PREFACE: Proactive Re-executions for Fault-aware Mixed-criticality Environments

Hwisoo So*, Byeonggil Jun†, Chanhee Lee†, Hokeun Kim†, and Aviral Shrivastava†
*Kyungpook National University, †Arizona State University
Email: *hwisoo.so@knu.ac.kr, †{byeonggil, chanheel, hokeun, aviral.shrivastava}@asu.edu

***Abstract*—Mixed Criticality Systems (MCSs) enable efficient utilization of hardware resources to execute safety-critical tasks along with non-critical tasks. Soft errors are a critical threat to MCSs, causing detectable as well as undetectable errors. State-of-the-art fault-tolerant MCSs protect the safety-critical tasks against soft errors by reactively re-executing them upon detecting failures. However, assuming that all failures can be detected, existing state-of-the-art failure formulations for fault-tolerant MCSs fail to consider *undetected* failures. Further, the reactive re-execution strategy cannot improve fault tolerance against undetected failures. To address this problem, we propose PREFACE, Proactive Re-Executions for Fault-Aware mixed-Criticality Environments. PREFACE formulates the failure rates of MCS tasks by differentiating *detectable* failures from *undetectable* ones. Based on our novel failure formulation, PREFACE proactively re-executes a task even when no fault is detected to cope with potential undetectable failures, only when it is necessary. Our evaluation demonstrates that PREFACE dramatically improves the scheduling feasibility and reliability compared to state-of-the-art fault-tolerant MCSs.**

***Index Terms*—mixed-criticality systems, real-time scheduling, fault tolerance, soft errors**

I. Introduction

Mixed-criticality systems (MCSs) must satisfy different criticality levels of tasks, including safety-critical, mission-critical, and non-critical tasks. Traditional MCSs have mainly focused on preventing timing failures by ensuring the schedulability of tasks. However, soft errors can result in fundamentally different types of failures [1] in MCS tasks; a task with a soft error can fail to complete execution or deliver correct output.

To satisfy failure requirements of the tasks under the potential soft errors, state-of-the-art (SOTA) fault (soft error)-aware MCSs [2], [3] apply *software-implemented hardware fault tolerance (SIHFT)* [4] solutions that can detect or correct most of the soft errors, and allow *reactive* re-executions for failed executions upon detection of the failures. Fig. 1 (a) illustrates the reactive re-execution strategy in SOTA fault-tolerant MCSs, which triggers re-execution upon detecting failures.

The reactive re-execution strategy in SOTA fault-tolerant MCSs [2], [3] is valid if *all failures are detectable* so that the system can trigger reactive re-executions for all failures. However, the premise that *all failures are detectable* is simply **not true** in reality. Executions under soft errors can result in undetectable failures, even with SIHFT protection. For further discussion, let us define terms for detectable and undetectable failures: 1) detected unrecoverable error (DUE), system-visible failures such as crash and hang, and 2) silent data corruption (SDC), cases in a task seemingly finished without a system-visible failure but delivered incorrect outputs.

The reactive re-execution strategy of SOTA fault-tolerant MCSs cannot mitigate SDCs as shown in Fig. 1 (b), since

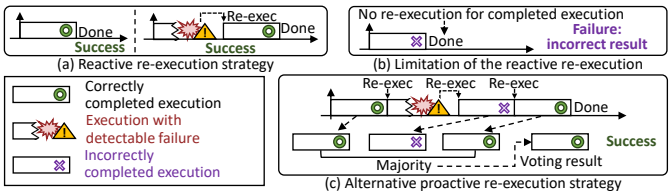


Fig. 1. Illustration of (a) re-execution strategy in SOTA fault-tolerant MCSs [2], [3], (b) their limitation, and (c) alternative proactive re-execution strategy.

they cannot trigger the reactive re-execution when no failure is detected. Further, SOTA MCSs estimate the failure probabilities of tasks by considering that the reactive re-execution strategy can mitigate all failures. Consequently, their formulations underestimate the failure probabilities of tasks by ignoring characteristics of SDC, resulting in dissatisfaction with the failure requirements of tasks without recognizing the dissatisfaction.

Contributions: To resolve the aforementioned limitations of SOTA fault-tolerant MCSs, this paper proposes **P**roactive **R**e-**E**xecutions for **F**ault-**A**ware mixed-**C**riticality **E**nvironments (PREFACE), which mitigates both DUEs and SDCs via its novel *proactive* re-execution strategy and provides corresponding failure formulation. First, PREFACE advances the failure formulation of SOTA fault-tolerant MCSs by separately considering SDCs and DUEs, and calculates probabilities for possible numbers of correct and incorrect completed executions under the proactive re-execution strategy. PREFACE's failure formulation for MCSs considering SDCs is the first approach that has never been explored so far, to the best of our knowledge. Second, PREFACE allows re-execution(s) of tasks even when no failure is detected and performs majority voting among completed executions even in the presence of SDCs, as shown in Fig. 1 (c). The completed executions in PREFACE form an adaptive redundancy to mitigate SDCs, where the task is already protected by a SIHFT solution and the number of completed executions can vary due to the DUEs. We emphasize that PREFACE is the first approach for fault-tolerant MCSs with a re-execution strategy that correctly considers SDCs and forms an adaptive redundancy with SIHFT-protected task.

Evaluation with RTL fault injection method shows PREFACE's dramatic improvements in scheduling feasibility under real-time and reliability requirements compared to SOTA fault-tolerant MCSs, providing enough slack time for non-critical tasks compared to naive triple modular redundancy (TMR).

II. Background: SOTA Fault-tolerant MCSs

This section discusses the failure formulation and re-executin strategy of SOTA fault-tolerant MCSs [2], [3]. Reghenzani *et al.* [2] extend traditional MC scheduling theory to provide fault

978-3-9826741-1-7/DATE26/© 2026 EDAA

tolerance against soft errors. They apply SIHFT to MC tasks to detect faults and estimate the **failure probability** of each task from the given **fault rate** and task information. If the failure probability of the task cannot satisfy the reliability requirement of the task, Reghenzani *et al.* allow reactive re-execution(s) of the task upon detecting failures. RTailor [3] improves the failure formulation of Reghenzani *et al.* by considering the masking effect and developing a selective SIHFT to satisfy the failure requirements at minimum overhead. For the simplicity of further discussion, we make the following assumptions:

1. We assume that a task solely occupies a single-core processor, and therefore, a fault on a core will affect the task executing on the core. This assumption is based on the system model of Reghenzani *et al.* [2], which considers the resource usage of a task for a core as either 0 or 1.
2. We only consider faults on the core and do not consider faults on memory subsystems since they can be protected by ECC or parity, following the same assumption in RTailor [3] and most recent SIHFT solutions [5]–[9].
3. We consider that crash and hang cases can be detected by the system and can trigger the re-executions in Reghenzani *et al.* and RTailor, following the re-execution models in RTailor.

A. Failure Formulation

1) Failure rate of a single task per time unit: For a given fault rate per hour λ, the probability that no fault occurs during H hours can be formulated as $(1-\lambda)^H$ based on a sequence of Bernoulli trials. Reghenzani *et al.* [2] consider an occurrence of a fault on a hardware resource executing a task as a failure of the corresponding task. Since we assume that one task occupies a single-core and the memory subsystem is protected by parity or ECC, the **failure rate** per specific time frame for a task in Reghenzani *et al.* is equal to the fault rate per the corresponding time frame on a core. Reghenzani *et al.* derives p'_i, the failure rate per a time unit $1/k$ hour, i.e., there are k time units per hour, for a task τ_i with the following Equation (1).

$$p'_i = 1 - (1-\lambda)^{\frac{1}{k}} \tag{1}$$

On the other hand, RTailor excludes masked and corrected faults from the failures as the following Equation (2).

$$p'_i = 1 - (1 - \lambda \times P(failure|fault)_i)^{\frac{1}{k}} \tag{2}$$

Where $P(failure|fault)_i$ represents the conditional probability that τ_i results in failure, including crash, hang, SDC, or detected cases (if detection-only SIHFT is used), given a fault.

2) Failure probability per execution: In the formulation of Reghenzani *et al.* and RTailor, failure of an execution of a task τ_i means that a failure occurs during the execution time E_i.

$$\mathbb{P}_i = 1 - (1-p'_i)^{E_i} \tag{3}$$

In Equation (3), $\mathbb{P}_i$ represents the failure probability per execution of τ_i for the failure rate per time unit p'_i from Equation (1) (Reghenzani *et al.*) or (2) (RTailor). E_i represents the execution time, i.e., the exposure time of the execution against soft errors.

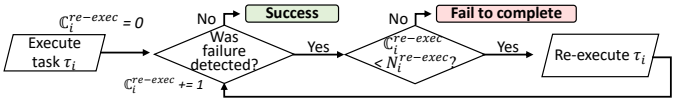


Fig. 2. Flow chart of reactive re-execution strategy in SOTA schemes [2], [3].

3) Required failure probability per execution: The required failure rate of a task is typically expressed in *per hour* $(/h)$. For example, the design assurance level (DAL) in DO-178C [10], which is the software considerations in airborne systems and equipment certification, specifies the maximum allowable failure probability per hour as $10^{-9}/h$ – $10^{-5}/h$ (DAL A – C) [2].

Fault-tolerant MCSs aims to satisfy the failure requirements of each task as well as its real-time requirements. The failure requirement of a task τ_i is satisfied if its failure rate is lower than its required failure rate. We derive the required failure probability per execution from a given required failure rate per hour. $\mathbb{P}_i^{req}$, the required failure probability per execution can be derived from the failure rate per hour (${p_{per-hour}}_i^{req}$) as a sequence of Bernoulli trials, when N_i^{act} executions of τ_i are activated per hour, as the following Equation (4).

$$\mathbb{P}_i^{req} = 1 - (1 - {p_{per-hour}}_i^{req})^{\frac{1}{N_i^{act}}} \tag{4}$$

If $\mathbb{P}_i < \mathbb{P}_i^{req}$, the system considers that it satisfies the failure requirement of τ_i. Otherwise, Reghenzani *et al.* and RTailor mitigate the failure probability by reactive re-execution strategy.

B. Reactive Re-execution Strategy

The reactive re-execution strategy in Reghenzani *et al.* and RTailor allows additional re-execution(s) upon detection of a failure, expecting success on the re-execution. Instead of allowing an infinite number of re-executions, the reactive re-execution selects a proper $N_i^{re-exec}$, the allowed maximum number of re-execution(s) for τ_i that can sufficiently decrease the failure probability of τ_i. Figure 2 shows the flow chart of the reactive re-execution strategy, which iterates the re-execution until no failure is detected or $\mathbb{C}_i^{re-exec}$, the count of triggered re-execution(s) for τ_i reaches $N_i^{re-exec}$.

Reghenzani *et al.* and RTailor consider that τ_i fails only when the original execution and total $N_i^{re-exec}$ re-execution(s) continuously fail so that no more re-execution is allowed. Therefore, $\mathbb{P}_i^{re-exec}$, the failure probability of τ_i considering reactive re-execution, is derived by the following Equation (5).

$$\mathbb{P}_i^{re-exec} = (\mathbb{P}_i)^{(1+N_i^{re-exec})} \tag{5}$$

Reghenzani *et al.* and RTailor select optimal $N_i^{re-exec}$ that satisfies $\mathbb{P}_i^{re-exec} < \mathbb{P}_i^{req}$. RTailor can trigger the re-execution with partially (selectively) protected task for better schedulability. For the sake of simplicity, we skip the selective SIHFT.

Limitations: the re-execution strategies in SOTA fault-tolerant MCSs [2], [3] are *reactive* mechanisms, where the re-execution is only triggered in reaction to the detected failures. Silent data corruption (SDC) is defined as an undetected failure with corrupted outputs. If an execution of τ_i resulted in SDC, the reactive re-execution strategy cannot trigger the re-execution upon SDCs regardless of $N_i^{re-exec}$, but just considers

TABLE I
GLOSSARY OF TERMS USED IN PREFACE FOR A TASK τ_i.

Symbol	Definitions
$P(due\|fault)_i$	Conditional DUE probability given a fault
$P(sdc\|fault)_i$	Conditional SDC probability given a fault
$p\_benign'_i$, $p\_due'_i$, $p\_sdc'_i$	Benign & DUE & SDC rates per time unit
$\mathbb{P}\_benign_i$	Benign probability per execution
$\mathbb{P}\_due_i$, $\mathbb{P}\_sdc_i$	DUE & SDC probabilities per execution
N_i^{total}	Maximum number of total execution
M_i	Maximum number of completed executions
$\mathbb{P}\_sdc_i{}^{re-exec}$	DUE probability considering re-execution
$\mathbb{P}\_due_i{}^{re-exec}$	SDC probability considering re-execution
$\mathbb{C}_i^{total}$, $\mathbb{C}_i^{comp.}$	Count of total & completed executions
$\mathbb{C}_i^{sdc}$	Count of executions resulted in SDC
$\mathbb{P}\_comp._i(\mathbb{C}_i^{comp.})$	Probability that task completes $\mathbb{C}_i^{comp.}$ executions
$\mathbb{P}\_sdc\_major_i(\mathbb{C}_i^{comp.})$	Probability that the majority of $\mathbb{C}_i^{comp.}$ completed executions resulted in SDC

the execution successfully completed. Consequently, the reactive re-execution strategy only mitigates the detectable failures, rendering the rate of failures due to SDCs unmitigated.

III. PROPOSED APPROACH: PREFACE

We propose PREFACE to address the limitations of SOTA fault-tolerant MCSs. PREFACE formulates the failure probabilities of tasks in MCSs by separately considering detected unrecoverable errors (DUEs) and SDCs. To cope with SDCs, PREFACE uses a novel, precise mechanism to trigger proactive re-executions, only when re-executions are necessary to meet failure requirements. PREFACE assumes that tasks are protected by a detection-only SIHFT scheme without selective protection, and shares the assumptions discussed in Section II.

PREFACE aims to provide adaptive redundancy of SIHFT-protected executions, and can be easily extended with detection and correction SIHFT schemes or selective protection. TABLE I provides glossary of terms for the following discussions.

A. Failure Formulation by Separating DUEs and SDCs

PREFACE separately formulates DUE and SDC probabilities per execution. This separated formulation for τ_i requires $P(due|fault)_i$ and $P(sdc|fault)_i$, the conditional DUE and SDC probabilities given a fault, which means the probabilities that a fault on τ_i results in DUE or SDC, respectively. $P(due|fault)_i$ and $P(sdc|fault)_i$ can be obtained from the fault injections for τ_i, by categorizing the results as follows.

- **Benign:** A fault is masked (does not affect execution) or corrected (if SIHFT supports correction), or no fault occurs.
- **DUE:** A fault causes a task to fail and is detected by the system. DUE includes (i) crash (a task is aborted), (ii) hang (a task is timed out and detected by a watchdog), and (iii) detection (a fault is detected by detection-only SIHFT).
- **SDC:** A fault causes the output of the task execution to be corrupted, but does not cause any visible failures.

PREFACE formulates the DUE and SDC rates per time unit for a task τ_i by extending Equation (1) and (2) as follows.

$$p\_due'_i = 1 - (1 - \lambda \times P(due|fault)_i)^{\frac{1}{k}} \quad (6)$$

$$p\_sdc'_i = 1 - (1 - \lambda \times P(sdc|fault)_i)^{\frac{1}{k}} \quad (7)$$

In Equations (6) and (7), $p\_due'_i$ and $p\_sdc'_i$ represent the DUE and SDC rates of τ_i per time unit, respectively. λ represents the fault rate per hour. $p\_benign'_i$, the benign rate per time unit of τ_i, i.e., no DUE- or SDC-inducing fault occurs during the time unit, is derived by Equation (8).

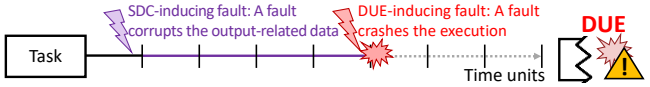


Fig. 3. Example scenario of an execution with DUE- and SDC-inducing faults.

$$p\_benign'_i = 1 - (p\_due'_i + p\_sdc'_i) \quad (8)$$

An execution results in benign, i.e., the task completes the execution without visible failure and with correct results, if no DUE- or SDC-inducing fault occur. $\mathbb{P}\_benign_i$, the benign probability per execution of τ_i, can be derived from $p\_benign'_i$ and E_i, the execution time of τ_i, as Equation (9).

$$\mathbb{P}\_benign_i = (p\_benign'_i)^{E_i} \quad (9)$$

An execution may encounter multiple faults. For example, in Fig. 3, an SDC-inducing fault affects an execution, and a DUE-inducing fault crashes the execution. This case is DUE, since the task crashes regardless of the correctness of the output, so the system is aware of the failure. Therefore, $\mathbb{P}\_due_i$, the DUE probability per execution of τ_i, can be derived from $p\_due'_i$ regardless of SDCs by calculating the probability that at least one DUE-inducing fault occurs during E_i as Equation (10).

$$\mathbb{P}\_due_i = 1 - (1 - p\_due'_i)^{E_i} \quad (10)$$

PREFACE can simply derive $\mathbb{P}\_sdc_i$, the SDC probability per execution, based on the fact that a completed execution, i.e., non-DUE execution should be benign or SDC as Equation (11).

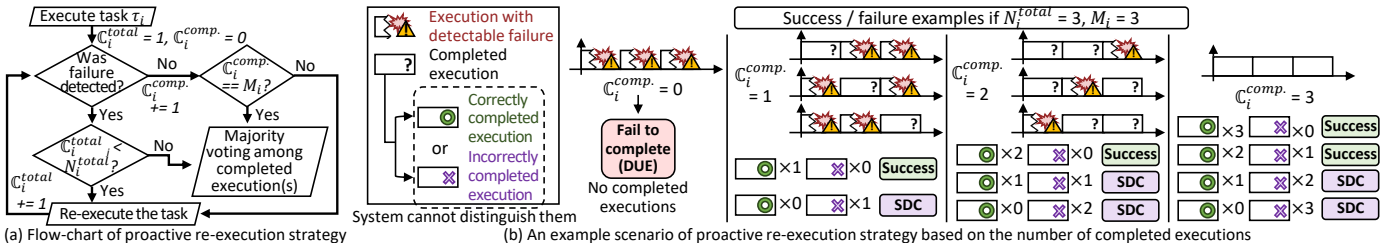
$$\mathbb{P}\_sdc_i = (1 - \mathbb{P}\_due_i) - \mathbb{P}\_benign_i \quad (11)$$

B. Proactive Re-execution Strategy

PREFACE introduces a systematic, proactive re-execution strategy that obtains adaptive redundancy of completed executions to mitigate $\mathbb{P}\_sdc_i$ and $\mathbb{P}\_due_i$. Continuously re-executing a task until obtaining fixed number of completed executions results in unbounded worst-case execution time (WCET) due to potential DUEs. Instead, PREFACE assigns the maximum number of total and completed executions for each task.

- N_i^{total}: The maximum number of **total** executions allowed for τ_i. From now on, we define total executions as the executions of a task for a single schedule, including the first execution and following re-executions, no matter if each execution fails (DUE) or succeeds to complete (benign or SDC). Note that $N_i^{re-exec}$ in Section II-B does not include the first execution while N_i^{total} of PREFACE includes it.
- M_i: The maximum number of **completed** executions allowed for τ_i. From now on, we define completed executions as the executions of a task for a single schedule that did not result in DUEs and produced outputs, but the system cannot distinguish the outputs between correct or wrong (SDC).

Figure 4 (a) shows the flow chart of the proactive re-execution strategy, where $\mathbb{C}_i^{total}$ and $\mathbb{C}_i^{comp.}$ represents the counts of total and completed execution(s) for τ_i, respectively.



(a) Flow-chart of proactive re-execution strategy (b) An example scenario of proactive re-execution strategy based on the number of completed executions

Fig. 4. Flow chart and example scenario of the proactive re-execution strategy.

C. Failure Formulation under Proactive Re-execution Strategy

The challenge to formulate DUE and SDC probabilities of τ_i considering re-execution(s) is that the count of completed execution(s) ($\mathbb{C}_i^{comp.}$) varies due to DUEs. Figure 4 (b) shows an example scenario of $\mathbb{C}_i^{comp.}$ when N_i^{total} and M_i are 3. DUE failure scenario under proactive re-execution strategy is $\mathbb{C}_i^{comp.} = 0$ case, which means all N_i^{total} executions for τ_i result in DUE. PREFACE can derive $\mathbb{P}\_due_i{}^{re-exec}$, the DUE probability of τ_i considering re-execution, by Equation (12).

$$\mathbb{P}\_due_i{}^{re-exec} = (\mathbb{P}\_due_i)^{(N_i^{total})} \tag{12}$$

When $\mathbb{C}_i^{comp.} > 0$, PREFACE adaptively performs majority voting among the outputs from completed executions. SDC failure scenario considering re-execution represents cases where the majority of outputs are incorrect (SDC). Figure 4 (b) shows the successful and SDC scenario considering variable $\mathbb{C}_i^{comp.}$.

$\mathbb{P}\_sdc_i{}^{re-exec}$, the SDC probability of τ_i considering re-execution, can be derived from the probability that τ_i completes $\mathbb{C}_i^{comp.}$ executions ($\mathbb{P}\_comp._i(\mathbb{C}_i^{comp.})$) and the probability that majority of $\mathbb{C}_i^{comp.}$ completed executions resulted in SDCs ($\mathbb{P}\_sdc\_major_i(\mathbb{C}_i^{comp.})$), by the following Equation (13).

$$\begin{aligned}\mathbb{P}\_sdc_i{}^{re-exec} = &\sum_{\mathbb{C}_i^{comp.}=1}^{M_i}(\mathbb{P}\_comp._i(\mathbb{C}_i^{comp.}) \\ &\times \mathbb{P}\_sdc\_major_i(\mathbb{C}_i^{comp.}))\end{aligned} \tag{13}$$

Formulation of $\mathbb{P}\_comp._i(\mathbb{C}_i^{comp.})$ varies depending on $\mathbb{C}_i^{comp.}$. $\mathbb{C}_i^{comp.} < M_i$ implies that PREFACE stopped the re-execution because $\mathbb{C}_i^{total} = N_i^{total}$; otherwise, PREFACE will trigger the re-execution as shown in Figure 4 (a). In such cases, the count of DUEs is $N_i^{total} - \mathbb{C}_i^{comp.}$. In contrast, $\mathbb{C}_i^{comp.} == M_i$ implies that PREFACE just stopped the re-execution when the last execution has been completed (not DUE). In such cases, $\mathbb{C}_i^{total}$ can vary from $\mathbb{C}_i^{comp.}$ to N_i^{total}. Based on these observations, PREFACE formulates $\mathbb{P}\_comp._i(\mathbb{C}_i^{comp.})$ as follows.

$$\mathbb{P}\_comp._i(\mathbb{C}_i^{comp.}) = \begin{cases} \binom{N_i^{total}}{\mathbb{C}_i^{comp.}}(\mathbb{P}\_due_i)^{N_i^{total}-\mathbb{C}_i^{comp.}}(1-\mathbb{P}\_due_i)^{\mathbb{C}_i^{comp.}}, \\ \text{when } \mathbb{C}_i^{comp.} < M_i \\ \hline \sum_{\mathbb{C}_i^{total}=M_i}^{N_i^{total}} \binom{\mathbb{C}_i^{total}-1}{M_i-1}(\mathbb{P}\_due_i)^{\mathbb{C}_i^{total}-M_i}(1-\mathbb{P}\_due_i)^{M_i}, \\ \text{when } \mathbb{C}_i^{comp.} == M_i \end{cases} \tag{14}$$

Note that Equation (14) considers permutations of set with $M_i - 1$ completed executions and $\mathbb{C}_i^{total} - 1$ DUEs when $\mathbb{C}_i^{comp.} == M_i$ since the last execution should be completed.

$\mathbb{P}\_sdc\_major_i(\mathbb{C}_i^{comp.})$ represents the probability that the majority of $\mathbb{C}_i^{comp.}$ completed executions ($\geq \lceil \mathbb{C}_i^{comp.}/2 \rceil$) resulted in SDCs. PREFACE derives $\mathbb{P}\_sdc\_major_i(\mathbb{C}_i^{comp.})$ based on $\mathbb{C}_i^{sdc}$, the count of SDC executions of τ_i, varying from $\lceil \mathbb{C}_i^{comp.}/2 \rceil$ to $\mathbb{C}_i^{comp.}$ as the following Equation (15).

$$\begin{aligned}&\mathbb{P}\_sdc\_major_i(\mathbb{C}_i^{comp.}) = \\ &\sum_{\mathbb{C}_i^{sdc}=\lceil \mathbb{C}_i^{comp.}/2 \rceil}^{\mathbb{C}_i^{comp.}} \binom{\mathbb{C}_i^{comp.}}{\mathbb{C}_i^{sdc}} \left(\frac{\mathbb{P}\_sdc_i}{\mathbb{P}\_sdc_i + \mathbb{P}\_benign_i}\right)^{\mathbb{C}_i^{sdc}} \\ &\times \left(\frac{\mathbb{P}\_benign_i}{\mathbb{P}\_sdc_i + \mathbb{P}\_benign_i}\right)^{\mathbb{C}_i^{comp.}-\mathbb{C}_i^{sdc}}\end{aligned} \tag{15}$$

The failure probability of τ_i considering re-execution is $\mathbb{P}\_due_i{}^{re-exec} + \mathbb{P}\_sdc_i{}^{re-exec}$. PREFACE should select optimal N_i^{total} and M_i that satisfy both $N_i^{total} \geq M_i$ and $\mathbb{P}\_due_i{}^{re-exec} + \mathbb{P}\_sdc_i{}^{re-exec} < \mathbb{P}_i^{req}$, where $\mathbb{P}_i^{req}$ represents the required failure probability of τ per execution from Equation (4). PREFACE explores optimal N_i^{total} and M_i for τ_i by gradually increasing N_i^{total}, selecting odd M_i from one to N_i^{total} and checking whether $\mathbb{P}\_due_i{}^{re-exec} + \mathbb{P}\_sdc_i{}^{re-exec} < \mathbb{P}_i^{req}$ is satisfied, since the majority bound of even number x is equal to the one of $x - 1$.

IV. Evaluation

We evaluate *schedulability* (feasibility of real-time scheduling), *reliability* (if meeting failure requirements), and *slack time* of PREFACE in comparison with Reghenzani *et al.* [2] and RTailor [3]. We first conduct fault injection experiments with SIHFT-applied benchmarks to measure/estimate $P(due|fault)$ and $P(sdc|fault)$, and generate synthetic task sets based on the $P(due|fault)$ and $P(sdc|fault)$. Finally, we test whether Reghenzani *et al.* [2], RTailor [3], Naive TMR, and PREFACE can satisfy the reliability and schedulability of each task set.

A. Fault-injected Datasets

We conducted fault injection experiments to generate datasets with $P(due|fault)$ and $P(sdc|fault)$ of SIHFT-applied applications. TABLE II shows the results of the fault injection.

1) Benchmarks with SIHFT: We selected seven benchmarks from MiBench [11] listed in TABLE II and reduced their input size feasible for low-level simulations. We compiled the benchmarks by LLVM [12], [13], reserving around half registers, and applied CHITIN [9], a SOTA in-thread SIHFT for fault detection, at assembly level with Python scripts.

2) Fault injectors: We built register-transfer level (RTL) fault injectors that inject a single bit-flip on flip-flops [14] of an open-source microprocessor. Specifically, we modified RTL codes of Mor1kx Cappuccino core [15] based on OpenRisc ISA [16] to inject faults by flipping a random bit of reg variables on their hardware components and simulated the modified Mor1kx Cappuccino on Icarus Verilog simulator [17].

TABLE II
FAULT INJECTION DATASETS FOR OUR EVALUATION.

Benchmark	Benign	DUE	SDC
ADPCM_e.	73.6%	26.4%	0.0185%
ADPCM_d.	73.1%	26.8%	0.0370%
bitcount	77.6%	22.4%	0.0185%
CRC32	72.1%	27.9%	0.0185%

Benchmark	Benign	DUE	SDC
qsort	73.4%	26.6%	0.0278%
sha	70.4%	29.6%	0.0093%
stringsearch	74.5%	25.5%	0.0185%

3) Generating datasets: We flipped one random bit of the processor executing a benchmark and classified the result as benign, DUE (detected by SIHFT, crash, and hang), or SDC. We consider an execution is hanged if the simulation time exceeds $2\times$ fault-free simulation time of the benchmark. For each benchmark, we repeated the fault injection trials with one random fault per trial 10,800 times. The number of trials for each hardware component is proportional to the number of bits in the component, following the evaluation in SOTA SIHFT [9].

B. MCS Environment and Task Sets

System model: In our evaluation, we use a simple MCS model with mixed-criticality tasks without dependencies, running on a single-core processor, with the following two time-criticality levels: critical tasks and non-critical tasks.

1. **Critical tasks** are periodic tasks with deadlines equal to the beginning of the next period. All critical tasks have the same scheduling priority, higher than non-critical tasks. Each critical task is randomly assigned a required failure rate among $10^{-9}/h$, $10^{-7}/h$, or $10^{-5}/h$ based on DO-178C [10], or $10^{-3}/h$ based on the upper bound of the "probable" failure condition in AC 25.1309-1B [18], and $P(due|fault)$ and $P(sdc|fault)$ from TABLE II. We assume that SIHFT protects critical tasks, and the system can detect DUEs including crash, hang, and detected by SIHFT cases at the WCET of the execution.
2. **Non-critical tasks** are best-effort tasks that run during the slack time between critical tasks when there is no critical task running on the processor. Therefore, more slack time will allow more execution of non-critical tasks.

Experimental scenarios and task sets: We generate experimental scenarios using the same method in RTailor [3], where each scenario is defined as a tuple (n, λ, U) where n is the number of tasks, λ is the fault rate ranging from $10^-7/h$ to $10^-1/h$ (selected based on the literature on fault rates [19]–[21]), and U is the total utilization without considering re-executions ranging from 0.1 to 0.5. For each tuple (n, λ, U), we randomly distribute the total utilization to tasks in the task set using the algorithm UUnifast [22]. Then, for each task in the task set, we randomly allocate the period ranging from 50 to 1,000 time units, required failure rate for critical tasks, and $P(due|fault)$ and $P(sdc|fault)$ from TABLE II. We set E_i, the WCET of one execution of task τ_i by the given utilization $\times$ period / N_i^{total} of PREFACE. We only generate task sets for critical tasks, assuming that non-critical tasks run best-effort while critical tasks are not running.

C. Experimental Results

We test the feasibility and reliability of each task set with different failure estimation models and re-execution strategies of Reghenzani *et al.* [2], RTailor [3], PREFACE, and Naive TMR.

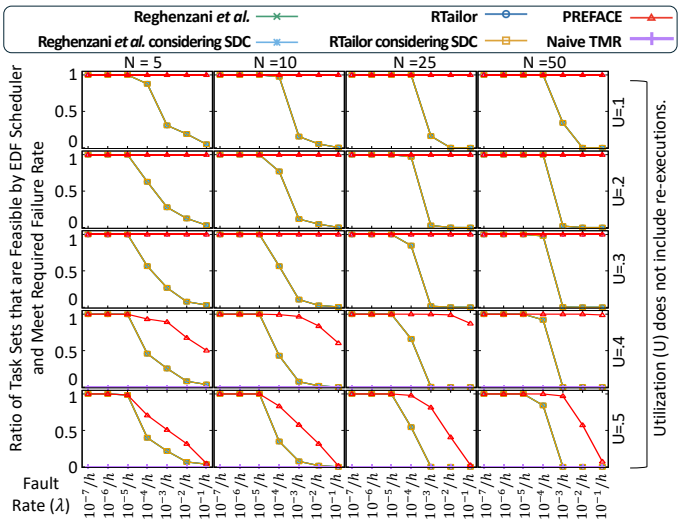


Fig. 5. The ratios of schedulable and reliable task sets out of 1,000 task sets. in total. N indicates the number of critical tasks in each task set.

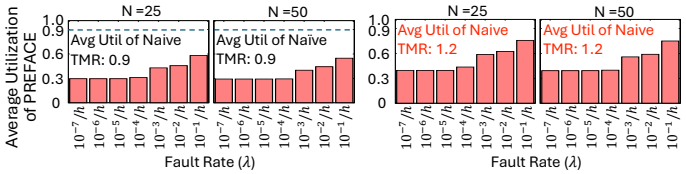
For Reghenzani *et al.* and RTailor, we compute the failure rate per execution with Equation (3) and find $N_i^{re-exec}$ satisfying Equation (5). For PREFACE, we compute N_i^{total} and M_i for each task τ_i that make the sum of $\mathbb{P}\_due_i$ (Equation (12)) + $\mathbb{P}\_sdc_i$ (Equation (13)) $<$ $\mathbb{P}_i^{req}$ (Equation (4)). Among multiple possible sets of N_i^{total} and M_i, we choose the set with the minimum N_i^{total} for better schedulability. Naive TMR adopts the re-execution strategy of PREFACE while uniformly assigning $N_i^{total} = 3$ and $M_i = 3$ for all tasks.

Reliability and schedulability of critical tasks: With the execution parameters, we check whether a critical task satisfies its required failure rate (i.e., *reliability*) by comparing the estimated failure rate with the required failure rate for each task. Then, we determine a "task set" is *reliable* if all tasks in the task set meet their required failure rates. We use the failure formulation of PREFACE for this reliability check since only PREFACE can precisely consider both DUEs and SDCs.

We also test a task set's *schedulability* using the earliest deadline first (EDF) scheduler [23], [24]. A task set is *schedulable* under the preemptive EDF scheduler if and only if the total utilization of the tasks is less than or equal to 1. We compute the total utilization by summing each task's utilization with re-execution. The utilization of τ_i with re-executions is computed by $E_i \times N^{total}/T_i$ where E_i and T_i are the worst-case execution time for one execution and period of τ_i, respectively.

Fig. 5 shows the ratio of schedulable and reliable task sets in each scenario. For each graph with N (number of critical tasks in each task set) and utilization, the x-axis represents the fault rate per hour (λ) while the y-axis represents the ratios of schedulable and reliable task sets. The ratios from Reghenzani *et al.* (green 'x') and RTailor (blue '$\circ$') are almost identical and thus, their plots are overlapped. Overall, PREFACE (red '$\triangle$') significantly improves the feasibility and reliability compared to SOTA approaches when $\lambda \geq 10^{-3}$.

Specifically, when $\lambda \geq 10^{-3}$, Reghenzani *et al.* and RTailor usually fail to meet the failure requirements of tasks, since their reactive re-execution strategy can only mitigate DUEs but cannot mitigate SDCs. On the other hand, PREFACE can



(a) When utilization w/o re-executions is **0.3**. (b) When utilization w/o re-executions is **0.4**.

Fig. 6. Average utilization by critical tasks for PREFACE and Naive TMR for the schedulable and reliable task sets in PREFACE.

mitigate both DUEs and SDCs with the proactive re-execution strategy. Still, as shown in Figure 5, the ratios of schedulable and reliable task sets from PREFACE decrease for some cases. This is because when the fault rate increases, PREFACE needs to allow higher maximum numbers of total executions (N_i^{total}) and maximum number of completed executions (M_i) for each task τ_i but such higher N_i^{total} and M_i can make the task not schedulable. Still, such tasks are also infeasible for Reghenzani *et al.* and RTailor due to the higher SDC probabilities.

Naive TMR (purple '|') shows the same ratio of schedulable and reliable task sets as PREFACE when the utilization is 0.3 or less, but the ratio becomes zero when the utilization is 0.4 or 0.5 since it always triples the utilization by $N_i^{total} = 3$ and $M_i = 3$. In contrast, PREFACE assigns $M_i = 3$ to τ_i only when majority voting is required, based on its accurate failure formulation. We never observed $M_i > 3$ in our evaluation.

We also make it as favorable as possible for the formulation of compared SOTA approaches. Specifically, we find N^{total} under the assumption that the other approaches also consider $\mathbb{P}\_sdc$; for each task unreliable with Reghenzani *et al.* or RTailor, we check if the task can meet the requirement by only increasing N^{total}. Still, as shown in Fig. 5 marked with "Reghenzani *et al.* considering SDC" (blue '*') and "RTailor considering SDC" ('□'), just improving the failure formulation cannot help the reliability of SOTA approaches, due to the lack of proper re-execution strategy to mitigate SDCs.

Slack time for non-critical tasks: We evaluate the performance of non-critical tasks by estimating the slack time between critical tasks. Concretely, we calculate the average utilization time of the task because the average utilization represents the proportion of time that is not slack time. For a task τ_i, we can compute the average utilization time by $\sum_n^{N^{total}} n \times E_i \times p_{executed}(n)$ where $p_{executed}(n)$ denotes the probability that the task is executed n times, i.e., DUE appears $n-1$ times.

Fig. 6 shows the estimated average utilization sets that are feasible and reliable for PREFACE, wherethere are 25 and 50 tasks and a total utilization without re-execution is 0.3 and 0.4. We only show the results of PREFACE and Naive TMR with feasible and reliable task sets in PREFACE, since Reghenzani *et al.* and RTailor cannot satisfy failure requirements for most of the task sets under higher fault rate, as shown in Figure 5.

PREFACE negligibly affects the average utilization when the fault rate, λ, is less than $10^{-4}/h$. When $\lambda \geq 10^{-4}/h$, the average utilization in PREFACE gradually increases, since PREFACE assigns $M_i = 3$ for any task τ_i that $\mathbb{P}\_sdc_i > \mathbb{P}_i^{req}$. Still, the average utilization of PREFACE is much lower than the one of Naive TMR, by identifying tasks to assign $M_i = 3$ with its failure formulation. Note that when PREFACE shows higher utilization than the total utilization without re-execution (higher than 0.3 in Fig. 6 (a) and higher than 0.4 in Fig. 6 (b)), it implies that there are task sets that have tasks requiring $M_i > 1$; SOTA MCSs cannot meet their failure requirements.

D. Limitations and Future Work

Our evaluation did not include selective SIHFT, which is the main contribution of RTailor [3], and the overhead and failures in the majority voting in PREFACE due to the negligible execution time of voting. We note that PREFACE can take failures in majority voting into consideration with minor changes.

V. Related Work

This section provides a further review of the literature on related work that has not been discussed in Section II.

Huang *et al.* [25] explicitly model the safety requirements of tasks based on the safety standards such as DO-178B [26] and assign maximum number of re-execution(s) for each task when the failure probabilities are given. Von der Brüggen *et al.* [27] discovered that WCETs with and without the fault recovery process differ largely while faults occur rarely, and defined timing-tolerant tasks that can occasionally miss their deadlines in uncertain or faulty execution environments. Caplan *et al.* [28] explored the design space of reliability and real-time requirements by dynamically adjusting hardware lock-step. Still, none of their formulations correctly address SDCs.

Pathan [29] utilized the backup(s) of a task upon failures for re-execution, where the backup can be a diverse implementation of the task. Huang *et al.* [30] utilized primary version with high-quality outputs and alternative version with acceptable outputs and validation of a task for re-execution. Zeng *et al.* [31] extended the re-execution with replications for multicores. Lin *et al.* [32] and Chen *et al.* [33] adopted checkpointing for re-executions so that the re-execution upon failures can go back to the checkpoint. The MSRP-FT protocol [34] addresses the resource contention problem in multiprocessor fault-tolerant MCSs. Above solutions focus on providing alternative re-executions rather than addressing SDCs. Contributions of PREFACE are orthogonal to such alternative re-executions.

VI. Conclusion

We propose PREFACE to address the challenging problem of SDCs in fault-tolerant MCSs. PREFACE advances SOTA approaches by formulating failure probabilities with considering SDCs and proactively triggering re-executions even when failures are not detected. Experiments show the schedulability and reliability of PREFACE compared to the SOTA approaches.

Acknowledgment

This material is based upon work supported by the National Science Foundation (NSF) under Award No. POSE-2449200 (An Open-Source Ecosystem to Coordinate Integration of Cyber-Physical Systems) and grants ACED 2436016; Semiconductor Research Corporation (SRC) project 3154; BK21 FOUR project (AI-driven Convergence Software Education Research Program) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (41202420214871).

REFERENCES

[1] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Symposium on High-Performance Computer Architecture*. IEEE, 2005, pp. 243–247.

[2] F. Reghenzani, Z. Guo, L. Santinelli, and W. Fornaciari, "A mixed-criticality approach to fault tolerance: integrating schedulability and failure requirements," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2022, pp. 27–39.

[3] S.-Y. Huang, J. Zeng, X. Deng, S. Wang, A. Sifat, B. Bharmal, J.-B. Huang, R. Williams, H. Zeng, and C. Jung, "RTailor: Parameterizing soft error resilience for mixed-criticality real-time systems," in *2023 IEEE Real-Time Systems Symposium*. IEEE, 2023, pp. 344–357.

[4] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante, *Software-implemented hardware fault tolerance*. Springer Science & Business Media, 2006.

[5] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "SWIFT: Software implemented fault tolerance," in *International Symposium on Code Generation and Optimization*. IEEE, 2005, pp. 243–254.

[6] F. Restrepo-Calle, A. Martínez-Álvarez, S. Cuenca-Asensi, and A. Jimeno-Morenilla, "Selective SWIFT-R: A flexible software-based technique for soft error mitigation in low-cost embedded systems," *Journal of Electronic Testing*, vol. 29, no. 6, pp. 825–838, 2013.

[7] M. Didehban, S. R. D. Lokam, and A. Shrivastava, "InCheck: An in-application recovery scheme for soft errors," in *Proceedings of the 54th Annual Design Automation Conference*, 2017, pp. 1–6.

[8] M. Didehban, A. Shrivastava, and S. R. D. Lokam, "NEMESIS: A software approach for computing in presence of soft errors," in *2017 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2017, pp. 297–304.

[9] H. So, M. Didehban, J. Jung, A. Shrivastava, and K. Lee, "CHITIN: A comprehensive in-thread instruction replication technique against transient faults," in *2021 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2021, pp. 1440–1445.

[10] B. Brosgol and C. Comar, "DO-178C: A new standard for software safety certification," in *Presented at the 22nd Systems and Software Technology Conference (SSTC)*, vol. 26, 2010, p. 29.

[11] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.

[12] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *International symposium on code generation and optimization, 2004. CGO 2004.* IEEE, 2004, pp. 75–86.

[13] OpenRISC, "llvm-or1k, GitHub," [accessed January-2026]. [Online]. Available: https://github.com/openrisc/llvm-or1k

[14] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–10.

[15] OpenRISC, "mor1kx Github repository," [accessed January-2026]. [Online]. Available: https://github.com/openrisc/mor1kx

[16] J. Tandon, "The OpenRISC processor: open hardware and linux," *Linux Journal*, vol. 2011, no. 212, p. 6, 2011.

[17] S. Williams and M. Baxter, "Icarus Verilog: Open-source Verilog more than a year later," *Linux Journal*, vol. 2002, no. 99, p. 3, 2002.

[18] F. A. Administration, "System design and analysis," Advisory Circular AC 25.1309-1B, [accessed January-2026]. [Online]. Available: https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_25.1309-1B.pdf

[19] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive fpga-based space computing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, pp. 1–30, 2012.

[20] X. Li, M. C. Huang, K. Shen, and L. Chu, "A realistic evaluation of memory hardware errors and software system susceptibility," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[21] L. A. Tambara, P. Rech, E. Chielle, J. Tonfat, and F. L. Kastensmidt, "Analyzing the impact of radiation-induced failures in programmable socs," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2217–2224, 2016.

[22] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1–2, p. 129–154, May 2005. [Online]. Available: http://dx.doi.org/10.1007/s11241-005-0507-9

[23] W. Horn, "Some simple scheduling algorithms," *Naval Research Logistics Quarterly*, vol. 21, no. 1, pp. 177–185, 1974.

[24] Z. A. Hammadeh, S. Quinton, and R. Ernst, "Weakly-hard real-time guarantees for earliest deadline first scheduling of independent tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 6, pp. 1–25, 2019.

[25] P. Huang, H. Yang, and L. Thiele, "On the scheduling of fault-tolerant mixed-criticality systems," in *Proceedings of the 51th Annual Design Automation Conference*, 2014, pp. 1–6.

[26] L. A. Johnson *et al.*, "Do-178b: Software considerations in airborne systems and equipment certification," *Crosstalk, October*, vol. 199, pp. 11–20, 1998.

[27] G. von der Brüggen, K.-H. Chen, W.-H. Huang, and J.-J. Chen, "Systems with dynamic real-time guarantees in uncertain and faulty execution environments," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 303–314.

[28] J. Caplan, Z. Al-Bayati, H. Zeng, and B. H. Meyer, "Mapping and scheduling mixed-criticality systems with on-demand redundancy," *IEEE Transactions on Computers*, vol. 67, no. 4, pp. 582–588, 2017.

[29] R. M. Pathan, "Fault-tolerant and real-time scheduling for mixed-criticality systems," *Real-Time Systems*, vol. 50, pp. 509–547, 2014.

[30] Y. Huang and Q. Deng, "Fault-tolerant scheduling of primary-alternate version based on variable workload," in *2016 International Symposium on System and Software Reliability (ISSSR)*. IEEE, 2016, pp. 119–128.

[31] L. Zeng, P. Huang, and L. Thiele, "Towards the design of fault-tolerant mixed-criticality systems on multicores," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded System*, 2016, pp. 1–10.

[32] J. Lin, "Towards a fault-tolerant, scheduling methodology for safety-critical certified information systems," *Journal of International Technology and Information Management*, vol. 27, no. 3, pp. 84–99, 2019.

[33] G. Chen, N. Guan, K. Huang, and W. Yi, "Fault-tolerant real-time tasks scheduling with dynamic fault handling," *Journal of Systems Architecture*, vol. 102, p. 101688, 2020.

[34] N. Chen, S. Zhao, I. Gray, A. Burns, S. Ji, and W. Chang, "MSRP-FT: Reliable resource sharing on multiprocessor mixed-criticality systems," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 201–213.