On the Safety of Mixed-Criticality Scheduling

Stefan Draskovic, Pengcheng Huang and Lothar Thiele
ETH Zurich, Switzerland
firstname.lastname@tik.ee.ethz.ch

***Abstract*—A common assumption for scheduling mixed-criticality systems is the degradation of less critical tasks when the system is in the emergency mode; which is entered when critical tasks overrun their expected WCET. In the meanwhile, common safety standards enforce strict safety requirements on all criticality levels. However, the impact of degrading less critical tasks on the overall system safety is not well understood. By introducing probabilistic distributions of task execution times, we show in this paper a first analysis of safety of mixed-criticality systems. Inspired by standards, we provide the probability-of-failure-per-hour (PFH) of a system when no overrun is detected. We also introduce and analyze the expected mode switching time, which describes how often emergency mode is entered. Our results reveal a fundamental trade-off between these two values. Finally, we discuss other possible safety measures that form a complete overview of a system's safety.**

I. Introduction

Today, mixed-criticality is becoming a significant trend for many major industries including automotive and avionics [1], where applications of different safety criticality levels are consolidated into a common computing platform to reduce cost, weight, and energy [2]. In the meanwhile, the design process of mixed-criticality systems is often regulated by industrial safety standards (see e.g. DO-178C [3] and ISO 26262 [4]), with the ultimate goal to meet different safety requirements on all criticality levels. To achieve this, a common safety measure, probability-of-failure-per-hour (PFH), is often suggested. For certification, it is required by the safety standards that different PFHs must be met on different criticality levels. In addition, criticality segregation is often suggested in safety standards as "best practice" towards ensuring system safety.

While traditional industrial practice favors static temporal and spatial isolation for mixed-criticality systems [3], [4], [5], a significant recent trend is to enforce dynamic and asymmetric isolation among different criticality levels [6], [1]. This means having several possible modes of operation for the system, among which one is chosen based on decisions made at run-time. The system starts in *normal* mode, and stays in it as long as there are no execution time overruns. In this mode, all tasks are guaranteed to meet their real-time requirements. If any task overruns its expected WCET, then the system switches to an *emergency* mode, where less critical tasks are given degraded service in order to free system resources for critical tasks. There can be several emergency modes, depending on which criticality levels are degraded. In the general case, there are as many modes of operation as criticality levels. A plethora of scheduling policies have been proposed based on this model [1] for both single-core and multi-cores – fixed-priority scheduling [7], [8], earliest deadline first scheduling [9], [10], [11], [12] and time-triggered scheduling [13], [6], [14]. Furthermore, the scheduling techniques are extended to consider fault-tolerance [15], [16], [17], [18], energy-conservation [19], [20], resource interference [21], [22], etc.

We observe that safety quantification of mixed-criticality scheduling techniques remains largely unexplored in the state-of-the-art [1], despite its importance from the certification's point of view. On the one hand, system resource efficiencies can be improved by enforcing asymmetric isolation among different criticality levels [1], since resources can be "shifted" to more critical tasks when they execute in emergency mode. On the other hand, degrading less critical tasks (dropping them in the extreme case [9], [10], [11], [13], [6], [14]) would naturally affect the safety of those tasks; this dependency needs to be understood and bounded in order to meet safety (PFH) requirements on all criticality levels, as required by existing safety standards [3], [4]. Existing work either focus on deadline missing probabilities of individual tasks [23], [24] or implicitly address system safety [25].

In light of this, we strive in this paper to answer the following question: *How safe is each criticality level for a mixed-criticality system?*

A. Contribution and Organization

As a step to answer the above question, we assume fixed-priority scheduling and dynamic mode-switched dual-criticality systems. Our contributions are as follows.

- A new probabilistic mixed-criticality system model is introduced in Section II. It features a normal and an emergency mode of operation, and the system choses one based on run-time events.
- For this model, we propose several safety metrics. These are the mode independent PFH (in Section II), the PFH inside normal and inside emergency mode (see Section III), and the expected mode switch time (Section IV).
- An analysis that produces the PFH in normal mode is given in Section III. It is based on the steady state analysis, presented in [26]. A way to safely use this analysis for our problem is proven, and it is extended to produce the PFH in normal mode.
- The expected mode switch time is calculated in Section IV. It is derived from execution time thresholds of higher criticality tasks. A fundamental trade-off, between normal mode safety and expected mode switch time, is presented and discussed in Section V.

- Experimental results in Section VI illustrate the PFH in normal mode and the expected mode switch time, with the trade-off between these two clearly visible.

II. System Model

We consider a mixed-criticality system with two criticality levels, scheduled using fixed-priority on a preemptive unicore. The criticality levels are denoted HI (high criticality) and LO (low criticality). The PFHs of criticality levels (PFH$_{\text{HI}}$ and PFH$_{\text{LO}}$) are drawn from the five criticality levels in DO-178C [3]. The values 10^{-7} and 10^{-5} can be used as an example.

We assume a periodic task set τ, and each instance of a task is called a job. Task τ_i is characterized by its criticality $\chi_i \in \{\text{HI}, \text{LO}\}$, period T_i, initial arrival phase ϕ_i, worst-case execution time (WCET) C_i, and deadline D_i. We define the hyper-period (HP) as the least common multiple of all task periods.

In a conventional model [1], [8], tasks in a dual-criticality system can have two levels of WCETs. While in the normal system mode, all tasks are guaranteed to meet their deadlines adhering to their LO level WCETs ($C_i(\text{LO}), \forall \tau_i$). If any HI criticality task overruns its LO level WCET, then the system switches to emergency mode. In this mode, all HI tasks are guaranteed with a more pessimistic WCETs ($C_i(\text{HI}), \forall \tau_i \in \tau_{\text{HI}}$). In emergency mode LO criticality tasks are guaranteed with reduced service. As an example, reduced service could imply less in precision. LO criticality tasks are not allowed to overrun $C_i(\text{LO})$.

In an attempt to match theoretical insights with applicable safety standards, we now introduce a probabilistic mixed-criticality task model. In this model we augument the WCETs model by making execution times independent random variables $\mathcal{C}_i, \forall \tau$, whose probability distributions are known. Well established methods like static or measurement based probabilistic timing analysis techniques already exist to derive safe estimations of task execution time distributions, see [27], [28], [29], [30]. For computational reasons, we assume a discrete distribution of task execution times.

Similar to the conventional model, we assume a normal and an emergency system mode. The system starts in normal mode and switches to emergency mode when any HI criticality task overruns an execution time threshold θ_i. LO criticality tasks are not allowed to exceed their execution thresholds, thus when the threshold is reached they are killed. In emergency mode, more resources are provided to HI tasks, at the expense of LO tasks – LO criticality tasks could be completely dropped [9], [10], [11], their service could be degraded with longer activation periods [12], [6], or they can be served as background tasks by lowering their priorities. We provide general safety analysis techniques considering all of these cases.

Once in emergency mode, the system could either stay in that mode forever, or switch back to normal mode when the system is idle [1], i.e. when there are no jobs executing or waiting to be executed. We shall assume the later case, the argument being, that degrading a criticality level indefinitely is not appropriate. The switch back to normal mode is left for future work.

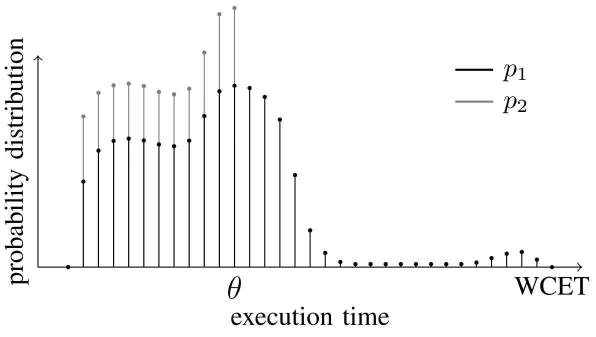


Fig. 1: p_2 created by limiting p_1 with θ

A value used often in this paper is the backlog. A backlog $b_{\lambda,i}$ is a random variable, describing how much more execution time is needed at time λ to finish old jobs of priority i and higher. Backlogs are important because they determine when an arriving job can start its execution.

A. Notations on Probabilities

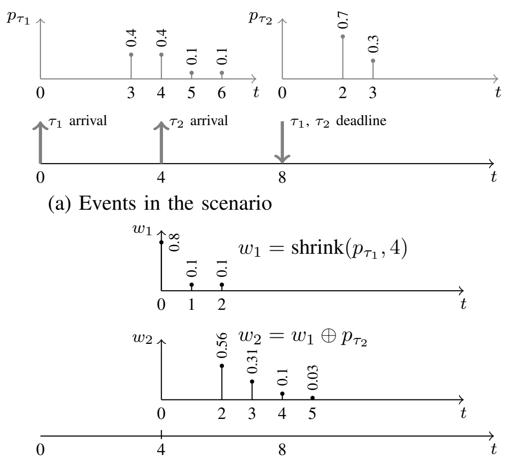
It is useful to introduce some notations that will be used, as well as a relation for ordering random variables.

1) Discrete PDFs: We deal with discrete non-negative random variables in this paper, the probability distributions of which are modeled with probability density functions (PDFs). Formally, for a random variable x and its probability density function $p_x(\cdot)$, it follows that $\sum_u p_x(u) = 1$ as u runs through the set of all possible values of x. Without loss of generality, we assume that the possible values of a random discrete variable x span the entire range of natural numbers, where $p_x(u) = 0$ if x can not take the value u. For notational convenience, we omit the random variable in our notation if it is irrelevant in the context. All probability distributions are density functions unless otherwise stated. Note that we alternate between function representation of a probability density function $p(\cdot)$ and its vector representation $[p(0), p(1), \cdots, p(u), \cdots]^T$ whenever it is convenient.

2) Comparison of PDFs: Let us define a relation $q \preceq p$, that is used to compare any two probability distributions p and q. With this relation, we say that p is greater or equal to q, if the sum from any arbitrary value l onward, is greater or equal for p than for q. Mathematically this is noted as in (1). Note that probability densities can be incomparable.

$$\forall l: \quad \sum_{u=l}^{\infty} q(u) \leq \sum_{u=l}^{\infty} p(u) \qquad \Leftrightarrow \qquad q \preceq p \tag{1}$$

3) Limiting PDFs: By limiting a probability distribution with a threshold θ, we find a conditional probability distribution, where values above the threshold θ are not allowed. This new probability distribution is calculated as in (2).



(a) Events in the scenario

(b) Analysis based on shrinking and convoluation

Fig. 2: Finding the failure probabilities of jobs in a simple scenario – at time 4, w_1 is the backlog (unfinished workload) from τ_1 while w_2 is the backlog including τ_2

$$p(x|x \leq \theta) = \frac{p(x, x \leq \theta)}{p(x \leq \theta)} = \begin{cases} \frac{p(x)}{p(x \leq \theta)} & x \leq \theta \\ 0 & x > \theta \end{cases} \quad (2)$$

We can see that this conditional distribution retains the same shape but is scaled up before the threshold, and is equal to zero afterwards. This operation is illustrated in Figure 1, where p_2 is the limit of p_1 with the threshold θ.

4) Convolution of PDFs: The well known convolution operation [26] defines the addition of independent random variables. For some discrete probability distributions p and q, it is defined as in (3).

$$(p \oplus q)(i) = \sum_{u=0}^{i} p(u) \cdot q(i-u) = \sum_{u=0}^{i} p(i-u) \cdot q(u) \quad (3)$$

5) Shrinking PDFs: Shrinking a probability distribution function p by m is defined as summing up the first $m+1$ elements into the first element, and left shifting the remaining distribution by m. (4) is a formal definition of this shrinking process.

$$\text{shrink}(p, m)(i) = \begin{cases} \sum_{j=0}^{m} p(j) & i = 0 \\ p(i+m) & i > 0 \end{cases} \quad (4)$$

We now use a minimal example to show the usage of the convolution and shrinking functions in analyzing real-time systems with probabilistic execution times.

Example 1. Let us consider the example shown in Figure 2. Two tasks τ_1 and τ_2 are scheduled via fixed-priority (τ_2 has higher priority), with task parameters also as shown in the figure. Their presented execution times are already limited by thresholds, $p_{\tau_i} = p_{\tau_i\text{full}}(x|x \leq \theta_i)$, where $p_{\tau_i\text{full}}$ are the execution times with no limiting thresholds. τ_1 starts its execution with no backlog at time 0. At time 4, we get the backlog $w_1 = \text{shrink}(p_{\tau_i}, 4)$. Intuitively, we see that this operation shifts the execution time distribution of τ_1 to the left by 4 time units. In addition, τ_2 is arriving at 4 and will preempt the execution of τ_1. For τ_2, it can be seen that its deadline miss probability is zero as it will finish before time 8 for both possible execution times. However, to get the failure probability of τ_1, we need to consider the interference from τ_2. This can be done by applying the convolution operation at time 4 to get the total backlog at this time: $w_2 = w_1 \oplus p_{\tau_2}$. From our derivation as shown in Figure 2 (b), τ_2 has a deadline miss probability of 0.03. Likewise, one could extend this analysis to arbitrary tasks under deterministic scheduling policies.

III. Plain Safety Analysis

In this section, we explain how to derive the probability-of-failure-per-hour (PFH) of a criticality level χ inside a single mode of operation M, $\text{PFH}_{\chi}^{\text{M}}$, with the assumption that the initial conditions, as well as the priority assignments and limiting thresholds, are known. It will be shown that this method can be applied to find the PFH when normal mode operation is assumed, noted as $\text{PFH}_{\chi}^{\text{N}}$.

Deriving the PFH directly, as in Example 1, is computationally prohibitive, as there could be millions of jobs inside an hour. An alternative, safe, analysis is presented, which focuses on one hyper-period, and extrapolates the results to find the probability of failure inside an interval of any length. The analysis presented here is built upon the seminal work Stochastic Analysis of Periodic Real-Time Systems [26]. There, a single-mode system model is assumed. [26] concludes that task deadline miss probabilities for every job inside a hyper-period converge, to a steady state value, as more hyper-periods are executed. The same is true for backlogs at specific times inside the hyper-period. In this work, we augment Our paper contributes by investigating when it is safe to use the steady state analysis to obtain safe PFH values, as required by safety standards.

Steady state deadline miss probabilities are obtained by first taking the steady state backlog at the beginning of a hyper-period, and then analyzing the execution of jobs inside the hyper-period when this backlog is present.

An important question is whether the steady state backlog exists in a given system. The existence of such a backlog is shown to relate to the system utilization, i.e. the ratio of time that the system is busy processing jobs. Using results from queuing theory, it has been proven that if the average system utilization is less than one, a steady state exists [26]. If the average utilization is greater than or equal to one, the steady state backlog is infinitely large, essentially making system safety guarantees impossible. Therefore, we focus on the case when a steady state exists.

However, we cannot use the steady state backlog for obtaining failure rates directly, as it is not known whether the steady state can be safely used to bound task deadline miss probabilities. In fact, deadline miss probabilities depend on the initial backlog, as stated in the following theorem.

Theorem 1. Let $\{b^n\}$ be a sequence of backlogs, where b^n is the backlog present at the beginning of hyper-period n. The sequence $\{b^n\}$ is monotone in the sense that, if $b^0 \preceq b^1$, then $b^{n+1} \preceq b^{n+2}, \forall n \in \mathbb{N}_0$, or if $b^0 \succeq b^1$, then $b^{n+1} \succeq b^{n+2}, \forall n \in \mathbb{N}_0$.

The proof is omitted here due to lack of space, but can be found in our technical report [31]. If the initial backlog at the start of operation in this mode is b^0, and the steady state backlog is π, then we present the following conclusions.

- If $b^0 \preceq \pi$, then the steady state deadline miss probabilities are the worst possible in the system, and they can be used to obtain safe failure rates.
- If $b^0 \succeq \pi$, then the deadline miss probabilities found in the first hyper-period are the worst in the system, and they can be used to find failure rates.
- If b^0 and π are neither comparable w.r.t. $\preceq$ nor $\succeq$, we need to find a third probability density q, such that $b^0 \preceq q$ and $\pi \preceq q$, and use this as a backlog in order to get safe failure rates. This value can be calculated as $\sum_{u=l}^{\infty} q(u) = \max\left(\sum_{u=l}^{\infty} b^0(u), \sum_{u=l}^{\infty} \pi(u)\right)$.

Note, a non-zero initial backlog is unusual at the start of a system, but can happen after a mode switch.

A. Failure Rates

As we know which backlog to use at a hyper-period's beginning, we can continue to determine the $\text{PFH}_{\chi}^{\text{M}}$. This is done firstly by calculating the deadline miss probability of every job of this hyper-period, and then aggregating these values into the probability of failure inside an interval of arbitrary length.

The detailed procedure for obtaining deadline miss probabilities of jobs is presented in [26]. It is in the same manner as Example 1, where convolutions and shrinking functions are used to determine when individual jobs will finish their execution.

Let us observe a time interval Δ. Regardless of the start time, the worst possible probability of failure inside the interval shall be denoted as PF_{Δ}. Because only jobs whose deadlines are inside Δ are of concern, we can link failure rates inside an interval to job deadline miss probabilities (dmp-s):

$$\text{PF}_{\Delta} = \max_{\lambda}\left(1 - \prod_{\substack{\text{job's deadline} \\ \text{in } [\lambda, \lambda+\Delta]}} (1 - \text{dmp}(\text{job}))\right) \tag{5}$$

Checking every possible interval start λ can be computationally prohibitive. Fortunately, we can reduce the problem by the following lemma.

Lemma 1. Surely $n_b = \left\lfloor \frac{\Delta}{\text{HP}} \right\rfloor - 1$ whole hyper-periods are present inside an interval Δ, regardless of its start time.

If we note the reminder of the interval as $\delta = \Delta - n_b \cdot \text{HP}$, we can calculate the failure rate inside an interval as $\text{PF}_{\Delta} = n_b \cdot \text{PF}_{\text{HP}} + \text{PF}_{\delta}$. The values PF_{HP} and PF_{δ} can be easily computed based on (5). First, the interval HP always has all jobs from one hyper-period inside it, so the search for maximum from (5) is not necessary. Second, the δ interval is limited in length, so the calculating (5) is not computationally prohibitive. More details, along with the proof of the lemma, can be obtained in the report [31].

B. Application to Normal Mode

In normal mode, by definition, tasks will not overrun their execution time thresholds. Thus, we could use the limited execution time distributions of all tasks and directly apply the plain safety analysis technique to get the failure rates on each criticality level, $\text{PFH}_{\text{LO}}^{\text{N}}$ and $\text{PFH}_{\text{HI}}^{\text{N}}$. However, special attention needs to be paid to LO criticality tasks due to the conventional mixed-criticality model: in normal mode, if a LO criticality task exceeds its execution θ threshold, it will be killed. This also constitutes a failed execution, which was not taken into account in the afore mentioned analysis. The probability of this event happening can be obtained directly from the execution time distribution function of the task, $P(\mathcal{C}_k > \theta_k)$. Therefore, we define that $\text{PFH}_{\text{LO}}^{\text{N}}$ is the probability that either a LO task is killed, or it misses its deadline. Because these two sources of failure are independent, calculating the cumulative probability of failure can be done directly. Nevertheless, for the scope of this paper and for simplicity reasons, when mentioning $\text{PFH}_{\text{LO}}^{\text{N}}$ only missed deadlines shall be implied.

IV. Expected Mode Switch Time

To show how long the system is expected to stay in normal mode, or how often the normal to emergency mode switch occurs, we introduce the *expected mode switch time*. If the normal mode starts at time zero, and $\mathcal{T}_{\text{sw}}$ is the time the mode switch happens, then $\text{E}[\mathcal{T}_{\text{sw}}]$ is the expected mode switch time. The normal mode start can either be the system start or a safe switch from emergency to normal mode.

The probability that a mode switch occurs in an interval $[\lambda, \lambda + \Delta]$ can be written as

$$p_{[\lambda,\lambda+\Delta]}^{\text{sw}} = 1 - \prod_{\substack{\text{all HI jobs} \\ \text{in interval}[\lambda,\lambda+\Delta]}} P(\mathcal{C}_k \leq \theta_k) \tag{6}$$

In a practical system, the mode switch is expected to be a rare event, with the time between the start of normal mode and the mode switch several orders of magnitude larger than an hour. Accordingly, as a precondition to the usefulness of this analysis, we expect the hyper-period to be much smaller than the expected time between two mode switches. Because of this, it is reasonable to view one hyper-period as a time unit. If $p_{\text{HP}}^{\text{sw}}$ is the probability of mode switch in one hyper-period, then the expected mode switch is after $1/p_{\text{HP}}^{\text{sw}}$ hyper-periods. This can be written as in (7).

$$\text{E}[\mathcal{T}_{\text{sw}}] = \frac{1}{p_{\text{HP}}^{\text{sw}}} \times \text{HP} \tag{7}$$

V. Choosing the Thresholds

An important and novel question not dealt within the previous analysis is how to determine the θ_i (execution time) thresholds that trigger a mode switch. These are thresholds for HI tasks. Choosing such thresholds is a design space exploration problem, but is not a straightforward task. As we explain, it involves a trade-off between two safety related values.

Let us first investigate the impact of the threshold on the expected mode switch time, $E[\mathcal{T}_{sw}]$. As (6) shows, with increased thresholds, the probability of mode switch per hyper-period (p^{sw}) will decrease; consequently, the expected mode switch time ($E[\mathcal{T}_{sw}]$) will increase, see (7). In practice, we would like to have the expected switch time to be as long as possible, in order to keep the system "away" from the emergency mode. This implies that thresholds should be large.

However, we need to still consider the impacts of selected thresholds on system safety. We continue to discuss the way the thresholds influence failure rates in normal mode, PFH^{N}_{χ}. Since the execution time of a HI task in normal mode ($\mathcal{C}^{N}_{i}$) depends on its threshold $\theta_i \geq \mathcal{C}^{N}_{i}$, we observe that the average normal mode execution time of a task $E[\mathcal{C}^{N}_{i}]$ decreases with decreasing θ_i; accordingly, we get smaller average job execution times and less average system utilization. This implies a smaller probability that jobs miss their deadlines and fail. In summary, we find that the failure rates on each criticality level in normal mode (PFH^{N}_{χ}) monotonically increase with increasing task execution time thresholds (θ). Thus, to have the failure rates for both LO and HI tasks as low as possible, we would favor thresholds to be small.

A corner case of this trade-off gives us the comparison between systems with and without modes of operation – the latter option being when HI task's thresholds are their WCETs, in which case the system never enters emergency mode. We can see that a system in normal mode of operation is safer than the same system operating with no modes at all. However, this comes at a cost, which is the introduction of the emergency mode.

VI. Simulation Results

To illustrate the afore presented analysis, as well as the impact of execution time thresholds on the system, we present generalized observations with random simulations.

First, we have used Gumbel distributions to model the execution times of tasks, as can be seen in Figure 3. The Gumbel distribution is a standard assumption for task execution time distributions [32]. It is used to model the distribution of maximum values, in this case long but unlikely execution times.

We simulated 54 random systems, each with an average utilization close to 0.8, a hyper-period of $300\mu s$, and up to eight jobs. The threshold varied was always the one of the task with the highest priority; this way the varied threshold effects tasks of all priority levels. This was varied from the respective WCET (right side in Figures 4 and 5), to a value $25\mu s$ less than that (left side of figures). All other thresholds, both for HI and LO criticality tasks, were $3\mu s$ less than their respective WCET. In order to present different values together, we normalized them, so instead of showing absolute values,

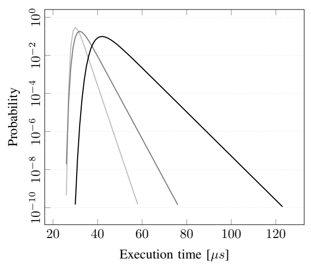


Fig. 3: Execution times as Gumbel distributions

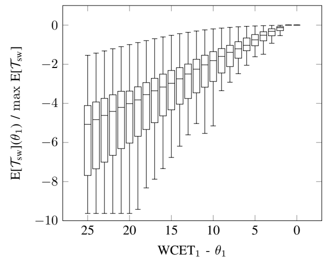


Fig. 4: Expected mode switch time for various thresholds, normalized

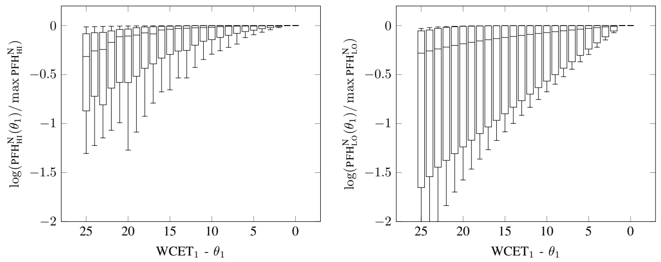


Fig. 5: Failure per hour rates, for different values of θ_1, normalized and in logarithmic scale

all values are relative to the case when the threshold of the highest priority task is at its WCET.

Figures 4 and 5 are box-whisker plots aggregating results from all of the systems. On one hand, we observe in Figure 5 that increasing threshold θ_1 increases the system failure rates in normal mode. On the other hand, Figure 4 shows that a higher threshold delays the expected time the system switches to emergency mode. This is in line with the analysis from Sections IV and V: a fundamental trade-off between normal mode safety and the expected mode switch time does exist.

VII. Future Work: Safety of Emergency Mode

The plain safety analysis, used to describe normal mode, would be pessimistic to be used for emergency mode. Thus, finding the safety of this mode is left for future work.

The reason is twofold. First, at the start of emergency mode, the backlog is expected to be large. This is because one of the jobs, the one causing the mode switch, is experiencing a high execution time. Backlogs later shall drop as time passes, and due to LO criticality tasks being degraded. However, Theorem 1 implies that this large backlog should be used to get deadline miss probabilities of all the jobs, and this potentially makes the failure rates unjustifiably high.

Second, the emergency mode is conceived by the system model to last as short as the safety requirements allow it, and having degraded service longer than necessary is not acceptable. This is why we can not take for granted that the steady state values will be reached during the operation of this mode.

Generally speaking, an analysis dealing with emergency mode should address the following challenges:

- The time of mode switch is unknown, as well as the job causing it. This causes the initial backlog to be unknown.
- The switch back to normal mode has to be understood, including proving when it is statistically safe to switch back. The statistically safe to switch back condition would be met if for example, after the switch the failure rates are in accordance with safety standards.
- For jobs in emergency mode, an algorithm to safely obtain deadline misses should be presented.

Apart from understanding the emergency mode, addressing these challenges would contribute to provide a holistic, mode independent PFH.

VIII. Conclusion

In this paper, we showed that it is possible to analyze the safety of mixed-criticality systems. We introduced a probabilistic model for dual-criticality systems, and defined safety metrics for this model in line with industry standards. These are the holistic or mode-independent PFH, the PFH in a single mode, and the expected mode switch time.

Next, we proved that the values we obtained for PFH in normal mode are safe. Expected mode switch times, from normal to emergency, were also analyzed.

Towards designing a feasible system, we showed that a trade-off has to be taken into account: this exists between the failure rates in normal mode for both HI and LO tasks, and the expected mode switch time. This was confirmed with the simulation results as well.

However, further work towards understanding emergency mode is still needed to form a complete view of the safety of a mixed-criticality system.

References

[1] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2016.

[2] S. Trujillo, R. Obermaisser, K. Grüttner, F. Cazorla, and J. Perez, "European project cluster on mixed-criticality systems," in *Design, Automation and Test in Europe (DATE) Workshop 3PMCES*, 2014.

[3] "Rtca/do-178c, software considerations in airborne systems and equipment certification," 1992.

[4] "ISO 26262, Road Vehicles - Functional Safety," 2011.

[5] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time embedded systems," *ACM Trans. on Embeddded Computing Systems*, vol. 14, no. 3, pp. 50:1–50:29, 2015.

[6] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *EMSOFT*, 2013, pp. 1–15.

[7] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*. IEEE, 2011, pp. 34–43.

[8] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 239–243.

[9] S. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Algorithms - ESA*, 2011.

[10] T. Park and S. Kim, "Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems," in *EMSOFT*, 2011, pp. 253–262.

[11] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *ECRTS*, 2012, pp. 135–144.

[12] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014, pp. 125–130.

[13] S. Baruah and G. Fohler, "Certification-cognizant time-triggered scheduling of mixed-criticality systems," in *RTSS*, 2011, pp. 3–12.

[14] P. Huang, G. Giannopoulou, R. Ahmed, D. B. Bartolini, and L. Thiele, "An isolation scheduling model for multicores," *ETH Zurich, Laboratory TIK, Tech. Rep*, vol. 361, 2015.

[15] P. K. Saraswat, P. Pop, and J. Madsen, "Task migration for fault-tolerance in mixed-criticality embedded systems," *ACM SIGBED Review*, vol. 6, no. 3, p. 6, 2009.

[16] S. Islam, R. Lindström, and N. Suri, "Dependability driven integration of mixed criticality sw components," in *Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on*. IEEE, 2006, pp. 11–pp.

[17] R. M. Pathan, "Fault-tolerant and real-time scheduling for mixed-criticality systems," *Real-Time Systems*, vol. 50, no. 4, pp. 509–547, 2014.

[18] S.-H. Kang, H. Yang, S. Kim, I. Bacivarov, S. Ha, and L. Thiele, "Reliability-aware mapping optimization of multi-core systems with mixed-criticality," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–4.

[19] V. Legout, M. Jan, and L. Pautet, "Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses," in *First Workshop on Real-Time Mixed Criticality Systems (ReTiMiCS)*, 2013, pp. 1–6.

[20] E. R. Wognsen, R. R. Hansen, and K. G. Larsen, "Battery-aware scheduling of mixed criticality systems," in *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*. Springer, 2014, pp. 208–222.

[21] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B. D. de Dinechin, "Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources," *Real-Time Systems*, pp. 1–51, 2015.

[22] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in *ECRTS*, 2012, pp. 299–308.

[23] A. Masrur, "A probabilistic scheduling framework for mixed-criticality systems," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 132.

[24] L. Santinelli and L. George, "Probabilities and mixed-criticalities: the probabilistic c-space," in *Proc. 3rd Workshop on Mixed Criticality Systems (WMC), RTSS*, 2015, pp. 30–35.

[25] Z. Guo, L. Santinelli, and K. Yang, "Edf schedulability analysis on mixed-criticality systems with permitted failure probability," in *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2015, pp. 187–196.

[26] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*. IEEE, 2002, pp. 289–300.

[27] M. Orshansky and K. Keutzer, "A general probabilistic framework for worst case timing analysis," in *Proceedings of the 39th annual Design Automation Conference*. ACM, 2002, pp. 556–561.

[28] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*. IEEE, 2012, pp. 91–101.

[29] A. Devgan and C. Kashyap, "Block-based static timing analysis with uncertainty," in *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society, 2003, p. 607.

[30] S. Malik, M. Martonosi, and Y.-T. S. Li, "Static timing analysis of embedded software," in *Proceedings of the 34th annual Design Automation Conference*. ACM, 1997, pp. 147–152.

[31] S. Draskovic, P. Huang and L. Thiele, "On the safety of mixed-criticality scheduling," *Technical Report, ETH Zurich, Laboratory TIK*, 2016.

[32] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quinones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu *et al.*, "Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study," in *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*. IEEE, 2013, pp. 241–248.