On the Scheduling of Fault-Tolerant Mixed-Criticality Systems

Pengcheng Huang, Hoeseok Yang, Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland
firstname.lastname@tik.ee.ethz.ch

ABSTRACT

We consider in this paper fault-tolerant mixed-criticality scheduling, where heterogeneous safety guarantees must be provided to functionalities (tasks) of varying criticalities (importances). We model explicitly the safety requirements for tasks of different criticalities according to safety standards, assuming hardware transient faults. We further provide analysis techniques to bound the effects of task killing and service degradation on the system safety and schedulability. Based on our model and analysis, we show that our problem can be converted to a conventional mixed-criticality scheduling problem. Thus, we broaden the scope of applicability of the conventional mixed-criticality scheduling techniques. Our proposed techniques are validated with a realistic flight management system application and extensive simulations.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Fault tolerance; C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms

Algorithms, Design, Performance, Reliability

Keywords

Mixed-Criticality, Real-time, Scheduling, Safety

1. INTRODUCTION

Many complex embedded systems are mixed-critical [20], where functionalities of varying criticalities (importances) co-exist. Examples can be seen in avionics applications (e.g. flight control and flight management systems) and automobile applications (e.g. smart car systems). For such systems, it is crucial to provide varying degrees of assurance for functionalities of different importances.

In the meanwhile, such systems are typically safety-critical [16]: the functionalities provided by such systems must be ensured under various stresses (e.g. random hardware errors, software errors, power shortage, etc). For example, the flight control and management systems are directly responsible for the safety of the airplanes and must be guaranteed to work under extreme situations. In order to guarantee safety, fault-tolerance [14] is inevitable in the design of such systems, e.g. tasks need to be replicated or re-executed in order to enhance their strengths against potential failures. Furthermore, due to the mixed-criticality nature of such systems, *different safety guarantees* need to be provided to functionalities of different criticalities.

Motivation. The current research on mixed-criticality systems has primarily focused on addressing the uncertainties in task worst-case execution times (WCETs) [7]. The problem investigated is to guarantee that critical tasks can still meet their deadlines even if the WCETs of tasks are violated at runtime. For this purpose, *killing* of less critical tasks or *degrading* their services is assumed to guarantee alternative (usually overly-pessimistic) WCETs for critical tasks. However, the mixed-criticality problem here has basically neglected fault-tolerance, which would be necessary to achieve safety for mixed-criticality systems. In addition, safety is not explicitly modeled. For most safety standards (e.g. IEC 61508 [6] and DO-178B [1]), it is required that varying probabilistic safety guarantees should be provided to functionalities of different criticalities. Such guarantees cannot be made with the current research results. Furthermore, task killing or service degradation could potentially affect the safety of less critical tasks. To the best of our knowledge, such impacts on the system safety are not quantified in existing approaches. Therefore, analysis techniques need to be developed in order to bound such impacts.

Contribution. We address in this paper mixed-criticality scheduling under hardware transient faults. We consider task re-execution as the fault-tolerance technique adopted. The problem we are studying is to provide safety guarantees for tasks of different criticalities while ensuring schedulability of the system. Our main results are as follows:

- We model explicitly the safety requirements on different criticality levels according to safety standards, which would enable us to design mixed-criticality systems complying with those safety standards.
- We provide analysis techniques to bound the system safety under various scenarios, i.e. with and without task killing or service degradation.
- We propose a scheduling algorithm unifying the concern of safety and schedulability for mixed-criticality systems. The algorithm is based on the insight that our problem can be converted to a conventional mixed-criticality

DAC '14, June 01-05 2014, San Francisco, CA, USA

scheduling problem. Thus, a broad class of existing mixed-criticality scheduling techniques can be applied.

- We show with a realistic flight management system (FMS) application the applicability of our proposed techniques. Furthermore, we evaluate with extensive experiments the effectiveness of task killing and service degradation.Our results indicate, if safety is a concern for less critical tasks, then service degradation is more proper than task killing, as the latter could directly violate the system safety.

Related Work. Recently, there is a wide-spread interest to study mixed-criticality systems [7]. To date, a common model exists to abstract such systems [21, 4, 17, 9, 19, 18, 12]: the WCETs of all tasks in the system are modeled on all existing criticality levels. And whenever a task exceeds its χ criticality WCET, all tasks with criticalities χ or lower are killed or degraded hereafter. The rationale behind this model is that: 1) the WCET on a higher criticality level is typically more conservative; and 2) less critical tasks can be compromised in emergent situations, e.g. when a critical task exhibits rarely a very large execution time. However, such a model is designed *without* any consideration of fault-tolerance. Moreover, safety is not explicitly modeled. This would be necessary as most safety standards do require probabilistic guarantees on safety.

So far, various fault-tolerance techniques have been proposed in literature, e.g. checking-point [8, 14], task re-execution [8, 10], task replication [15, 14], etc. In addition, existing works on fault-tolerance do maintain some notions of mixed-criticality, see [11] for an overview. In [15], each criticality level is modeled by a numerical value representing its importance with the probability of transient hardware faults given, the authors use task replication and propose an evolutionary algorithm to explore the trade off between system run-time, dependability and cost. In [2, 5], a loose form of mixed-criticality is used: the authors differentiate applications from fault-tolerance, fault-detection, and fault-ignore and take this into account when performing design space exploration to balance dependability and cost. However, a unified model of mixed-criticality is *lacking* in those approaches. Moreover, those works primarily focus on system level design space explorations, and useful scheduling techniques still need to be developed.

2. PRELIMINARY

2.1 System Model

We consider the scheduling of a set of independent sporadic tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ on a uniprocessor. Each task τ_i is characterized by a minimal inter-arrival time T_i, a relative deadline D_i, a WCET C_i and a criticality χ_i. Each task may issue an infinite number of instances (jobs). We assume tasks have arbitrary deadlines.For notational convenience, we use $\boldsymbol{\chi}$ to denote the set of all existing criticalities, and use $\tau_\chi = \{\tau_i \mid \chi_i = \chi\}$ to represent all criticality χ tasks. For ease of presentation, we focus on dual-criticality systems in this paper, where only a high criticality (HI) and a low criticality (LO) exist, i.e. $\boldsymbol{\chi} = \{\text{HI}, \text{LO}\}$.

Fault Model and Fault Tolerance. We assume for every job issued by task τ_i, there is an associated probability f_i that this job does not finish properly by its deadline. We assume the cause be the transient hardware errors and consider *task re-execution* as the fault-tolerance technique adopted in this paper. For this purpose, sanity checks are assumed to detect whether tasks execute correctly. In case of faults detected, the faulty task instances are re-executed up to a given number of times in order to achieve the required safety. We assume that any instance of τ_i can execute at most n_i times ($n_i \in \mathbb{N}$). We call n_i as the *re-execution profile* of task τ_i. For notational convenience, we use $N = \{n_i \mid \tau_i \in \tau\}$ to denote the re-execution profile of all tasks, and use $N_\chi = \{n_i \mid \tau_i \in \tau \wedge \chi_i = \chi\}$ to denote the re-execution profile of all criticality level χ tasks.

Table 1: DO-178B Safety Requirements

χ	A	B	C	D	E
PFH_χ	$< 10^{-9}$	$< 10^{-7}$	$< 10^{-5}$	$\geq 10^{-5}$	–

Safety Requirements. We consider in this paper the failure of tasks in the temporal domain, i.e. an instance of a task is said to fail if it does not successfully finish by its deadline. To characterize the safety of the system, we use probability-of-failure-per-hour (PFH) as our metric, which is adopted in most safety standards (e.g. IEC 61508 [6] and DO-178B [1]) for the measurement of safety of continuous functions. Specifically, for each criticality level χ, there is an associated safety requirement, PFH_χ, which represents the PFH that must be met by all criticality χ tasks. PFH_χ strictly decreases with increasing criticality χ. Furthermore, according to the definition stated in IEC 61508, PFH can be represented as the *average failure rate* in one hour over an operation duration of the system lasts O_S hours (e.g. the typical range of O_S for commercial aircrafts is $1 \leq O_S \leq 10$).

In this paper, we will stick to the DO-178B [1] safety standard, where in total 5 criticality levels are defined, with A being the highest and E being the lowest. The considered dual-criticality task sets can have any two criticalities out of the 5 criticality levels. Furthermore, for all 5 criticality levels, different safety requirements are specified, which are summarized into Table 1. As shown in this table, for DO-178B, the level D and level E tasks are essentially not safety-related. For criticality levels A, B and C, the required PFHs are essentially very small (at least less than 10^{-5}).

Problem Definition. Given the system model, the problem we are studying in this paper is defined as follows:

DEFINITION 2.1. (Fault-Tolerant Mixed-Criticality Scheduling) *Given a dual-criticality sporadic task set τ, and given the probability of failure f_i for all instances of each task τ_i. Find a re-execution profile N for all tasks and a scheduling technique $\mathcal{S}$, such that both safety requirements on all criticality levels and the schedulability of the system are satisfied.*

2.2 Conventional Mixed-Criticality Scheduling

Vestal proposed the state-of-the-art model for mixed-criticality systems in his seminal work [21]. The key idea is to model the WCETs of tasks on all criticality levels. The WCETs of one task when considering from low to high criticality levels are strictly non-decreasing, assuming that the WCET on a high criticality level is typically more conservative than that on a low criticality level. A task is not allowed to exceed the WCET on its own criticality level. The WCET of task τ_i on criticality level χ is denoted by $C_i(\chi)$.

Based on this model, the problem studied in literature is to provide *dynamic guarantees* to all tasks based on their run-times: whenever any task τ_i exceeds its χ criticality WCET ($C_i(\chi)$), only tasks with criticalities higher than χ are guaranteed to meet their deadlines thereafter, and all other less critical tasks are *killed* or their services are *degraded* to guarantee critical tasks. The system is said to be *schedulable* if there exists a scheduling technique, which can provide such dynamic guarantees to all tasks.

3. QUANTIFY THE SAFETY METRIC

We present in this section how to quantify the safety on different criticality levels without and with task killing or service degradation. To the best of our knowledge, this has not been studied for mixed-criticality systems. Explicitly quantifying safety will enable the certification of mixed-criticality systems according to established safety standards.

3.1 Plain Safety Quantification

We first study the quantification of safety on different criticality levels without task killing or service degradation.Recall that we assume any job of τ_i can execute at most n_i times. In the worst-case, each job of τ_i will execute n_i times until we know it fails or not. Let us call n_i times of executions of one job of τ_i as *one round*. The following result is presented.

LEMMA 3.1. *Given the re-execution profile N for all tasks, the maximum number of rounds of τ_i that the time domain $[0,t]$ can accommodate, is given by:*

$$r_i(n_i,t) = \max\{\left\lfloor \frac{t - n_i \cdot C_i}{T_i} + 1 \right\rfloor, 0\}.^1 \quad (1)$$

*The probability-of-failure-per-hour (*PFH*) on criticality level χ can be upper-bounded by:*

$$pfh(\chi) = \sum_{\tau_i \in \tau_\chi} r_i(n_i,t) \cdot f_i^{n_i}, \quad t = 1 \text{ hour}. \quad (2)$$

PROOF. All proofs can be found in [13]. □

3.2 The Need for Task Killing

As convinced by existing results on conventional mixed-criticality scheduling [7], killing of less critical tasks will help to improve the schedulability of critical tasks. For our fault-tolerant mixed-criticality scheduling problem (Definition 2.1), we show here a concrete example that motivates the need of task killing *under our problem setup*. Our detailed scheduling algorithm will be presented in Section 4.

Table 2: Example 3.1 task set

τ	τ_1	τ_2	τ_3	τ_4	τ_5
χ	HI	HI	LO	LO	LO
T/D	60	25	40	90	70
C	5	4	7	6	8

EXAMPLE 3.1. *Given a set of 5 sporadic tasks as shown in Table 2 with task parameters in units of* ms*. The tasks have criticalities* HI *and* LO*, with* HI $\in$ {A,B,C} *and* LO $\in$ {D,E}*. The failure probability of each job for every task is assumed to be a constant 10^{-5}.*

Since the LO *criticality tasks are either level D or level E tasks, the* PFH *on the* LO *criticality is of no interest and we can set $n_3 = n_4 = n_5 = 1$. Furthermore, for the* HI *criticality tasks, we can derive according to (2) their minimal re-execution profiles: $n_1 = n_2 = 3$. In this case the calculated* PFH *on the* HI *criticality is 2.04×10^{-10}, which satisfies the safety requirement on the* HI *criticality level. This will lead to an unschedulable system as the the total system utilization is greater than 1: $U = 3 \times \sum_{\tau_i \in \tau_{\text{HI}}} \frac{C_i}{T_i} + \sum_{\tau_i \in \tau_{\text{LO}}} \frac{C_i}{T_i} = 1.08595$.*

However, since there is no requirement on the upper bound of PFH *for level D/E tasks (they are intrinsically not safety-related, and can be killed without jeopardizing the sytem safety). Hence, to guarantee the schedulability of* HI *criticality tasks, one may kill* LO *criticality tasks, when e.g. any* HI *criticality task instance executes a third time. As we will show in Section 4, this will indeed make the task set schedulable.*

The above example motivates from the schedulability point of view the need of killing less critical tasks in order to guarantee critical tasks. However, killing of less critical tasks should be assessed such that the less critical tasks can still meet their safety requirement. For example, if the low criticality tasks in Example 3.1 are criticality level C tasks, then they still need to meet the safety requirement on criticality level C (pfh(C) $< 10^{-5}$), even though they could be killed.

3.3 Safety Quantification with Task Killing

We proceed to present how to quantify the system safety on different criticalities when killing of LO criticality tasks is adopted. This is not a trivial problem, as the safety of those LO criticality tasks now depends on *when* they are killed by the HI criticality tasks.

Recall that for any instance of task τ_i, it is executed up to n_i times in order to meet the safety requirement. We assume in this paper that, for each HI criticality task τ_i, killing of LO criticality tasks is controlled by another parameter n_i' ($n_i' \in \mathbb{N} \wedge n_i' < n_i$): whenever an instance of τ_i executes the $(n_i'+1)$th time, all LO criticality tasks are killed thereafter. We call n_i' as the *killing profile* of the HI criticality task τ_i, and we use $N'_{\text{HI}} = \{n_i' \mid \chi_i = \text{HI}\}$ to denote the killing profile of all HI criticality tasks. Notice that N'_{HI} is required by the scheduling technique in order to guarantee schedulability.

We now first quantify the probability that the LO criticality tasks are killed within the time domain $[0,t]$ under this setting. Formally, we have the following result.

LEMMA 3.2. *Within the time domain $[0,t]$, the probability that* no *instance of any* HI *criticality task τ_i executes the $(n_i'+1)$*th *time, is lower-bounded by:*

$$R(N'_{\text{HI}}, t) = \prod_{\tau_i \in \tau_{\text{HI}}} (1 - f_i^{n_i'})^{r_i(n_i', t)}. \quad (3)$$

The probability that the LO *criticality tasks are killed within $[0,t]$ is upper-bounded by $1 - R(N'_{\text{HI}}, t)$.*

Notice that according to (3), $R(N'_{\text{HI}}, t)$ will decrease with increasing t. This implies that the probability that the LO criticality tasks will be killed increases as time elapses. As $R(N'_{\text{HI}}, t)$ can approach 0 if t is sufficiently large, this means that the LO criticality tasks will eventually be killed for sure. Based on Lemma 3.2, we now quantify the safety of the LO criticality tasks providing that they can be killed by the HI criticality tasks. The following result is presented.

LEMMA 3.3. *Given the re-execution profile N for all tasks and the killing profile N'_{HI} for the* HI *criticality tasks. Define $\pi_i(t)$ as a sequence of timing points unique to τ_i:*

$$\pi_i(t) = \{t - n_i C_i - m T_i + D_i \mid m \in \mathbb{N} \wedge m \geq 1 \wedge m < r_i(n_i, t)\} \cup \{t\}. \quad (4)$$

The PFH *on the* HI *criticality level can be calculated as shown in (2). The* PFH *on the* LO *criticality level can be upper-bounded by:*

$$\text{pfh(LO)} = \left(\sum_{\tau_i \in \tau_{\text{LO}}} \sum_{\alpha \in \pi_i(t)} 1 - R(N'_{\text{HI}}, \alpha) \cdot (1 - f_i^{n_i}) \right) \Big/ O_S \ , \quad (5)$$

where $t = O_S$ hours.

[1] We assume in this paper that each job of task τ_i takes C_i to finish at runtime. If this assumption does not hold, then C_i in (1) , (4) and (6) should be modified to 0.

The result as shown in Lemma 3.3 confirms that the safety of the LO criticality tasks under task killing depends on when they are killed by the HI criticality tasks (i.e. N'_{HI}). In particular, for any HI criticality task τ_i, if we decrease n'_i, then the PFH for the LO criticality tasks will be increased (safety comprised). On the contrary, if n'_i is increased, then the PFH of the LO criticality tasks will decrease (safety improved). This can be intuitively explained: with increasing killing profiles, the LO criticality tasks will be killed "less often", leading to improved system safety.

3.4 Service Degradation Instead of Task Killing

Analogous to our discussion in Example 3.1, service degradation of the LO criticality tasks can also help to alleviate the system load if the HI criticality tasks are re-executed "too many" times. In addition, for many real-life mixed-criticality applications, task killing may not be the best choice. For example, for the flight management system application, there are B criticality tasks for location computation and C criticality tasks for flightplan computation. However, as the system may constantly require the flightplan information, it would not be a good design choice to kill the flightplan tasks if any B criticality task overruns. Instead, a degraded service should be provided to the flightplan tasks.

We assume that service degradation is only allowed for the LO criticality tasks, and it is facilitated by a new inter-arrival time $\hat{T}_i$ ($\forall \tau_i \in \tau_{\mathrm{LO}}$). We use a factor d_f (> 1) to characterize the service degradation: $\forall \tau_i \in \tau_{\mathrm{LO}}, \hat{T}_i = d_f \times T_i$. Similar to task killing, the service degradation is triggered when any instance of a HI criticality task τ_i executes the $(n'_i + 1)$th time. We call n'_i the *degradation profile* of task τ_i in this case (N'_{HI} represents the degradation profile for all HI criticality tasks). For ease of presentation, we call both the killing and degradation profiles as the *adaptation* profile when this does not cause any ambiguity.

To safely facilitate service degradation, its impact on the system safety should also be quantified. We present the following result in this regarding.

LEMMA 3.4. *Given the re-execution profile N for all tasks, the degradation profile N'_{HI} for the* HI *criticality tasks, and the service degradation factor d_f for the* LO *criticality tasks. Define function $\omega(d_f, t)$ as follows:*

$$\omega(d_f, t) = \sum_{\tau_i \in \tau_{\mathrm{LO}}} \max\{\left\lfloor \frac{t - n_i \cdot C_i}{d_f \cdot T_i} + 1 \right\rfloor, 0\} \cdot f_i^{n_i}. \quad (6)$$

The PFH *on the* LO *criticality level, if service degradation is allowed, can be upper-bounded by:*

$$\mathrm{pfh(LO)} = (1 - R(N'_{\mathrm{HI}}, t)) \cdot \omega(1, t)/O_S \ , t = O_S \text{ hour}. \quad (7)$$

According to (7), one can verify that the PFH on the LO criticality level is decreased if service degradation is adopted as compared to (2) when it is not. This is intuitive: for increased inter arrival times of LO criticality tasks, we have "less" such tasks to execute within a given time interval, and the failure probability of LO criticality tasks is decreased.

4. FAULT-TOLERANT MIXED-CRITICALITY SCHEDULING

Clearly, task re-execution will have an impact on the actual run-times of tasks. From another point of view, depending on how many re-executions are performed, a list of different WCETs is generated for a task. Hence, if the LO criticality tasks are killed/degraded when the HI criticality tasks exceed certain times of re-execution, this can be *alternatively* viewed as the LO criticality tasks are killed/degraded when the HI criticality tasks exceed certain WCETs. In light of this, there exists an underlying link between our problem and the conventional mixed-criticality scheduling problem. We will show in this section how our problem can be solved leveraging existing results on mixed-criticality scheduling.

Table 3: Converted mixed-criticality task set (Example 3.1)

τ	τ_1	τ_2	τ_3	τ_4	τ_5
χ	HI	HI	LO	LO	LO
T/D	60	25	40	90	70
C(HI)	15	12	7	6	8
C(LO)	10	8	7	6	8

4.1 Problem Conversion

We first present a concrete example to show how our problem can be converted to a corresponding conventional mixed-criticality scheduling problem.

EXAMPLE 4.1. *Consider the same task set as presented in Example 3.1. Since any instance of a* HI *criticality task τ_i can execute up to 3 times, we can set its* HI *criticality WCET as $3C_i$. Furthermore, all* LO *criticality tasks are killed whenever any* HI *criticality task instance executes a third time. Therefore, the* LO *criticality WCET of any* HI *criticality task τ_i can be set as $2C_i$. For the* LO *criticality tasks, their* HI *and* LO *criticality WCETs can be set as their original WCETs (i.e. all* LO *criticality task instances can execute only once). In this way, we construct a conventional mixed-criticality task set for our problem, which is shown in Table 3. Notice that our problem conversion is* conservative*: if an instance of the* HI *criticality task τ_i exceeds $2C_i$ units of execution at runtime, we know that this instance is executing a third time. However, the reverse is not true as a task may take less than its WCET to finish.*

For the converted mixed-criticality task set, one can verify that, it can be scheduled by existing mixed-criticality scheduling techniques, e.g. EDF-VD [3]. Hence, we get a schedulable dual-criticality task set with the safety requirements met.

We now formalize our findings in Example 4.1.

LEMMA 4.1. *Given the re-execution profile N for all tasks and the adaptation profile N'_{HI} for the* HI *criticality tasks. Construct a conventional mixed-criticality task set by: 1) for any* HI *criticality task τ_i, let its* HI *criticality WCET be $n_i C_i$, and let its* LO *criticality WCET be $n'_i C_i$; 2) for any* LO *criticality task τ_i, let its* HI *criticality and* LO *criticality WCETs both be $n_i C_i$. Then the system safety and schedulability are guaranteed if: 1) the* PFH *on each criticality is satisfied under N and N'_{HI}; and 2) the converted mixed-criticality task set is schedulable.*

Based on Lemma 4.1, our problem is then twofold:

- How to decide the re-execution profile N for all tasks and the adaptation profile N'_{HI} for the HI criticality tasks?
- How to schedule the system?

For scheduling the system, we may use existing mixed-criticality scheduling techniques assuming the problem conversion. However, it is not trivial to solve both problems together, as they are *correlated*: how we choose the re-execution and adaptation profiles will have an impact on how the system can be scheduled. Therefore, it is necessary to first gain understandings of such impacts.

Algorithm 1: Pseudo code – $\mathcal{FT}$-$\mathcal{S}$

Input: τ, mixed-criticality scheduling algorithm $\mathcal{S}$

1 **foreach** $\chi \in \{\text{HI}, \text{LO}\}$ **do**
2 | $n_\chi \leftarrow \inf\{n \in \mathbb{N} : \text{pfh}(\chi) \leq \text{PFH}_\chi\}$ ((2));
3 **end**
4 $n^1_{\text{HI}} \leftarrow \inf\{n \in \mathbb{N} : \text{pfh(LO)} < \text{PFH}_{\text{LO}}\}$ ((5)/(7));
5 **if** $n^1_{\text{HI}} > n_{\text{HI}}$ **then**
6 | **return** FAILURE;
7 **end**
8 $n^2_{\text{HI}} = \sup\{n \in \mathbb{N} : \Gamma(n_{\text{HI}}, n_{\text{LO}}, n) \text{ schedulable by } \mathcal{S}\}$;
9 **if** $n^1_{\text{HI}} \leq n^2_{\text{HI}}$ **then**
10 | $n'_{\text{HI}} \leftarrow n^2_{\text{HI}}$;
11 | schedule the converted mixed-criticality tasks $\Gamma(n_{\text{HI}}, n_{\text{LO}}, n'_{\text{HI}})$ by $\mathcal{S}$;
12 | **return** SUCCESS;
13 **else**
14 | **return** FAILURE;
15 **end**

4.2 A General Scheduling Method

Intuitively, if we decrease the task re-execution or adaption profiles, the system schedulability can be improved: for decreasing re-execution profiles, the load of the system will also be decreased; while for decreasing adaptation profiles, the LO criticality tasks will be killed/degraded "more often", leading to "more help" on alleviating system loads. On the contrary, decreasing the task re-execution and adaptation profiles will lead to compromised system safety (Lemma 3.1, Lemma 3.3 and Lemma 3.4).

Thus, safety and schedulability are *"conflicting"* forces on choosing the re-execution profile for all tasks and the adaptation profile for the HI criticality tasks.In order to simplify the problem, we restrict that all tasks of the same criticality have the same re-execution profile, i.e. $\forall \tau_i, \tau_j \in \tau_\chi, n_i = n_j$. Furthermore, we assume that the adaptation profiles for all HI criticality tasks are the same, i.e. $\forall \tau_i, \tau_j \in \tau_{\text{HI}}, n'_i = n'_j$. We use n_{HI} (n_{LO}) to denote the re-execution profile for each HI (LO) criticality task. And we use n'_{HI} to denote the adaptation profile for each HI criticality task. We further use a function $\Gamma(n_{\text{HI}}, n_{\text{LO}}, n'_{\text{HI}})$ to denote the constructed mixed-criticality task set according to the re-execution and adaptation profiles (Lemma 4.1).

We now explain our proposed algorithm (as shown in Algorithm 1). We assume a mixed-criticality scheduling technique $\mathcal{S}$ is used to schedule the system (the converted mixed-criticality task sets). We first calculate the *minimal* re-execution profiles for all tasks with the guarantee of safety on both criticality levels. For this first step calculation, we assume that no task will be killed/degraded (line 1-3). We then calculate the adaptation profiles for the HI criticality tasks, providing that the safety requirement on the LO criticality level can still be satisfied. For this purpose, we first calculate according to (5) (or (7), if service degradation is adopted instead of task killing) the *minimal* adaptation profiles (n^1_{HI}) for all HI criticality tasks, which will still guarantee the safety of the LO criticality tasks (line 4). We then calculate from the schedulability point of view, the *maximal* adaptation profiles (n^2_{HI}) for all HI criticality tasks in order to guarantee the schedulability of the system (line 8). Based on n^1_{HI} and n^2_{HI}, our algorithm can then decide the feasible adaptation profiles for the HI criticality tasks (line 9-12).

Notice that the calculation of n^2_{HI} depends on $\mathcal{S}$, we will give detailed explanations on how to compute it for some well-known mixed-criticality scheduling algorithms in [13]. Notice also that our proposed algorithm is general in the sense any mixed-criticality scheduling algorithm can be integrated. Formally, our result in this section is summarized as follows:

THEOREM 4.1. *Given a dual-criticality task set τ, the failure probability f_i for any instance of each task τ_i ($\forall \tau_i \in \tau$) and a mixed-criticality scheduling technique $\mathcal{S}$. Both safety and schedulability of the system are satisfied if the $\mathcal{FT}$-$\mathcal{S}$ algorithm signals success.*

5. EVALUATION

5.1 Flight Management System

We first validate the proposed techniques with a real-life flight management system (FMS) application. In particular, we evaluate the impacts of task killing and service degradation on the system safety and schedulability. The considered FMS is a subset of the original FMS and consists of 11 criticality level B and C tasks. The failure probability of each task's instance is assumed to be a constant (10^{-5}) for our experiments. We select EDF-VD [3] with task killing and its variant [12] with service degradation as our explored mixed-criticality scheduling techniques ([13]). The detailed experiment setup can also be found in [13].

According to Algorithm 1, the re-execution profiles are set as the minimal profiles ($n_{\text{HI}} = 3, n_{\text{LO}} = 2$), such that safety on both criticality levels is satisfied *without* task killing or service degradation. The FMS application is not schedulable with the task re-execution profiles. Hence, we consider the option of killing or degrading the level C tasks in order to improve schedulability. To evaluate the impact of task killing and service degradation on the system schedulability, we use the mixed-criticality system utilization metric (U_{MC}, with detailed explanation given in [13]).

Our results are shown in Fig. 1 and Fig. 2. First, we see that with increasing adaptation profiles for the HI criticality tasks (i.e. with increasing n'_{HI}), U_{MC} will continuously increase. This is because the system load is increased in scenarios when the LO criticality tasks are killed/degraded "less often". We see that, in both cases, the system will no longer be schedulable when $n'_{\text{HI}} > 2$ ($n'_{\text{HI}} \in \mathbb{N}$). Second, we see that, with increasing n'_{HI}, the PFH on the LO criticality level under task killing or service degradation can be decreased, i.e. safety can be improved. This also matches with our analysis: with increasing n'_{HI}, the chance that the LO criticality tasks will be killed or degraded is decreased, leading to enhanced safety. Furthermore, we see that task killing has stronger impact on the system safety than service degradation, e.g. when $n'_{\text{HI}} = 2$, if task killing is adopted, then the order of magnitude of pfh(LO) is 10^{-1}, compared to 10^{-11} when service degradation is adopted. This is due to the reason that, if tasks are killed, there is no functionality/safety they can still deliver. This also suggests that service degradation is more proper than task killing from the safety point of view. Notice that in both cases, U_{MC} is incomparable as different schedulability analyses are used.

5.2 Extensive Simulation

It is known in theoretical study of conventional mixed-criticality scheduling [7] that task killing and service degradation can improve the system schedulability, assuming that the LO criticality tasks can be "safely" killed or degraded.

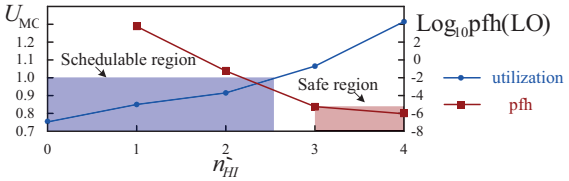


Figure 1: The impacts of task killing

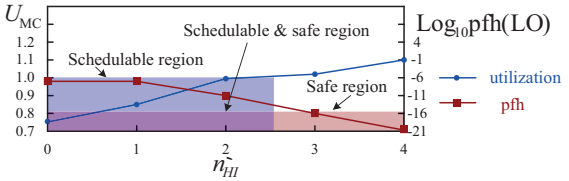


Figure 2: The impacts of service degradation

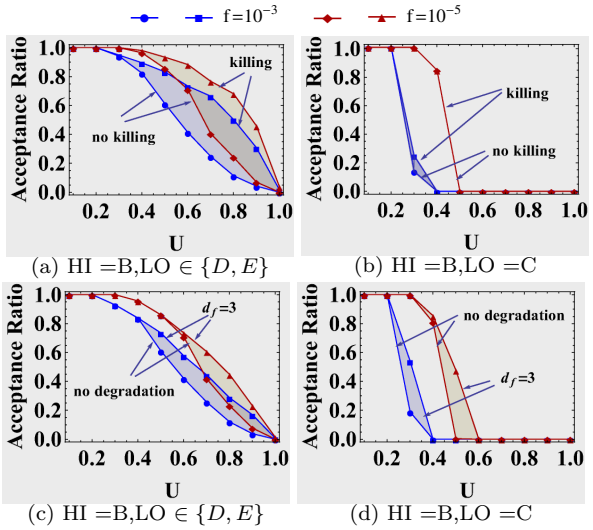
We will investigate now, with explicit quantification of safety under task killing or service degradation, their impacts on the system schedulability. For this purpose, we apply our algorithms to synthetic dual-criticality task sets (500 at each data point) and compare the acceptance ratios (ratio of the number of schedulable task sets to the number of tested task sets) under different settings. Details of our experiment setup can be found in [13]. Our results are shown in Fig. 3, where x-axis represents the system utilization and y-axis represents the acceptance ratio. f represents the universal probability of failure for all task instances.

Based on the experimental results, we see that with safer and more expensive hardware (decreased f), the system schedulability will be improved. We further observe:

- As shown in Fig. 3a and Fig. 3c,if the LO criticality tasks are not relevant to the system safety (i.e. they have criticality level D or E), then the system schedulability for our problem can also be improved considerably by adopting task killing or service degradation.
- According to the results Fig. 3b, if the LO criticality tasks have explicit safety requirements (e.g. they have criticality level C in our experiments), then killing those tasks could rarely help the design of fault-tolerant mixed-criticality systems. The underlying reason is that task killing could directly violate the system safety.
- Service degradation is more proper than task killing if safety is a concern for the LO criticality tasks: by comparing Fig. 3d to Fig. 3b, we see that service degradation helps more to improve the system schedulability in this case. This observation also matches our analysis: service degradation has less effects on the system safety. In fact according to Lemma 3.4, the safety on the LO criticality level can only be improved with service degradation.

6. CONCLUSION

We study in this paper fault-tolerant mixed-criticality scheduling under hardware transient faults. We explicitly model the safety requirements on different criticality levels according to established safety standards. We analytically bound the impacts of task re-execution, task killing and service degradation on the system safety and schedulability. We propose a scheduling algorithm for our problem and show that it can be converted to a conventional mixed-criticality scheduling problem. Our proposed techniques are validated with a real-life flight management system application and extensive simulations.



(a) HI =B,LO $\in \{D, E\}$ (b) HI =B,LO =C

(c) HI =B,LO $\in \{D, E\}$ (d) HI =B,LO =C

Figure 3: Schedulability evaluation – shadows represent schedulability gaps (Fig. 3a and Fig. 3b – with/without task killing, Fig. 3c and Fig. 3d – with/without degradation)

References

[1] RTCA/DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.

[2] P. Axer, M. Sebastian, and R. Ernst. Reliability analysis for mpsocs with mixed-critical, hard real-time constraints. CODES+ISSS, 2011.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, 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 *ECRTS*, 2012.

[4] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS*, 2008.

[5] C. Bolchini and A. Miele. Reliability-driven system-level synthesis for mixed-critical embedded systems. 2012.

[6] S. Brown. Overview of iec 61508. design of electrical/electronic/programmable electronic safety-related systems. *Computing & Control Engineering Journal*, 2000.

[7] A. Burns and R. Davis. Mixed criticality systems-a review. 2013.

[8] A. Burns, R. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *Real-Time Systems*, 1996.

[9] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *ECRTS*, 2012.

[10] J. Huang, J. Blech, A. Raabe, C. Buckl, and A. Knoll. Reliability-aware design optimization for multiprocessor embedded systems. In *DSD*, 2011.

[11] J. Huang, A. Raabe, K. Huang, C. Buckl, and A. Knoll. A framework for reliability-aware design exploration on mpsoc based systems. *Design Automation for Embedded Systems*, 2013.

[12] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. In *Design Automation Conference (ASP-DAC)*, Jan 2014.

[13] P. Huang, H. Yang, and L. Thilele. On the scheduling of fault-tolerant mixed-criticality systems. Technical Report 351, ETH Zurich, Laboratory TIK, Dec 2013.

[14] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems. In *DATE*, 2005.

[15] A. Jhumka, S. Klaus, and S. Huss. A dependability-driven system-level design approach for embedded systems. In *DATE*, 2005.

[16] J. C. Knight. Safety critical systems: challenges and directions. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, 2002.

[17] H. Li and S. Baruah. Load-based schedulability analysis of certifiable mixed-criticality systems. In *EMSOFT*, 2010.

[18] T. Park and S. Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *EMSOFT*, 2011.

[19] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *ECRTS*, 2012.

[20] L. Sha. Resilient mixed-criticality systems, 2009.

[21] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, 2007.