IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 12, 2021, accepted March 8, 2021, date of publication March 15, 2021, date of current version March 24, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3066100

Service Improvements in Real-Time Uniprocessor Scheduling With Single Errors

ROBERT SCHMIDT, (Graduate Student Member, IEEE), AND ALBERTO GARCÍA-ORTIZ, (Senior Member, IEEE)
Institute of Electrodynamics and Microelectronics, University of Bremen, 28359 Bremen, Germany
Corresponding author: Robert Schmidt (rschmidt@uni-bremen.de)

ABSTRACT Mixed-criticality scheduling in modern deeply embedded mission and safety-critical systems needs to consider delivered service, that is, the runtime in low criticality mode. If the change into a higher criticality mode is triggered by the first overrunning job, the service is severely reduced. With earliest deadline first with virtual deadlines for single errors (EDF-VD-SE) we show how to reserve additional time to tolerate a single overrunning job by formulating and solving an optimization problem, and that EDF-VD-SE is feasible without assumptions about error probabilities for safety guarantees. We conduct extensive simulation experiments to report on average doubled service figures, and show how EDF-VD-SE results in a nearly constant acceptance rate of random task systems.

INDEX TERMS Real-time systems, scheduling algorithms, system recovery.

I. INTRODUCTION

To meet non-functional requirements of embedded systems in the avionics and automotive sector, components of different criticality, or required level of assurance against failure, are integrated onto a common hardware platform [1], [2], resulting in mixed-criticality systems [3].

For mixed-criticality real-time control systems, like on-board flight computers, a criticality is assigned to each task, or reoccurring computation. Computations, or jobs, need to finish prior to their deadline to meet the timing requirements of the system, and share the processor with other jobs. If job arrival times and their execution times are not known in advance, it is to decide during system run-time on a schedule when jobs are allowed to access the processor. This makes schedulability verification prior to system deployment a necessity. For schedulability verification, jobs' worst case execution times (WCETs) are modeled as constant upper bounds, which are impossible or hard to derive in a safe and precise manner [4], and tend to increase with the criticality [5]. In specifying multiple estimates for a job's execution time, mixed-criticality scheduling approaches are able to provide different levels of assurance, increasing the chance to verify the schedulability of the system.

For a preemptive uniprocessor, scheduling jobs from independent, sporadic, implicit deadline tasks, mixed-criticality scheduling approaches like earliest deadline first with virtual deadlines (EDF-VD) guarantee that all high criticality tasks finish prior to their deadline, irrespective of the low criticality tasks [6]. This guarantee is achieved among other things by separating the run-time in modes, where change from the initial low criticality mode into the higher criticality mode, executing only high criticality tasks, is triggered by a high criticality job executing longer than accounted for. But abandoning low criticality tasks, or rather the service they implement, is not sensible [7], [8] if the system is expected to experience an actual mode change [9]. Especially in modern fault-tolerant hard real-time systems, where scheduling is not deterministic [10], mode change needs to be considered. Therefore holistic approaches need to go beyond scheduling guarantees for high criticality tasks, and strive to provide as much service as possible to low criticality tasks as well.

To extend the service of low criticality tasks, the change from low to high criticality mode should be delayed as long as possible [11]. Changing modes is tied to a particular event which is integral to the approach's schedulability check. For approaches like EDF-VD, the event is the overrun of a single high criticality job, which happens if the job's computation takes longer than anticipated by its low criticality budget.

We propose to reserve additional time during system design to accommodate for a single overrun, which results in a delayed or even prevented change into high criticality mode. With our approach, called EDF-VD-SE, we show

The associate editor coordinating the review of this manuscript and approving it for publication was Laxmisha Rai.

- how to find and reserve additional time by virtual deadlines using sequential least squares programming (SLSQP) (Section IV-C);
- that our adaptive approach results in a nearly constant acceptance rate of random task systems (Section V-A); and
- how this approach improves the quality of service by effectively doubling the run-time on average in low criticality mode (Section V-B).

II. PRELIMINARIES

In the following subsections we introduce the reader to our notation, and the basics of earliest deadline first (EDF) scheduling for mixed-criticality task systems. Once major terms are defined, we review how task systems are randomly generated in a controlled manner, which is important to ensure reproducible experiments.

A. NOTATION

We consider independent, sporadic, implicit deadline dual-criticality task systems, scheduled on a preemptive uniprocessor. The processor is the only shared resource, and the overhead of scheduler operation and context switch can be bounded by a constant. We adopt the standard dual-criticality task system model [9], [12], [13]: Each task τ_i in a dual-criticality sporadic task set $\tau = \{\tau_1, \ldots, \tau_n\}$ is characterized by

- a criticality level $\chi_i \in \{1, 2\}$, often referred to by symbolic names low $L = 1$ and high $H = 2$;
- execution time budgets in both criticality modes $c_i(1), c_i(2)$;
- a relative deadline d_i of the jobs of τ_i; and
- the minimum interarrival time, or period p_i between two jobs of τ_i.

Execution time budgets are estimates of the time it takes to execute a task's job on the uniprocessor. They summarize the distribution of actual execution times by an upper limit. With multiple execution time budgets the distribution is sliced into multiple parts, which allows to differentiate an average case, and further pessimistic cases. In dual-criticality tasks we use up to two execution time budgets $c_i(1), c_i(2)$ to differentiate the average- and worst case. A constant is added to all execution time budgets to account for scheduler and context switch overhead.

Furthermore, the execution time budgets, deadlines, and periods are constrained to obtain an implicit deadline system:

$$\forall \tau_i \in \tau : c_i(1) \leq c_i(2) \leq d_i = p_i \tag{1}$$

Tasks generate an unbounded sequence of jobs, where jobs are characterized by their arrival time α_{ij}, actual execution time γ_{ij}, and absolute deadline $d_{ij} = \alpha_{ij} + d_i$. At run-time, high criticality jobs might execute longer than $c_i(1)$, but never exceed their WCET $c_i(2)$. Low criticality jobs are constrained to never execute longer than $c_i(1)$. The task system is mixed-criticality schedulable if every high criticality job can receive execution time γ_{ij} during $[\alpha_{ij}, d_{ij})$, to meet their deadline [14].

At system level k a single task τ_i has a utilization of $u_i(k) = c_i(k)/p_i$. We can summarize the utilization in low- and high criticality mode for a dual-criticality system with $K = 2$ as follows [14]:

$$U_l(k) \equiv \sum_{i:\chi_i=l} \frac{c_i(k)}{p_i} \qquad l = 1, \ldots, K, \quad k = 1, \ldots, l \tag{2}$$

For the reader's convenience we refer to the criticality levels by their symbolic names with the following notation: $U_1(1) = U_L$, $U_2(1) = U_H^L$, $U_2(2) = U_H^H$. Moreover for a *low* criticality task τ_i the WCET is c_i and the task's utilization is u_i. For *high* criticality tasks τ_i the WCETs are c_i^L, c_i^H and the task's utilizations are u_i^L, u_i^H.

B. EARLIEST DEADLINE FIRST WITH VIRTUAL DEADLINES

EDF-VD is a preemptive uniprocessor dynamic scheduling approach for mixed-criticality task systems. It is an extension of EDF scheduling to mixed-criticality task systems by introducing earlier, virtual deadlines. As in EDF, the priority of a job is defined by its deadline [15]. The closer the deadline, the higher the priority of the job. The job with the highest priority is granted access to the processor, and can preempt a currently running job, which returns to the scheduler's job queue.

In contrast to EDF scheduling, EDF-VD introduces the concepts of criticalities and modes, which separates the system's run-time into two modes: In the first mode, jobs from all tasks are scheduled. Once a job from a high criticality task requires more computation than anticipated ($\gamma_{ij} > c_i^L$), the system changes to the second mode, where only jobs from high criticality tasks are scheduled.

To guarantee that all high criticality jobs have enough time to finish prior to their deadlines, EDF-VD introduces earlier, virtual deadlines which reserve time necessary for a successful mode change. These earlier deadlines increase the priority of the task's jobs. Once the switch to high criticality mode is triggered, jobs from high criticality tasks are scheduled with their original deadline.

We can interpret the scheduling in both modes as regular EDF scheduling with different sets of tasks. If both modes satisfy the constraints of regular EDF scheduling, with the transitional phase taken care of by earlier deadlines, the system is guaranteed to never miss a deadline [14].

C. RANDOM TASK SYSTEM GENERATION

It is interesting to know which, or how many, task systems are deemed schedulable by a given algorithm. With a fixed set of task systems, the fraction of task systems deemed schedulable can be compared between algorithms as an indication of their practical applicability. If an algorithm is able to schedule a larger fraction of task systems, it seems more practical, because the likelihood of scheduling a specific task system increases. Comparing algorithms in this regard requires a large set of task systems, hence randomly-generated synthetic task systems are used.

Require: Utilization U to distribute over n tasks
Ensure: Uniform distributed utilizations u_i
1: $u \leftarrow \emptyset$
2: $s \leftarrow U$
3: **for** $j \leftarrow [0, n-1) \cap \mathbb{Z}$ **do**
4: $\quad v \leftarrow \mathcal{U}(0,1)$
5: $\quad i \leftarrow n - (j+1)$
6: $\quad s_{\text{next}} = s \cdot v^{1/i}$
7: $\quad u \leftarrow u \cup \{s - s_{\text{next}}\}$
8: $\quad s \leftarrow s_{\text{next}}$
9: **end for**
10: $u \leftarrow u \cup \{s\}$

FIGURE 1. `UUnifast` **algorithm [16]. The algorithm computes each task's utilization u_i iteratively by nested splitting of U between i tasks. Each iteration, a utilization u_i is calculated by inverse transform sampling a CDF of the sum of i uniform random variables limited to the iteratively decreasing remaining utilization s.**

The random generation is controlled by several parameters, which can affect results and bias the judgment about algorithms [16]. A major parameter is the task system's utilization U, because the difficulty increases with U. The distribution of U over all tasks in a system needs to be controlled to avoid any bias. It is favorable to have uniform distributed task utilization u_i; the `UUnifast` algorithm provides a controlled way of distributing U over n tasks in a system [16].

To calculate uniformly distributed task utilizations u_i in linear time the `UUnifast` algorithm splits U iteratively into n slices, as shown in Fig. 2. Given a CDF $F_i(x)$ of the sum of i independent uniform distributed random variables and their sum limited to s, the algorithm draws a random value $x \leq s$ by inverse transform sampling from $F_i(x)$. The difference $s-x$ is the remaining utilization which needs to be split up between $n-1$ tasks. This procedure, formalized in Fig. 1, is repeated for the remaining $n-1$ tasks.

The CDF $F_i(x)$ is derived from the probability density function (PDF) $f_i(x)$ of the sum of independent, uniformly distributed random variables in $[0, b]$:

$$f_i(x) = f_{\mathcal{U}_1+\mathcal{U}_2+\cdots+\mathcal{U}_i}(x) = \left(f_{\mathcal{U}_1} * f_{\mathcal{U}_2} * \cdots * f_{\mathcal{U}_i}\right)(x) \quad (3)$$

$$f_i(x) = \begin{cases} \frac{1}{b}x^{i-1}, & x \in [0,b] \\ 0, & \text{else} \end{cases} \quad (4)$$

$$F_i(x) = \begin{cases} 0, & x \leq 0 \\ \left(\frac{x}{b}\right)^i, & 0 < x \leq b \\ 1, & x > b \end{cases} \quad (5)$$

The sampling of random values from $F_i(x)$ requires the inverse function $F^{-1}(x)$, which is used in line six of the `UUnifast` algorithm in Fig. 1 in case $0 < x \leq b$:

$$F_i\left(F^{-1}(x)\right) = x \quad (6)$$

$$\left(\frac{F^{-1}(x)}{b}\right)^i = x \quad (7)$$

$$F^{-1}(x) = b\sqrt[i]{x} = bu^{1/i} \quad (8)$$

TABLE 1. **Example of executing** `UUnifast` **algorithm for $U = 1$ and $n = 4$.**

j	i	$s(j)$	$s_{\text{next}}(j)$	u_j
0	3	U	0.757	0.243
1	2	0.757	0.652	0.105
2	1	0.652	0.253	0.399
-	-	-	-	0.253

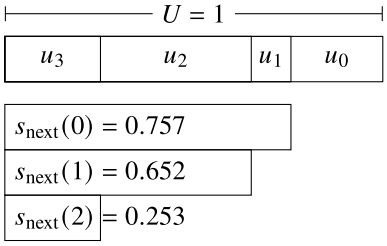

FIGURE 2. **Example slicing of $U = 1$ by the** `UUnifast` **algorithm with $n = 4$ tasks.**

As an example consider $U = 1$ and $n = 4$, which can result in $u = (0.243, 0.105, 0.399, 0.253)$ as shown in Table 1. The slices are iteratively calculated, where $v \in \mathcal{U}(0,1)$ is a uniform distributed random variable:

$$s(0) = U \quad s_{\text{next}}(0) = U \cdot v^{1/3} \quad (9)$$

$$s(1) = s_{\text{next}}(0) \quad s_{\text{next}}(1) = s_{\text{next}}(0) \cdot v^{1/2} \quad (10)$$

$$s(2) = s_{\text{next}}(1) \quad s_{\text{next}}(2) = s_{\text{next}}(1) \cdot v \quad (11)$$

The task's actual utilizations are the differences between two slices, except the last one:

$$u_0 = s(0) - s_{\text{next}}(0) \quad (12)$$

$$u_1 = s(1) - s_{\text{next}}(1) \quad (13)$$

$$u_2 = s(2) - s_{\text{next}}(2) \quad (14)$$

$$u_3 = s_{\text{next}}(2) \quad (15)$$

The remaining parameters in the task system generation are the limits of uniform random distributions for the task's period, deadline-to-period ratio, and factor between low- and high level utilization. Usually one can assume task periods within typical ranges, for example between $p_l = 200\,\text{ms}$ and $p_h = 5000\,\text{ms}$ depending on the kind of application. A uniform distribution $\mathcal{U}(p_l, p_u)$ is justified for general comparisons of algorithms, but other distributions can be considered if the applications of interest have a bias towards shorter or larger periods in a task system. For non-implicit-deadline tasks the deadline of each task is a multiple of the period, $d_i = \epsilon_i p_i$ with a deadline-to-period factor drawn uniformly $\epsilon_i \in \mathcal{U}(\epsilon_l, \epsilon_u)$. Implicit-deadline task systems are generated with $\epsilon_l = \epsilon_h = 1$. The execution time budget of each task is the product of the task's utilization u_i and period $p_i \in \mathcal{U}(p_l, p_u)$: $c_L = u_i p_i$.

For our experiments we generate synthetic task systems, and convert them into dual-criticality task systems. A uniformly distributed factor $z_i \in \mathcal{U}(z_l, z_u)$ determines the higher level execution time budget $c_H = z_i c_L$ [17], if by random chance of $p = 0.5$ the task is of high criticality. The factor z_i describes the pessimism in estimating the task's WCET

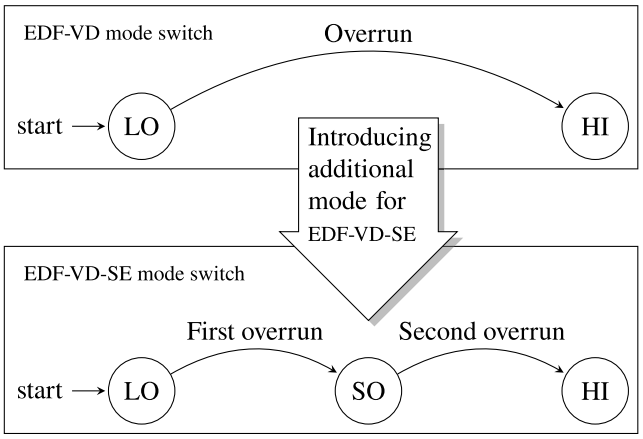

FIGURE 3. Mode switch in EDF-VD and EDF-VD-SE. With EDF-VD-SE we can tolerate a single overrun without abandoning low criticality tasks. With the second overrun, low criticality tasks are abandoned in favor of high criticality tasks. EDF-VD represents the mode switch behavior of classic dual-criticality mode switch schemes. Compared to classic dual-criticality mode switching, EDF-VD-SE has an intermediate mode labeled "SO" (single overrun).

with c_H. Moreover, by selecting $z_l = z_h = z \in \{x \in \mathbb{Z} : x > 1\}$ we can model different software error correction approaches where jobs are repeated, which we describe in more detail in Section IV. Resulting dual-criticality task systems have different utilizations U_L, U_H in low- and high criticality mode, with $U_H > U_L$ due to increased pessimism in estimating the task's WCET. During random generation, the utilization in low criticality mode is chosen, which influences the utilization of low criticality tasks in low criticality mode U_L^L and of high criticality task in low criticality mode U_L^H:

$$U = U_L = U_L^L + U_H^L \tag{16}$$

This parameterized random generation approach is used to reproduce task system sets used in the literature and allows to investigate a schedulability test under explicit parameters and task system properties.

III. OVERVIEW OF EDF-VD-SE

The general idea of EDF-VD-SE is to delay the mode change in EDF-VD to high criticality mode as shown in Fig. 3. The delay is achieved by tolerating a single overrun of execution time budget for a high criticality task, under the assumption that overruns are possible, but rare. In EDF-VD the scheduler would kill all low criticality tasks and serve only high criticality tasks after the first overrun. But with EDF-VD-SE, we can continue to service jobs from low criticality tasks, resulting in a better quality of service (QoS) for them.

With EDF-VD-SE, we consider both the schedulability of high criticality tasks, and the QoS for low criticality tasks during system operation. Compared to analysis-only approaches like EDF-VD, whose scope is the schedulability of high criticality tasks, EDF-VD-SE is motivated from a different perspective on the problem: how to guarantee schedulability of high criticality tasks *and* improve the QoS for low criticality tasks. We deem this perspective more appropriate for safety-critical systems, and EDF-VD-SE as a holistic approach to solve the problem.

By reserving additional time during system design, EDF-VD-SE can tolerate an overrun of a high criticality task. The additional time is sourced from static slack, or by reducing the utilization of low criticality tasks. The question remains how much time do we need to reserve, and how to ensure that the approach works with the task system at hand. We adopt the idea of virtual, earlier deadlines as in EDF-VD to reserve time for the tolerated overrun. We formulate the question as an optimization problem and solve it with SLSQP to acquire a suitable virtual deadline scaling parameter and the maximum allowed utilization of low criticality tasks. Comparing with the task system at hand, we might find out that we can even add further low criticality tasks, which is valuable information for the system designer.

To derive a suitable virtual deadline scaling parameter for a task system with n high criticality tasks, we create for each high criticality task a virtual task system where the task's execution time budget in low criticality mode is equal to the budget in high criticality mode. If we can find a virtual deadline scaling parameter x suitable for all n virtual task systems, we can use this x to create the virtual deadlines $\hat{d}_i = xd_i$ in the original task system, and schedule the original task system with confidence that one overrun is tolerable.

IV. MODEL

Our approach works well for modern fault-tolerant hard real-time systems build from commercial off-the-shelf (COTS) components. Ensuring the continuous correct service of such a system and thus making it reliable requires addressing faults during system design and operation. Faults are classified based on whether their duration is permanent or transient, their extend is local or distributed, and their value is determinate or indeterminate [18]. Once a fault is activated, the system deviates from its correct service state. This deviation is an *error*. If the error affects the delivered service a failure occurs [19]. Fault tolerance techniques are used to prevent system failure and differ in which class of fault they are able to tolerate.

Therefore our model, which we introduce in the following sections, can describe systems where error detection and correction is used to ensure the service of high criticality tasks, but not exclusively. First, we introduce the aspects of our model which are relevant during system operation, including mode change, application of virtual deadlines, and implications for the scheduler implementation (Section IV-A). Motivated by the system operation, we continue with the static aspects of our model which cover error modeling and the schedulability check (Sections IV-B and IV-C).

A. SYSTEM OPERATION

The standard mode change scheme [9], [12], [14] for mode switched EDF scheduling of dual-criticality task systems separates the run-time in modes, with the system starting in

low criticality mode, where all jobs are scheduled according to their execution time budget c_i or c_i^L. Low criticality jobs are prevented to execute longer than c_i, but high criticality tasks can overrun their budget up to c_i^H. As soon as a high criticality tasks executes longer than c_i^L, the system changes into the high criticality mode, where only jobs from high criticality tasks are scheduled. Once the system enters high criticality mode, low criticality tasks, or rather the service they implement, are abandoned.

The mode change scheme in EDF-VD-SE is an expansion of the standard mode change scheme as shown in Fig. 3. The additional intermediate mode between high- and low criticality mode accounts for the single tolerable error, and allows to continue servicing jobs from low criticality tasks. Once the second error is detected, the system changes into high criticality mode, which is identical to the standard mode change scheme.

In EDF-VD-SE the deadlines of high criticality jobs are scaled by x. When the system starts, jobs are EDF scheduled according to their earlier, virtual deadlines from the scaling. In case of a single error, which results in a job from a high criticality task τ_i to execute longer than c_i^L, the system records the error and continues to service all tasks. If a second error happens, all low criticality tasks and their jobs are removed from the system, and jobs from high criticality tasks are scheduled according to their original deadlines. Therefore the run-time operation is similar to EDF-VD except the additional flag to record the first error and to delay the mode change.

B. ERROR MODEL

In EDF-VD-SE, we model the error detection and correction techniques used to ensure the service of high criticality tasks with execution time budgets. During error-free operation, protected high criticality tasks don't exceed their low mode execution time budget. Once errors are detected, the error correction requires additional time, which is reflected in the high mode execution time budget.

Execution time budget overrun is a powerful abstraction for transient errors in embedded systems because a lot of soft- and hardware errors manifest as delays or errors in timing. For example, a single energetic particle strike causing a voltage spike at a node in an integrated circuit can result in a single event upset (SEU) [20]. Such an SEU might flip a bit in a central processing unit (CPU) register which holds a loop counter, or a calculated result gets corrupted and needs to be recalculated. These longer run-times are considered in the high mode execution time budget c_i^H. To model a task with error correction, we consider $c_i^H = z_i c_i^L$ as a multiple of c_i^L, where $z_i \in \{x \in \mathbb{Z} : x > 1\}$ captures the pessimism in estimating the task's WCET for error correction approaches where jobs are repeated.

C. SCHEDULABILITY CONDITIONS

To account for the unforeseen switch to high criticality mode, approaches like EDF-VD reserve processor capacity by introducing earlier, virtual deadlines $\hat{d}_i = xd_i$ for high criticality tasks. Therefore virtual deadlines should be chosen in such a way that they ensure schedulability during the mode switch, or transitional phase, under the assumptions of the implemented mode switch scheme. For a dual-criticality system, the necessary conditions to ensure steady-state schedulability under EDF-VD are [6]:

$$U_L^L + \frac{U_H^L}{x} \leq 1 \tag{17}$$

$$xU_L^L + U_H^H \leq 1 \tag{18}$$

In Eq. (17) the term U_H^L/x accounts for the utilization increase of high criticality tasks from earlier virtual deadlines. We replace this term with $\hat{U}_H^L$, which reserves time for a *single* high criticality task τ_j in low criticality mode to tolerate an overrun without switching modes:

$$\hat{U}_H^L = \frac{c_j^H}{d_j} + \sum_{\substack{i \in \tau_H \\ i \neq j}} \frac{c_i^L}{xd_i} \tag{19}$$

The resulting schedulability condition for the low criticality mode considering τ_j is:

$$U_L^L + \frac{c_j^H}{d_j} + \sum_{\substack{i \in \tau_H \\ i \neq j}} \frac{c_i^L}{xd_i} \leq 1 \tag{20}$$

To account for a single error in *all* tasks we check if the task system is still schedulable if any high criticality task requires more time. This increases the number of equations we need to consider from two to $n + 1$ with n as the number of high criticality tasks in τ:

$$\forall j \quad U_L^L + \frac{c_j^H}{d_j} + \sum_{\substack{i \in \tau_H \\ i \neq j}} \frac{c_i^L}{xd_i} \leq 1 \tag{21}$$

The schedulability condition for the high criticality mode is unchanged.

In EDF-VD-SE, we seek a numerical solution of which deadline scaling parameter x of our task system with n high criticality tasks allows to schedule the task system while supporting the maximum amount of low criticality work U_L^L. Our vector of variables is $\boldsymbol{y} = \begin{bmatrix} x & U_L^L \end{bmatrix}$, our objective function is $f(\boldsymbol{y}) = U_L^L$, and we can formulate our search as an optimization problem, with the constraints stemming from the schedulability conditions in low- and high criticality mode for each high criticality task j:

$$\begin{aligned} &\underset{\boldsymbol{y}}{\text{maximize}}\ U_L^L \\ &\text{subject to } \forall j\ 1 - U_L^L - u_j^H - \sum_{\substack{i \in \tau_H \\ i \neq j}} \frac{u_i^L}{x} \geq 0 \\ &\qquad 1 - xU_L^L - U_H^H \geq 0 \end{aligned} \tag{22}$$

The resulting optimization problem has a scalar objective function and $n + 1$ nonlinear inequality constraints, and can

TABLE 2. Example task system with two high criticality tasks and two low criticality tasks. The task system is schedulable with EDF-VD-SE, as demonstrated in Section IV-C1. The virtual deadline scaling parameter for all high criticality tasks τ_j is $x = 4/5$.

j	c_i^L	c_i^H	$p_i = d_i$	u_i^L	u_i^H	xd_i
1	2	3	10	4/20	3/10	8
2	4	8	16	5/20	5/10	64/5
-	3	-	20	3/20	-	-
-	1	-	20	1/20	-	-

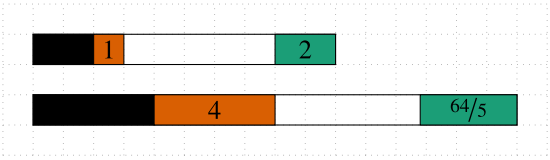

FIGURE 4. Visualization of both high criticality tasks from the example task system in Table 2. The length of each bar is equal to the task's period p_i and deadline d_i, with ratio of bar length to filled length as utilization. The black part of each bar is equal to the task's execution time budget in low criticality mode c_i^L. In high criticality mode, the execution time budget of each task is larger. The difference between low- and high criticality execution time budget is shown in red. Deadline scaling with x results in earlier deadlines, with the difference between original and virtual deadline indicated by the green part of the bar.

be solved by nonlinear programming (NLP). Because the objective function and constraints are twice continuously differentiable we can solve the problem with SLSQP [21], [22] to calculate the *maximum* utilization of low criticality tasks in low criticality mode U_L^m and working virtual deadline scaling parameter x. If the optimization succeeds, and U_L^L of the task system is below or equal to U_L^m, the task system is schedulable. The optimization is done prior to target system operation, during system design, and the results are fixed and valid for the whole system operation time. Therefore there is no overhead from optimization during operation.

1) EXAMPLE

As an example let us consider the task system in Table 2. The task system has two high criticality tasks τ_1, τ_2, and two low criticality tasks. As it stands, the utilization of low criticality tasks in low criticality mode is $U_L^L = 4/20$. Solving the optimization problem Eq. (22) results in $U_L^m = 1/4$ and $x = 4/5$. Because $U_L^m \geq U_L^L$ the task system is schedulable by EDF-VD-SE.

The example's deadline scaling is visualized in Fig. 4, which shows how earlier deadlines reserve time for the mode change. Despite earlier, virtual deadlines the period is not changed, and the mode is changed once one of the high criticality tasks executes longer than c_i^L.

2) MODIFICATION OF U_L^L

Solving the optimization problem Eq. (22) can results in 1) a schedulable task system $U_L^m \geq U_L^L$; or 2) a non-schedulable task system $U_L^m < U_L^L$. Contrary to EDF-VD, we can use the additional information of U_L^m to modify our task system in both cases: 1) $U_L^m \geq U_L^L$ allows us to add additional low criticality tasks to our task system until $U_L^L = U_L^m$; and 2) $U_L^m < U_L^L$ guides us in removal of tasks until $U_L^L = U_L^m$. In both cases, a difference in utilization $\Delta U_L^L = U_L^m - U_L^L$ of zero indicates the maximum load for the processor. We investigate how modification of U_L^L by ΔU_L^L can help to improve the acceptance rate for EDF-VD-SE in Section V-A.

V. EXPERIMENTS

To understand the usefulness of EDF-VD-SE, we investigate how many task systems are actually schedulable with EDF-VD-SE. For this we generate a set of random task systems and apply the schedulability check described in Section IV-C, noting which task systems are schedulable and which not.

Moreover, we want to know the benefit in QoS, or additional system run-time beyond the first error. We use `Thready` [23] to simulate a large set of random task systems to investigate the run-time until the first- and second error.

A. ACCEPTANCE RATE OF `UUnifast` RANDOM TASK SYSTEMS FOR DIFFERENT UTILIZATIONS

The acceptance rate is the number of schedulable task systems divided by the total number of task systems. We generate random task systems to investigate how many can be scheduled by EDF-VD-SE in comparison to EDF-VD. One benefit of EDF-VD-SE is that we know the maximum allowed utilization by low criticality tasks after the schedulability check. Therefore we investigate the acceptance rate of EDF-VD-SE in two ways: First, we compare if the utilization from low criticality tasks in the random task system is below or equal to the maximum utilization allowed by EDF-VD-SE. If this is the case, we count the task system as schedulable. In the second investigation we use our knowledge about the maximum allowed utilization to modify the random task system, and report if it is schedulable after modification. While we do this, we take note of the low task utilization delta, which can be positive, if EDF-VD-SE allows to add further low criticality tasks to the system, or negative, if EDF-VD-SE requires to lower the utilization of low criticality tasks. Such a negative utilization delta can be interpreted as the cost to allow a single error.

In Fig. 5, we compare the acceptance rate of EDF-VD-SE to EDF-VD, with and without modification of the random task system. Each data point is the result of calculating the acceptance rate of 1024 task systems for a specific utilization in low criticality mode $U_L = U_L^L + U_H^L$. We generate random task systems with specific U_L as described in Section II-C, following a parameterization for task systems which are mostly schedulable by EDF-VD [6]: For each task periods are uniformly drawn between $p_l = 50$ and $p_u = 200$, and pessimism is uniformly drawn between $z_l = 1$ and $z_u = 2$.

While EDF-VD shows a superior schedulability for task systems with utilization $U_T < 0.9$, EDF-VD-SE can use the knowledge about maximum low criticality task utilization to provide a near constant acceptance rate around 0.5.

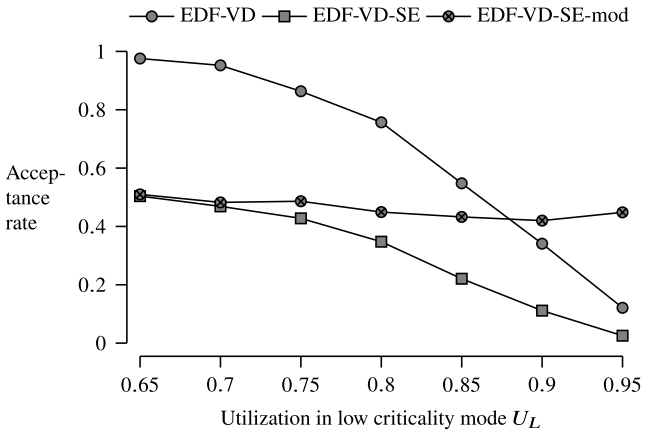

FIGURE 5. Acceptance rate of EDF-VD-SE and EDF-VD for `UUnifast` random task system. A higher acceptance rate is better. With increasing utilization on low criticality mode U_L the difficulty to find acceptable deadline scales increases. The data labeled "EDF-VD-SE-mod" is for EDF-VD-SE with adjustment to the low criticality task utilization.

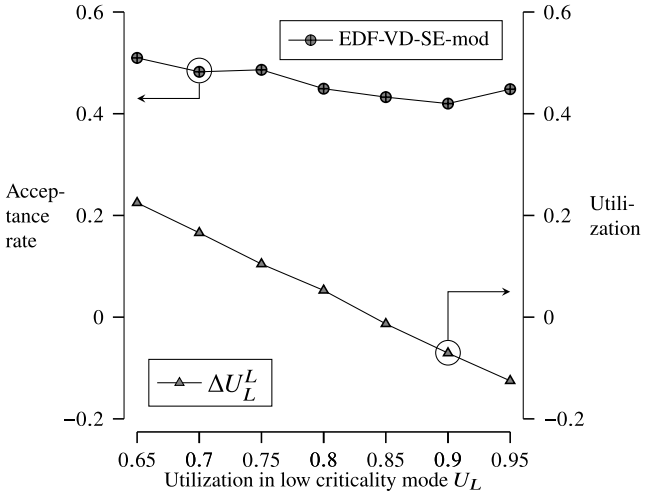

FIGURE 6. Acceptance rate for EDF-VD-SE with adjustment to the low criticality task utilization, described by the average ΔU_L^L. Arrows indicate the relevant axis for each data. A higher acceptance rate is better. A negative ΔU_L^L requires to decrease the utilization of low criticality tasks, and a positive value allows to add more.

Moreover, the general decline of the acceptance rate is less emphasized for EDF-VD-SE without adaption compared to EDF-VD.

If we adjust our task systems by ΔU_L^L on average, as shown in Fig. 6, we achieve a near constant acceptance rate. It is interesting to see how the average ΔU_L^L is positive for utilizations $U_L < 0.85$, and turns into a cost for task systems with higher utilization.

B. QUALITY OF SERVICE COMPARISON BY MODE SWITCH TIME

Estimating the QoS improvement requires to simulate how task systems are actually scheduled on a uniprocessor. It is not sufficient to apply the schedulability check, which only ensures that the task system is schedulable. To investigate the increased service for low criticality tasks, we need to measure how long jobs from such tasks are scheduled, and when the system changes modes. The general idea is to simulate each task system multiple times until it switches to high criticality mode, to cover different job arrival patterns, and to analyze the scheduling response of the system.

These simulations are random by nature, and allow us to gather quantitative results. Random decisions during simulation are 1) when jobs from a task arrive at the scheduler; 2) how much computation the job requires; and 3) does the job overrun its execution time budget if it is from a high criticality task. We control the randomness in these decisions in our simulation experiment by parameters, which we introduce prior to the actual simulation results.

1) SIMULATION PARAMETERS

Our simulation setup allows to change several parameters which influence the random decisions during simulation. The time between two jobs of the same sporadic task is at least its period, but by random chance it might be longer. We assume exponential distributed job inter-arrival times. The exponential distribution is parameterized with β, with the CDF as $F(x) = 1 - \mathrm{e}^{-x/\beta}$ for $x \geq 0$. The time between two jobs is $p_i + e_i p_i$. We choose $\beta \ll 1$ to set the time between two jobs of the same task to its period p_i, to demonstrate our independence of dynamic slack time. In general, longer job inter-arrival times are beneficial for scheduling, because it creates dynamic slack time, which can be used to service jobs that else might have failed to meet their deadline. Our $\beta \ll 1$ is the worst case assumption in terms of dynamic slack, because it reduces the accumulation of dynamic slack to dynamic slack from unused computation budgets.

The environment is reflected in the probability p to have an error, which is observable by the scheduler as an overrun in computation when a high criticality job executes for $c_i > c_L$. With $p = 0.001$ we have a rather harsh environment, where it is not uncommon to have an error which results in an overrunning high criticality task. Each job's overrun probability is independent and equal to p. It is interesting to note that choosing a larger p is not influencing the fairness of the QoS comparison between EDF-VD and EDF-VD-SE. While a larger p increases the chance to have an error, which results in EDF-VD dropping the low criticality tasks while EDF-VD-SE can continue, it increases the chance to have a second error as well, which limits the QoS improvement of EDF-VD-SE over EDF-VD too.

The computation for each job from low criticality tasks is chosen uniformly between 1 and c_L. Jobs from high criticality tasks can overrun. If they overrun, the computation is chosen uniformly between $c_L + 1$ and c_H, else between 1 and c_L. This behavior is identical in both low criticality mode, and the intermediate "single overrun" mode.

2) SIMULATION RESULTS

We create dual criticality `UUnifast` random task systems with generation parameters similar to Baruah *et al.* [6] which are schedulable by EDF-VD-SE as described in Section IV-C.

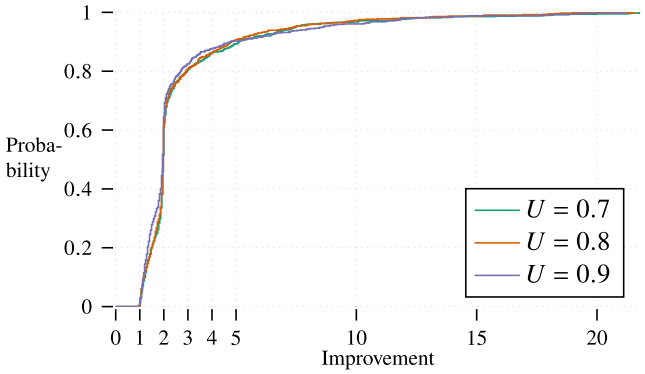

FIGURE 7. CDF of EDF-VD-SE QoS improvement. The QoS improvement is the ratio of time until the second error to first error: QoS = t_2/t_1. Task systems are `UUnifast` random generated with target utilizations of $U = 0.7$, $U = 0.8$, and $U = 0.9$.

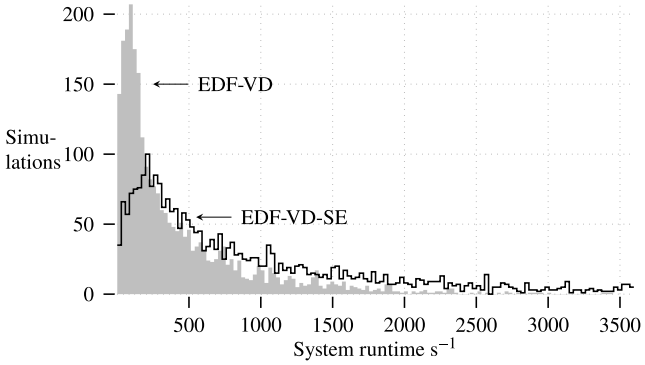

FIGURE 8. System run-time histogram of baseline EDF-VD and improved EDF-VD-SE.

We use `Thready` to simulate each EDF-VD-SE schedulable task system with an error probability of $p = 0.001$ for one hour with a millisecond time step resolution. Because EDF-VD-SE can tolerate a single error, which results in a single overrunning high criticality task, we record the system run-time up to the second overrun, where EDF-VD-SE would switch into high criticality mode and abandon all low criticality tasks. The prolonged run-time is the additional service for low criticality tasks. We calculate the QoS improvement as the ratio of time until the second error to first error: QoS = t_2/t_1. EDF-VD-SE either achieves the same or better QoS than EDF-VD, and Fig. 7 show the distribution of all instances with QoS improvement, excluding improvements beyond three standard deviations. The steep incline of the CDF at an improvement of two is the result of a constant error probability p where on average an error happens every t_1 time steps, effectively doubling the run-time on average in low criticality mode with EDF-VD-SE.

The CDF of the QoS is independent of the actual value of p, because p influences both t_2 and t_1. Nevertheless, a higher value of p results in earlier t_2 and t_1, which is interesting to know for a specific task system in an actual application.

The run-time histogram in Fig. 8 compares the distributions of system run-time until switch to high criticality mode between EDF-VD and EDF-VD-SE. For each approach,

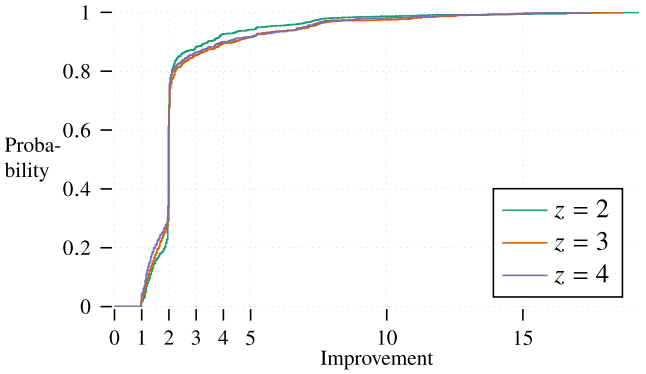

FIGURE 9. CDF of EDF-VD-SE QoS improvement for fixed pessimism factor z. The QoS improvement is the ratio of time until the second error to first error: QoS = t_2/t_1. Task systems are `UUnifast` random generated with target utilizations of $U = 0.7$.

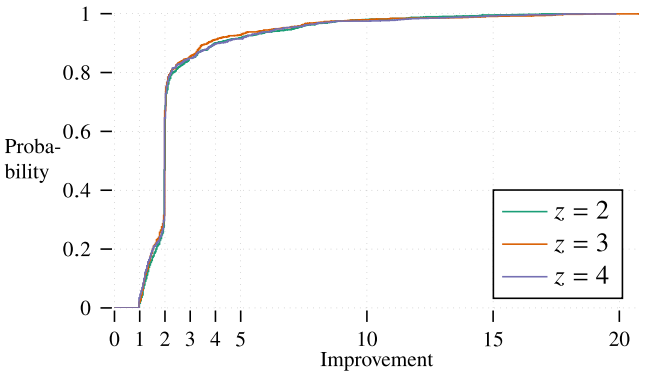

FIGURE 10. CDF of EDF-VD-SE QoS improvement for fixed pessimism factor z. The QoS improvement is the ratio of time until the second error to first error: QoS = t_2/t_1. Task systems are `UUnifast` random generated with target utilizations of $U = 0.8$.

the same task systems with utilizations between $U = 0.65$ and $U = 0.95$ have been simulated. The distributions look like Log-normal, which is a result of the independent job overrun probability p, where the system run-time random variable is a multiplicative product of many independent random variables.

If we set the pessimism factor $z_l = z_h = z$, we can investigate how a controlled over-allocation of budgets influences the QoS. We select $z = 2$ to model software error correction approaches where jobs are repeated in case of detected errors. Larger values of z represent multiple recomputations and possibly majority voting of results to correct errors in software. As shown in Figs. 9 to 11, the impact of the pessimism factor is nearly identical over all shown utilizations U. Looking at Fig. 12 for $U = 0.7$, the pessimism factor emphasizes the improvement of EDF-VD-SE over EDF-VD, as more systems increase their run-time with increasing pessimism.

VI. DISCUSSION

EDF-VD-SE can tolerate a single job overrun without dropping jobs or missing deadlines, and represents an approach which makes a task system fail operational (FO) with a count of one [24]. Reserving additional time to tolerate an

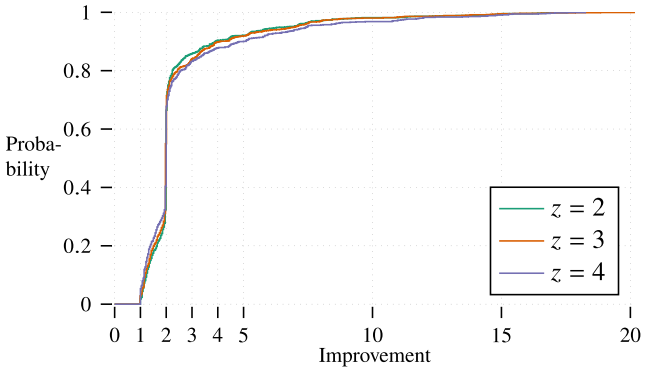

FIGURE 11. CDF of EDF-VD-SE QoS improvement for fixed pessimism factor z. The QoS improvement is the ratio of time until the second error to first error: QoS $= t_2/t_1$. Task systems are `UUnifast` random generated with target utilizations of $U = 0.9$.

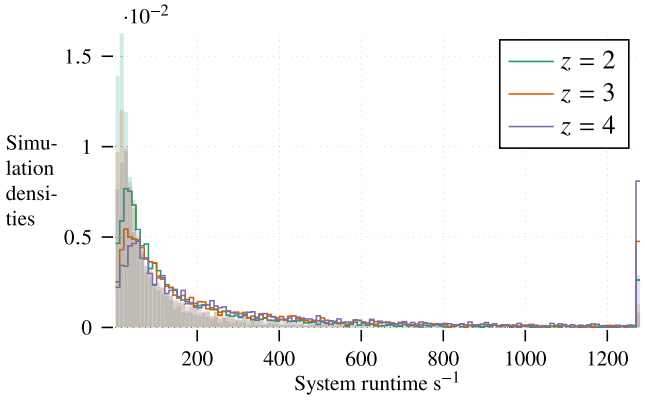

FIGURE 12. System run-time histogram of EDF-VD-SE for different pessimism factors z. Filled areas show the densities of EDF-VD for the corresponding EDF-VD-SE results of the same color. Bin size is 10 s, and the last bin contains the accumulated results of all simulations with run-times between 1280 s and 3600 s. Increased pessimism emphasizes the improvement of EDF-VD-SE over EDF-VD, as more systems increase their run-time. This is shown in the smaller spikes below 200 s, and increased number of simulations beyond 1280 s.

overrunning task successfully delays or prevents the change to high criticality mode. On average the QoS doubles, and some task systems manage to improve QoS by a factor of more than five. The static approach requires only minimal extension to the EDF-VD scheduler run-time to record the single overrun, which is beneficial for deeply embedded systems.

EDF-VD-SE is limited to handle a single error, multiple errors can cause utilization spikes which can lead to deadline violations depending on the currently available dynamic slack. Nevertheless tolerating a single error already improves QoS, and multiple errors during system operation are rare if the error probability and system run-time are suitable for the application.

Compared to EDF-VD, the acceptance rate is fair considering that EDF-VD represents an optimal non-clairvoyant algorithm [14] which is not designed to tolerate a single error. Moreover, EDF-VD-SE allows the designer to choose virtual deadline scaling such that the utilization of low criticality tasks in low criticality mode is optimized.

VII. RELATED WORK

Mixed criticality scheduling is a major aspect in modern fault-tolerant hard real-time systems which are built from COTS components to satisfy size, weight, and power (SWaP) constraints [3], [25]. Fault tolerance in such systems can be achieved by hard- and software-based error detection and correction techniques, as long as the error rate is manageable by the selected techniques.

Dual modular redundancy [26] and error detection codes like parity [27] are hardware-based error detection techniques which exploit spatial redundancy to detect errors. Detection based on temporal redundancy is possible as well, and allows to trade performance for reliability and energy savings [28], [29].

Software solutions for error detection mostly resort to temporal redundancy by duplicating the execution of instructions [30], procedures [31], or whole programs [32]. Another approach to software error detection is to generate two versions of a program with the same functionality, which are executed with different input data. Due to the systematic program generation the results are comparable and errors are detectable [33], [34]. Executable assertions introduce additional conditional checks in the program to test the plausibility of intermediate data, resulting in the detection of errors during program execution [35], [36].

Scheduling for modern fault-tolerant hard real-time systems is not deterministic due to the probabilistic nature of errors [10]. Therefore scheduling approaches can only strive to extend the system lifetime, or guarantee schedulability for a limited amount of errors in a given time window [11], [37], which is again only probabilistic.

Most dynamic priority scheduling approaches for fault-tolerant mixed criticality systems are related to, or based on, preemptive uniprocessor EDF scheduling. Preemptive uniprocessor EDF scheduling is an optimal solution which can schedule jobs from independent, sporadic, implicit deadline tasks [15], [38]. While fault-tolerance extensions of EDF exist [25], [39], [40], they do not support mixed criticality systems natively. The extension of EDF to mixed criticality systems by EDF-VD is an optimal non-clairvoyant algorithm [14] without explicit fault tolerance considerations, and the basis for EDF-VD-SE.

Regarding EDF-based scheduling approaches, fault tolerance can be achieved by reserving additional time, by smarter mode changes, or by slack management. The elastic mixed criticality model with variable periods for low criticality tasks [41] or period scaling [42] provide a way of reducing the utilization to free additional time, which is exploited in EDF-VD-SE to tolerate a single fault. Reserving some time for an overrunning task [43] requires knowledge about the error probability of each task during system design, which is at best hard to estimate, and wrong estimates risk overloading the system. Contrary EDF-VD-SE is always safe for a single overrun, no matter the error probability, which can only reduce the additional service for low criticality tasks.

Assigning low criticality tasks a smaller execution time budget on higher levels to avoid a mode change [44], or letting a subset of low criticality tasks execute in high mode [45] can improve QoS for low criticality tasks, similar to EDF-VD-SE. Contrary to EDF-VD-SE, which is a static approach, service for low criticality tasks can be adapted dynamically by assuming independence of high criticality tasks and their mode switches [46]. Transitioning from high criticality mode back to low criticality mode [47] allows to improve QoS even further, but is not implemented in EDF-VD-SE yet.

An orthogonal approach is extensive dynamic slack monitoring [48], where opposed to EDF-VD-SE a slack-aware run time management is mandatory. This is a burden for deeply embedded systems, which lack additional computation resources to implement the slack-aware run time management.

VIII. CONCLUSION

Scheduling mixed-criticality fault-tolerant hard real-time systems needs to consider the QoS for low criticality tasks. Our static approach EDF-VD-SE reserves additional time to tolerate a single error without risking deadline violations while optimizing the amount of possible low criticality work, doubling on average the QoS for low criticality tasks. EDF-VD-SE requires no slack-aware run-time operation, which is beneficial for deeply embedded systems with limited computation resources, and no assumptions about error probabilities for safety guarantees. Therefore EDF-VD-SE is a viable and certification friendly approach to schedule tasks in modern deeply embedded mission and safety-critical systems.

REFERENCES

[1] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," in *Proc. IEEE/AIAA 26th Digit. Avionics Syst. Conf.*, Oct. 2007, pp. 2.A.1-1–2.A.1-10.

[2] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 7, pp. 956–965, Jul. 2009.

[3] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surveys*, vol. 50, no. 6, pp. 1–37, Jan. 2018.

[4] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, pp. 1–53, Apr. 2008.

[5] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 239–243.

[6] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 145–154.

[7] T. Fleming and A. Burns, "Incorporating the notion of importance into mixed criticality systems," in *Proc. 2nd Workshop Mixed Crit. Syst.*, 2014, pp. 33–38.

[8] S. Baruah and A. Burns, "Incorporating robustness and resilience into mixed-criticality scheduling theory," in *Proc. IEEE 22nd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2019, pp. 155–162.

[9] A. Burns and S. K. Baruah, "Towards a more practical model for mixed criticality systems," in *Proc. 1st Int. Workshop Mixed Crit. Syst.*, 2013, pp. 1–6.

[10] A. Burns, S. Punnekkat, L. Strigini, and D. R. Wright, "Probabilistic scheduling guarantees for fault-tolerant real-time systems," in *Proc. Dependable Comput. Crit. Appl. 7*, Jan. 1999, pp. 361–378.

[11] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 155–165.

[12] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Syst.*, vol. 50, no. 1, pp. 48–86, Jun. 2013.

[13] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *Proc. Euromicro Conf. Real-Time Syst.*, Jul. 2008, pp. 147–155.

[14] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems," *J. ACM*, vol. 62, no. 2, pp. 1–33, May 2015.

[15] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a Hard-Real-Time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[16] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, May 2005.

[17] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Improving the scheduling of certifiable mixed-criticality sporadic task systems," Dept. Inf. Technol., Uppsala Univ., Uppsala, Sweden, Tech. Rep. 2013-008, Apr. 2013.

[18] A. Avižienis, "Fault-tolerant systems," *IEEE Trans. Comput.*, vol. C-25, no. 12, pp. 1304–1312, Dec. 1976.

[19] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[20] *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*, JEDEC Solid State Technol. Assoc., Arlington, VA, USA, JEDEC Standard JESD89A, Oct. 2006.

[21] D. Kraft, "A software package for sequential quadratic programming," Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt, Tech. Rep. DFVLR-FB 88-28, 1988.

[22] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[23] R. Schmidt and A. Garcia-Ortiz, "Thready: A fast scheduling simulator for real-time task systems," in *Proc. 9th Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, Sep. 2020, pp. 1–6.

[24] A. Burns, R. I. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1478–1491, Oct. 2018.

[25] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Fault tolerant scheduling of mixed criticality real-time tasks under error bursts," *Procedia Comput. Sci.*, vol. 46, pp. 1148–1155, 2015.

[26] D. Brière and P. Traverse, "AIRBUS A320/A330/A340 electrical flight controls: A family of fault-tolerant systems," in *Proc. 23rd Int. Symp. Fault-Tolerant Comput.*, 1993, pp. 616–623.

[27] G. Aydos and G. Fey, "Exploiting error detection latency for parity-based soft error detection," in *Proc. IEEE 19th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2016, pp. 1–6.

[28] M. Nicolaidis and M. Dimopoulos, "Advanced double-sampling architectures," in *Proc. IEEE 22nd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2016, pp. 130–132.

[29] R. Schmidt, A. Garcia-Ortiz, and G. Fey, "Temporal redundancy latch-based architecture for soft error mitigation," in *Proc. IEEE 23rd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2017, pp. 240–243.

[30] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Trans. Rel.*, vol. 51, no. 1, pp. 63–75, Mar. 2002.

[31] N. Oh and E. J. McCluskey, "Error detection by selective procedure call duplication for low energy consumption," *IEEE Trans. Rel.*, vol. 51, no. 4, pp. 392–402, Dec. 2002.

[32] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proc. 27th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2000, pp. 25–36.

[33] H. Engel, "Data flow transformations to detect results which are corrupted by hardware faults," in *Proc. IEEE High-Assurance Syst. Eng. Workshop*, 1996, pp. 279–285.

[34] N. Oh, S. Mitra, and E. J. McCluskey, "ED4I: Error detection by diverse data and duplicated instructions," *IEEE Trans. Comput.*, vol. 51, no. 2, p. 180, 2002.

[35] M. Hiller, "Executable assertions for detecting data errors in embedded control systems," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, 2000, pp. 24–33.
[36] J. Vinter, J. Aidemark, P. Folkesson, and J. Karlsson, "Reducing critical failures for control algorithms using executable assertions and best effort recovery," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2001, pp. 347–356.
[37] R. M. Pathan, "Fault-tolerant and real-time scheduling for mixed-criticality systems," *Real-Time Syst.*, vol. 50, no. 4, pp. 509–547, Jul. 2014.
[38] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. 11th Real-Time Syst. Symp.*, Dec. 1990, pp. 182–190.
[39] Q. Han, L. Niu, G. Quan, S. Ren, and S. Ren, "Energy efficient fault-tolerant earliest deadline first scheduling for hard real-time systems," *Real-Time Syst.*, vol. 50, nos. 5–6, pp. 592–619, Sep. 2014.
[40] H. Beitollahi, S. G. Miremadi, and G. Deconinck, "Fault-tolerant earliest-deadline-first scheduling algorithm," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Mar. 2007, pp. 1–6.
[41] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2013, pp. 147–152.
[42] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 125–130.
[43] X. Zhao, Y. Wei, and W. Li, "The improved earliest deadline first with virtual deadlines mixed-criticality scheduling algorithm," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPA/IUCC)*, Dec. 2017, pp. 444–448.
[44] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Nov. 2016, pp. 25–46.
[45] K. Yang and Z. Guo, "EDF-based mixed-criticality scheduling with graceful degradation by bounded lateness," in *Proc. IEEE 25th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2019, pp. 1–6.
[46] G. Chen, N. Guan, D. Liu, Q. He, K. Huang, T. Stefanov, and W. Yi, "Utilization-based scheduling of flexible mixed-criticality real-time tasks," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 543–558, Apr. 2018.
[47] H. Su, N. Guan, and D. Zhu, "Service guarantee exploration for mixed-criticality systems," in *Proc. IEEE 20th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2014, pp. 1–10.
[48] B. Hu, L. Thiele, P. Huang, K. Huang, C. Griesbeck, and A. Knoll, "FFOB: Efficient online mode-switch procrastination in mixed-criticality systems," *Real-Time Syst.*, vol. 55, no. 3, pp. 471–513, Dec. 2018.

ROBERT SCHMIDT (Graduate Student Member, IEEE) received the Diploma degree in electrical engineering from the University of Bremen, Germany, in 2015.

From 2015 to 2017, he worked as a Research Associate with the Institute of Space Systems, German Aerospace Center. Since 2017, he has been working as a Research Associate with the Institute of Electrodynamics and Microelectronics, University of Bremen.

ALBERTO GARCÍA-ORTIZ (Senior Member, IEEE) received the Diploma degree in telecommunication systems from the Polytechnic University of Valencia, Spain, in 1998, and the Ph.D. degree *(summa cum laude)* from the Department of Electrical Engineering and Information Technology, Institute of Microelectronic Systems, Darmstadt University of Technology, Germany, in 2003. For two years, he worked with NewLogic, Austria. From 2003 to 2005, he worked as a Senior Hardware Design Engineer with the IBM Deutschland Research and Development, Böblingen. Then, he joined the start-up AnaFocus, Spain, where he was responsible for the design and integration of AnaFocus' next generation Vision Systems-on-Chip. He is currently a Full Professor with the Chair of Integrated Digital Systems, University of Bremen. His research interests include low-power design and estimation, communication-centric design, SoC integration, and variations-aware design. He received the Outstanding Dissertation Award from the European Design and Automation Association, in 2004, and the Innovation Award from IBM, in 2005, for his contributions to leakage estimation, and he holds two issued patents for that work. He serves as an Editor for *JOLPE* and a reviewer for several conferences, journals, and European projects.