarantees that one can find, in polynomial time for any fixed $\epsilon > 0$, values $\tilde{\lambda}_1$, $\tilde{\lambda}_2$ such that

$$\begin{aligned} \lambda_1 \leq \tilde{\lambda}_1 \leq (1+\epsilon)\lambda_1, \\ \lambda_2 \leq \tilde{\lambda}_2 \leq (1+\epsilon)\lambda_2. \end{aligned} \tag{25}$$

We observe that if one defines further

$$\tilde{\lambda} \stackrel{\text{def}}{=} \tilde{\lambda}_1 + \tilde{\lambda}_2 \tag{26}$$

and uses $\tilde{\lambda}$, $\tilde{\lambda}_1$, $\tilde{\lambda}_2$ in place of λ, λ_1, λ_2, respectively, then all the claims in Sections 4.1 and 4.2 continue to hold. This is due to the fact that such approximate loads never underestimate the correct load values. For example, Lemma 4.4 used the hypothesis $\lambda \leq 1$, but since $\tilde{\lambda} \geq \lambda_1 + \lambda_2 \geq \lambda$ (Proposition 4.3), the hypothesis $\tilde{\lambda} \leq 1$ implies $\lambda \leq 1$. Similarly, the hypotheses of Lemma 4.5 and Lemma 4.6 are stronger when expressed in terms of the approximated loads. Proposition 4.2, on the other hand, is easily adapted using the fact that $\lambda_k \geq \tilde{\lambda}_k/(1+\epsilon)$, for $k \in \{1, 2\}$.

PROPOSITION 4.12. *Let τ be a 2-level task system and $\epsilon > 0$. If $\tilde{\lambda}_1 > 1+\epsilon$ or $\tilde{\lambda}_2 > 1+\epsilon$, then τ is not feasible.*

We conclude that one needs simply generalize the statement of Theorem 4.10 by increasing the stated speedup from $1 + \sqrt{3}/2$ to $(1+\epsilon)(1+\sqrt{3}/2)$.

THEOREM 4.13. *Let $\epsilon > 0$. If a 2-level task system τ is feasible on a unit-speed processor, then it is schedulable by* EDF-VD *on a processor of speed $(1+\epsilon)(1+\sqrt{3}/2)$ whenever the loads λ, λ_1, λ_2 are approximated as in* (25)–(26).

PROOF. Let $\sigma_\epsilon \stackrel{\text{def}}{=} (1+\epsilon)(1+\sqrt{3}/2)$. Reasoning as in the proof of Theorem 4.10 yields that the normalized estimated loads $\tilde{\lambda}_1/\sigma_\epsilon$ and $\tilde{\lambda}_2/\sigma_\epsilon$ are both not larger than $4 - 2\sqrt{3}$,

and the proof of Theorem 4.10 already establishes that a task set is schedulable by EDF-VD on a unit-speed processor whenever the largest of its (normalized) estimated loads does not exceed $4 - 2\sqrt{3}$. □

5. EFFICIENT IMPLEMENTATION OF THE RUNTIME DISPATCHING

For traditional (non-MC) sporadic task systems consisting of n tasks, uniprocessor EDF can be implemented efficiently to have a runtime complexity of $\mathcal{O}(\log n)$ per event, where an event is either the arrival of a job, or the completion of the execution of a job (see, e.g., Mok [1988]). A direct application of such implementations can be used to obtain an implementation of the runtime dispatching of EDF-VD that has a runtime of $\mathcal{O}(\log n)$ per job-arrival and job-completion event. However, EDF-VD potentially needs to deal with an additional runtime event: the change in the criticality level of the behavior from at most k to more than k (this is the event that is triggered at the instant that `current_level()` $> k$). Since this event requires that each subsequent scheduling be done according to each task's original deadline, explicitly recomputing priorities according to these original deadlines would take time linear in the number of tasks of criticality more than k – in the worst case, $\mathcal{O}(n)$ time. We now describe an implementation of EDF-VD's runtime system that has a worst-case runtime of $\mathcal{O}(\log n)$ per event for all three kinds of events: job arrival, job completion, and change in the criticality level of the behavior from LO (i.e., at most k) to HI (more than k).

Recall that a priority queue supports the operations of inserting (`insert`) and deleting the smallest item (`deleteMin`) in logarithmic time, and the operation of finding the smallest item (`min`) in constant time. In addition, the standard priority queue data structure can be enhanced to support the deletion of a specified item (the `delete` operation), also in logarithmic time (see, e.g., Cormen et al. [2009, Sect. 6.5]). We maintain two such enhanced priority queues, Q_{LO} and Q_{HI}. We also use a timer that is used to indicate whether the currently executing job has executed for more than its kth level WCET (thereby triggering the event `current_level()` $> k$).

Initially, `current_level()` $= 1$ and there are three kinds of events to be dealt with: (1) the arrival of a job; (2) the completion of a job; and (3) `current_level()` reaching a value larger than k. We consider each separately. Suppose that the event occurs at time-instant t_c, and let J_c denote the currently executing job.

(1) A job of task τ_i arrives at time t_c.
   (a) Insert the newly arrived job into Q_{LO}, prioritized according to its virtual scheduling deadline.
   (b) If $\chi_i > k$, then also insert it into Q_{HI}, prioritized according to its unmodified (i.e., actual) scheduling deadline.
   (c) If J_c is no longer the minimum job in Q_{LO}, it must be the case that the newly arrived job has an earlier virtual deadline than J_c's virtual deadline. In that case, the newly inserted job becomes J_c, and a timer for it is switched on and set to go off after it has clocked $c_i(k)$. The timer of the interrupted job is stopped (and resumed as soon as the job will get executed again).

(2) The currently executing job J_c completes execution at time t_c.
   (a) Delete this job from Q_{LO}, using the `deleteMin` operation supported by priority queue implementations.
   (b) If it was a job of criticality larger than k, also delete it from Q_{HI} – this would be accomplished by a `delete` operation.
   (c) Set the current-job indicator J_c to denote the new minimum (virtual) deadline job – the "minimum" job in Q_{LO}; and set the timer to go off at $t_c +$ this job's remaining level-k WCET (when the job would exceed its level-k WCET if allowed to execute without interruption).

(3) The timer goes off, indicating that the currently executing job has executed beyond its level-k WCET without signaling completion. We switch to scheduling according to Q_{HI}. Henceforth, all runtime dispatch decisions are taken as indicated by this priority queue.

After the event `current_level()` $> k$, no jobs of criticality at most k need execution, and jobs of criticality larger than k are executed according to EDF with respect to their original (unmodified) deadlines. Hence, subsequent runtime dispatching is done as for traditional EDF scheduling (as described in, e.g., Mok [1988]), with Q_{HI} being the priority queue used for this purpose.

6. EXTENSIONS

The basic algorithm EDF-VD can be extended in a number of ways that enhance its applicability. In this section, we consider two such extensions.

6.1. Best-Effort Runtime Dispatching and Return to Low-Level Dispatching Mode

The basic algorithm EDF-VD was designed for best performance under worst-case scenarios; in particular, it may sometimes be overly pessimistic by completely discarding low-criticality tasks after observing a high-criticality behavior, and by never returning to the low-level mode of runtime dispatching (lines 1–8 of Algorithm 2) after a high-level behavior has been observed. We briefly discuss how to modify the algorithm in order to correctly schedule a larger number of jobs whenever possible, while maintaining the same worst-case guarantee in terms of speedup bounds.

The first modification is straightforward: instead of completely discarding low-criticality tasks after entering the higher-level behavior, they can simply be scheduled with a priority lower than that of all high-criticality tasks. This, clearly, will meet the deadlines of at least all the jobs that meet their deadlines under the basic EDF-VD schedule; in particular, it preserves the correct scheduling of the high-criticality tasks. On the other hand, this more optimistic schedule may meet the deadlines of many otherwise ignored tasks, especially when the high-level behavior is exhibited marginally and by only a few tasks.

Regarding the return to the low-level mode of runtime dispatching, the modification is perhaps less straightforward. Let us limit the discussion to task systems with two criticality levels, 1 (LO) and 2 (HI). We note that we can preserve the schedulability guarantees with the following modification: on top of its criticality level, the runtime dispatcher classifies at all times each job as either *standard* or *best-effort*. All standard jobs have priority over best-effort jobs; standard jobs are scheduled as before by Algorithm 2, while the relative priorities of best-effort jobs can be arbitrary. Initially, the best-effort queue is empty and newly released jobs are always classified as standard.

Whenever the criticality-level counter `current_level()` is 2, but the set of active jobs having criticality level HI is empty, the algorithm (i) resets the criticality-level counter `current_level()` to 1 – and consequently, returns the runtime dispatcher to its LO mode of execution; (ii) at the same time, it demotes all active LO-criticality job to best-effort priority. This may violate the deadline of some active LO-criticality job; in particular, of a job demoted to the best-effort queue. However, we observe that such jobs need not be guaranteed anyway, since `current_level()` reaches the value 2 at some point and so the criticality level of the scenario is HI according to the definition (Section 2). The advantage of returning to the low-level dispatching mode is that *subsequently* released jobs of LO-criticality will be correctly scheduled, at least until a HI-level behavior is exhibited again. Note that tasks having criticality level HI are always guaranteed,

since, by construction, best-effort jobs never interfere with standard jobs (and HI-criticality jobs are never moved to the best-effort queue).

6.2. Nonuniform Virtual Deadlines

In our basic approach, there was a single scaling factor x and the virtual deadlines for the high-criticality tasks were defined uniformly, via the rule $\hat{d}_i \stackrel{\text{def}}{=} x \cdot d_i$. However, in many situations, it may be beneficial to define nonuniform scaling factors x_i – one for every high-criticality task. For concreteness, we limit this discussion to the case of two criticality levels and implicit-deadline task systems.

Although, in light of Theorem 3.12, this will not help to achieve a better worst-case speedup bound than 4/3, setting the virtual deadlines nonuniformly may potentially schedule task sets that are not deemed schedulable by EDF-VD. We proceed to show that this is indeed sometimes the case and discuss how the schedulability condition is affected and how to set the deadline-scaling factors.

Consider the following variant of EDF-VD, called EDF with Non-Uniform Virtual Deadlines (EDF-NUVD). Given a task system τ, we first check whether τ satisfies the hypothesis of Theorem 3.4; if it does, we can simply apply EDF-VD. If the hypothesis of Theorem 3.4 is not satisfied, then instead of defining $\hat{d}_i \stackrel{\text{def}}{=} x \cdot d_i$ for every high-criticality task τ_i, we set $\hat{d}_i \stackrel{\text{def}}{=} x_i \cdot d_i$, where $x_i \in (0, 1)$ is a task-dependent scaling parameter; we will see shortly how to set the scaling parameters and how to complete the schedulability test. Apart from the revised definition of the virtual deadlines, the runtime dispatching algorithm is otherwise unaffected.

As starting point of the analysis, we use the following alternative schedulability condition. For a task set τ, let $L_2 \stackrel{\text{def}}{=} \{i \in [n] : \chi_i = 2\}$ be its set of high-criticality tasks.

LEMMA 6.1. *Let τ be a 2-level task system and let $0 < x_i < 1$, for each $i \in L_2$. If*

$$\begin{cases} U_1(1) + \sum_{i \in L_2} u_i(1)/x_i & \leq 1 \\ \sum_{i \in L_2} u_i(2)/(1 - x_i) & \leq 1, \end{cases} \tag{27}$$

then τ is schedulable by EDF-NUVD.

PROOF. First observe that the condition

$$U_1(1) + \sum_{i \in L_2} u_i(1)/x_i \leq 1$$

ensures that no deadline (virtual or otherwise) is missed by EDF-NUVD while the system exhibits level-1 behavior. Let t^* denote the time at which the scenario first exhibits level-2 behavior. Consider a job J_{ij} of a high-criticality task τ_i that is active at t^*. Let a_{ij} denote its arrival time. The absolute deadline of this job is $d_{ij} = a_{ij} + d_i$ and before t^*, J_{ij} is EDF-scheduled by EDF-NUVD assuming a deadline $\hat{d}_{ij} = a_{ij} + x_i d_i$. Since all jobs would meet their virtual deadlines in any level-1 scenario, we know that $t^* \leq \hat{d}_{ij}$; otherwise, the job would no longer be active at t^*. Note that

$$d_{ij} - t^* \geq d_{ij} - \hat{d}_{ij} = d_i - x_i d_i = (1 - x_i) d_i.$$

In the worst case, we can therefore regard the system after time t^* as a single-criticality task system in which task τ_i has period $(1 - x_i)d_i$ and utilization $u_i(2)/(1 - x_i)$. Then, condition

$$\sum_{i \in L_2} u_i(2)/(1 - x_i) \leq 1$$

ensures that this system is itself EDF-schedulable. □

It remains to show how to choose the scaling parameter x_i for each high-criticality task τ_i. We show that, when using (27) as the sufficient schedulability condition, a best possible choice of the x_i is given by

$$x_i \stackrel{\text{def}}{=} \left(1 + \lambda\sqrt{u_i(2)/u_i(1)}\right)^{-1} \qquad \text{for all } i \in L_2 \tag{28}$$

for some positive parameter λ. The particular form (28) is not arbitrary; its choice is justified by the following result.

THEOREM 6.2. *For a 2-level task system τ, let*

$$S_{12} \stackrel{\text{def}}{=} \sum_{i \in L_2} \sqrt{u_i(1)u_i(2)}.$$

If τ satisfies

$$\begin{cases} U_1(1) + U_2(1) + \lambda\, S_{12} & \le 1, \\ U_2(2) + \lambda^{-1}\, S_{12} & \le 1, \end{cases} \tag{29}$$

for some $\lambda > 0$, then τ is schedulable by EDF-NUVD *when the x_i are as in* (28)*. Moreover, whenever* (27) *holds for some choice of the x_i, then* (29) *holds for some $\lambda > 0$.*

PROOF. For the first part of the theorem, by Lemma 6.1, we just need to check if (27) holds for the given choice of the x_i, which are determined up to the constant λ. We substitute the value of λ from (28) and the definition of S_{12} into (29) to obtain the condition

$$\begin{cases} U_1(1) + U_2(1) + \sum_{i \in L_2} (1/x_i - 1)\sqrt{u_i(1)/u_i(2)} \cdot \sqrt{u_i(1)u_i(2)} & \le 1 \\ U_2(2) + \sum_{i \in L_2} (x_i/(1-x_i))\sqrt{u_i(2)/u_i(1)} \cdot \sqrt{u_i(1)u_i(2)} & \le 1, \end{cases}$$

which is easily seen to be equivalent to (27), proving the first part of the theorem.

To prove the second part, we form the Lagrange dual function of (27) [Boyd and Vandenberghe 2009, Sect. 5.8.1]. Since (27) has two constraints, the dual function g involves two arguments μ_1, μ_2 ($\mu_1, \mu_2 \ge 0$):

$$g(\mu_1, \mu_2) = \inf_{0 < x_i < 1 \,:\, i \in L_2} (\mu_1 f_1(x) + \mu_2 f_2(x)), \tag{30}$$

where

$$f_1(x) = U_1(1) + \sum_{i \in L_2} \frac{u_i(1)}{x_i} - 1,$$

$$f_2(x) = \sum_{i \in L_2} \frac{u_i(2)}{1 - x_i} - 1.$$

Since the set $\{0 < x_i < 1,\ i \in L_2\}$ is open, the infimum with respect to x in (30) must be achieved for values x such that $\frac{\partial}{\partial x_i}\left(\mu_1 f_1(x) + \mu_2 f_2(x)\right) = 0$ for all $i \in L_2$, yielding the condition

$$\mu_1 \left(-\frac{u_i(1)}{x_i^2}\right) + \mu_2 \frac{u_i(2)}{(1-x_i)^2} = 0.$$

Solving for x_i yields

$$x_i = \left(1 + \sqrt{\frac{\mu_2}{\mu_1}}\sqrt{\frac{u_i(2)}{u_i(1)}}\right)^{-1} \qquad \text{for all } i \in L_2,$$

which is equivalent to (28) after defining $\lambda = \sqrt{\mu_2/\mu_1}$. The weak alternative form of Lagrange duality implies that if the inequality system (27) is feasible, the inequality

$$\mu_1 f_1(x) + \mu_2 f_2(x) > 0$$

must be infeasible for any choice of $\mu_1, \mu_2 \geq 0$. This in turn implies that $f_1(x),\ f_2(x) \leq 0$, which for the particular choice of x given by (28) implies (29). □

Similarly as in Section 3, we can rewrite the system of inequalities (29) as a single condition:

$$\frac{S_{12}}{1 - U_2(2)} \leq \frac{1 - U_1(1) - U_2(1)}{S_{12}}. \tag{31}$$

If (31) holds, then λ can be chosen as any value in the interval

$$\left[\frac{S_{12}}{1 - U_2(2)}, \frac{1 - U_1(1) - U_2(1)}{S_{12}}\right],$$

and the values of the x_i can be chosen accordingly. In summary, whenever either the assumptions of Theorem 3.4 hold, or Condition (31) holds, then τ is schedulable by EDF-NUVD.

Example 6.3. Consider the following task set $\tau = (\tau_1, \tau_2, \tau_3)$, where $\epsilon = 1/1000$.

τ_i	χ_i	$u_i(1)$	$u_i(2)$
τ_1	1	$3/4 - \epsilon$	$3/4 - \epsilon$
τ_2	2	$1/8$	$1/8$
τ_3	2	ϵ	$5/8$

We observe that $U_1(1) = 3/4 - \epsilon$, $U_2(1) = 1/8 + \epsilon$, $U_2(2) = 3/4$. Conditions (2) and (3) are both violated (for $\epsilon = 1/1000$), so that the system is deemed unschedulable by EDF-VD. On the other hand, Condition (31) is satisfied since $S_{12} = \sum_{i \in L_2} \sqrt{u_i(1)u_i(2)} = 3/20$. Therefore, we can choose any $\lambda \in [3/5, 5/6]$ and conclude that τ is schedulable by EDF-NUVD.

7. SUMMARY AND CONCLUSIONS

Devising more cost-efficient techniques for obtaining certification for safety-critical embedded systems has been identified as a prime research challenge [Barhorst et al. 2009]. We believe that in mixed-criticality systems, these certification considerations give rise to fundamental new resource allocation and scheduling challenges that are not adequately addressed by conventional real-time scheduling theory. In this article, we consider the scheduling, upon preemptive uniprocessors, of mixed-criticality systems that can be modeled using a mixed-criticality generalization of the sporadic task model.

We propose an algorithm called EDF-VD for scheduling such systems. We have proved that EDF-VD is speedup-optimal by showing that (i) it has a processor speedup factor equal to 4/3 for 2-level implicit-deadline systems (Theorem 3.8); and (ii) no non-clairvoyant algorithm can have a smaller speedup factor (Theorem 3.12). We have also proved bounds on the speedup factor of EDF-VD for implicit-deadline systems with up to 13 criticality levels (Theorem 3.11), and a bound of $1 + \sqrt{3}/2$ on the speedup factor of EDF-VD for 2-level arbitrary-deadline systems (Theorem 4.10).

We have shown how EDF-VD admits efficient schedulability tests and how it can be implemented to have a runtime complexity per scheduling decision that is logarithmic in the number of tasks for any K-level implicit-deadline task system, and thus demonstrated the practical applicability of the algorithm. Finally, we discussed extensions

of EDF-VD by analyzing the possibility of using nonuniform scaling parameters and more optimistic runtime dispatching rules.

For further research, we observe that the mixed-criticality model proposed by Vestal [2007] can be extended to take into account other aspects that arise in the design of safety-critical systems. A detailed discussion of the extensions is not the focus of this article. We observe, however, that a very interesting contribution of Ekberg and Yi [2014] concerns the generalization of the mixed-criticality task model to models where not only the execution times change after a critical event, but all task parameters are allowed to change.

ACKNOWLEDGMENTS

We thank the anonymous referees for their observations and criticisms that helped improve the presentation and quality of this work.

REFERENCES

K. Albers and F. Slomka. 2004. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*. IEEE, Los Alamitos, CA, 187–195.

J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. S. P. Stanfill, D. Stuart, and R. Urzi. 2009. A research agenda for mixed-criticality systems. White paper. http://www.cse.wustl.edu/∼cdgill/CPSWEEK09_MCAR/.

S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. 2012. Scheduling real-time mixed-criticality jobs. *IEEE Trans. Comput.* 61, 8, 1140–1152.

S. K. Baruah, R. R. Howell, and L. E. Rosier. 1993. Feasibility problems for recurring tasks on one processor. *Theor. Comput. Sci.* 118, 1, 3–20.

S. K. Baruah, H. Li, and L. Stougie. 2010a. Mixed-Criticality scheduling: Improved resource-augmentation results. In *Proceedings of the ISCA International Conference on Computers and their Applications*. ISCA, Los Alamitos, CA, 217–223.

S. K. Baruah, H. Li, and L. Stougie. 2010b. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the 16th IEEE Real-Time Technology and Applications Symposium.* IEEE, Los Alamitos, CA, 13–22.

P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. 2009. Branching and bounds tightening techniques for non-convex MINLP. *Optimi. Meth. Softw.* 24, 4–5, 597–634.

S. Boyd and L. Vandenberghe. 2009. *Convex Optimization*. Cambridge University Press, Cambridge, UK.

A. Burns and R. I. Davis. 2013. Mixed criticality systems - A Review. http://www-users.cs.york.ac.uk/∼burns/review.pdf.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms* 3rd Ed. MIT Press, Cambridge, MA.

M. L. Dertouzos. 1974. Control robotics: The procedural control of physical processes. In *Proceedings of the International Federation for Information Processing Congress*. North-Holland, Amsterdam, 807–813.

A. Easwaran. 2013. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Proceedings of 34th IEEE Real-Time Systems Symposium*. IEEE, Los Alamitos, CA, 78–87.

P. Ekberg and W. Yi. 2012. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Proceedings of 24th Euromicro Conference on Real-Time Systems*. IEEE, Los Alamitos, CA, 135–144.

P. Ekberg and W. Yi. 2014. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-Time Systems* 50, 1, 48–86.

N. Guan, P. Ekberg, M. Stigge, and W. Yi. 2011. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium.* IEEE, Los Alamitos, CA, 13–23.

J. Y.-T. Leung and J. Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perf. Eval.* 2, 4, 237–250.

H. Li and S. K. Baruah. 2010. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proceedings of the 31st IEEE Real-Time Systems Symposium.* IEEE, Los Alamitos, CA, 183–192.

C. L. Liu and J. W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* 20, 1, 46–61.

A. K. 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 Technical Report No. MIT/LCS/TR-297.)

A. K. Mok. 1988. Task management techniques for enforcing ED scheduling on periodic task set. In *Proceedings of the 5th IEEE Workshop on Real-Time Software and Operating Systems*. USENIX Association, Washington, DC, 42–46.

P. J. Prisaznuk. 1992. Integrated modular avionics. In *Proceedings of the IEEE National Aerospace and Electronics Conference*, Vol. 1. IEEE, Los Alamitos, CA, 39–45.

H. Su and D. Zhu. 2013. An elastic mixed-criticality task model and its scheduling algorithm. In *Proceedings of the Conference on Design, Automation & Test in Europe*. EDA Consortium, San Jose, CA, 147–152.

S. Vestal. 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*. IEEE, Los Alamitos, CA, 239–243.

R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al. 2008. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.* 7, 3, 36.