Towards the Design of Fault-Tolerant Mixed-Criticality Systems on Multicores

Luyuan Zeng
zengl@student.ethz.ch

Pengcheng Huang
phuang@tik.ee.ethz.ch

Lothar Thiele
thiele@tik.ee.ethz.ch

Computer Engineering Group (TEC), ETH Zurich

ABSTRACT

Mixed-criticality is a significant recent trend in the embedded system industry, where common computing platforms are utilized to host functionalities of varying criticality levels. To date, most scheduling techniques have focused on the timing aspect of this problem, while functional safety (i.e. fault-tolerance) is often neglected.

This paper presents design methodologies to guarantee both safety and schedulability for real-time mixed-criticality systems on identical multicores. Assuming hardware/software transient errors, we model safety requirements on different criticality levels explicitly according to safety standards; based on this, we further propose fault-tolerant mixed-criticality scheduling techniques with task replication and re-execution to enhance system safety. To cope with runtime urgencies where critical tasks do not succeed after a certain number of trials, our techniques can perform system reconfigurations (task killing or service degradation) in those situations to reallocate system resources to the critical tasks. Due to explicit modeling of safety, we can quantify the impact of task killing and service degradation on system feasibility (safety and schedulability), enabling a rigorous design. To this end, we derive analysis techniques when reconfigurations are triggered either globally (synchronously) on all cores or locally (asynchronously) on each core. To our best knowledge, this is the first work on fault-tolerant mixed-criticality scheduling on multicores, matching theoretical insights with industrial safety standards. Our proposed techniques are validated with an industrial application and extensive simulations.

Keywords

Reliability, Modeling, Multicores

1. INTRODUCTION

Complex embedded systems are often mixed-critical, where applications of different criticality (importance) levels coexist on the same computing platform [21, 28]. For instance, the avionics industry is currently undergoing a significant trend to consolidate functionalities of different criticality levels (e.g. flight control of DO-178B [3] criticality A and flight management of criticality B, with A being the highest and B being the second highest criticality levels in DO-178B) onto the same hosting platform to reduce system cost, weight and energy [15]. For mixed-criticality systems [21], it is of vital importance to provide different levels of assurance for applications of different criticality levels. Such a task is challenging, because tasks of different criticality levels could mutually interfere with each other on the shared resources (e.g. CPU and memory systems) [11], jeopardizing their guarantees made in isolation.

To facilitate the design of mixed-criticality systems, the industry often follows well-established safety standards, e.g. DO-178B [3] for avionics and ISO 26262 [2] for automotive, with the goal to guarantee both functional safety and non-functional (real-time) requirements on all criticality levels. For functional safety, various stresses (e.g. hardware/software errors) [21] need to be mitigated through different hardening techniques including task re-execution and replication [9, 19]. For real-time guarantees, while conventional techniques favor strict temporal and spatial isolations among varying criticality levels [16, 25], recent advances in mixed-criticality advocate asymmetric isolation among them [11] – whenever timing threats (task overrun) are detected, less critical tasks are degraded and their occupied resources are freed to guarantee more critical tasks.

1.1 Motivation

To a large extent, guaranteeing functional safety and timeliness for mixed-criticality systems have been studied in isolation in the literature [11]. In particular, making the above two guarantees on a commercial-off-the-shelf (COTS) platform is still not much explored. To date, the mixed-criticality community has mainly focused on the scheduling aspect of such a challenging problem [11], e.g. scheduling techniques and schedulability analysis [6–8, 14, 23, 26, 27, 29], communication methods and analysis [12, 13], and interference analysis with respect to shared resources [16, 30].

Other works, though trying to address both safety and schedulability together [9, 19, 21], might incur some drawbacks. In [21], the authors tackled this problem on unicores. However, their techniques do not trivially extend to multicores due to additional complexities related to task replication and scheduling on multicores. In [9, 19, 22], design methodologies are provided to ensure both system safety and schedulability on multicores. However, they do not model safety explicitly according to common safety standards, see e.g. DO-178B [3] and ISO 26262 [2]. Furthermore, mixed-criticality scheduling techniques have some peculiarities that they can react to runtime urgencies by performing reconfigurations, i.e. by killing less critical tasks or degrading their services. To our best knowledge, the impact of such reconfigurations on system safety has not been considered for multicore platforms yet.

CASES '2016 Pittsburgh, Pennsylvania USA

 ISBN 978-1-4503-4482-1/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2968455.2968515

1.2 Contribution

We tackle in this paper mixed-criticality fault-tolerant scheduling on identical multicore platforms, where both system safety and schedulability requirements must be satisfied. Focusing on hardware/software transient errors and dual-criticality systems, task re-execution (on each core) and replication (on multiple cores) are considered as our fault-tolerance mechanisms. For this class of systems, we provide fault-tolerance mechanisms, we provide analysis techniques that can safely bound the impact of run-time system reconfigurations on safety and schedulability. In detail, our contributions can be summarized as follows:

- We explicitly model system safety requirements by pfh (probability-of-failure-per-hour), as commonly used in safety standards [3, 10].
- We formulate our problem using redundancy and adaptation profiles, where task re-execution, task replication and system reconfigurations are considered jointly to achieve a feasible design. Based on this, we propose a problem transformation, leveraging classical mixed-criticality scheduling techniques to guarantee task deadlines.
- We develop diverse analysis techniques to bound system safety when system reconfigurations are triggered on each core locally or on all cores globally; this enables the trade-off between analysis complexity and system feasibility. Our analytical solutions have significantly extended the original techniques in [21].
- We validate our proposed techniques with a realistic flight management system [21] and extensive simulations, where the impact of online reconfigurations on the overall system feasibility is evaluated. Our results reveal quantitatively that service degradation for mixed-criticality systems can greatly improve system feasibility, while task killing only helps if performed on non-safety related tasks.

The remainder of this paper is organized as follows. In Section 2, we introduce the background and define our studied problem. Section 3 and 4 describe two diverse analysis techniques to bound system safety. Section 5 presents our evaluation results while Section 6 concludes this paper.

2. BACKGROUND & PROBLEM STATEMENT

In this section, we first present some background on mixed-criticality systems and scheduling. We subsequently introduce our fault models, common safety requirements as found in [3, 10] and corresponding fault-tolerance mechanisms. We finally present a concrete problem definition. All notations of our mixed-criticality system can be found in Table 1.

2.1 Mixed-Criticality Task Model

We consider dual-criticality systems; one such system consists of $|\tau|$ independent sporadic tasks $\{\tau_\mathrm{i}|\tau_\mathrm{i} \in \tau\}$ running on K identical cores $P = \{p_1, \cdots, p_\mathrm{k}, \cdots, p_\mathrm{K}\}$. Each task is characterized by a 4-tuple $\tau_\mathrm{i} = (T_\mathrm{i}, D_\mathrm{i}, C_\mathrm{i}, \chi_\mathrm{i})$, where T_i is the minimal inter-arrival time and D_i is the relative deadline ($D_\mathrm{i} \leq T_\mathrm{i}$). C_i is the single worst case execution time (WCET) of τ_i. χ_i is the criticality level (being either high (HI) or low (LO)) of τ_i. For notational convenience, we use $\mathcal{X}$ to denote the set of existing criticality levels and $\tau(\chi)$ to represent all tasks with criticality level $\chi \in \mathcal{X}$. In addition, we assume that all tasks run for a system operation duration of O_s hours, after which the system state is reset.

2.2 Fault Model and Safety Requirements

Due to intrinsic transient hardware/software errors [21], for any instance of task τ_i, we assume there is a probability that it does not finish successfully, denoted as f_i.

notation	description
τ	a set of independent sporadic tasks τ_i
$P = \{p_1, \cdots, p_\mathrm{k}, \cdots, p_\mathrm{K}\}$	a set of K identical cores
$\mathcal{X} = \{\mathrm{HI}, \mathrm{LO}\}$	criticality levels: high level (HI) and low level (LO)
$C_\mathrm{ik}(\chi)$	WCET of task τ_i on criticality level χ on core p_k
$\tau(\chi)$	set of tasks with criticality χ
τ_k	set of tasks executed on core p_k
$\tau_\mathrm{k}(\chi)$	set of tasks with criticality level χ on core p_k
$N : \tau \times P \to \mathbb{N}$	Redundancy Profile (Definition 2.1)
n_ik	abbreviation of $N(\tau_\mathrm{i}, p_\mathrm{k})$
n_i	$n_\mathrm{i} = \sum_{k=1}^{K} n_\mathrm{ik}$
$\mathrm{pfh}(\chi)$	probability of failure per hour of tasks with criticality level χ
f_i	probability that any instance of task τ_i fails with C_i units of execution
d_f	service degradation factor
$N' : \tau(\mathrm{HI}) \times P \to \mathbb{N}$	Adaptation Profile (Definition 2.2)
n'_ik	abbreviation of $N'(\tau_\mathrm{i}, p_\mathrm{k})$
$\Gamma(\tau, P, N(\cdot), N'(\cdot))$	constructed mixed-criticality task set of τ based on its $N(\cdot)$ and $N'(\cdot)$.

Table 1: Important notations

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

Table 2: DO-178B safety requirements [21]

Because of the potential faults, safety measures need to be applied to indicate how safe a system is. We use the probability-of-failure-per-hour (pfh), which is widely adopted in safety standards [3, 10]; pfh can be efficiently calculated as the average system failure rate in an hour [10, 21]. According to the general safety standard IEC 61508 [10], pfh estimates failure probability of safety functions and is further specified on each criticality level.

In this paper, we follow the DO-178B [3] safety standard, which defines 5 criticality levels (A is the highest and E is the lowest). The corresponding safety requirements are shown in Table 2. As we can see, pfh decreases with increasing χ, i.e. safety requirements are more stringent on higher criticality levels. The results of our paper can be applied to a system that contains any two criticality levels in DO-178B.

2.3 Fault Tolerance

To mitigate errors and to enhance system safety, two established approaches exist. First, one could detect such errors at runtime by performing sanity-checks [9, 19, 22]. We suppose at the end of a job's WCET, it is known whether the job has failed or not, and re-execute a faulty job in the hope that it will succeed. Second, one could replicate the job a priori on multiple processors, enhancing the chance that one of them will succeed. We assume both in this paper: Each instance of any task is replicated on multiple cores offline due to safety requirements, where migrating task instances from one core to another at runtime is excluded in the analysis due to high runtime overheads [24]; in case faults are detected on one core, the faulty task instance is re-executed locally up to a given number of times to achieve the required safety. Note that, if a replicate of one task's instance succeeds on some core, replicates on the other cores actually do not need to proceed. In addition, not all tasks need to have replicas. However, for our safety analysis, we need to consider the worst-case (all replicas of each task instance fail) to bound the failure probability.

We proceed to introduce some notations. We use τ_k to represent all tasks mapped to core p_k, and assume any instance of task τ_i can execute at most n_ik times on core p_k and at most n_i times on all cores, such that $\sum_{k=1}^{K} n_\mathrm{ik} = n_\mathrm{i}$,

where we call n_{ik} the re-execution profile of task τ_i on core p_k. Formally, we can define a function N to jointly represent task replication and re-execution (referred to as the system redundancy profile), for all LO and HI criticality tasks.

Definition 2.1 (Redundancy Profile). *We define the redundancy profile $N : \tau \times P \to \mathbb{N}$, where $N(\tau_i, p_k)$, abbreviated as n_{ik}, is the maximum number of executions of any instance of task τ_i on core p_k.*

Note that, with the redundancy profile, we have a general problem formulation: if a task τ_i is not replicated on core p_k, then $n_{ik} = 0$; if a task τ_i has no replicas, then $n_i = 1$. Furthermore, it is subject the system safety and schedulability analysis to find the feasible redundancy profile, as elaborated in the remainder of this paper.

For notational convenience, we use $N_k(\chi) = \{n_{ik} \mid \tau_i \in \tau_k \wedge \chi_i = \chi\}$ to denote the redundancy profile of all χ criticality tasks on core p_k.

2.4 System Reconfiguration

Real-time embedded systems are typically resource constrained, while task replication and re-execution are costly in terms of required resources. To reallocate resources occupied by less critical tasks to more critical ones when there is an urgency (i.e. when any critical task instance does not succeed after a certain number of trials), a straightforward approach is to drop less critical tasks or degrade their services. We note that the embedded system industry is already investigating/adopting such runtime adaptations in safety-critical or mixed-criticality systems, see e.g. [4, 31].

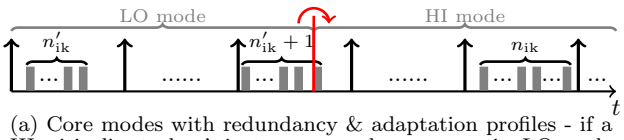
However, this would naturally affect the safety of the less critical tasks and might only work if those tasks are not safety relevant, e.g. LO = E as shown in Table 2. Otherwise, the impact of system reconfigurations on safety needs to be addressed. It is important to notice that the replicates and re-executions of all LO criticality tasks execute unobstructedly if system reconfigurations are not triggered.

To formally model reconfiguration on each core, we use $\tau_k(\chi)$ to denote all $\mathcal{X}$ criticality tasks executed on core p_k. For $\tau_i \in \tau_k(\text{HI})$, we assume that dropping LO criticality tasks or degrading their services is controlled by the parameter n'_{ik} ($n'_{ik} \in \mathbb{N} \wedge n'_{ik} \leqslant n_{ik}$): If no instance of any τ_i executes the $(n'_{ik}+1)$th time on p_k, we say p_k is in a normal mode and all local tasks can be guaranteed; otherwise, we say it is in an urgent mode and system reconfigurations must be performed. We call n'_{ik} the adaptation (killing or service degradation) profile of any HI criticality task τ_i on core p_k. We can now define a function N' to specify the system adaptation profile for all HI criticality tasks.

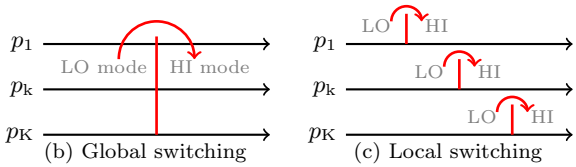
Definition 2.2 (Adaptation Profile). *We define the adaptation profile $N' : \tau(\text{HI}) \times P \to \mathbb{N}$. If any instance of τ_i executes $(n'_{ik}+1)$th time on any core p_k, then all LO criticality tasks on p_k are immediately dropped or degraded with a service degradation factor d_f.*

The service degradation factor d_f characterizes the new minimal inter-arrival time of any task $\tau_i \in \tau(\text{LO})$ after degradation (i.e. the minimal inter-arrival time T_i increases to $d_f \cdot T_i$). We assume its value is provided by the system designer; our proposed analysis techniques in this paper will further help to validate whether the provided degradation factor is feasible or not regarding both, system schedulablity and safety. For notational convenience, we denote with $N'_k(\text{HI}) = \{n'_{ik} \mid \tau_i \in \tau_k(\text{HI})\}$ the adaptation profile of all HI criticality tasks on core p_k. With $N'_k(\text{HI})$, we define LO and HI modes on any core p_k as follows.

Definition 2.3 (Core Mode). *If no instance of any $\tau_i \in \tau_k(\text{HI})$ executes the $(n'_{ik}+1)$th time, then p_k is in LO mode. Otherwise, p_k is in HI mode, see Figure 1a.*



(a) Core modes with redundancy & adaptation profiles - if a HI criticality task τ_i's instance exceeds on core p_k its LO mode re-execution profile (n'_{ik}, also called its adaptation profile), p_k enters HI mode where any HI criticality task τ_i's job can execute up to n_{ik} times (HI mode re-execution profile) and LO criticality tasks are dropped or degraded (Definition 2.2).



(b) Global switching (c) Local switching

Figure 1: Core modes, global & local switching

Moreover, it is still open how different cores coordinate their mode switch from LO to HI mode. There exists options to either trigger system reconfigurations synchronously on all cores (global switching) or asynchronously on each core (local switching). We formally define the options as follows.

Definition 2.4 (Global Switching). *All cores in our system switch from LO to HI mode (Definition 2.3) at the same time synchronously, see Figure 1b.*

Definition 2.5 (Local Switching). *Any core in our system switches from LO to HI mode (Definition 2.3) independently of other cores, see Figure 1c.*

As we will show later on in this paper, we will explore the tradeoffs between global and local switching – global switching incurs much reduced system states to be explored when analyzing system safety, as compared to local switching. However, global switching is pessimistic as reconfiguration on any core can be triggered by any other core.

2.5 Fault-Tolerant Mixed-Criticality Scheduling

We have introduced above the redundancy and adaptation profiles to achieve a safe and efficient design of mixed-criticality systems, assuming transient errors. The remaining question is how to schedule a system accordingly.

Our key observation is that the scheduling problem here can be directly *transformed* into a classical mixed-criticality scheduling problem. We now first briefly discuss the classical scheduling problem [11], we then establish a link between the fault-tolerant mixed-criticality scheduling problem and the well-know mixed-criticality scheduling problem.

Classical mixed-criticality scheduling problem. This problem [11] deals with the different timing guarantees made on different criticality levels – for higher criticality tasks, more pessimistic WCETs are used since those tasks are more important. To further increase system resource efficiency, a dynamic mode-switched scheduling approach is often used. Analogous to Section 2.4, the system starts with a nominal (LO) mode, where all tasks are guaranteed with their "normal" WCETs $(C_i(\text{LO}), \forall \tau_i)$. Whenever any HI criticality task overruns its LO mode WCET, the system enters the HI mode. Thereafter, HI criticality tasks are allowed more pessimistic WCETs $(C_i(\text{HI}), \forall \tau_i \in \tau(\text{HI}))$, while LO criticality tasks are dropped or degraded. In this context, both partitioned [17] and global [20] scheduling techniques are proposed for mixed-criticality systems on multicores. Note that tasks of different criticality levels are *asymmetrically isolated* as LO criticality tasks cannot interfere with HI criticality tasks, while the opposite is allowed. This improves resource

efficiency and differs from industrial approaches which enforce static temporal and spatial isolations [3].

Establish the link. For the problem in this paper, first, since task instances are statically mapped to each core and task migration is forbidden due to excessive cost (see Definition 2.1), it follows that we have a classical partitioned scheduling problem. Second, on each core p_k, system reconfiguration is triggered when any HI criticality task τ_i's instance executes too many times, see Definition 2.2. We can translate this into a sufficient condition that, whenever any HI criticality task τ_i's instance exceeds its LO WCET $C_{ik}(\text{LO}) = n'_{ik}C_i$ on core p_k, then system reconfiguration is triggered, as similarly done in [21]. Afterwards, core p_k is in HI mode, and we guarantee a HI criticality WCET for any HI criticality task τ_i, defined as $C_{ik}(\text{HI}) = n_{ik}C_i$. Furthermore, any instance of a LO criticality task τ_i cannot exceed its allocated number of redundancies, and we have $C_{ik}(\text{HI}) = C_{ik}(\text{LO}) = n_{ik}C_i$ for all LO criticality tasks. We shall use $C_{ik}(\chi)$ to uniformly denote the WCET of τ_i on χ criticality level and on core p_k. Notice that, the original mixed-criticality problem deals with timing errors (i.e. task overrun), thus it uses multiple WCETs for a task on different criticality levels. In our problem, we focus on hardware/software transient errors, and task re-executions are modeled as overrun in the standard mixed-criticality model.

In summary, the impact of reconfigurations on schedulability is considered implicitly assuming classical mixed-criticality scheduling techniques. For scheduling in our problem, a mixed-criticality task set $\Gamma(\tau, P, N(\cdot), N'(\cdot))$ can be constructed given τ, P, $N(\cdot)$ and $N'(\cdot)$, as discussed above. In this notation, any task can be duplicated into multiple subtasks running on different cores due to replication. Thus, we can adopt well-studied mixed-criticality scheduling techniques [11] to schedule tasks locally on each core. Due to the intrinsic computational intractability of the mixed-criticality scheduling problem [11], we focus in this paper on the partitioned EDF-VD method [7], where EDF-VD [6] scheduling is adopted locally on each core. However, such an approach can be easily extended to consider other scheduling techniques. EDF-VD follows the standard mixed-criticality model and adopts EDF scheduling in both LO and HI modes. In addition, it uses shortened virtual deadlines (VD) for HI criticality tasks in LO mode, such that enough slack time is left between the virtual and actual deadlines to accommodate job overrun. Finally, we define the problem studied in this paper as follows.

Definition 2.6 (Fault-Tolerant Mixed-Criticality Scheduling on Multicores (FMSM))**.** *Given a set of identical cores P, a dual-criticality sporadic task set τ and the probability of failure f_i for any instance of each task τ_i. Assume partitioned EDF-VD scheduling [7], find $N(\cdot)$ (Definition 2.1), $N'(\cdot)$(Definition 2.2), such that both safety and schedulability of $\Gamma(\tau, P, N(\cdot), N'(\cdot))$ are satisfied with global switching (Definition 2.4) or local switching (Definition 2.5).*

3. SAFETY QUANTIFICATION UNDER GLOBAL SWITCHING

We quantify in this section system safety on different criticality levels assuming global switching (Definition 2.4), with or without task killing or service degradation.

3.1 Basic Safety Quantification

First, we study safety quantification on different criticality levels without applying task killing or service degradation, i.e. basic safety quantification. For HI criticality tasks, this is apparently needed as they can never be killed or degraded in our model; for LO criticality tasks, basic safety quantification is also interesting as it will serve as a baseline to justify the other cases when reconfigurations are applied.

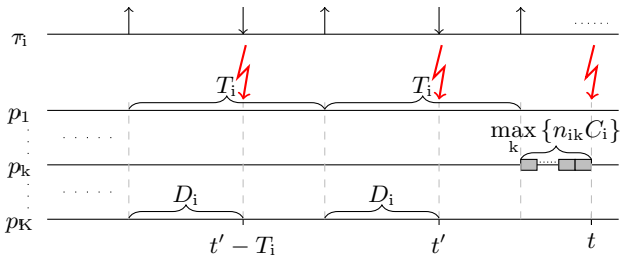


Figure 2: Induction process for Lemma 3.1

Recall that any instance of τ_i can execute at most n_{ik} times on core p_k and n_i times on all cores for one job arrival. We refer to n_i times of executions of one arrived instance of τ_i as *one round*. In the worst-case, a round of τ_i fails if all n_i instances of τ_i fail for this round. To further quantify the worst-case pfh in any interval, we need to upper bound the maximum number of rounds any task τ_i can accommodate in any interval. Formally, we have the following result.

Lemma 3.1. *Given τ, P and $N(\cdot)$, the maximum number of rounds that τ_i can fail (or accommodate) in any time interval of length t, $r_i(N(\cdot), t)$, is given by:*

$$r_i(N(\cdot), t) = \max\left\{\left\lfloor (t + D_i - \max_k\{n_{ik}\} \cdot C_i) \Big/ T_i \right\rfloor, 0\right\} + 1. \quad (1)$$

Proof. This lemma is proved by an induction process as shown in Figure 2. The shortest time interval length that can accommodate one round failure is simply zero, supposing the failure just falls in this interval (say, it happens at time instant t). In order to have a shortest time interval to accommodate one more round failure, we have to let the two failures happen as close as possible. The latest time instant the last round can start is $\left(t - \max_k\{n_{ik}\} \cdot C_i\right)$, hence the second last round starts at $\left(t - \max_k\{n_{ik}\} \cdot C_i - T_i\right)$ and fails at $\left(t' = t - \max_k\{n_{ik}\} \cdot C_i - T_i + D_i\right)$, which is the time instant that this round ends and the latest time instant it can fail. Similarly, the third last round fails at time instant $(t' - T_i)$. Therefore, the shortest time interval lengths that can accommodate one, two and three failures of τ_i are zero, $\left(\max_k\{n_{ik}\} \cdot C_i + T_i - D_i\right)$ and $\left(\max_k\{n_{ik}\} \cdot C_i + 2 \cdot T_i - D_i\right)$, respectively. By induction, the shortest length of time interval that can accommodate n round failures of τ_i equals

$$\max\left\{\left(\max_k\{n_{ik}\} \cdot C_i + (n-1) \cdot T_i - D_i\right), 0\right\}. \quad (2)$$

Setting (2) equal t to solve n, we get the maximum number of rounds that τ_i can accommodate in any time interval of length t, as shown in (1). Note that we use floor($\lfloor\ \rfloor$) here to calculate the value of $(n-1)$. □

Notice that Lemma 3.1 derives the worst-case number of rounds for any task in a time interval *on all cores*. Similarly, we can also upper-bound the local rounds of τ_i in any time interval on a core p_k, as shown in the following result.

Corollary 3.1. *Consider core p_k, given τ and $N(\cdot)$, the maximum number of local rounds τ_i can accommodate in any time interval of length t, $r_{ik}(n_{ik}, t)$, is given by:*

$$r_{ik}(n_{ik}, t) = \max\{\lfloor (t + D_i - n_{ik} \cdot C_i)/T_i \rfloor, 0\} + 1. \quad (3)$$

Proof. Proof can be similarly derived as in Lemma 3.1. □

With Lemma 3.1, we can upper-bound the pfh on any criticality level. The worst-case failure rate of any task τ_i in a given time interval happens when this interval accommodates the maximum number of rounds of τ_i and each round fails; adding up those failure rates over all tasks of one critical level, we get the pfh on this criticality level. This is summarized into the following result.

Theorem 3.1. *Given τ, P and $N(\cdot)$, pfh(χ) (probability-of-failure-per-hour on χ criticality) can be upper bounded by:*

$$pfh(\chi)^1 = \sum_{\tau_i \in \tau(\chi)} r_i(N(\cdot), t) \cdot f_i^{n_i}, \quad t = 1 \ hour. \tag{4}$$

Proof. $\mathrm{pfh}(\chi)$ does not vary from one hour to the next as the number of rounds issued by the same task is the same for two consecutive hours and each round is subject to the same constant probability of failure [21]. For any task $\tau_\mathrm{i} \in \tau(\chi)$, the probability that τ_i does not fail in any time interval of length t is lower bounded by:

$$(1 - f_\mathrm{i}^{n_\mathrm{i}})^{r_\mathrm{i}(N(\cdot),t)}. \tag{5}$$

Thus, the probability that any $\tau_\mathrm{i} \in \tau(\chi)$ fails at any time interval of length t, can be upper bounded by:

$$1 - \prod_{\tau_\mathrm{i} \in \tau(\chi)} (1 - f_\mathrm{i}^{n_\mathrm{i}})^{r_\mathrm{i}(N(\cdot),t)}. \tag{6}$$

Due to $f_\mathrm{i}^{n_\mathrm{i}} \ll 1$, we only consider its first order terms. Hence (6) can be tightly upper-bounded by:

$$1 - \left(1 - \sum_{\tau_\mathrm{i} \in \tau(\chi)} r_\mathrm{i}(N(\cdot), t) \cdot f_\mathrm{i}^{n_\mathrm{i}}\right) = \sum_{\tau_\mathrm{i} \in \tau(\chi)} r_\mathrm{i}(N(\cdot), t) \cdot f_\mathrm{i}^{n_\mathrm{i}}. \tag{7}$$

Thus, the upper bound of $\mathrm{pfh}(\chi)$ can be computed by setting $t = 1$ *hour* as shown in (4). □

Summarizing, Lemma 3.1 and Theorem 3.1 enable us to quantify the upper bound of probability of failure per hour on any criticality level without system reconfigurations.

3.2 Safety Quantification with Task Killing

We proceed to quantify system safety under task killing and global switching. As discussed, this only has an impact on the safety of LO criticality tasks.

First of all, for our analysis here, we will consider a system operation of O_s hours, and the time interval referred in this section represents such a system operation period.

Recall that on any core p_k, for any $\tau_\mathrm{i} \in \tau_\mathrm{k}(\mathrm{HI})$, killing LO criticality tasks is controlled by n'_ik: if any instance of τ_i executes the $(n'_\mathrm{ik}+1)$th time, then all LO criticality tasks are killed immediately on all cores (i.e. global switching).

We first quantify the probability that killing LO criticality tasks is triggered by core p_k within a time interval of length t (we calculate a complementary probability of this). Subsequently, we can derive the probability of task killing globally triggered by any core.

Lemma 3.2. *If $\tau_k(\mathrm{HI}) \neq \emptyset$, given $N'_k(\mathrm{HI})$, in a time interval of length t, the probability that no instance of any $\tau_i \in \tau_k(\mathrm{HI})$ executes the $(n'_{ik}+1)$th time on p_k, is lower bounded by:*

$$R_k\left(N'_k(\mathrm{HI}), t\right) = \prod_{\tau_i \in \tau_k(\mathrm{HI})} (1 - f_i^{n'_{ik}})^{r_{ik}(n'_{ik},t)}. \tag{8}$$

[1]Our analysis in this paper gives the tight upper bound of $\mathrm{pfh}(\chi)$ if $f_\mathrm{i}^{n_\mathrm{i}} \ll 1$, otherwise we should use $\min\{\mathrm{pfh}(\chi), 1\}$.

Proof. $\forall \tau_\mathrm{i} \in \tau_\mathrm{k}(\mathrm{HI})$, in each round, the probability that an instance does not execute the $(n'_\mathrm{ik}+1)$th time is $(1 - f_\mathrm{i}^{n'_\mathrm{ik}})$, since this is the complementary probability that all the first n'_ik executions of τ_i's instance fail. In addition, the maximum number of rounds that τ_i can accommodate on core p_k is $r_\mathrm{ik}(n'_\mathrm{ik}, t)$ in a time interval of length t according to (3). Thus, the probability that no instance of τ_i executes the $(n'_\mathrm{ik}+1)$th time in a time interval of length t is lower bounded by $(1 - f_\mathrm{i}^{n'_\mathrm{ik}})^{r_\mathrm{ik}(n'_\mathrm{ik},t)}$. (8) is the product of such probability over all tasks in $\tau_\mathrm{k}(\mathrm{HI})$, which lower-bounds the probability that no instance of any $\tau_\mathrm{i} \in \tau_\mathrm{k}(\mathrm{HI})$ executes the $(n'_\mathrm{ik}+1)$th time in a time interval of length t on core p_k. □

According to (8), $R_\mathrm{k}(N'_\mathrm{k}(\mathrm{HI}), t)$ decreases with increasing t, which implies the probability of task killing increases as time elapses – $R_\mathrm{k}(N'_\mathrm{k}(\mathrm{HI}), t)$ approaches 0 if t is infinity, thus killing LO criticality tasks eventually occurs for certain [21]. For each instance of any $\tau_\mathrm{i} \in \tau(\mathrm{LO})$, it does not fail if it is not killed and one of its executions is successful. Subsequently, by adding the failure probability of each round in such a time interval with the maximum number of rounds of τ_i in it, the upper bound of failure rates of any $\tau_\mathrm{i} \in \tau(\mathrm{LO})$ in a time interval can be computed. This is summarized in Theorem 3.2.

Theorem 3.2. *Given τ, P, $N(\cdot)$ and $N'(\cdot)$, assuming task killing and global switching, pfh(HI) can be calculated the same as shown in Theorem 3.1, since HI criticality tasks are not affected. Furthermore, for any $\tau_i \in \tau(\mathrm{LO})$, define $y_i(t)$ as a sequence of timing points unique to τ_i:*

$$\begin{aligned} y_i(t) = \{t - \max_k \{n_{ik}\} \cdot C_i - m \cdot T_i + D_i \\ \mid m \in \mathbb{N} \wedge m \geq 1 \wedge m < r_i(N(\cdot), t)\} \cup \{t\}. \end{aligned} \tag{9}$$

pfh(LO) can then be upper-bounded by:

$$\frac{1}{O_s} \sum_{\tau_i \in \tau(\mathrm{LO})} \sum_{\alpha \in y_i(t)} 1 - \left(\prod_{k=1}^{K} R_k(N'_k(\mathrm{HI}), \alpha)\right) \cdot (1 - f_i^{n_i}), \tag{10}$$

where $t = O_s$ denotes the system operation hours.

Proof. For one round of any $\tau_\mathrm{i} \in \tau(\mathrm{LO})$ finishing at time t, the probability that τ_i fails in this round can be upper bounded by:

$$1 - \left(\prod_{k=1}^{K} R_\mathrm{k}(N'_\mathrm{k}(\mathrm{HI}), t)\right) \cdot (1 - f_\mathrm{i}^{n_\mathrm{i}}). \tag{11}$$

Our explanations are as follows: In a time interval of length t, the probability that no instance of any $\tau_\mathrm{i} \in \tau_\mathrm{k}(\mathrm{HI})$ executes the $(n'_\mathrm{ik}+1)$th time on all cores can be lower bounded by:

$$\prod_{k=1}^{K} R_\mathrm{k}\left(N'_\mathrm{k}(\mathrm{HI}), t\right). \tag{12}$$

In addition, (12) also lower bounds the probability that LO criticality tasks are not killed under global switching. $(1 - f_\mathrm{i}^{n_\mathrm{i}})$ is the probability that τ_i does not fail by itself in this round. Thus, the product $\left(\prod_{k=1}^{K} R_\mathrm{k}(N'_\mathrm{k}(\mathrm{HI}), t)\right) \cdot (1 - f_\mathrm{i}^{n_\mathrm{i}})$ lower bounds the probability that τ_i does not fail in this round; its complement (11) computes the upper bound of the probability that τ_i fails in this round.

To get the worst-case failure rate for $\tau_\mathrm{i} \in \tau(\mathrm{LO})$ in a time interval of length t, we need to get the maximum number of rounds in it, simply because this leads to the maximum number of rounds that can fail. In addition, (8) decreases with increasing t; hence, each round should finish as late as

possible to maximize failure probability (see (11)). Applying Lemma 3.1, the $r_{\mathrm{i}}(N(\cdot), t)$th round, in the latest case, starts at $(t - \max_{\mathrm{k}}(n_{\mathrm{ik}}) \cdot C_{\mathrm{i}})$ and ends at t; similarly, the $(r_{\mathrm{i}}(N(\cdot), t) - 1)$th round starts at $(t - \max_{\mathrm{k}}(n_{\mathrm{ik}}) \cdot C_{\mathrm{i}} - T_{\mathrm{i}})$ and finishes at $(t - \max_{\mathrm{k}}(n_{\mathrm{ik}}) \cdot C_{\mathrm{i}} - T_{\mathrm{i}} + D_{\mathrm{i}})$. By induction, in the worst case, the $(r_{\mathrm{i}}(N(\cdot), t) - m)$th round starts at $(t - \max_{\mathrm{k}}(n_{\mathrm{ik}}) \cdot C_{\mathrm{i}} - m \cdot T_{\mathrm{i}})$ and must finish at $(t - \max_{\mathrm{k}}(n_{\mathrm{ik}}) \cdot C_{\mathrm{i}} - m \cdot T_{\mathrm{i}} + D_{\mathrm{i}})$ (except for the last round, finishing at t). Thus, for any $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$, we use $y_{\mathrm{i}}(t)$ to denote this sequence of latest finishing times for all rounds of τ_{i}, while the total number of rounds is maximum.

With the above results, we can upper bound the probability of failure of the round of τ_{i} finishing at $\alpha \in y_{\mathrm{i}}(t)$ by:$1 - \left(\prod_{k=1}^{K} R_{\mathrm{k}}(N'_{\mathrm{k}}(\mathrm{HI}), \alpha\right) \cdot (1 - f_{\mathrm{i}}^{n_{\mathrm{i}}})$. Therefore, pfh (LO) can be upper bounded by (10) (upper bound of failure rate during system operation of O_{s} hours divided by O_{s}) under global switching and task killing. □

Theorem 3.2 confirms that the safety of LO criticality tasks depends on the adaptation profiles of HI criticality tasks – according to (10), with decreasing adaptation profiles, LO criticality tasks are killed more often, leading to reduced system safety if killed tasks are safety related.

3.3 Service Degradation instead of Task Killing

Though task killing can greatly improve system schedulability [11], in practice, service degradation is usually preferred due to constant service requirements on all criticality levels. On the contrary, task killing might directly violate system safety, because LO criticality tasks (could still be safety-related) are completely removed from the system in such a drastic approach.

Recall that, when triggered, LO criticality tasks will have a service degradation factor d_{f} larger than one – for any $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$, its minimal inter-arrival time becomes $d_{\mathrm{f}} \cdot T_{\mathrm{i}}$. In addition, service degradation on all cores are synchronously triggered due to global switching. We now quantify the impact of service degradation on system safety under global switching – for a system operation interval of O_s hours, find the worst-case switching time, such that the total failure rates of any task in this interval are maximized (worst-case). This is summarized in Theorem 3.3.

Theorem 3.3. *Given τ, P, $N(\cdot)$ and $N'(\cdot)$, assuming service degradation under global switching, pfh(HI) can be upper-bounded by Theorem 3.1 as HI criticality tasks are not affected. Additionally, we define $s_i(N(\cdot), d_f, t)$ as the maximum number of rounds that $\tau_i \in \tau(\mathrm{LO})$ can fail (or accommodate) in an interval of length t with service degradation factor d_f if service degradation is triggered at the beginning of this time interval:*

$$s_i(N(\cdot), d_f, t) = \max\left\{\left\lfloor \frac{t + D_i - \max_k\{n_{ik}\} \cdot C_i}{d_f \cdot T_i} \right\rfloor, 0\right\} + 1. \tag{13}$$

Furthermore, $z(d_f, t)$ is defined as the upper bound of failure probability of LO criticality tasks in this time interval:

$$z(d_f, t) = \sum_{\tau_i \in \tau(\mathrm{LO})} s_i(N(\cdot), d_f, t) \cdot f_i^{n_i}. \tag{14}$$

pfh(LO) can then be upper bounded by

$$\frac{1}{O_s}\left(1 - \prod_{k=1}^{K} R_k(N'_k(\mathrm{HI}), t)\right) \cdot z(1, t), \tag{15}$$

where $t = O_s$ hours.

Proof. First of all, suppose service of any $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ is degraded with a service degradation factor d_{f} in a time interval of length t on all cores. (13) can be derived according to Lemma 3.1. In addition, according to Theorem 3.1, (14) upper-bounds the probability of failure of LO criticality tasks in a time interval of length t if service degradation of LO criticality tasks is already triggered. We assume global switching happens at time instant λt $(0 \leq \lambda \leq 1)$. Based on (12), the probability that LO criticality tasks are not degraded till λt under global switching is lower bounded by:

$$\prod_{k=1}^{K} R_{\mathrm{k}}\left(N'_{\mathrm{k}}(\mathrm{HI}), \lambda t\right). \tag{16}$$

The complement of (16) upper bounds the probability that LO criticality tasks are degraded at time instant λt. For a system operation interval of length t, from starting to λt, service degradation of LO criticality tasks on all cores is not triggered thus probability of failure of LO criticality tasks is upper bounded by $z(1, \lambda t)$. From λt to t, service of LO criticality tasks is degraded with d_{f} on all cores; hence, failure probability of LO criticality tasks is upper bounded by $z(d_{\mathrm{f}}, (1 - \lambda)t)$. Therefore, probability of failure of LO criticality tasks on all cores in a system operation interval of length t under global switching and service degradation can be upper bounded by:

$$\left(1 - \prod_{k=1}^{K}(R_{\mathrm{k}}(N'_{\mathrm{k}}(\mathrm{HI}), \lambda t)\right)\left(z(1, \lambda t) + z(d_{\mathrm{f}}, (1 - \lambda)t)\right). \tag{17}$$

Based on the proof of Lemma 3.2, (16) decreases while λ increases. Furthermore, with increasing λ, $z(1, \lambda t)$ increases while $z(d_{\mathrm{f}}, (1 - \lambda)t)$ decreases. However, the increasing rate of $z(1, \lambda t)$ is larger than the decreasing rate of $z(d_{\mathrm{f}}, (1 - \lambda)t)$. This is because for a time interval of fixed length, if service degradation of $\tau(\mathrm{LO})$ is triggered, any $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ accommodates less number of failures than it can accommodate without service degradation. Thus, $(z(1, \lambda t) + z(d_{\mathrm{f}}, (1 - \lambda)t))$, accordingly the upper bound of pfh (LO), is maximum when $\lambda = 1$. □

4. SAFETY QUANTIFICATION UNDER LOCAL SWITCHING

We proceed to perform safety analysis under local switching (Definition 2.5). Unlike global switching as presented in the previous section, local switching provides the flexibility for each core to switch from LO to HI mode independently, reducing the pessimism that LO criticality tasks must be killed or degraded on all cores at once. However, this complicates the analysis as the possible system states explode (i.e. when and which cores incur reconfigurations). In this section, we focus on analyzing the case with task killing. An analysis for service degradation under local switching is provided in the technical report [5] due to lack of space.

4.1 Task Killing Under Local Switching

Our key intuition is that, despite the combinatorial nature of the problem, we can still perform a round per round analysis to pessimistically bound system failure rates. The main reason is that, for any LO criticality task τ_{i} on any core p_{k}, we can bound the probability of executing each round (note that a round is only executed on core p_{k} with a probability that killing is not triggered, as give in Lemma 3.2). This enables us to further bound the failure probability of each round on each core independently of other cores.

Recall that any instance of $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ can execute at most n_{ik} times on core p_{k} for one job arrival. A round of τ_{i} fails on p_{k} if it is killed or all n_{ik} instances of τ_{i} fail in this round on p_{k}. We quantify the upper bound of failure probability of one round of τ_{i} finishing at time t on each core p_{k} individually. The product of such probability across all cores bounds the failure probability of one round of τ_{i} finishing at time t. Furthermore, by taking the maximum number of rounds τ_{i} can encounter in a time interval and adding up the failure probability of each round, we get the worst-case failure rate of a single task; from this, pfh(LO) can be further derived. This analysis is formally presented in Theorem 4.1.

Theorem 4.1. *Given τ, P, $N(\cdot)$ and $N'(\cdot)$, assuming task killing under local switching, pfh(HI) can be upper-bounded by Theorem 3.1 as HI criticality tasks are not affected. We further define $u_{ik}(t)$ as the upper bound of failure probability of one round of $\tau_i \in \tau_k(\mathrm{LO})$ finishing at time t on p_k:*

- *if $n_{ik} = 0$,*

$$u_{ik}(t) = 0, \tag{18}$$

- *if $\exists \tau_j \in \tau_k(\mathrm{HI}) : n_{jk} \neq 0 \wedge n'_{jk} = 0$,*

$$u_{ik}(t) = 1, \tag{19}$$

- *if $((N_k(\mathrm{HI}) = \emptyset) \vee (N_k(\mathrm{HI}) = N'_k(\mathrm{HI}))) \wedge (n_{ik} \neq 0)$,*

$$u_{ik}(t) = f_i^{n_{ik}}, \tag{20}$$

- *if $(N'_k(\mathrm{HI}) \neq \emptyset) \wedge (N_k(\mathrm{HI}) \neq N'_k(\mathrm{HI})) \wedge (n_{ik} \neq 0)$,*

$$u_{ik}(t) = 1 - R_k\left(N'_k(\mathrm{HI}), t\right) \cdot \left(1 - f_i^{n_{ik}}\right). \tag{21}$$

Using $y_i(t)$ (see (9)), pfh(LO) can be upper bounded by:

$$\frac{1}{O_s}\left(\sum_{\tau_i \in \tau(\mathrm{LO})} \sum_{\alpha \in y_i(t)} \left(\prod_{\substack{k=1 \\ u_{ik}(\alpha)\neq 0}}^{K} u_{ik}(\alpha)\right)\right), \tag{22}$$

where $t = O_s$ hours.

Proof. Recall that, for any core p_{k}, killing $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ is adopted when any $\tau_{\mathrm{j}} \in \tau_{\mathrm{k}}(\mathrm{HI})$ executes the $\left(n'_{\mathrm{jk}} + 1\right)$th time; we derive the upper bound of failure probability of one round of τ_{i} finishing at t on p_{k} as $u_{\mathrm{ik}}(t)$ in four cases.

1. If $n_{\mathrm{ik}} = 0$, then there is no instance of τ_{i} executing on p_{k}. Hence, τ_{i} does not fail on p_{k}, we have (18).
2. If $\exists \tau_{\mathrm{j}} \in \tau_{\mathrm{k}}(\mathrm{HI}) : n_{\mathrm{jk}} \neq 0 \wedge n'_{\mathrm{jk}} = 0$, then τ_{j} executes on p_{k} with "zero" adaptation profile, meaning that any $\tau_{\mathrm{i}} \in \tau_{\mathrm{k}}(\mathrm{LO})$ will be killed when the first instance of τ_{j} starts to execute. Thus, in the worst case, any $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ will not execute (i.e. simply fail), we have (19).
3. If $((N_{\mathrm{k}}(\mathrm{HI}) = \emptyset) \vee (N_{\mathrm{k}}(\mathrm{HI}) = N'_{\mathrm{k}}(\mathrm{HI}))) \wedge (n_{\mathrm{ik}} \neq 0)$, then on p_{k}, there are no HI criticality tasks or redundancy profile of HI criticality tasks equals their adaptation profile; in both cases, killing LO criticality tasks will not be triggered on p_{k}. Hence, one round of $\tau_{\mathrm{i}} \in \tau_{\mathrm{k}}(\mathrm{LO})$ fails with probability $f_{\mathrm{i}}^{n_{\mathrm{ik}}}$ on p_{k}, we have (20).
4. If $(N'_{\mathrm{k}}(\mathrm{HI}) \neq \emptyset) \wedge (N_{\mathrm{k}}(\mathrm{HI}) \neq N'_{\mathrm{k}}(\mathrm{HI})) \wedge (n_{\mathrm{ik}} \neq 0)$, then killing LO criticality tasks is triggered on p_{k} when any task $\tau_{\mathrm{j}} \in \tau_{\mathrm{k}}(\mathrm{HI})$ executes the $(n'_{\mathrm{jk}} + 1)$th time. Based on our system model, for one round of $\tau_{\mathrm{i}} \in \tau_{\mathrm{k}}(\mathrm{LO})$ ends at t, $R_{\mathrm{k}}(N'_{\mathrm{k}}(\mathrm{HI}), t)$ lower bounds the probability that it is not killed until t. Therefore, the probability that it does not fail is lower bounded by $R_{\mathrm{k}}(N'_{\mathrm{k}}(\mathrm{HI}), t) \cdot (1 - f_{\mathrm{i}}^{n_{\mathrm{ik}}})$, whose complement yields the upper bound of the probability that it fails, see (21).

Second, to get the worst-case failure rate for $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ in a time interval of length t, we need to get the maximum number of rounds it can accommodate in this time interval so that the number of rounds it can fail is maximum. In addition, all rounds on all cores should end as late as possible to get the maximum value of (21). According to the proof of Theorem 3.2, for any $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$, $y_{\mathrm{i}}(t)$ is a sequence of the latest time that each round of τ_{i} finishes while there are maximum number of rounds in the considered interval.

As shown above, (18), (19), (20) and (21) upper bound $u_{\mathrm{ik}}(t)$ for any $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ in all possible cases. Thus, based on our system model, one round of $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ finishing at time t fails if it fails on all cores with probability upper bounded by:

$$\prod_{\substack{k=1 \\ u_{\mathrm{ik}}(t)\neq 0}}^{K} u_{\mathrm{ik}}(t). \tag{23}$$

The complement of (23),

$$1 - \prod_{\substack{k=1 \\ u_{\mathrm{ik}}(t)\neq 0}}^{K} u_{\mathrm{ik}}(t), \tag{24}$$

lower-bounds the probability that one round of $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ finishing at t does not fail. Thus, the probability that $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ does not fail in a time interval of length t can be lower bounded by the probability that $r_{\mathrm{i}}(N(\cdot), t)$ rounds have all executed successfully:

$$\prod_{\alpha \in y_{\mathrm{i}}(t)} \left(1 - \prod_{\substack{k=1 \\ u_{\mathrm{ik}}(\alpha)\neq 0}}^{K} u_{\mathrm{ik}}(\alpha)\right). \tag{25}$$

Taking the complement of (25), the probability of failure of $r_{\mathrm{i}}(N(\cdot), t)$ rounds of $\tau_{\mathrm{i}} \in \tau(\mathrm{LO})$ in a time interval of length t can be upper bounded by

$$1 - \prod_{\alpha \in y_{\mathrm{i}}(t)} \left(1 - \prod_{\substack{k=1 \\ u_{\mathrm{ik}}(\alpha)\neq 0}}^{K} u_{\mathrm{ik}}(\alpha)\right). \tag{26}$$

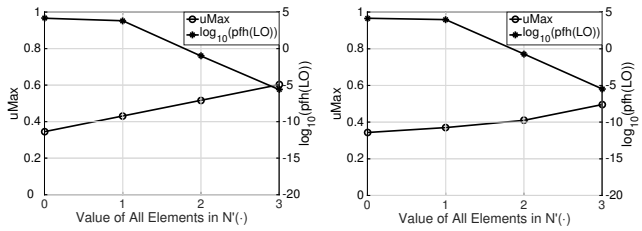
Due to $\prod_{\substack{k=1 \\ u_{\mathrm{ik}}(\alpha)\neq 0}}^{K} u_{\mathrm{ik}}(\alpha) \ll 1$, we only consider its first order terms, (26) can be tightly upper-bounded by

$$\sum_{\alpha \in y_{\mathrm{i}}(t)} \left(\prod_{\substack{k=1 \\ u_{\mathrm{ik}}(t)\neq 0}}^{K} u_{\mathrm{ik}}(t)\right). \tag{27}$$

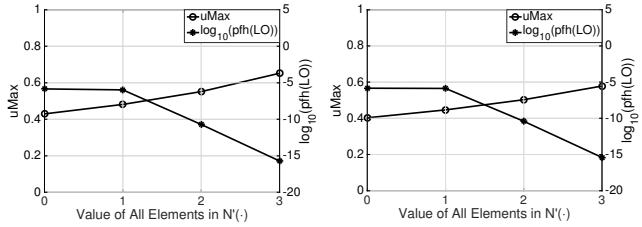
Therefore, the upper bound of pfh(LO) in a system operation of O_{s} hours can be calculated as the average overall failure rates in this interval, as shown in (22). □

5. EVALUATION

We validate in this section our proposed techniques with both a real life flight management system (FMS) application and extensive simulations. As for FMS, we show the impact of task killing and service degradation on system safety and schedulability; with extensive simulations on synthetic



(a) Dual-Core; Task Killing (b) Quad-Core; Task Killing



(c) Dual-Core; Service Degradation (d) Quad-Core; Service Degradation

Figure 3: FMS: Global Switching

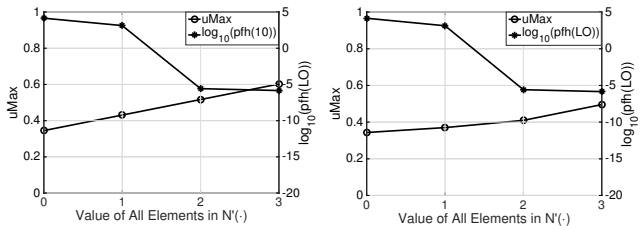
task sets, we present the improvements of system feasibility achieved by task killing or service degradation.

5.1 Flight Management System

Here, task killing or service degradation under global/local switching are evaluated with a real-life flight management system, and their impact on system safety and schedulability is studied. The considered FMS consists of 7 DO-178B criticality level B tasks and 4 criticality level C tasks, with detailed parameters found in [21]. As we consider HI = B and LO = C, the safety requirements are pfh(HI) $< 10^{-7}$ and pfh(LO) $< 10^{-5}$. We assume any instance of each task is subject to a constant failure probability 10^{-5} [21]. We conduct our experiments on dual-core/quad-core platforms: For all cases, we first find a feasible redundancy profile $N(\cdot)$ to satisfy system safety and schedulability without system reconfigurations, while load balancing is achieved at the same time by applying a genetic optimization algorithm [1]; we then fix this $N(\cdot)$ and increase the values of adaptation profile $N'(\cdot)$ of all HI criticality tasks to show the impact (assuming a uniform adaptation profile for all HI criticality tasks). The service degradation factor d_f is set to 5 in case of degrading LO criticality tasks. We present our results in Figures 3 and 4, where the x-axis represents the values of the adaptation profile of all HI criticality tasks, the left y-axis (uMax) represents the maximum utilization factor across all cores and the right y-axis represents the upper-bound of pfh (LO) in logarithmic scale. In our experiments, the upper-bound of pfh (HI) always satisfies safety requirements. pfh (HI) in logarithmic scale equals -35.17 and -75.17 on dual-core and quad-core systems respectively. Additionally, the impact of task criticality and the adoption of task killing or service degradation is evaluated.

5.1.1 Analysis of Experiment Results

First of all, we can observe that in all cases, with increasing values of adaptation profile of all HI criticality tasks, system schedulability is hampered while safety is enhanced. With a larger adaptation profile, LO criticality tasks are killed/degraded less likely (often), see our analysis in Section 3 and Section 4. This will not help for system schedulability, as we would kill or degrade LO criticality tasks frequently to improve schedulability. However, system safety can be improved, simply because LO criticality tasks are killed/degraded less likely. For example, with task killing



(a) Dual-Core; Task Killing (b) Quad-Core; Task Killing

Figure 4: FMS: Local switching

under global switching on a dual-core platform (Figure 3a), the maximum utilization across all cores is increased from 0.345 to 0.603 with adaptation profile increasing from 0 to 3; at the same time, pfh(LO) decreases by 10 orders of magnitude.

Next, as expected, we can observe that with more cores, system schedulability can be improved. For instance, considering an adaptation profile of 3 under service degradation and global switching (Figure 3c and 3d), the maximum utilization factor across all cores on dual-core (0.652) is larger than that (0.578) on quad-core. However, the difference here is not large: In order to evaluate the impact of adaptation profile on system feasibility, we fixed the redundancy profile of all 7 HI criticality tasks on each core to 4. Thus, load balancing is only performed for 4 LO criticality tasks of low utilizations, which will not result in a large difference in the maximum utilization across all cores.

Furthermore, if we compare task killing with service degradation under global switching (e.g. Figure 3a and Figure 3c), we can observe that task killing improves system schedulability more, while jeopardizing more system safety. Task killing is more abrupt than service degradation; when triggered, it removes *all* LO criticality tasks either locally (local switching) or globally (global switching), freeing more resources to guarantee schedulability of HI criticality tasks while affecting the safety of LO criticality tasks. For example, as shown in Figure 3, task killing satisfies system safety only with an adaptation profile of 3, while service degradation satisfies safety requirements in all cases.

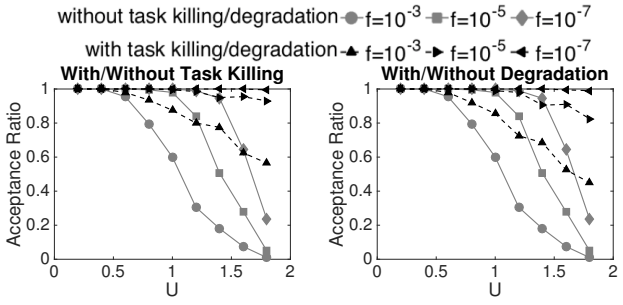
Last, by comparing global with local switching (e.g. task killing in Figure 3b and Figure 4b), we can observe that both incur similar impacts on system schedulability, since the worst-case is that all LO criticality tasks are killed to guarantee schedulability of HI criticality tasks. However, global switching has a stronger impact on system safety – in global switching, killing LO criticality tasks can be triggered by any core, leading to a significantly higher killing probability for LO criticality tasks than in case of local switching. Regarding task killing in all cases (Figure 3a, 3b, 4a and 4b), system safety is guaranteed with a larger adaptation profile under global switching ($\forall \tau_i \in \tau(\text{HI}), \forall p_k \in P : n'_{ik} = 3$) than that under local switching ($\forall \tau_i \in \tau(\text{HI}), \forall p_k \in P : n'_{ik} = 2$).

5.1.2 Summary of Results

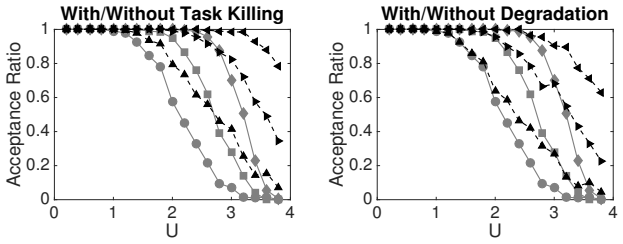
We conclude that system safety and schedulability contradicts with each other under system reconfiguration. To improve system safety, either the redundancy or the adaptation profile needs to be increased. However, this increases system utilization and hampers schedulability. Those tradeoffs can be quantified by the analysis techniques proposed in this paper.

5.2 Extensive Simulation

We proceed to validate the analysis and design methods on synthetic task sets. It is well known that task killing or service degradation can improve mixed-criticality system schedulablity [11]. With explicit safety analysis under sys-

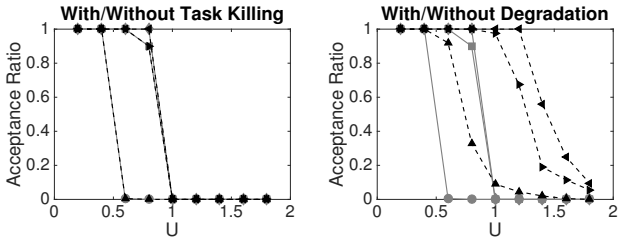


(a) Dual-Core; HI = A, LO = E (b) Dual-Core; HI = A, LO = E

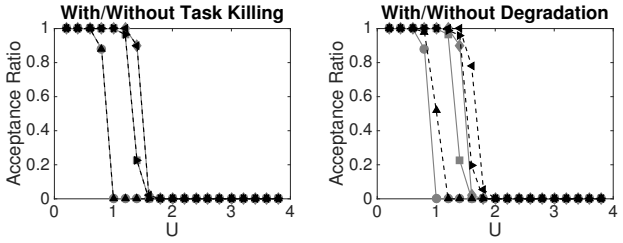


(c) Quad-Core; HI = A, LO = E (d) Quad-Core; HI = A,LO = E

Figure 5: Feasibility evaluation with global switching

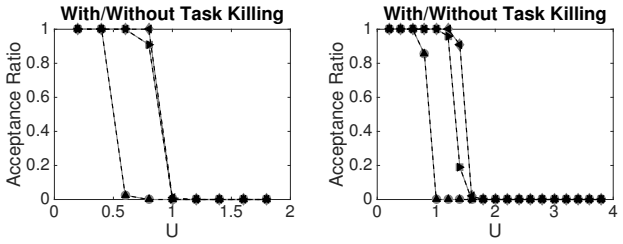


(a) Dual-Core; HI = A, LO = C (b) Dual-Core; HI = A, LO = C



(c) Quad-Core; HI = A, LO = C (d) Quad-Core; HI = A, LO = C

Figure 6: Feasibility evaluation with global switching



(a) Dual-Core; HI = A, LO = C (b) Quad-Core; HI = A, LO = C

Figure 7: Feasibility evaluation with local switching

tem reconfigurations, we study here whether they can help improve system feasibility if safety is also involved. To this end, we adopt a well-known random mixed-criticality task generator to generate 200 task sets at each system utilization point [11]. The task generation is slightly modified as compared to [11] to fit our system model in Section 2. We repetitively apply random search to each task set to find feasible design parameters under various assumptions (e.g. local/global switching, task killing/service degradation); if the search process is successful then the task-set is said to be feasible (otherwise not). We then compare the system acceptance ratios (the ratio of the number feasible task sets to the number of tested task sets) for those different assumptions at each data point.

5.2.1 Task Generation

Random implicit deadline dual-criticality task-sets are generated [11], with the following controlling parameters:

- $[u_-, u_+]$: the range $(0 < u_- < u_+ \leqslant 1)$ from which the utilization $(\frac{C_i}{T_i})$ of any task τ_i is uniformly drawn.
- U: the system utilization defined as $\sum_{\tau_i} \frac{C_i}{T_i}$.
- $[T_-, T_+]$: the minimal inter-arrival time of each task is generated uniformly from this range.
- P_{HI}: the probability a generated task has HI criticality.

For our experiments, we set $u_-/u_+ = 0.02/0.2$, $T_-/T_+ = 2/2000ms$, and $P_{\text{HI}} = 0.2$. The random task generator starts with system utilization factor $U = 0.2$ and randomly adds new tasks into this task set until a certain U is reached (U in increments of 0.2). Notice that, system reconfiguration is applied only if it is not feasible without reconfiguration.

5.2.2 Analysis of Simulation Results

We present our results in Figures 5, 6 and 7, where the x-axis represents system utilization before applying redundancies or adaptations, the y-axis represents the acceptance ratio, and f represents the universal failure probability of any task instance. Due to space limitations, we provide our results for octa-core systems in [5].

First, as expected, it can be observed that system feasibility constantly improves with more reliable hardware platforms (decreasing f) or more cores. For example, consider system utilization $U = 1$ on a dual-core/quad-core platform (LO = E) without task killing. The acceptance ratio increases from 60% to 100% as f decreases from 10^{-3} to 10^{-7} on a dual-core platform (Figure 5a). With $f = 10^{-3}$, the acceptance ratio increases from 60% (Figure 5a) to 99% (Figure 5c) as the number of cores increases from 2 to 4.

Next, according to the results shown in Figure 5, if LO criticality tasks are not safety related (LO = E), system feasibility can be improved by both task killing and service degradation. In addition, task killing performs better than service degradation here, since it can safely remove all LO criticality tasks to improve schedulability. For example, consider $U = 2$ and $f = 10^{-3}$ on a quad-core platform (Figure 5c and 5d). Acceptance ratio increases by 38.26%/11.30% under killing/degradation as compared to the case when only redundancies are applied.

Moreover, as shown in Figure 6, if LO criticality tasks are safety related (LO = C), then global switching with task killing hardly helps the design of fault-tolerant mixed-criticality systems because task killing can immediately violate system safety. In detail, the reason is two-fold: First, after killing, those safety-related LO criticality tasks constantly fail. Second, as time elapses, the probability that killing is triggered constantly increases, eventually reaching 1 (see Lemma 3.2). However, global switching with service degradation improves system feasibility and it is more proper when LO criticality tasks are safety concerned – as long as the system accepts degraded services, it is always beneficial for safety as less task instances are executed. Thus,

LO criticality tasks can only fail less. For instance, consider $U = 1$ and $f = 10^{-3}$ on a dual-core platform (Figure 6a and 6b), acceptance ratio does not improve under task killing, whereas it increases by 9% under service degradation.

Last, similar to the case with global switching, for local switching (Figure 7), killing LO criticality tasks when they are safety-related could rarely help system feasibility, as this could immediately violate system safety.

5.2.3 Summary of Results

We conclude that task killing greatly improves system feasibility if LO criticality tasks are not safety related. However, if safety is a concern for LO criticality tasks, service degradation is a proper choice as task killing could directly violate system safety. Again, the analysis methods derived in this paper enables us to quantify those findings.

6. CONCLUSION

Fault-tolerant mixed-criticality scheduling on multicores under hardware/software transient errors is studied in this paper. We propose techniques that could reconfigure the system at run-time such that critical tasks can still be guaranteed under urgent situations, i.e. when they do not succeed after a certain number of trials. Furthermore, with explicit modeling of system safety on different criticality levels according to well-known safety standards [3], we can quantify the impacts of online reconfigurations (task killing or service degradation) on safety. To guarantee schedulability, we perform a problem transformation and reduce our problem to a classical mixed-criticality scheduling problem. Moreover, we distinguish in our techniques between global reconfiguration and local reconfiguration, allowing the trade-off between analysis complexity and system feasibility. Finally, with both a flight management system and extensive simulations, we validate our proposed techniques and demonstrate quantitatively two important findings: first, task killing as commonly assumed for mixed-criticality systems can hardly help improving system feasibility if it is performed on safety-related functions; second, system feasibility can be improved to a great extent for all other cases under run-time system reconfigurations.

Our proposed techniques are built upon the classical mixed-criticality model [11], for which possible extensions are discussed in e.g. [18]. As future work, it would be interesting and important to extend the methods in this paper to incorporate such extensions, e.g. allowing the system to switch back from HI to LO mode while guaranteeing system safety and schedulablity. Furthermore, it would be of practical importance to extend and integrate the proposed techniques on sub-systems (e.g. the flight management system in a commercial aircraft) with system wide analysis (e.g. on the level of the aircraft) to achieve certifiable designs of industrial mixed-criticality systems.

References

[1] Genetic Algorithm. http://ch.mathworks.com/help/gads/genetic-algorithm.html.

[2] ISO—26262 Road Vehicles – Functional Safety.

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

[4] Safeadapt, 2016. http://www.safeadapt.eu/.

[5] Authors removed due to blind review. Towards the Design of Fault-Tolerant Mixed-Criticality Systems on Multicores. https://www.dropbox.com/s/u67yyko4lncfsp8/report.pdf?dl=0.

[6] 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 *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 145–154. IEEE, 2012.

[7] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin. Mixed-criticality scheduling on multiprocessors. *Real-Time Systems*, 50(1):142–177, 2014.

[8] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 147–155. IEEE, 2008.

[9] V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. *eraerts*, page 204, 2011.

[10] S. Brown. Overview of iec 61508. design of electrical/electronic/programmable electronic safety-related systems. *Computing & Control Engineering Journal*, 11(1):6–12, 2000.

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

[12] A. Burns, R. Davis, et al. Mixed criticality on controller area network. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 125–134. IEEE, 2013.

[13] A. Burns, J. Harbin, and L. Indrusiak. A wormhole noc protocol for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 184–195. IEEE, 2014.

[14] P. Ekberg and W. Yi. Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 135–144. IEEE, 2012.

[15] Mixed Criticality Systems, 2012. Report from the Workshop on Mixed Criticality Systems, Brussels, Belgium.

[16] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *International Conference on Embedded Software (EMSOFT)*, pages 17:1–17:15, Montreal, Oct 2013.

[17] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Mapping mixed-criticality applications on multi-core architectures. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

[18] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. 2013.

[19] 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*, 16(4):189–220, 2012.

[20] P. Huang, G. Giannopoulou, R. Ahmed, D. B. Bartolini, and L. Thiele. An isolation scheduling model for multicores. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, San Antonio, Texas, USA, Dec 2015.

[21] P. Huang, H. Yang, and L. Thiele. On the scheduling of fault-tolerant mixed-criticality systems. Technical report, Computer Engineering and Networks Laboratory, ETH Zurich, 2014.

[22] S.-h. Kang, H. Yang, S. Kim, I. Bacivarov, S. Ha, and L. Thiele. Static mapping of mixed-critical applications for fault-tolerant mpsocs. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 31:1–31:6, New York, NY, USA, 2014. ACM.

[23] H. Li and S. Baruah. Load-based schedulability analysis of certifiable mixed-criticality systems. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 99–108. ACM, 2010.

[24] D. McNulty, L. Olson, and M. Peloquin. A comparison of scheduling algorithms for multiprocessors, 2010.

[25] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, J. Scoredos, et al. Mixed-criticality real-time scheduling for multicore systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1864–1871. IEEE, 2010.

[26] T. Park and S. Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 253–262. ACM, 2011.

[27] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 155–165. IEEE, 2012.

[28] L. Sha. Resilient mixed-criticality systems. *CrossTalk: The Journal of Defense Software*, 2009.

[29] 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*, pages 239–243. IEEE, 2007.

[30] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 299–308. IEEE, 2012.

[31] M. Zeller, C. Prehofer, D. Krefft, and G. Weiss. Towards runtime adaptation in autosar. *ACM SIGBED Review*, 10(4):17–20, 2013.