The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems

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

The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems

S. Baruah V. Bonifaci G. D'Angelo H. Li A. Marchetti-Spaccamela S. van der Ster L. Stougie[1]

***Abstract*—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 a lower level of assurance. An algorithm called EDF-VD (for Earliest Deadline First with Virtual Deadlines) is described for the scheduling of such mixed-criticality task systems. Analyses of EDF-VD significantly superior to previously-known ones are presented, based on metrics such as processor speedup factor (EDF-VD is proved to be *optimal* with respect to this metric) and utilization bounds.**

I. 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) [19] in aerospace and AUTOSAR (AUTomotive Open System ARchitecture — see `www.autosar.org`) in the automotive industry. This trend towards integration means that even in highly safety-critical systems, typically only a relatively small fraction of the overall system is actually of critical functionality and subject to mandatory certification by statutory certification authorities (CAs). Such systems are called *mixed-criticality (MC)* systems.

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.

To show that real-time systems meet timing properties, *worst-case execution times (WCETs)* must be estimated for certain pieces of code, denoting an upper bound on the amount of time the piece of code would take to execute. 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.

Multiple WCET parameters. Why would we wish to specify multiple WCET parameters for a single piece of code? It is well known that determining exact worst-case execution times of pieces of code is a very difficult problem; instead, system engineers work with upper bounds on the exact value. However, for many non-trivial kinds of code strict upper bounds are extremely pessimistic, and represent scenarios that are highly unlikely, or indeed impossible, to occur in practice [10]. For such code, less pessimistic upper bounds on their WCET's may be obtained at lower degrees of confidence than absolute certainty. Based on the observation that "the more confidence one needs in a task execution time bound, the larger and more conservative that bound tends to be in practice," Vestal [20] proposed that multiple different WCET values be specified, with the different values being determined at different levels of assurance. These different values may be 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.

Dual-criticality systems. We have considered two criticality levels – needing certification, and not needing certification – in the discussion above. 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. We are eventually interested in systems with arbitrarily many criticality levels; however in this document we will, for ease of presentation, focus primarily on systems with just two criticality levels that we will call LO and HI; we call such systems *dual-criticality systems.* We postpone a discussion regarding how our results generalize to more than two criticality levels to an extended paper combining these results with those in [8], currently under preparation.

[1]Baruah and Li are with the University of North Carolina. Bonifaci is with Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti". D'Angelo is with MASCOTTE project, INRIA. Marchetti-Spaccamela is with Sapienza Università di Roma. Van der Ster and Stougie are with Vrije Universiteit Amsterdam; Stougie is also affiliated with CWI.

Context and Related work. In traditional (i.e., not mixed-criticality) real-time systems, a sporadic task [17], [14] τ_i is characterized by a WCET C_i, a relative deadline D_i, and a period T_i; such a task generates an unbounded sequence of jobs with successive jobs arriving at least T_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. Vestal [20] 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. Further results on this problem appear in [3], [9].

The preemptive uniprocessor scheduling of collections of mixed-criticality *independent jobs* was studied in [5], [4], [7], [6]. An efficient scheduling algorithm and associated polynomial-time schedulability test was proposed that makes the following guarantee: any dual-criticality system that can be scheduled by an optimal clairvoyant algorithm on a given processor can be scheduled by this algorithm on a processor that is $(1+\sqrt{5})/2 \approx 1.62$ times as fast. In [15], [12], this result was extended to mixed-criticality sporadic task systems: a scheduling algorithm and associated pseudopolynomial-time schedulability test was proposed that makes the same guarantee. These scheduling algorithms, however, have too large a run-time complexity to be implementable in practice: the run-time complexity per scheduling decision of the algorithm in [15] is pseudo-polynomial in the representation of the task system, while the one in [12] is quadratic in the number of tasks.

An important special case of sporadic task systems are task systems in which each task τ_i satisfies the property that $D_i = T_i$ — such systems are called *implicit-deadline* or *Liu & Layland* task systems. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems was studied in [8]. An algorithm called EDF-VD was proposed that has the same speedup guarantee as above — any task system that can scheduled by an optimal clairvoyant algorithm on a given processor can be scheduled by EDF-VD on a processor that is $(1 + \sqrt{5})/2$ times as fast. Moreover, the schedulability test of EDF-VD has polynomial run-time complexity, and the run-time complexity per scheduling decision was logarithmic in the number of tasks. Based on these run-time properties it is evident that EDF-VD, in contrast to the algorithms in [15], [12], can be considered suitable for implementation in actual systems.

This research. Our main contribution in this paper is a more refined analysis of EDF-VD showing that EDF-VD can actually make a better performance guarantee: *any task system that can be scheduled by an optimal clairvoyant algorithm on a given processor can be scheduled by EDF-VD on a processor that is* $4/3$ *times as fast*. This new analysis is based upon some sophisticated new techniques and deep insights that we have recently developed, and represents a substantial improvement over the bound proved in [8]. It was previously shown [6, Prop. 2] that $(1+\sqrt{5})/2$ is a lower bound on the speedup of any non-clairvoyant algorithm for scheduling collections of independent jobs; it is somewhat surprising that this bound does *not* hold for the more expressive implicit-deadline task model. 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 optimal clairvoyant algorithm, thereby proving that EDF-VD is an optimal non-clairvoyant algorithm from the perspective of this metric. In addition, we spell out the details as to how EDF-VD can actually be implemented to have the logarithmic run-time complexity claimed in [8]. We also perform further analysis on the behavior of EDF-VD, deriving a utilization-based schedulability test and exploring its behavior under certain extremal conditions.

Organization. The remainder of this paper is organized as follows. In Section II we formally describe the mixed-criticality model that we will be using in the remainder of this paper. In Section III we provide a high-level overview of EDF-VD. We provide a detailed description of the schedulability test in Section IV, and of the run-time algorithm in Section V. We provide a formal analysis of the properties and behavior of EDF-VD in Sections VI-VIII, and describe the outcome of some simulation experiments in Section IX.

II. Model and definitions

A mixed-criticality (MC) implicit-deadline sporadic task system τ consists of a finite specified collection of MC implicit-deadline sporadic tasks, each of which may generate an unbounded number of MC jobs.

MC jobs. As stated in Section I above, we will, for the most part, restrict our attention here to *dual*-criticality systems: systems with two distinct criticality levels, which we denote as LO and HI.

Each such dual-criticality job is characterized by a 5-tuple of parameters: $J_i = (a_i, d_i, \chi_i, c_i(\text{LO}), c_i(\text{HI}))$, where

- $a_i \in R^+$ is the release time, and $d_i \in R^+$ the deadline. We require that $d_i \geq a_i$.
- $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes the criticality of the job. A HI-criticality job (a J_i with $\chi_i = \text{HI}$) is one that is subject to certification, whereas a LO-criticality job (a J_i with $\chi_i = \text{LO}$) is one that does not need to be certified.
- $c_i(\text{LO})$ specifies the worst case execution time (WCET) estimate of J_i that is used by the system designer (i.e., the WCET estimate at the LO criticality level).
- $c_i(\text{HI})$ specifies the WCET estimate of J_i that is used by the certification authorities (i.e., the WCET estimate at the HI criticality level).

System *behavior*. The MC job model has the following semantics. Job J_i is released at time a_i, has a deadline at d_i, and needs to execute for some amount of time γ_i. The value of γ_i is not known beforehand, but only becomes revealed by actually executing the job until it *signals* that

it has completed execution. These values of γ_i for a given run of the system defines the kind of *behavior* exhibited by the system during that run. If each J_i signals completion without exceeding $c_i(\text{LO})$ units of execution, we say that the system has exhibited LO-*criticality behavior*; if even one job J_i signals completion after executing for more than $c_i(\text{LO})$ but no more than $c_i(\text{HI})$ units of execution, we say that the system has exhibited HI-*criticality behavior*. If any job J_i does not signal completion despite having executed for $c_i(\text{HI})$ units, we say that the system has exhibited *erroneous behavior*. Informally, the system-designer expects LO-criticality behavior, while the CA is allowing for the possibility of HI-criticality behavior.

MC implicit-deadline sporadic tasks. Analogously to traditional (non-MC) implicit-deadline sporadic tasks, an MC implicit-deadline sporadic task τ_k is characterized by a four-tuple $(\chi_k, C_k(\text{LO}), C_k(\text{HI}), T_k)$, with the following interpretation. Task τ_k generates an unbounded sequence of jobs, with successive jobs being released at least T_k time units apart. Each such job has a deadline that is T_k time units after its release. The criticality of each such job is χ_k, and it has LO-criticality and HI-criticality WCET's of $C_k(\text{LO})$ and $C_k(\text{HI})$ respectively; we assume that $C_k(\text{LO}) \leq C_k(\text{HI})$ for all tasks τ_k.

An MC *implicit-deadline sporadic task system* is specified by specifying a finite number of such sporadic tasks. As with traditional (non-MC) systems, such a MC sporadic task system can potentially generate infinitely many different MC instances (collections of jobs), each instance being obtained by taking the union of one sequence of jobs generated by each task.

Correctness criteria. We define an algorithm for scheduling MC task systems to be *correct* if it is able to schedule any system such that

- During all LO-criticality behaviors of the system, all jobs receive enough execution between their release time and deadline to be able to signal completion; and
- During all HI-criticality behaviors of the system, all HI-criticality jobs receive enough execution between their release time and deadline to be able to signal completion.

Note that *if any job executes for more than its* LO-*criticality WCET, we do not require any* LO-*criticality jobs (including those that may have arrived before this happened) to complete by their deadlines*. This is an implication of the requirements of certification: informally speaking, the system designer fully expects that all jobs will exhibit LO-criticality behavior, and hence is only concerned that they behave as desired under these circumstances. The CA, on the other hand, allows for the possibility that some jobs may exhibit HI-criticality behavior and requires that all HI-criticality jobs nevertheless meet their deadlines; however, the CA is not concerned with the fate of the LO-criticality jobs.

Utilization parameters. The *utilization* of a (regular, i.e., non-MC) implicit-deadline sporadic task denotes the ratio of its WCET to its period; the utilization of a task system denotes the sum of the utilizations of all the tasks in the system. We now define analogous concepts for mixed-criticality sporadic task systems.

Let τ denote a MC implicit-deadline sporadic task system. For each of x and y in $\{\text{LO}, \text{HI}\}$, we define a utilization parameter as follows:

$$U_x^y(\tau) = \sum_{\tau_i \in \tau \wedge \chi_i = x} \frac{C_i(y)}{T_i} \tag{1}$$

Thus for example, $U_{\text{HI}}^{\text{LO}}(\tau)$ denotes the sum of the utilizations of the HI-criticality tasks in τ, under the assumption that each job of each task executes for no more than its LO-criticality WCET.

III. An overview of Algorithm EDF-VD

Let τ denote the MC implicit-deadline sporadic task system that is to be scheduled on a unit-speed preemptive processor. Prior to run-time, EDF-VD performs a schedulability test to determine whether τ can be successfully scheduled by it or not. If τ is deemed schedulable, then an additional parameter, which we call a *modified period* denoted $\hat{T}_i$, is computed for each HI-criticality task $\tau_i \in \tau$. The algorithm for computing these parameters is described in pseudo-code form in Figure 1; this pseudo-code is proved correct in Section IV. Observe that it is always the case that $\hat{T}_i \leq T_i$.

Run-time scheduling is done according to the Algorithm EDF, with ***virtual deadlines***: deadlines that EDF-VD computes (in a manner to be described below) and assigns to jobs before handing them off to the EDF scheduler. The EDF scheduler will then use these virtual deadlines for the purpose of determining scheduling priority.

These virtual deadlines are assigned as follows. Suppose that a job of task τ_i arrives at time-instant t_a:

- If $\chi_i = \text{LO}$, then this job is assigned a virtual deadline equal to $t_a + T_i$.
- If $\chi_i = \text{HI}$, then this job is assigned a virtual deadline equal to $t_a + \hat{T}_i$.

If some job does execute beyond its LO-criticality WCET without signaling that it has completed execution, the following changes occur:

1) All currently-active LO-criticality jobs are immediately discarded; henceforth, no LO-criticality job will receive any execution.
2) Subsequent run-time scheduling of the HI-criticality tasks (including their jobs that are currently active) continue to be done according to EDF. But the *actual* job deadlines (arrival time plus period) are used.

IV. Pre-runtime processing

We now provide a detailed description of the pre-runtime processing conducted by EDF-VD. We describe, and prove correct, the strategy used to determine whether a system is schedulable, and for computing the modified period parameters (the $\hat{T}_k$'s) for systems deemed schedulable. This is also represented in pseudo-code form in Figure 1.

Task system $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ to be scheduled on a unit-speed preemptive processor.

1) Compute x as follows:

$$x \leftarrow \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)}$$

2) If $\left(x\, U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1\right)$ **then**

$\hat{T}_i \leftarrow x\, T_i$ for each HI-criticality task τ_i
declare success and **return**

else declare failure and **return**

Figure 1. EDF-VD: The preprocessing phase.

As shown in Figure 1, EDF-VD first computes a parameter x (the reason why x is assigned this value is derived below – see Expression 4) and then assigns values to the $\hat{T}_i$ parameters for all HI-criticality tasks as follows:

$$\hat{T}_i \leftarrow x \times T_i \tag{2}$$

Theorem 1: The following condition is sufficient for ensuring that EDF-VD successfully schedules all LO-criticality behaviors of τ:

$$x \geq \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \tag{3}$$

Proof: If EDF is able to schedule all LO-criticality behaviors of the task system obtained from τ by replacing each HI-criticality task τ_i by one with a reduced period, then it follows from the *sustainability* property [2] of preemptive uniprocessor EDF that EDF is able to schedule all LO-criticality behaviors of τ as well. Note that scaling down the period of each HI-criticality task by a factor x is equivalent to inflating its utilization by a factor $1/x$. From the utilization-bound result of EDF [16], we therefore conclude that

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{x} \leq 1 \\ \Leftrightarrow \quad & \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{x} \leq 1 - U_{\text{LO}}^{\text{LO}}(\tau) \\ \Leftrightarrow \quad & x \geq \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \end{aligned}$$

is sufficient for ensuring that EDF-VD successfully schedules all LO-criticality behaviors of τ. ■

Algorithm EDF-VD thus chooses for x the smallest value such that Theorem 1 is satisfied:

$$x \leftarrow \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \tag{4}$$

With this value of x, we now determine a sufficient condition for ensuring that EDF-VD successfully meets all HI-criticality deadlines during all HI-criticality behaviors of τ:

Theorem 2: The following condition[1] is sufficient for ensuring that EDF-VD successfully schedules all HI-criticality behaviors of τ:

$$x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 \tag{6}$$

Proof: Suppose that τ satisfies Condition 3 but EDF-VD cannot meet all deadlines in all HI-criticality behaviors of τ. Let I denote a minimal instance of jobs released by τ, on which a deadline is missed. (By *minimal*, we mean that EDF-VD will meet all deadlines if scheduling any proper subset of I.) Without loss of generality, assume that the earliest job-release in I occurs at time zero, and let t_f denote the instant of the (first) deadline miss — since τ is assumed to satisfy Condition 3, this must be the deadline of a HI-criticality job, in a HI-criticality behavior. Let t^* denote the time-instant at which HI-criticality behavior is first flagged (i.e., the first instant at which some job executes for more than its LO-criticality worst-case execution time without signaling that it has completed execution).

We observe that all jobs in I, except perhaps the one that misses a deadline at t_f, experiences some execution; else, the job could be removed from I; this would contradict the assumed minimality of I.

We now introduce some **notation** for the remainder of this section:

1) For each i, $1 \leq i \leq n$, let η_i denote the amount of execution over the interval $[0, t_f]$ that is needed by jobs in I that are generated by task τ_i.
2) For each i, $1 \leq i \leq n$, let $u_i(\chi)$ denote the quantity $\frac{C_i(\chi)}{T_i}$. (That is, $u_i(\text{LO})$ denotes τ_i's LO-criticality utilization, and $u_i(\text{HI})$ denotes its HI-criticality utilization).
3) Let J_1 denote the job with the earliest release time amongst all those that execute in $[t^*, t_f)$. Let a_1 denote its release time, and d_1 its deadline.

Fact 1: All jobs that execute in $[t^*, t_f)$ have deadline $\leq t_f$.

Proof: Suppose not. Consider the latest instant t' in $[t^*, t_f)$ when a job with deadline $> t_f$ executes. Only those jobs in I that have release time $\geq t'$ and deadline $\leq t_f$ are sufficient to cause a deadline miss; this contradicts the assumed minimality of I. ■

Fact 2: Any LO-criticality task τ_i has

$$\eta_i \leq u_i(\text{LO})\big(a_1 + x(t_f - a_1)\big) \tag{7}$$

Proof: No LO-criticality job will execute after t^*. For it to execute after a_1, it must have a deadline no larger than J_1's virtual deadline, which is $(a_1 + x(d_1 - a_1))$. Therefore, no LO-criticality job with deadline $> (a_1 + x(t_f - a_1))$ will execute after a_1.

Suppose that some LO-criticality job with deadline $> (a_1 + x(t_f - a_1))$ were to execute, at some time $< a_1$. Let t' denote the latest instant at which any such job executes. This means that at this instant, there were no jobs with effective

[1] We note here that this theorem is one of the reasons that the results presented in this paper dominate the ones in [8]; the corresponding condition derived in [8] is

$$x + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 \tag{5}$$

Note that $U_{\text{LO}}^{\text{LO}}(\tau) \leq 1$ is a necessary condition for τ to be schedulable. It is evident that any schedulable system satisfying Condition 5 also satisfies Condition 6 while the converse is not true: there are task systems satisfying Condition 6 that violate Condition 5.

deadline $\leq (a_1 + x(t_f - a_1))$ awaiting execution. Hence the instance obtained by considering only those jobs in I that have release times $\geq t'$ also misses a deadline; this contradicts the assumed minimality of I. ■

Fact 3: Any HI-criticality task τ_i has

$$\eta_i \leq \frac{u_i(\text{LO})}{x} a_1 + (t_f - a_1) u_i(\text{HI}) \tag{8}$$

Proof: We consider separately the cases when τ_i does not have a job with release time $\geq a_1$, and when it does.

Case: If τ_i does not release a job at or after a_1. We claim that each job of τ_i has a modified deadline $\leq (a_1 + x(t_f - a_1))$. To see why this is so, consider some job with a modified deadline $> (a_1 + x(t_f - a_1))$, and let t' denote the latest instant at which this job executes. All jobs in I that have release times $\geq t'$ also miss a deadline; this contradicts the assumed minimality of I.

Since each job has a modified deadline $\leq (a_1 + x(t_f - a_1))$, their actual deadlines are all $\leq \frac{a_1}{x} + (t_f - a_1)$. Therefore, their cumulative execution requirement is at most

$$\begin{aligned} & \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{LO}) \\ & \quad \leq \quad \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \end{aligned}$$

Case: If τ_i releases a job at or after a_1. Let a_i denote the first release $\geq a_1$. The cumulative execution requirement of all jobs of τ_i is at most

$$\begin{aligned} & a_i u_i(\text{LO}) + (t_f - a_i) u_i(\text{HI}) \\ & \quad \leq \quad \text{(Since } a_1 \leq a_i \text{ and } u_i(\text{LO}) \leq u_i(\text{HI})) \\ & \qquad a_1 u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \\ & \quad \leq \quad \text{(Since } x \leq 1) \\ & \qquad \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \end{aligned}$$

■

Let us sum the cumulative demand of all the tasks over $[0, t_f)$:

$$\begin{aligned} & \sum_{\chi_i = \text{LO}} \eta_i + \sum_{\chi_i = \text{HI}} \eta_i \\ \leq \quad & \sum_{\chi_i = \text{LO}} u_i(\text{LO}) \big(a_1 + x(t_f - a_1)\big) \\ & + \sum_{\chi_i = \text{HI}} \frac{a_1}{x} u_i(\text{LO}) + (t_f - a_1) u_i(\text{HI}) \\ = \quad & a_1 \big(U_{\text{LO}}^{\text{LO}}(\tau) + \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{x}\big) \\ & + (t_f - a_1)(x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau)) \\ \leq \quad & \text{(By choice of } x \text{ [Eqn. 3], } (U_{\text{LO}}^{\text{LO}}(\tau) + \tfrac{U_{\text{HI}}^{\text{LO}}(\tau)}{x}) \leq 1) \\ & a_1 + (t_f - a_1)(x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau)) \end{aligned}$$

It follows from the infeasibility of this instance that

$$\begin{aligned} & a_1 + (t_f - a_1)(x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau)) > t_f \\ \Leftrightarrow \quad & (t_f - a_1)(x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau)) > t_f - a_1 \\ \Leftrightarrow \quad & x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) > 1 \end{aligned}$$

Taking the contrapositive, it follows that $(x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1)$ is sufficient to ensure HI-criticality schedulability by EDF-VD, as is claimed in this theorem. ■

We have thus established the correctness of Algorithm EDF-VD: by Theorem 1 the value assigned to x ensures the correctness of all LO-criticality behaviors whereas Theorem 2 guarantees the correct scheduling of all HI-criticality behaviors.

Observation. Note that Theorem 1 requires that $x \geq \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)}$, while Theorem 2 requires that $x \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau)}{U_{\text{LO}}^{\text{LO}}(\tau)}$. When these upper and lower bounds on x are not equal to each other, a pragmatic choice would be to choose a value for x that lies somewhere within the interval (e.g., at the midpoint), rather than at either of the boundaries – this would increase the robustness of the algorithm and its tolerance to, e.g., any arrival jitter.

V. Run-time dispatching

During the execution of the system, jobs are selected for execution according to the following rules:

1) There is a *criticality level indicator* Γ, initialized to LO.
2) While $(\Gamma \equiv \text{LO})$,
   a) Suppose a job of some task $\tau_i \in \tau$ arrives at time t
      - if $\chi_i \equiv$ LO, the job is assigned a scheduling deadline equal to $t + T_i$.
      - if $\chi_i \equiv$ HI, the job is assigned a scheduling deadline equal to $t + \hat{T}_i$.
   b) At each instant the waiting job with earliest scheduling deadline is selected for execution (ties broken arbitrarily).
   c) If the currently-executing job executes for more than its LO-criticality WCET without signaling completion, then the behavior of the system is no longer a LO-criticality behavior, and $\Gamma \leftarrow$ HI.
3) Once $(\Gamma \equiv \text{HI})$,
   a) The scheduling deadline of each HI-criticality job that is currently active is changed to its release time plus the unmodified period parameter (the T_i, not the $\hat{T}_i$) of the task that generated it. That is, if a job of τ_i that was released at some time t is active, its deadline, for scheduling purposes, is henceforth $t + T_i$.
   b) When a future job of τ_i arrives at some time t, it is assigned a scheduling deadline equal to $t + T_i$.
   c) LO-criticality jobs will *not* receive any further execution. Therefore at each instant the earliest-deadline waiting job generated by a HI-criticality task is selected for execution(ties broken arbitrarily).
4) An additional rule could specify the circumstances when Γ gets reset to LO. This could happen, for instance, if no HI-criticality jobs are active at some instant in time. (We will not discuss the process of resetting $\Gamma \leftarrow$ LO any further in this document, since this is not relevant to the certification process — LO-criticality certification assumes that the system *never* exhibits any HI-criticality behavior, while HI-criticality certification is not interested in the behavior of the LO-criticality tasks.)

A. An efficient implementation of run-time dispatching

For traditional (non-MC) sporadic task systems consisting of n tasks, uniprocessor EDF can be implemented efficiently

to have a run-time 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., [18]). A direct application of such implementations can be used to obtain an implementation of the run-time dispatching of EDF-VD that has a run-time of $\mathcal{O}(\log n)$ per job-arrival and job-completion event. However, EDF-VD potentially needs to deal with an additional run-time event: the change in the criticality level of the behavior from LO to HI (this is the event that is triggered at the instant that Γ gets assigned the value HI). Since this event requires that each subsequent scheduling be done according to each HI-criticality task's original deadline, explicitly recomputing priorities according to these original deadlines would take time linear in the number of HI-criticality tasks — in the worst case, $\mathcal{O}(n)$ time. We now describe an implementation of EDF-VD's run-time system that has a worst-case run-time 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 to HI.

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, [11, Sec. 6.5]). We maintain two such enhance 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 LO-criticality WCET (thereby triggering the assignment $\Gamma \leftarrow$ HI).

Initially, $\Gamma \equiv$ LO 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)* Γ being assigned the value HI. We consider each separately, below. 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 modified scheduling deadline.
   b) If $\chi_i =$ HI (i.e., if it is a HI-criticality job), then also insert it into Q_{HI}, prioritized according to its *un*modified (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 modified deadline than J_c's modified deadline. In that case, the newly-inserted job becomes J_c, and the timer is set to go off at $t_c + C_i(\text{LO})$ (when this newly-inserted job would exceed its LO-criticality WCET if allowed to execute without interruption).
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 HI-criticality job, 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 (modified) deadline job — the "minimum" job in Q_{LO}; and set the timer to go off at t_c+ this job's remaining LO-criticality WCET (when the job would exceed its LO-criticality WCET if allowed to execute without interruption).
3) The timer goes off, indicating that the currently-executing job has executed beyond its LO-criticality WCET without signaling completion. The system is therefore now in HI-criticality mode, and we switch to scheduling according to Q_{HI}. Henceforth, all run-time dispatch decisions are taken as indicated by this priority queue.

After Γ becomes HI, no LO-criticality jobs need execute, and HI-criticality jobs are executed according to EDF with their original (unmodified) deadlines. Hence subsequent run-time dispatching is done as for traditional EDF scheduling (as described in, e.g., [18]), with Q_{HI} being the priority queue used for this purpose.

VI. Some properties of EDF-VD

A. Comparison with *worst-case reservations*

Under the *worst-case reservations* strategy that is widely used in the design of mixed-criticality systems, computing capacity is provisioned to each task at its own criticality level. That is, the MC task system τ is mapped on to the traditional (non-MC) task system

$$\bigcup_{\tau_i \in \tau} \Big\{ \big(C_i(\chi_i), T_i\big) \Big\}$$

and scheduled using regular EDF. It directly follows from the utilization-bound result of EDF [16] that the condition

$$U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 \tag{9}$$

is necessary and sufficient for ensuring that EDF can schedule τ to meet all deadlines, provided each LO-criticality job executes for up to its LO-criticality WCET and each HI-criticality job executes for up to its HI-criticality WCET. This covers all LO-criticality and all HI-criticality behaviors of τ.

We now show that Algorithm EDF-VD strictly dominates the worst-case reservations approach: any task system that can be scheduled using worst-case reservations can be scheduled by EDF-VD.

Theorem 3: Any task system τ that is correctly scheduled using worst-case reservations is also correctly scheduled by EDF-VD.

Proof: Any task system that can be scheduled using worst-case reservations satisfies Condition 9 above.

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 \\ \Rightarrow \quad & \text{(Since } U_{\text{HI}}^{\text{LO}}(\tau) \leq U_{\text{HI}}^{\text{HI}}(\tau)\text{)} \\ & U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq 1 \\ \Rightarrow \quad & \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq 1 \end{aligned}$$

From this and Equation 4, we conclude that x is assigned a value ≤ 1 by Algorithm EDF-VD.

Now, Theorem 2 contains the following sufficient condition for EDF-VD schedulability:

$$x U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 \ ,$$

which always holds since Condition 9 holds and $x \leq 1$. ■

B. Task systems τ with $U_{\text{HI}}^{\text{LO}}(\tau) = 0$

It is interesting to analyze the manner in which EDF-VD deals with task systems in which the LO-criticality WCET of each HI-criticality task is equal to zero. (This would be the case for systems in which a "mode change" can be thought to occur when the high-criticality behavior is triggered.)

For such a system τ, observe that $U_{\text{HI}}^{\text{LO}}(\tau) = 0$. Therefore, the value assigned to the scaling parameter x in Step 1 of Figure 1 is equal to zero, and the test in Step 2 of Figure 1 evaluates to true to all τ with $U_{\text{HI}}^{\text{HI}}(\tau) \leq 1$. Thus EDF-VD can schedule any task system for which $U_{\text{HI}}^{\text{LO}}(\tau) = 0$ that satisfies the condition

$$U_{\text{LO}}^{\text{LO}}(\tau) \leq 1 \text{ and } U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 \ .$$

I.e., EDF-VD can schedule any such system provided the LO-criticality and the HI-criticality behaviors are separately schedulable.

Continuing to analyze the pseudo-code in Figure 1 for this special case, we observe that EDF-VD assigns each HI-criticality task τ_i a modified period $\hat{T}_i$ equal to zero. Thus during run-time each job of a HI-criticality task immediately becomes an earliest-deadline (and hence highest-priority) one upon arrival. If it is discovered to have a non-zero execution time, the criticality-level indicator is immediately assigned the value HI (i.e., $\Gamma \leftarrow$ HI), and all LO-criticality jobs are immediately discarded.

VII. Speedup bounds

The *speedup factor* of an algorithm A for scheduling mixed-criticality systems is defined to be the smallest real number f such that any task system τ that is schedulable on a unit-speed processor by a hypothetical optimal clairvoyant[2] algorithm is successfully scheduled on a speed-f processor by algorithm A. 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 optimal algorithm.

Theorem 4: The speedup factor of EDF-VD is $\leq \frac{4}{3}$.

Proof: To prove this theorem, we will show that any MC implicit-deadline sporadic task system that is clairvoyantly schedulable on a speed-$\frac{3}{4}$ processor is schedulable by EDF-VD on a unit-speed processor.

Let b denote an upper bound on both the LO-criticality utilization and the HI-criticality utilization of task system τ:

$$b \geq \max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) \tag{10}$$

[2]Informally, a clairvoyant algorithm for scheduling MC systems is one that knows prior to run-time whether the system is going to exhibit LO-criticality or HI-criticality behavior.

By Theorems 1 and 2, we know that if an x satisfying both theorems exists, there will be no deadline miss. Since Theorem 1 requires that

$$\frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq x$$

while Theorem 2 requires that

$$x \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau)}{U_{\text{LO}}^{\text{LO}}(\tau)} \ ,$$

we can derive Expression 11 below as a sufficient condition for τ to be successfully scheduled using EDF-VD:

$$\begin{aligned}
& \frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau)}{U_{\text{LO}}^{\text{LO}}(\tau)} \\
\Leftarrow \quad & \text{(Since } U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq b \Rightarrow U_{\text{HI}}^{\text{LO}}(\tau) \leq b - U_{\text{LO}}^{\text{LO}}(\tau)) \\
& \frac{b - U_{\text{LO}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau)}{U_{\text{LO}}^{\text{LO}}(\tau)} \\
\Leftarrow \quad & \text{(Since } U_{\text{HI}}^{\text{HI}}(\tau) \leq b\text{)} \\
& \frac{b - U_{\text{LO}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq \frac{1 - b}{U_{\text{LO}}^{\text{LO}}(\tau)} \\
\Leftrightarrow \quad & \\
& \Big(U_{\text{LO}}^{\text{LO}}(\tau)\Big)^2 - U_{\text{LO}}^{\text{LO}}(\tau) + (1 - b) \geq 0
\end{aligned} \tag{11}$$

Now if we set $b \leftarrow \frac{3}{4}$, Expression 11 becomes

$$\begin{aligned}
& \Big(U_{\text{LO}}^{\text{LO}}(\tau)\Big)^2 - U_{\text{LO}}^{\text{LO}}(\tau) + \frac{1}{4} \geq 0 \\
\Leftrightarrow \quad & \\
& \Big(U_{\text{LO}}^{\text{LO}}(\tau) - \frac{1}{2}\Big)^2 \geq 0
\end{aligned}$$

which is true for all values of $U_{\text{LO}}^{\text{LO}}(\tau)$.

We have thus shown that any task system that is clairvoyant schedulable on a speed-$\frac{3}{4}$ processor is scheduled by EDF-VD to meet all deadlines on a unit-speed processor. It therefore follows that any task system that is clairvoyant schedulable on a unit-speed processor is scheduled by EDF-VD to meet all deadlines on a speed-$\frac{4}{3}$ processor, as claimed by this theorem. ■

We now show that EDF-VD is optimal with regard to speedup factor:

Theorem 5: No non-clairvoyant algorithm for scheduling dual-criticality implicit-deadline sporadic task systems can have a speedup bound better than $\frac{4}{3}$.

Proof: Consider the example task system $\tau = \{\tau_1, \tau_2\}$, with the following parameters, where ϵ is an arbitrarily small number > 0:

τ_i	χ_i	C_i(LO)	C_i(HI)	T_i
τ_1	LO	$1 + \epsilon$	$1 + \epsilon$	2
τ_2	HI	$1 + \epsilon$	3	4

This system is schedulable by a clairvoyant scheduler: EDF would meet all deadlines in LO-criticality behaviors (since $U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq 1$), while only jobs of τ_2 would get to execute in HI-criticality behaviors.

To see that τ cannot be scheduled correctly by an on-line 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

$U_{\text{HI}}^{\text{LO}}(\tau)$	$U_{\text{LO}}^{\text{LO}}(\tau)$ 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	0.98	0.97	0.9	0.93	0.90	0.85	0.76	0.60	0.10
0.2	0.97	0.95	0.914	0.86	0.80	**0.70**	0.53	0.20	
0.3	0.96	0.92	0.87	0.80	0.70	0.55	0.30		
0.4	0.95	0.90	0.82	0.73	0.60	0.40			
0.5	0.94	0.87	0.78	0.66	0.50				
0.6	0.93	0.85	0.74	0.60					
0.7	0.9	0.82	0.70						
0.8	0.911	0.80							
0.9	0.90								

Table I
UTILIZATION BOUNDS: UPPER BOUND ON $U_{\text{HI}}^{\text{HI}}(\tau)$, FOR GIVEN VALUES OF $U_{\text{LO}}^{\text{LO}}(\tau)$ AND $U_{\text{HI}}^{\text{LO}}(\tau)$

the behavior is a LO-criticality or a HI-criticality one. 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 behavior is revealed to be a HI-criticality one. 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 behavior is revealed to be a LO-criticality one, in that τ_2's job signals that it has completed execution. But 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 $\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big)$ exceeds $3/4$ by an arbitrarily small amount. ■

VIII. UTILIZATION BOUNDS

It is evident that in order for task system τ to be schedulable by any algorithm on a unit-speed processor, it is *necessary* that both

$$U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq 1, \text{ and}$$

$$U_{\text{HI}}^{\text{HI}}(\tau) \leq 1.$$

We now derive *sufficient* conditions on $U_{\text{LO}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{LO}}(\tau)$ and $U_{\text{HI}}^{\text{HI}}(\tau)$ for the system to be successfully scheduled by EDF-VD.

As we had observed in Section VII above, it follows from Theorems 1 and 2 that if an x satisfying both theorems exists, then EDF-VD will schedule the system correctly. Since Theorem 1 requires that

$$\frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq x$$

while Theorem 2 requires that

$$x \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau)}{U_{\text{LO}}^{\text{LO}}(\tau)},$$

we have

$$\begin{aligned}
&\frac{U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \leq \frac{1 - U_{\text{HI}}^{\text{HI}}(\tau)}{U_{\text{LO}}^{\text{LO}}(\tau)} \\
&\Leftrightarrow \quad U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{HI}}(\tau) - U_{\text{LO}}^{\text{LO}}(\tau)\Big(U_{\text{HI}}^{\text{HI}}(\tau) - U_{\text{HI}}^{\text{LO}}(\tau)\Big) \leq 1 \\
&\Leftrightarrow \quad U_{\text{HI}}^{\text{HI}}(\tau)\big(1 - U_{\text{LO}}^{\text{LO}}(\tau)\big) \leq 1 - U_{\text{LO}}^{\text{LO}}(\tau) - U_{\text{LO}}^{\text{LO}}(\tau)U_{\text{HI}}^{\text{LO}}(\tau) \\
&\Leftrightarrow \quad U_{\text{HI}}^{\text{HI}}(\tau) \leq \frac{1 - U_{\text{LO}}^{\text{LO}}(\tau) - U_{\text{LO}}^{\text{LO}}(\tau)U_{\text{HI}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)} \\
&\Leftrightarrow \quad U_{\text{HI}}^{\text{HI}}(\tau) \leq 1 - U_{\text{HI}}^{\text{LO}}(\tau)\Big(\frac{U_{\text{LO}}^{\text{LO}}(\tau)}{1 - U_{\text{LO}}^{\text{LO}}(\tau)}\Big) \qquad (12)
\end{aligned}$$

as a sufficient condition for τ to be schedulable by EDF-VD. In Table I we list this upper bound on $U_{\text{HI}}^{\text{HI}}(\tau)$ for task systems τ with specified $U_{\text{LO}}^{\text{LO}}(\tau)$ and $U_{\text{HI}}^{\text{LO}}(\tau)$. Since the schedulability of LO-criticality behaviors of τ requires that $U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau) \leq 1$, bounds are only computed when this condition is satisfied.

We note that the model assumption that $C_i(\text{LO}) \leq C_i(\text{HI})$ for all tasks mandates that $U_{\text{HI}}^{\text{LO}}(\tau) \leq U_{\text{HI}}^{\text{HI}}(\tau)$; thus the range of values that $U_{\text{HI}}^{\text{HI}}(\tau)$ may take for given values of $U_{\text{LO}}^{\text{LO}}(\tau)$ and $U_{\text{HI}}^{\text{LO}}(\tau)$ is bounded from below by the value of $U_{\text{HI}}^{\text{LO}}(\tau)$, and from above by the table entry (as computed according to Expression 12). Hence for example if we have a system τ with $U_{\text{LO}}^{\text{LO}}(\tau) = 0.6$ and $U_{\text{HI}}^{\text{LO}}(\tau) = 0.2$, then $U_{\text{HI}}^{\text{HI}}(\tau)$ may take on any value in $[0.2, 0.7]$ (this entry is highlighted in bold font in Table I).

IX. EVALUATION VIA SIMULATION

We have conducted a series of simulation experiments to evaluate the effectiveness of EDF-VD in finding certifiably correct scheduling strategies. Our experiments were conducted upon randomly-generated task systems that were generated according to (a slight modification of) the workload-generation algorithm introduced by Guan et al. [13]. The input parameters for our workload generation algorithm are as follows:

- U_{bound}: The desired value of the larger of LO-criticality and HI-criticality utilization of the task system:
$$\max\Big(U_{\text{LO}}^{\text{LO}}(\tau) + U_{\text{HI}}^{\text{LO}}(\tau), U_{\text{HI}}^{\text{HI}}(\tau)\Big) = U_{\text{bound}} \qquad (13)$$
- $[U_L, U_U]$: Utilizations are uniformly drawn from this range; $0 \leq U_L \leq U_U \leq 1$.
- $[Z_L, Z_U]$: The ratio of the HI-criticality utilization of a task to its LO-criticality utilization is uniformly drawn from this range; $1 \leq Z_L \leq Z_U$.
- P: The probability that a task is a HI-criticality task; $0 \leq P \leq 1$.

Gives these parameters, the task-generation algorithm initializes the task system τ to be empty and repeatedly adds tasks τ_i, $i = 1, 2, \ldots$, until the utilization bound is met. (For further detail please see [13].)

In our experiments, we determined the fraction of randomly-generated task systems that are deemed to be schedulable by the algorithm under consideration, as a function of the system utilization U_{bound}. Some of our results are depicted graphically in Figures 2-5. In each

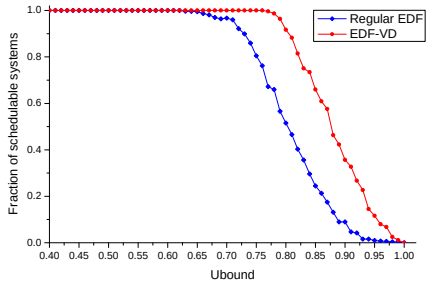

Figure 2. $U_L = 0.02, U_U = 0.2, Z_L = 1, Z_U = 2, P = 0.5$

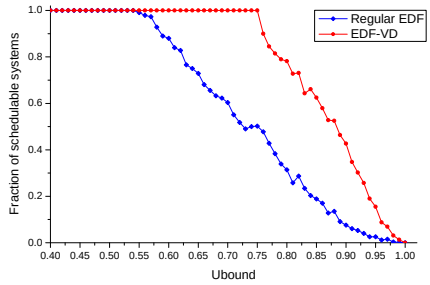

Figure 4. $U_L = 0.02, U_U = 0.2, Z_L = 1, Z_U = 8, P = 0.5$

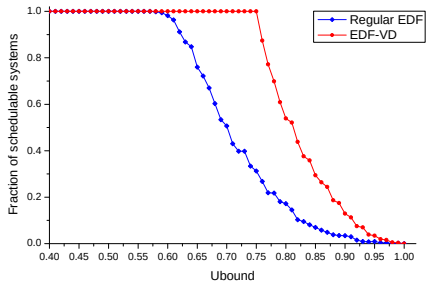

Figure 3. $U_L = 0.02, U_U = 0.2, Z_L = 1, Z_U = 4, P = 0.5$

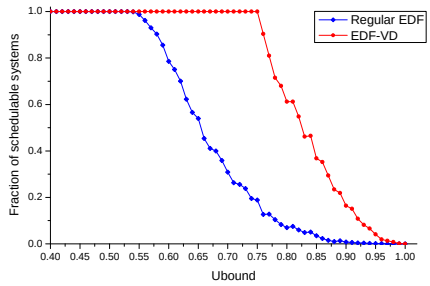

Figure 5. $U_L = 0.02, U_U = 0.2, Z_L = 1, Z_U = 8, P = 0.3$

graph, the fraction of systems that were determined to be schedulable is depicted on the y-axis, and the utilization on the x-axis. Each data-point was obtained by randomly generating 1000 task systems, testing each for schedulability according to the appropriate algorithm, and calculating the fraction of systems deemed schedulable. The parameters used in generating these task systems (other than the system utilization, which is depicted on the x-axis) are provided in the caption of the graph; e.g., the task systems for Figure 2 were generated using the parameters $U_L = 0.02$, $U_U = 0.2$, $Z_L = 1$, $Z_U = 2$, and $P = 0.5$. For each set of parameters, two different algorithms were compared: regular EDF (i.e., EDF on the task system $\bigcup_i \{(C_i(\chi_i), T_i)\}$)– this is the "worst-case reservations" strategy discussed in VI); and Algorithm EDF-VD (as depicted in Figure 1).

It is evident from the graphs that EDF-VD consistently exhibits noticeable improvement over a simple EDF scheduler. We do not seek to make quantitative claims about the degree of such improvement based on simulation data, but we do notice some trends.

1) When the system utilization is smaller than 0.5, the task system is always schedulable by worst-case reservations (regular EDF); when the system utilization is smaller than 0.75, the task system is always schedulable by EDF-VD. These observation match the known utilization bounds.
2) The improvement becomes more significant as the ratio of the HI-criticality WCET to LO-criticality WCET increases (compare Figures 2-4). The intuitive explanation for this is that EDF-VD takes more advantage from processing LO-criticality and HI-criticality behaviors separately when the LO-criticality and HI-criticality behaviors overlap less.
3) For a similar reason, the improvement becomes more pronounced as the ratio of the HI-criticality utilization ($U_{\text{HI}}^{\text{HI}}(\tau)$) to the LO-criticality utilization ($U_{\text{LO}}^{\text{LO}}(\tau)+U_{\text{HI}}^{\text{LO}}(\tau)$) approaches 1 (compare Figures 4-**??**, in which the ratios are respectively 1.6, 1.06 and 0.34).

X. Summary and Conclusions

Devising more cost-efficient techniques for obtaining certication for safety-critical embedded systems has been identified as a prime research challenge [1]. We believe that in mixed-criticality systems, these certication considerations give rise to fundamental new resource allocation and scheduling challenges that are not adequately addressed by conventional real-time scheduling theory. In this paper, we consider the scheduling, upon preemptive uniprocessors, of mixed-criticality systems that can be modeled using a mixed-criticality generalization of the implicit-deadline sporadic tasks model. An algorithm called EDF-VD was proposed in [8] 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 $4/3$ (Theorem 4); and

(ii) no non-clairvoyant algorithm can have a smaller speedup factor (Theorem 5). The result in Theorem 4 improves on an earlier result presented in [8], which had shown a speedup factor of $(\sqrt{5}+1)/2 \approx 1.62$. This improved speedup factor was obtained by first deriving a superior sufficient schedulability condition (Theorem 2); this sufficient schedulability condition strictly dominates earlier sufficient schedulability tests in that any task system deemed schedulable by these earlier tests is also deemed schedulable by the test of Theorem 2 while the converse of this statement is not true.

As further contributions, we have shown how EDF-VD can be implemented to have a run-time complexity per scheduling decision that is logarithmic in the number of tasks, and thus demonstrated the practical applicability of the algorithm. We have explored further properties of EDF-VD, and have conducted extensive simulation experiments to reveal its behavior on randomly-generated task systems.

REFERENCES

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

[2] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 159–168, Rio de Janeiro, December 2006. IEEE Computer Society Press.

[3] S. Baruah, A. Burns, and R. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[4] S. Baruah, H. Li, and L. Stougie. Mixed-criticality scheduling: improved resource-augmentation results. In *Proceedings of the ICSA International Conference on Computers and their Applications (CATA)*. IEEE, April 2010.

[5] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)*. IEEE, April 2010.

[6] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*. To appear.

[7] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. In P. Hlinený and A. Kucera, editors, *Proceedings of the 35th International Symposium on the Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010.

[8] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proceedings of the 19th Annual European Symposium on Algorithms*, pages 555–566, Saarbrucken, Germany, September 2011. Springer-Verlag.

[9] A. Burns and S. Baruah. Timing faults and mixed criticality systems. In Jones and Lloyd, editors, *Dependable and Historic Computing*, volume LNCS 6875, pages 147–166. Springer, 2011.

[10] A. Burns and B. Littlewood. Reasoning about the reliability of multi-version, diverse real-time systems. In *Proceedings of 31st IEEE Real-Time Systems Symposium*, pages 73–81. IEEE Computer Society Press, December 2010.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, third edition, 2009.

[12] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling for certifiable mixed criticality sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[13] N. Guan and W. Yi. Improving the scheduling of certifiable mixed criticality sporadic task systems. Technical report, Uppsala University, 2012.

[14] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[15] H. Li and S. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proceedings of the Real-Time Systems Symposium*, pages 183–192, San Diego, CA, 2010. IEEE Computer Society Press.

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

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

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

[19] P.J.Prisaznuk. Integrated modular avionics. In *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference (NAECON 1992)*, volume 1, pages 39–45, May 1992.

[20] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the Real-Time Systems Symposium*, pages 239–243, Tucson, AZ, December 2007. IEEE Computer Society Press.