Leveraging Weakly-hard Constraints for Improving System Fault Tolerance with Functional and Timing Guarantees

Hengyi Liang, Zhilu Wang, Ruochen Jiao, Qi Zhu
{hengyiliang2018@u.,zhilu.wang@u.,RuochenJiao2024@u.,qzhu@}northwestern.edu
Department of Electrical and Computer Engineering, Northwestern University
Evanston, Illinois, USA

ABSTRACT

Many safety-critical real-time systems operate under harsh environment and are subject to soft errors caused by transient or intermittent faults. It is critical and yet often very challenging to apply fault tolerance techniques in these systems, due to resource limitations and stringent constraints on timing and functionality. In this work, we leverage the concept of weakly-hard constraints, which allows task deadline misses in a bounded manner, to improve system's capability to accommodate fault tolerance techniques while ensuring timing and functional correctness. In particular, we **a)** quantitatively measure control cost under different deadline hit/miss scenarios and identify weak-hard constraints that guarantee control stability; **b)** employ typical worst-case analysis (TWCA) to bound the number of deadline misses and approximate system control cost; **c)** develop an event-based simulation method to check the task execution pattern and evaluate system control cost for any given solution; and **d)** develop a meta-heuristic algorithm that consists of heuristic methods and a simulated annealing procedure to explore the design space. Our experiments on an industrial case study and synthetic examples demonstrate the effectiveness of our approach.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

Fault tolerance, weakly-hard, EED, EOC, timing guarantees

ACM Reference Format:
Hengyi Liang, Zhilu Wang, Ruochen Jiao, Qi Zhu. 2020. Leveraging Weakly-hard Constraints for Improving System Fault Tolerance with Functional and Timing Guarantees. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415717

1 INTRODUCTION

Many real-time embedded systems, such as automotive, avionics, and industrial automation systems, often operate under harsh environment and are subject to soft errors caused by transient or intermittent faults (e.g., those from radiation [3]). As those systems are often safety-critical, it is important to improve their resiliency by applying fault tolerance techniques [1, 19].

In the literature, various error detection and recovery mechanisms have been proposed [9, 16, 28, 35]. For instance, to address soft errors, there are both hardware based approaches [2, 30, 31] and software approaches [9, 25, 26]. In this work, we focus on addressing transient soft errors through software layer, by relying on error detection techniques to detect potential soft errors and possibly performing recovery jobs to correct them. As defined in [9], there are two main categories of error detection techniques, i.e., embedded error detection (EED) and explicit output comparison (EOC). EED-type techniques have built-in error detection mechanisms and do not reply on redundant execution. Some common EED approaches include watchdog timer [25], control flow checking and instruction signature checking [26]. In contrast, EOC-type techniques rely on explicit redundant execution with either temporal redundancy or spatial redundancy, e.g., executing the same task at least twice and compare the outputs. One common approach of EOC is the triple modular redundancy scheme [23]. In this work, we consider the general type of EED techniques and an EOC technique based on temporal redundancy, i.e., EOC tasks are executed twice on the same computation resource and in the case of a soft error, a re-execution job is scheduled immediately on the same resource.

Both EED and EOC techniques incur significant timing overhead, and thus quantitative schedulability analysis is needed to ensure system timing correctness. For instance, the work in [9] presents an offline scheduling algorithm for EOC-type techniques. The work in [35] explores the tradeoff between EOC- and EED-type techniques and presents an algorithm to optimize their selection and scheduling, while considering timing constraints. However, applying fault tolerance techniques to resource-constrained real-time systems is quite challenging and sometimes infeasible, as it is often difficult to meet the stringent hard timing constraints with the additional overhead from those fault tolerance techniques.

In this work, we present a novel approach for improving system fault tolerance that leverages the concept of *weakly-hard constraints* as defined in [4], where bounded deadline misses are allowed, to provide more slack in task execution and enable the addition of more error detection and correction measurements. Unlike traditional hard real-time constraints, weakly-hard constraints allow occasional deadline misses in a bounded manner, which are often specified as the maximum number of deadline misses allowed within a given number of consecutive job instances (or a window of time) [4, 12].

The exploration of weakly-hard constraints is motivated by the fact that many system functions can tolerate certain degree of

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415717

deadline misses while still satisfy functional correctness requirements. For example, recent works have studied control performance and stability under deadline misses specified by weakly-hard constraints [10, 14, 27]. The work in [10] proves an analytical upper bound of deadline miss ratio to ensure the stability of a distributed embedded control platform. In [27], the impact of deadline miss pattern on control performance is studied. The work in [14] presents a method to formally verify the safety of certain control systems under weakly-hard constraints, which is further improved in [13]. On the other hand, a number of approaches have been presented for schedulability analysis of real-time system with weakly-hard constraints [4, 6, 21, 29, 32, 34]. In [4], the response time analysis for periodic task is discussed. In [32], the schedulability analysis is modeled as a mixed integer linear programming (MILP) problem and applied to periodical tasks with unknown task activation offset. In [29], a model is proposed to describe task activation pattern, and typical worst-case analysis (TWCA) is introduced to bound the number of deadline misses due to overload. The work in [34] further improves the approach from [29]. Then, there is also limited work on trying to leverage the scheduling flexibility from weakly-hard constraints to improve other design objectives. For instance in [22], a co-design approach is presented to improve system security while ensuring control safety. In [15], application of weakly-hard paradigm to networked systems is discussed.

Our work is the **first to leverage weakly-hard constraints for improving fault tolerance**. There are two unique challenges to address for solving this problem: 1) While exploring weakly-hard constraints, we have to ensure that the allowed deadline misses will not cause functional incorrectness. In this work, we focus on the stability of control tasks under deadline misses, and the behavior of these tasks is particularly difficult to analyze when we consider the possible faults on them. 2) We need to analyze the system schedulability under the possible deadline misses from weakly-hard constraints *and* the potential redundant task execution from fault tolerance techniques. Addressing these two challenges requires new methods for both control and schedulability analysis.

We address these two challenges by developing new methods to analyze control stability and system schedulability under deadline misses, faults, and the application of EED or EOC fault-tolerance techniques. Based on these analysis methods, we also develop an optimization algorithm for exploring the design space to improve a system-level fault-tolerance metric. More specifically, our work makes the following novel contributions:

- We develop a control analysis method for linear time-invariant (LTI) systems to formally derive the weakly-hard constraints that can ensure system stability (e.g., the system can be brought back to the equilibrium state under deadline misses), and to quantitatively measure the control cost under different deadline hit/miss patterns.
- We develop two schedulability analysis methods. One is to model tasks as the superposition of typical and overload activation and provide an upper-bound of the deadline misses (the control cost can be approximated based on this upper-bound). The other method uses an event-based simulation to record the exact pattern of deadline hits and misses (the worst-case control cost can be calculated under single transient error in this method).
- We develop a meta-heuristic optimization algorithm to explore the design space, including task allocation, priority assignment, and the choice of fault tolerance techniques (EED, EOC, or none). We conduct experiments on an industrial case study and a set of synthetic examples. Our experiments demonstrate the effectiveness of our approach in improving system fault tolerance and trading off between control cost and error coverage.

The rest of the paper is organized as follows. Section 2 introduces our system mode, including task execution model and control model. Section 3 presents our problem analysis and formulation, including the analysis on control stability and cost. Section 4 introduces our schedulability analysis methods and our meta-heuristic optimization algorithm. Section 5 presents the experimental results. Secion 6 concludes this work.

2 SYSTEM MODEL

We consider a real-time distributed platform, with multiple homogeneous single-core CPUs (communication is not considered in this work). Let $\mathcal{E} = \{e_1, \ldots, e_n\}$ be the set of CPUs. The functional layer is described by a set of independent tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$. Each task τ_i has a fixed period t_i, a deadline d_i, a worst case execution time (WCET) c_i and a static priority p_i. We assume that the system is subject to uncertainties such as external disturbance and transient soft errors. To alleviate the impact of uncertainties, we assume that **a)** some tasks can be equipped with error detection and recovery techniques, and **b)** some control tasks can tolerate certain degree of deadline miss. In this study, we consider two types of fault-tolerance techniques, EED and EOC. For each task $\tau_i \in \mathcal{T}$, we use a variable o_{τ_i} to denote the choice of fault-tolerance technique. Once a transient soft error is detected, corresponding task re-execution is followed to correct the soft error. Due to the difference between the two fault-tolerance techniques and the random arrival of soft errors, we model each task as a superposition of typical and overload activation, as explained below in details.

2.1 Error Detection Strategy and Modeling

For simplicity, we consider a single-error model in this work, where we assume that there is at most one transient soft error within the task set hyper-period (in practice this covers vast majority of the cases). Let C_i be the worst case execution time with error detection for task τ_i, and c_i be the original WCET when error detection is not applied. Then the worst case execution time with error detection for any task τ_i can be defined as in [35]:

$$C_i = c_i + \rho_i\left(o_i(c_i + \Lambda_i) + (1 - o_i)\Delta c_i\right), \tag{1}$$

where ρ_i denotes whether any error detection (EOC or EED) is applied for task τ_i, Λ_i the time for output comparison and Δc_i the EED overhead. Moreover, $o_i = 1$ if EOC is selected, otherwise $o_i = 0$. Note that C_i only includes WCET and error detection overhead. Once an error is detected, a re-execution is scheduled immediately. Let CR_i be the error recovery/re-execution time for a task τ_i. We have

$$CR_i = \rho_i\left(c_i + (1 - o_i)\Delta c_i\right).$$

As we will discuss later, CR_i corresponds to the execution time of an overload activation due to transient soft errors, while C_i is the execution time of regular periodic activation.

2.2 Task Execution Model

Considering the sporadic nature of transient soft errors, we characterize our task model by its activation pattern and execution time pattern, similarly as in [20]. Each pattern is further distinguished by a typical component and an overload component. More specifically, for each task with any error detection technique, the periodical activation pattern corresponds to the typical model, whereas the sporadic overload is due to addressing the transient soft errors. They are formally defined below.

Definition 2.1. Event models [20]: The event models $\eta_i^{-,(tp)}(\Delta t)$ and $\eta_i^{+,(tp)}(\Delta t)$ ($\eta_i^{-,(o)}(\Delta t)$ and $\eta_i^{+,(o)}(\Delta t)$, respectively) provide lower and upper bound on the number of typical (overload, respectively) activations of task τ_i during any time interval $[t, t+\Delta t)$.

Due to the periodicity of task events, we have $\eta_i^{-,(tp)} = \eta_i^{+,(tp)} = \lceil \frac{\Delta t}{l_i} \rceil$. There is a minimal interval Δt_{error} between two consecutive soft errors and $\eta_i^{-,(o)} = \eta_i^{+,(o)} = \lceil \frac{\Delta t}{\Delta t_{error}} \rceil$, if applicable. For simplicity, we assume that the worst-case event model $\eta_i^{+} = \eta_i^{+,(tp)} + \eta_i^{+,(o)}$.

Definition 2.2. Execution time model [20]: The execution time model $\gamma_i^{-,(tp)}(n)$ and $\gamma_i^{+,(tp)}(n)$ ($\gamma_i^{-,(o)}(n)$ and $\gamma_i^{+,(o)}(n)$, respectively) provide lower and upper bound on the typical (overload, respectively) share of the service demand required by any n consecutive activation of task τ_i.

In this work, we have $\gamma_i^{-,(tp)}(n) = \gamma_i^{+,(tp)}(n) = n \times C_i$, where C_i is the WCET with error detection as defined in (1). Similarly, $\gamma_i^{-,(o)}(n) = \gamma_i^{+,(o)}(n) = n \times CR_i$.

Throughout the paper, we assume that some tasks can miss certain number of deadlines. We employ a general representation to characterize such task timing requirement. Let $\zeta_i = \{(k_i^1, N_i^1), \ldots, (k_i^{n_i}, N_i^{n_i})\}$ be the set of the weakly-hard constraints of task τ_i, where (k_i^j, N_i^j) means for any N_i^j consecutive activations of task τ_i, at most k_i^j deadline misses are allowed. $(0, 1)$ is the special case for hard deadline tasks. The system is schedulable if

$$dmm_i(N_i^j) \leq k_i^j, \forall j, 1 \leq j \leq n_i, \forall i, \tag{2}$$

where $dmm_i(N_i^j)$ is the maximum number of deadline misses of task τ_i in any N_i^j consecutive activations. We further assume that when a task instance misses its deadline, it will continue running until completion.

2.3 Control Model

We consider linear time-invariant (LTI) control tasks. The system dynamic is modeled as:

$$\dot{x}(t) = Ax(t) + Bu(t),$$
$$y(t) = Cx(t),$$

where A, B and C are system matrices, and $x(t)$, $u(t)$ and $y(t)$ are vectors representing the system state, control input and system output at time t, respectively. We further assume that the control task is activated periodically and follows the Logical Execution Time (LET) diagram. The LET implementation applies control input at the deadlines and provides fixed closed-loop delay [8, 27]. The corresponding discrete-time system dynamics with certain sampling period h is given by [17]:

$$x[k+1] = A_d x[k] + B_{d,0} u[k] + B_{d,1} u[k-1], \tag{3}$$

where $B_{d,0} = \int_0^{h-D} e^{As} \cdot B ds$ and $B_{d,1} = \int_{h-D}^{h} e^{As} \cdot B ds$. D is the relative deadline. By defining an augmented state matrix $z[k] = \begin{bmatrix} x[k] \\ u[k-1] \end{bmatrix}$, we can rewrite the delayed system in (3) as:

$$z[k+1] = A_{aug} z[k] + B_{aug} u[k], \tag{4}$$

where $A_{aug} = \begin{bmatrix} A_d & B_{d,1} \\ 0 & 0 \end{bmatrix}$, $B_{aug} = \begin{bmatrix} B_{d,0} \\ \mathbf{I} \end{bmatrix}$, $C_{aug} = \begin{bmatrix} C & 0 \end{bmatrix}$, with $\mathbf{0}$ and $\mathbf{I}$ denoting zero matrix and identity matrix of suitable dimensions, respectively. The control law $u[k] = -Kz[k]$ is calculated by pole place technique [18].

3 PROBLEM ANALYSIS AND FORMULATION

In this section, we introduce our analysis and formulation of the problem, including the definition for a system-level error coverage metric and the analysis for control stability and cost.

Illustrating Example: To illustrate how we leverage the weakly-hard constraints, let us consider 4 tasks running on a single-core CPU as defined in Table 1 and shown in Figure 1. If there is no error detection applied to these tasks, the taskset is schedulable under hard timing constraints. If we want to add EOC to the control task τ_4, the WCET (with error detection) of the control task becomes 2. The system is still schedulable when no soft error occurs. However, if there is an error, the control task has to schedule a re-execution job and the system with hard timing constraints is no long schedulable (as the re-execution job of the control task will miss its deadline), as shown in Figure 1.

If the control task is robust enough and can tolerate some deadline misses, we can leverage weakly-hard constraints to improve its fault tolerance. For instance, let us assume τ_4 satisfies $(2, 10)$ weakly-hard constraint, the system can be proven schedulable with EOC applied to τ_4.

Table 1: Task set of the illustrating example.

Task name	τ_1	τ_2	τ_3	Controller τ_4
Period	5	6	3	10
WCET	1	1	1	1

3.1 Error Coverage

We define a system-level *error coverage* metric as the probability that either the transient soft errors are either detected by the error detection technique or the errors occur during the idle time[1]. For a single-core CPU, assuming K uniformly distributed soft errors can happen within a hyper-period, the error coverage can be

[1]For system idle time, transient errors like memory errors may still occur. We assume that the probability that the program is affected by such error is negligible, although we can extend our formulation to cover idle time error.

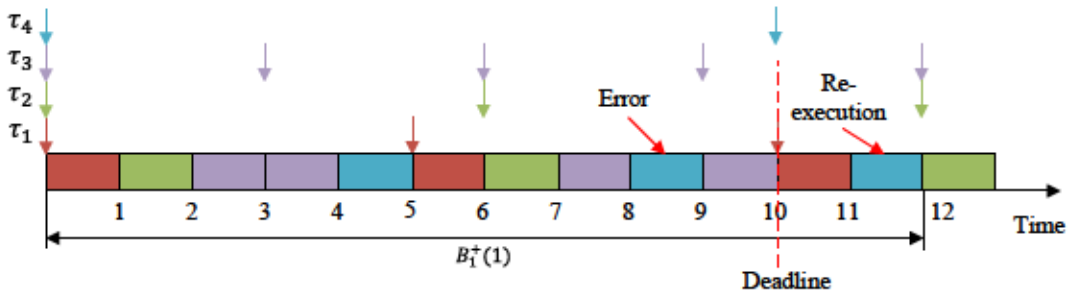

Figure 1: Illustrating example: task τ_4 cannot be applied with EOC under hard timing constraints.

approximated as defined in [35]:

$$P \approx \sum_{i=0}^{K} \sum_{j=0}^{i} \binom{K}{i} \cdot \binom{i}{j} \cdot (\frac{\alpha t_{eed}}{T_{hyper}})^{j} \cdot (\frac{\beta t_{eoc}}{T_{hyper}})^{i-j} \cdot (\frac{t_{idle}}{T_{hyper}})^{K-i},$$

where α and β are the average probabilities that an error is detected by EED and EOC, respectively. Here, t_{eed}, t_{eoc} and t_{none} are the time spent by tasks using EED, EOC and no error detection, respectively. $T_{hyper} = t_{eed} + t_{eoc} + t_{none} + t_{idle}$. For our study, we assume $K = 1$ and the above equation can be further approximated as follows according to [35]:

$$P \approx 1 - \frac{\sum_{\tau_i \in \mathcal{T}} (1 - \varepsilon_{\tau_i}) C_i}{t_i},$$

where ε_{τ_i} is the error detection rate for task τ_i. In our experiment, we set $\alpha = 0.7$ and $\beta = 1$, similarly as in [9, 35]. In our work, we assume that there is an error coverage requirement $EC\_Threshold$ defined, such that $P \leq EC\_Threshold$.

3.2 Control Stability and Cost

We consider stabilization controller that can bring the system back to the equilibrium state after a disturbance. Moreover, due to potential deadline misses, some control inputs may not always be applied on time. Following the LTE diagram, we assume that if a control task misses its deadline, the last control input will be used. The control cost is defined as the number of sampling periods needed to bring the system back to the equilibrium state [33]. The control input delay at the k-th instance can be bounded by:

$$\psi_k \leq \psi_{max} = \lceil \frac{r_i}{t_i} \rceil,$$

where r_i is the worst-case response time obtained by the schedulability analysis (detailed later in Section 4.1). Under such deadline miss case, the system state can be captured by the augmented state vector:

$$\xi[k] = [x^T[k], u^T[k-1], \ldots, u^T[k-\psi_{max}]]^T$$

and the system dynamic can be re-written as:

$$\xi[k+1] = A_\xi \xi[k] + B_\xi u[k] \tag{5}$$

$$A_\xi[k] = \begin{bmatrix} A_d & B_1 & \ldots & B_{\psi_{max}-1} & B_{\psi_{max}} \\ 0 & I & \ldots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \ldots & I & 0 \end{bmatrix}, B_\xi = \begin{bmatrix} 0 \\ I \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6}$$

where $B_{\psi_k} = B_{d,1}$ and $B_i = 0, \forall i \neq \psi_k$. $u[k-\psi_k]$ is the latest control input. The above system dynamic can be simplified as $\xi[k+1] = (A_\xi[k] - B_\xi[k]K_\xi)\xi[k] = \phi[k]\xi[k]$, where $K_\xi = [K, \mathbf{0}]$.

Control Stability: Assuming we are given the deadline hit/miss pattern of a control task within a hyper-period, the delay ψ_k in a hyper-period and the transition matrix $A_\xi[k]$ of each control period are known ($\forall k \in [0, N]$). Thus, we have:

$$\begin{aligned} \xi[k+N] &= \phi[k+N-1] \ldots \phi[k+1]\xi[k] \\ &= \prod_{i=k+N-1}^{0} \phi[i] \prod_{j=N-1}^{k} \phi[j]\xi[k] \\ &= \Phi_k \xi[k] \end{aligned}$$

The above system under deadline misses is asymptotically stable if the eigenvalues of Φ_k are within the unit circle for all k [33].

Control Cost: We define the cost of a control task as its ability to reject an external disturbance. Formally, let us assume that an external disturbance occurs at the k-th job and brings the system state to $x[k]$. We consider the disturbance is rejected if the residual disturbance after r sampling period satisfies:

$$\frac{||x[k+r]||}{||x[k]||} \leq J_{th}, \quad \forall r \geq h_k,$$

where J_{th} is a pre-defined threshold. Let $\xi[k+r] = \Phi_{k+r,k}\xi[k]$. Then $x[k+r]$ can be expressed as:

$$x[k+r] = \hat{I}\xi[k+r] = \hat{I}\Phi_{k+r,k}\xi[k] = \hat{I}^T\Phi_{k+r,k}\hat{I}x[k],$$

where $\hat{I} = \begin{bmatrix} I_{n\times n} & 0_{n\times m} \end{bmatrix}^T$. Then, the control cost is defined as:

Definition 3.1. The control cost metric of a control task τ_i is defined as $J_i = \max\{h_k | \forall k\}$, where h_k satisfies:

$$||\hat{I}^T\Phi_{k+r,k}\hat{I}|| \leq \frac{||x[k+r]||}{||x[k]||} \leq J_{th}, \quad \forall r \geq h_k. \tag{7}$$

Approximation of Control Cost: Calculating the precise control cost through event-based simulation is time-consuming. We try to approximate it by utilizing the deadline miss bound from the schedulability analysis (detailed in Section 4.1). Assuming that a control task satisfies the (k, n) weakly-hard constraint, where k is an upper-bound of the number of deadline misses in n consecutive activations. Then, for any $i \in \mathbb{N}$, $A_\xi[i]$ can take at most k different forms. Similarly, $\phi[i]$ can take at most k different forms. Since k and n are typically small numbers, we can exhaustively search all patterns of length n that satisfy the (k, n) deadline miss bound and find out the worst-case pattern as our approximation. Figure 2 shows result of a demo cruise control controller [10], when $n = 10$ and k changes from 0 to 4. The red line is the approximated cost[2].

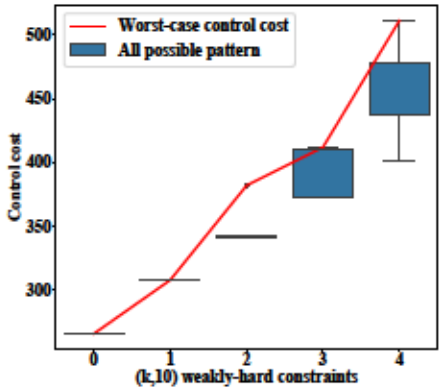

Figure 2: Control cost approximation of a demo controller, where k ranges from 0 to 4, when $n = 10$.

3.3 Overall Problem Formulation

Our optimization objective is the overall system-level control cost defined as the weighted sum of each individual control cost:

$$\mathcal{J} = \sum_{\tau_i \in \mathcal{T}_J} \omega_i \frac{J_i}{J_i^{des}}, \tag{8}$$

where ω_i are the given weights, $\mathcal{T}_J$ is the set of control tasks, and J_i^{des} is the desired control cost of task τ_i assuming no deadline miss occurs. We can formulate our problem as:

(P1) Given $\mathcal{E}, \mathcal{T}, \mathcal{T}_J$, optimize task assignment, selection of error detection mechanism $O = \{o_{\tau_1}, \ldots, o_{\tau_{|\mathcal{T}_J|}}\}$ such that

- schedulability constraints as defined in Equation (2) are satisfied,
- the stability of control tasks as defined above are satisfied, and
- the error coverage requirement $EC\_Threshold$ is satisfied.

.

4 META-HEURISTIC ALGORITHM FOR DESIGN SPACE EXPLORATION

Our meta-heuristic algorithm relies on the schedulability analysis under weakly-hard constraints and the fault-tolerance model. In this section, we first introduce the two schedulability analysis methods and then we present our optimization algorithm.

[2]In this demo example, for $k = 2$, there is an outlier point when the deadline misses are evenly distributed.

4.1 Schedulability Analysis

We assume that tasks running on the same CPU is scheduled by the static priority preemptive (SPP) scheduling policy. In this study, we leverage two different schedulability analysis methods for weakly-hard systems. One extends the work from [34], where the deadline miss model can be upper-bounded by employing typical worst-case analysis (TWCA). The other extends the event-based schedulability analysis in [22]. The event-based schedulability simulates the execution of tasks within a hyper-period and derives the deadline miss patterns for all tasks in a single run. In the following, we will briefly introduce TWCA and then a detailed explanation of the event-based simulation with error injection.

4.1.1 Bounding Deadline Miss Model Using TWCA. Our schedulability analysis extends the ideas from [20, 34]. The state-of-art technique [34] is an improved version of [29]. In [29, 34], the task activation model is a superposition of typical activation and sporadic overload. The typical activation is assumed to be feasible whereas the overload activations can cause at most m deadline misses out of k consecutive activation of a task. However, neither [29] nor [34] distinguishes the execution time of different activation classes (i.e. regardless of whether typical or overload activation). In [20], the authors extend the task model with sporadic long execution time overload and the TWCA algorithm is extended based on the result in [29]. In this study, we borrow the task model in [20] and bound the deadline misses by counting the number of possible overload activations with the consideration of fault-tolerance techniques.

Worst-case Response Time of Typical Activation: Due to error detection techniques considered in this work, our approach to calculate the worst-case response time is different compared to the common practice as in [20, 29, 34]. More specifically, we explicitly consider the potentially varying execution time of recovery jobs. Let $B_i^+(q)$ denote the maximum time needed to process q typical activations of task τ_i within any busy window, where the transient soft error may occur[3]:

$$B_i^+(q) = \gamma_i^{+,(tp)}(q) + \sum_{\tau_j \in hp(\tau_i)} \gamma_j^{+,(tp)}(\eta_j^{+,(tp)}(B_i^+(q))) \tag{9}$$

$$+ max\{\gamma_j^{+,(o)}(\eta_j^{+,(o)}(B_i^+(q)))|\forall \tau_j \in hp(\tau_i)\}. \tag{10}$$

Here, the first term is the service demand of the q typical activations; the second term is the interference from higher priority tasks; and the last term is the maximum possible overload service demand due to transient soft error.

Definition 4.1. Worst-case level-i busy window [34]: A worst-case level-i busy window, denoted as BW_i, is the maximal time window during which tasks of equal or higher priority than task τ_i have pending jobs.

BW_i can be calculated as following:

$$BW_i = B_i^+(K_i), \tag{11}$$

where

$$K_i = \min\{q \geq 1 : B_i^+(q) \leq \eta_i^{-,(tp)}(q+1)\}.$$

[3]We assume that the time interval between two consecutive soft errors are large enough such that soft error will not happen during the execution of recovery jobs.

The worst-case response time can be calculated as:

$$r_i = \max_{1 \le q \le K_i} \{B_i^+(q) - \delta_i^{-,(tp)}(q)\},$$

where $\delta_i^{-,(tp)}$ is the event distance function of typical activations.

TWCA assumes that task τ_i is schedulable in the typical model (i.e., no transient soft error). However, in the worst case, out of the K_i activations in the worst-case busy window, some may miss their deadlines. Let us denote these deadline misses by N_i and thus $N_i = \{q \in \mathbb{N} | 1 \le q \le K_i \wedge B_i^+(q) - \delta_i^{-,(tp)}(q) > d_i\}$. These deadline misses are caused by the overload activations due to transient soft errors. Thus, in order to bound the maximum number of deadline misses, we just need to find how many recovery jobs may affect the k consecutive typical activations. The maximum time window $\Delta T_k^{j,i}$ during which the overload activation of τ_j may impact the k activation of task τ_i can be calculated by [20]:

$$\Delta T_k^{j,i} = BW_i + \delta_i^{+,(tp)}(k) + r_i.$$

Finally, the number of deadline misses is then bounded by:

$$dmm_i(k) = |N_i| \times \left\lceil \frac{\Delta T_k^{j,i}}{\Delta t_{error}} \right\rceil. \tag{12}$$

4.1.2 Event-based Simulation for Exact Deadline Miss Pattern. The aforementioned schedulability analysis only guarantees a pessimistic upper-bound to the number of deadline misses within k consecutive activations. However, the exact deadline hit/miss patterns sometimes have a non-negligible effect on the control cost. Moreover, due to the randomness of the soft errors, the activation patterns of recovery jobs are not clear. Thus, we build an event-based simulation with error injection. The pseudo-code is shown in Algorithm 1.

Our event-based simulation records the time-stamp of each event such as the job release time, finish time, etc. we denote the j-th invocation of task τ_i as job $\theta_{ij} = (s_{\theta_{ij}}, c_{\theta_{ij}})$, where $s_{\theta_{ij}} = j \cdot t_{\tau_i}$ is the release time of the job. $c_{\theta_{ij}}$ keeps track of the remaining computation time of the job. For each task τ_i, we record its deadline miss patterns in an array $Miss[i]$, where $Miss[i][j] = true$ if τ_i's j-th job θ_{ij} misses its deadline. *event_queue* and *job_queue* are two job priority queues to store the unreleased jobs and pending jobs, respectively. *event_queue* is sorted by the job release time $s_{\theta_{ij}}$ while *job_queue* is sorted by the task priority.

We assume that the soft error can arrive any time during the hyper-period. The algorithm first pushes all typical activations into the *event_queue*. Function $InjectError()$ tries to inject a soft error for each event and the corresponding re-execution job is pushed into *event_queue*. Then we conduct an efficient even-based simulation of the whole hyper-period (lines 10-28). If the event belongs to a task without error detection technique, we just skip to the next event. For a time point *cur_time*, any jobs that can be released are popped from the *event_queue* and then pushed into the *job_queue*, and the highest priority job in the *job_queue* is scheduled to run. Here, θ_{kl} is the scheduled job at *cur_time* and θ_{ij} is the next job to release. Afterwards, the simulation moves to the next time point.

If the scheduled job has not finished at time *next*, it will update its remaining execution time $c_{\theta_{ij}}$ and be pushed back to the *job_queue*. Every time a job θ_{kl} finishes, the simulation records whether it misses its deadline in $Miss[k][l]$. After the simulation completes, the function $VerifyWHConstraint()$ counts the maximum deadline misses of any consecutive N_i^j activations (i.e., $dmm_i(N_i^j)$) to verify whether tasks have met corresponding weakly-hard constraints. Once we have finished the simulation of the whole hyper-period, we clear the error and move to next possible event. The return value indicates whether current system configuration is schedulable under the worst-case soft error scenario.

Algorithm 1: *EventSim*: Event-based Simulation with Error Injection

```
1:  WCRTAnalysis(T)
2:  if ∀τ_i ∈ T, r_τi ≤ d_τi then
3:    return true
4:  for task τ_i ∈ T do
5:    for j ∈ {0, ..., HyperPeriod/t_i − 1} do
6:      event_queue.push(θ_ij)
7:  for e ∈ event_queue do
8:    if InjectError(e) then
9:      θ_ij = event_queue.pop(),   cur_time = s_θij
10:     while cur_time ≤ HyperPeriod do
11:       while s_θij ≤ cur_time do
12:         θ_ij = event_queue.pop(), job_queue.push(θ_ij)
13:       if job_queue is empty then
14:         cur_time = s_θij
15:       else
16:         θ_kl = job_queue.pop(),   next = s_θij
17:         response = cur_time + c_θkl
18:         if response ≤ next then
19:           if response ≤ s_θkl + d_τk then
20:             Miss[k][l] = false
21:             cur_time = response
22:           else
23:             Miss[k][l] = true
24:             cur_time = max(s_θkl + d_τk, cur_time)
25:         else
26:           c_θkl = c_θkl − (next − cur_time)
27:           job_queue.push(θ_kl), cur_time = next
28:     ClearError(e)
29:     schedulability = VerifyWHConstraint(Miss)
30:     if !schedulability then
31:       return false
32: return true
```

4.2 Optimization Algorithm

In this section, we develop a meta-heuristic that tries to optimize control cost while meeting various constraints. We first use a simple heuristic to decide the initial choice of error detection technique for each task. Then an initial solution for the whole system is generated by using a bin-packing scheme [7]. Finally, we use simulated annealing (SA) to further explore the design space.

4.2.1 Initial Solution. The initial system configuration is generated by two steps. First, we sort the tasks based on their utilizations in an ascending order. Then, starting from the task with the lowest utilization, we assign EED to each task until the error coverage requirement is met. After the decision for error detection technique

is made, we map the taskset onto the underlying hardware platform using a bin-packing algorithm. The priority of each task is then assigned using the deadline monotonic priority assignment. The pseudo-code for obtaining the initial solution is in Algorithm 2.

Algorithm 2: Obtaining Initial Solution

Input: taskset $\mathcal{T}$, error coverage requirement $EC\_Threshold$

$\mathcal{T}_{sort} = utilSort()$
$p = getEC()$
while $p < EC\_Threshold$ **do**
  for $\tau_i \in \mathcal{T}_{sort}$ **do**
    $\tau_i.o_{\tau_i} = max(1, tau_i.o_{\tau_i} + 1)$
    $p = getEC()$
return $binPacking(\mathcal{T}_{sort})$

4.2.2 Overall Meta-heuristic with SA. Algorithm 3 shows our meta-heuristic optimization. First, the function $ObtainInitialSolution()$ generates the initial solution as in Algorithm 2. Then, the function $SchedulabilityAnalysis()$ checks system schedulability by using either **a)** the TWCA analysis to give an upper-bound to the deadline miss number of each task and check whether it meets the weakly-hard constraints or **b)** the event-based simulation to obtain the exact deadline hit/miss pattern. Then $CalculateCost()$ returns the control cost: **a)** if TWCA is used, $CalculateCost()$ gets current control cost by approximation as discussed in Section 3.2, or **b)** if event-based simulation is used, $CalculateCost()$ calculates the exact control cost using the obtained deadline hit/miss pattern. Function $AddPenalty()$ adds a penalty to the cost value if current solution is unschedulable or the error coverage is below the error coverage requirement $EC\_Threshold$. During each step of the simulated annealing, S_{cur} will randomly move to another configuration S_{new} by either swapping the priority of two tasks or changing the error detection technique or the allocation of a task. If the new configuration cannot be guaranteed to be schedulable under the schedulability analysis, a penalty will be added to the cost value. The new configuration will be accepted if it has a better objective value; otherwise, the acceptance probability will be calculated based on the current temperature and the objective difference.

Algorithm 3: Meta-heuristic Optimization with SA

$S_0 = ObtainInitialSolution()$
$S_{best} = S_{cur} = S_{new} = S_0$
$is\_sched = SchedulabilityAnalysis(S_0)$
$current\_cost = CalculateCost()$
$current\_cost = AddPenalty()$
$\eta_{best} = \eta_{cur} = \eta_{new} = current\_cost$
while $T > T^*$ **do**
  $k = 1$
  while $k \leq iter\_max$ **do**
    $S_{new} = RandomMove(S_{cur})$
    $\eta_{new} = CalculateCost() + AddPenalty()$
    if $\eta_{new} < \eta_{cur}$ **then**
      $S_{cur} = S_{new}, \eta_{cur} = \eta_{new}$
      if $S_{new}.is\_sched == \textbf{true} \wedge S_{new}.EC \geq EC\_Threshold \wedge \eta_{cur} < \eta_{best}$ **then**
        $S_{best} = min(S_{cur}, S_{best})$
        $\eta_{best} = min(\eta_{cur}, \eta_{best})$
    else if $AccepProb(\eta_{new} - \eta_{cur}, T) > rand()$ **then**
      $S_{cur} = S_{new}, \eta_{cur} = \eta_{new}$
    $k = k + 1$
  $T = T * cooling\_factor$
return S_{best}, η_{best}

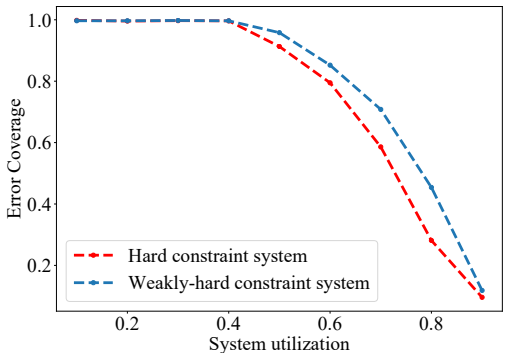

Figure 3: Comparison of average error coverage for weakly-hard-constraint systems and hard-constraint systems.

5 EXPERIMENTAL RESULTS

We evaluate our proposed approach with an industrial case study and a set of synthetic examples. Our controller tasks are derived based on 4 example LTI systems [10, 24]. The weakly-hard constraints for them are chosen such that the control stability of each task is guaranteed based on the analysis in Section 3. Each non-control task is randomly assigned with a (k, n) constraint where k ranges from 0 to 4 and n ranges from 10 to 20. All experiments are conducted on a server with Intel Xeon Gold 6130 CPU at 2.1 GHz.

5.1 Synthetic Examples

We conduct experiments with a set of 50 synthetic examples. Each synthetic example consists of 4 non-control tasks and 4 control tasks, all mapped onto a single-core CPU.

Hard Constraints vs. Weakly-hard Constraints: To see how much improvement can be obtained from leveraging weakly-hard constraints, we use the event-based SA to explore the maximum error coverage. The average maximum error coverage over the 50 synthetic examples is shown in Figure 3. Both hard-constraint and weakly-hard-constraint systems can achieve 100% error coverage when the system utilization is below 0.4. As the utilization increases, our approach can make use of the scheduling slack obtained from the weakly-hard constraints, i.e., allowing certain tasks to miss their deadlines can enhance the systems fault-tolerance capability. When utilization is 0.9, both weakly-hard and hard-constraint systems are not able to achieve meaningful error coverage.

Impact of System Utilization: Then, we study how the system control cost can be affected by the system utilization and error coverage requirement. Figure 4 shows the control cost of different system utilization while the actual error coverage increases from 0.1 to 0.7. As expected, when system utilization is 0.9, we can hardly improve the maximum error coverage and the control

cost increases dramatically with small increase of error coverage requirement. For system utilization of 0.7 and 0.8, we are able to find a solution for most of the cases. The maximum error coverage that can be achieved by 0.8 system utilization is around 0.5, while the maximum error coverage of 0.7 system utilization is around 0.7. This information can facilitate the choice among various design options under different system utilizations.

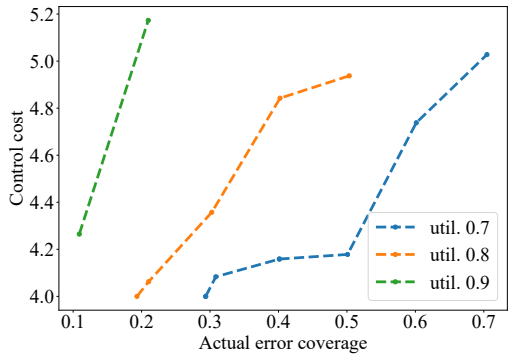

Figure 4: Control cost of different system utilization when error coverage requirement changes from 0.1 to 0.7

Comparison of Heuristic Algorithms: We also compare the effectiveness of different heuristic algorithms, i.e., the initial solution (bin-packing), the event simulation based simulated annealing, and the TWCA based simulated annealing. We run the three algorithms on a set of synthetic examples with 0.7 system utilization and error coverage requirement increasing from 0.4 to 0.7. Note that the x-axis in Figure 5 is the actual error coverage. As we can see, among the three heuristic algorithms, event simulation based SA produces the best solution as it offers the lowest control cost under different error coverage requirements.

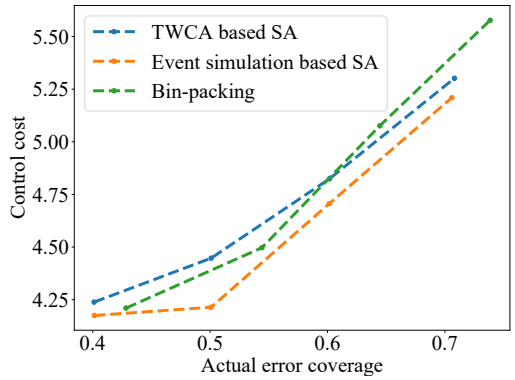

Figure 5: Control cost obtained by different heuristics.

5.2 Industrial Case Study: WATERS Challenge

Our industrial case is derived from the WATERS 2019 Challenge [11], which consists of 9 tasks and covers a prototype of an advanced driver-assistance system (ADAS). The underlying reference platform is NVIDIA Jetson TX-2 consisting of 6 heterogeneous cores and an integrated GPU. [5] provides a detailed discussion of task modeling and response time analysis, and shows that the original taskset as presented in WATERS 2019 Challenge is unschedulable.

For the purpose of our study, we assume a homogeneous platform and that all tasks are running on ARMv8 A57 cores. To make to the taskset schedulable, we scale the WCET of each task by a scaling factor. We also add four additional control tasks. Table 2 shows the maximum error coverage when the scaling factor changes from 0.3 to 0.7, and the number of CPUs changes from 3 to 5. We can see that lower utilization and more number of CPUs lead to better error coverage. The error coverage saturates when the scaling factor is 0.3 and 5 CPUs are used.

Table 2: Error Coverage under different scaling factor and number of CPUs.

CPU number \ Scaling factor	0.3	0.4	0.5	0.6	0.7
3	0.68	0.3	n.a.	n.a.	n.a.
4	0.85	0.67	0.46	0.15	n.a.
5	1.0	0.86	0.76	0.56	0.14

Figure 6 shows the trade-off between error coverage and control cost when the scaling factor is 0.5 and the number of CPUs is 4. During the experiments, we increase the error coverage requirement changes from 0.1 to 0.45. The x-axis in the figure is the actual error coverage after SA. Note that the minimal error coverage is 0.27 since there are some idle time and OS overhead is not counted towards the error coverage. As the error coverage requirement increases from 0.3 to 0.5, the control cost rises accordingly. This study shows that our approach can enable quantitative tradeoff analysis between error coverage and control cost for designers.

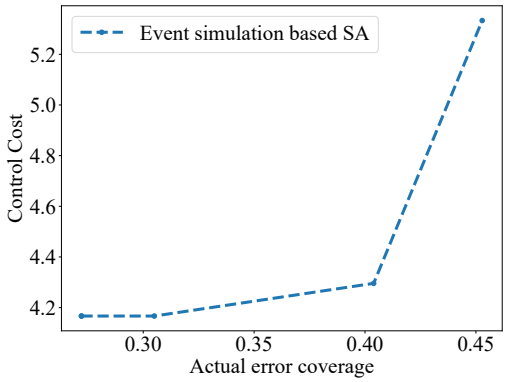

Figure 6: Tradeoff analysis enabled by our approach between error coverage and control cost for the WATERS example.

6 CONCLUSION

In this work, we present a novel approach for improving system fault tolerance by leveraging weakly-hard constraints. Our approach includes novel control analysis and scheduling analysis methods under deadline misses, and a meta-heuristic for exploring the design space. Experimental results demonstrate its effectiveness in improving fault tolerance and enabling system-level tradeoffs between control cost and error coverage.

ACKNOWLEDGMENTS

We gratefully acknowledge the support from NSF grants 1834701, 1834324, 1839511, 1724341, and ONR grant N00014-19-1-2496.

REFERENCES

[1] Thomas Anderson and John C. Knight. 1983. A Framework for Software Fault Tolerance in Real-Time Systems. *IEEE Transactions on Software Engineering* SE-9, 3 (1983), 355–364.

[2] Rodrigo Possamai Bastos and Frank Sill Torres. 2020. *Effectiveness of Hardware-Level Techniques in Detecting Transient Faults.* Springer International Publishing, Cham, 17–27. https://doi.org/10.1007/978-3-030-29353-6_2

[3] Robert C. Baumann. 2005. Radiation-induced Soft Errors in Advanced Semiconductor technologies. *IEEE Transactions on Device and Materials Reliability* 5, 3 (2005), 305–316.

[4] Guillem Bernat, Alan Burns, and Albert Liamosi. 2001. Weakly Hard Real-time Systems. *IEEE Trans. Comput.* 50, 4 (2001), 308–321.

[5] Daniel Casini, Paolo Pazzaglia, Alessandro Biondi, Giorgio Buttazzo, and Marco Di Natale. 2019. Addressing Analysis and Partitioning Issues for the Waters 2019 Challenge. In *10th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2019).*

[6] Hyunjong Choi, Hyoseung Kim, and Qi Zhu. 2019. Job-Class-Level Fixed Priority Scheduling of Weakly-Hard Real-Time Systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 241–253.

[7] Edward G. Coffman, Michael R. Garey, and David S. Johnson. 1984. *Approximation Algorithms for Bin-Packing — An Updated Survey.* Springer Vienna, Vienna, 49–106. https://doi.org/10.1007/978-3-7091-4338-4_3

[8] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. 2014. Formal Analysis of Timing Effects on Closed-Loop Properties of Control Software. In *RTSS.* https://doi.org/10.1109/RTSS.2014.28

[9] Yue Gao, Sandeep K. Gupta, and Melvin A. Breuer. 2013. Using Explicit Output Comparisons for Fault Tolerant Scheduling (FTS) on Modern High-performance Processors. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE).* 927–932.

[10] Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. 2014. Relaxing Signal Delay Constraints in Distributed Embedded Controllers. *IEEE Transactions on Control Systems Technology* 22, 6 (2014), 2337–2345.

[11] Arne Hamann, Dakshina Dasari, and Falk Wurst. [n.d.]. WATERS Industrial Challenge 2019. 2019. In *URL: https://www. ecrts. org/forum/viewtopic. php*, Vol. 124.

[12] Moncef Hamdaoui and Parameswaran Ramanathan. 1995. A Dynamic Priority Assignment Technique for Streams with (m, k)-firm Deadlines. *IEEE Trans. Comput.* 44, 12 (Dec 1995), 1443–1451. https://doi.org/10.1109/12.477249

[13] Chao Huang, Kai-Chieh Chang, Chung-Wei Lin, and Qi Zhu. 2020. SAW: A Tool for Safety Analysis of Weakly-Hard Systems. In *Computer Aided Verification*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer International Publishing, Cham, 543–555.

[14] Chao Huang, Wenchao Li, and Qi Zhu. 2019. Formal Verification of Weakly-hard Systems. In *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control* (Montreal, Quebec, Canada) *(HSCC '19).* ACM, New York, NY, USA, 197–207. https://doi.org/10.1145/3302504.3311811

[15] Chao Huang, Kacper Wardega, Wenchao Li, and Qi Zhu. 2019. Exploring Weakly-Hard Paradigm for Networked Systems. In *Proceedings of the Workshop on Design Automation for CPS and IoT* (Montreal, Quebec, Canada) *(DESTION '19).* Association for Computing Machinery, New York, NY, USA, 51–59. https://doi.org/10.1145/3313151.3313165

[16] Viacheslav Izosimov, Ilia Polian, Paul Pop, Petru Eles, and Zebo Peng. 2009. Analysis and Optimization of Fault-tolerant Embedded Systems with Hardened Processors. In *2009 Design, Automation Test in Europe Conference Exhibition.* 682–687.

[17] Karl J.Åström and Björn Wittenmark. 1997. *Computer-controlled Systems: Theory and Design.* Prentice Hall.

[18] Jaroslav Kautsky, Nancy K. Nichols, and Paul Van Dooren. 1985. Robust Pole Assignment in Linear State Feedback. *Internat. J. Control* 41, 5 (1985), 1129 - 1155.

[19] Arvind Kumar, Rama Shankar Yadav, and Anjali Jain Ranvijay. 2011. Fault Tolerance in Real Time Distributed System. *International Journal on Computer Science and Engineering* 3, 2 (2011), 933–939.

[20] Leonie Köhler and Rolf Ernst. 2019. Improving a Compositional Timing Analysis Framework for Weakly-Hard Real-Time Systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 228–240.

[21] Jian Li, YeQiong Song, and Françoise Simonot-Lion. 2006. Providing Real-Time Applications With Graceful Degradation of QoS and Fault Tolerance According to(m, k)-Firm Model. *IEEE Transactions on Industrial Informatics* 2, 2 (2006), 112–119.

[22] Hengyi Liang, Zhilu Wang, Debayan Roy, Soumyajit Dey, Samarjit Chakraborty, and Qi Zhu. 2019. Security-Driven Codesign with Weakly-Hard Constraints for Real-Time Embedded Systems. In *2019 IEEE 37th International Conference on Computer Design (ICCD).* 217–226. https://doi.org/10.1109/ICCD46524.2019.00035

[23] Robert E. Lyons and Wouter Vanderkulk. 1962. The Use of Triple-Modular Redundancy to Improve Computer Reliability. *IBM Journal of Research and Development* 6, 2 (1962), 200–209.

[24] William C Messner, Dawn M Tilbury, and Asst Prof Rick Hill. 1999. Control Tutorials for MATLAB® and Simulink®.

[25] Ghassem Miremadi, Johan Harlsson, Ulf Gunneflo, and Jan Torin. 1992. Two Software Techniques for On-line Error Detection. In *[1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing.* 328–335.

[26] Nahmsuk Oh, Philip P. Shirvani, and Edward J. McCluskey. 2002. Control-flow Checking by Software Signatures. *IEEE Transactions on Reliability* 51, 1 (2002), 111–122.

[27] Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. 2018. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *ECRTS.*

[28] Paul Pop, Viacheslav Izosimov, Petru Eles, and Zebo Peng. 2009. Design Optimization of Time- and Cost-Constrained Fault-Tolerant Embedded Systems With Checkpointing and Replication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17, 3 (2009), 389–402.

[29] Sophie Quinton, Matthias Hanke, and Rolf Ernst. 2012. Formal Analysis of Sporadic Overload in Real-time Systems. In *DATE.*

[30] Amir Rajabzadeh and Seyed Ghassem Miremadi. 2005. A Hardware Approach to Concurrent Error Detection Capability Enhancement in COTS Processors. In *11th Pacific Rim International Symposium on Dependable Computing (PRDC'05).* 8 pp.–.

[31] Semeen Rehman, Florian Kriebel, Bharath Srinivas Prabakaran, Faiq Khalid, and Muhammad Shafique. 2018. Hardware and Software Techniques for Heterogeneous Fault-Tolerance. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS).* 115–118.

[32] Youcheng Sun and Marco Di Natale. 2017. Weakly Hard Schedulability Analysis for Fixed Priority Scheduling of Periodic Real-Time Tasks. *ACM TECS* 16, 5s (2017), 19.

[33] Zhilu Wang, Hengyi Liang, Chao Huang, and Qi Zhu. 2020. Cross-Layer Design of Automotive Systems. arXiv:2005.11842 [eess.SY]

[34] Wenbo Xu, Zain AH Hammadeh, Alexander Kroller, and Rolf Ernst. 2015. Improved Deadline Miss Models for Real-time Systems using Typical Worst-case Analysis. In *ECRTS.*

[35] Bowen Zheng, Yue Gao, Qi Zhu, and Sandeep Gupta. 2015. Analysis and Optimization of Soft Error Tolerance Strategies for Real-time Systems. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).* 55–64. https://doi.org/10.1109/CODESISSS.2015.7331368