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
Northwestern University
Evanston, Illinois

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 their 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 a set of synthetic examples demonstrate the effectiveness of our approach.

CCS CONCEPTS
•Computer systems organization → Embedded and cyber-physical systems;

ACM Reference format:
Hengyi Liang, Zhilu Wang, Ruochen Jiao, Qi Zhu. 2016. Leveraging Weakly-hard Constraints for Improving System Fault Tolerance with Functional and Timing Guarantees. In Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17), 9 pages.
DOI: 10.1145/nmnnnn.nmnnnn

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, 17].

In the literature, various error detection and recovery mechanisms have been proposed [8, 13, 25, 32]. For instance, to address soft errors, there are both hardware based approaches [2, 27, 28] and software approaches [8, 22, 23]. 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 [8], 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 [22], control flow checking and instruction signature checking [23]. 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 [20]. 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 [8] presents an offline scheduling algorithm for EOC-type techniques. The work in [32] 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, 11].

The exploration of weakly-hard constraints is motivated by the fact that many system functions (e.g., control or sensing functions)
can tolerate certain degree of 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 [9, 12, 24]. In [9], the authors prove an analytical upper bound of deadline miss ratio to ensure the stability of a distributed embedded control platform. In [24], the authors study the impact of deadline miss pattern on control performance. In [12], the authors present a method to formally verify the safety of certain control systems under weakly-hard constraints. On the other hand, a number of approaches have been presented for schedulability analysis of real-time system with weakly-hard constraints [4, 18, 26, 29, 31]. In [4], the response time analysis for periodic task is discussed. In [29], the authors model the schedulability analysis as a mixed integer linear programming (MILP) problem and apply it to periodical tasks with unknown task activation offset. In [26], 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 [31] further improves the approach from [26]. 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 [19], a co-design approach is presented to improve system security while ensuring control safety.

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 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 for 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 model, 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. Section 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 \( E = \{e_1, \ldots, e_n\} \) be the set of CPUs. The functional layer is described by a set of independent tasks \( T = \{\tau_1, \ldots, \tau_m\} \). Each task \( \tau_i \) 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 \( \tau_i \in T \), we use a variable \( o_{i} \) 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 \( \tau_i \), 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 \( \tau_i \) can be defined as in [32]:

\[
C_i = c_i + p_i(o_i(c_i + \Lambda_i) + (1 - o_i)\Delta c_i),
\]

where \( p_i \) denotes whether any error detection (EOC or EED) is applied for task \( \tau_i \), \( \Lambda_i \) the time for output comparison and \( \Delta c_i \) the EED overhead. Moreover, \( o_1 = 1 \) if EOC is selected, otherwise \( o_1 = 0 \). 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 \( \tau_i \). We have

\[
CR_i = p_i(c_i + (1 - o_i)\Delta c_i).
\]

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 [16]. 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 [16]: The event models \( \eta_i^{-}(tp)(\Delta t) \) and \( \eta_i^{+}(tp)(\Delta t) \) (\( \eta_i^{-}(\omega)(\Delta t) \) and \( \eta_i^{+}(\omega)(\Delta t) \)), respectively) provide lower and upper bound on the number of typical (overload, respectively) activations of task \( \tau_i \) during any time interval \([t, t + \Delta t]\).

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

Definition 2.2. Execution time model [16]: The execution time model \( Y_i^{-}(tp)(n) \) and \( Y_i^{+}(tp)(n) \) (\( Y_i^{-}(\omega)(n) \) and \( Y_i^{+}(\omega)(n) \), respectively) provide lower and upper bound on the number of typical (overload, respectively) share of the service demand required by any \( n \) consecutive activation of task \( \tau_i \).

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

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 \( \zeta_i = (K_{i,1}^1, N_{i,1}^1), \ldots, (K_{i,n_i}^{N_i}, N_{i,n_i}^{N_i}) \) be the set of the weakly-hard constraints of task \( \tau_i \), where \( (K_{i,j}^{N_j}, N_{i,j}^{N_j}) \) means for any \( N_j^{N_i} \) consecutive activations of task \( \tau_i \), at most \( K_{i,j}^{N_j} \) deadline misses are allowed. (0, 1) is the special case for hard deadline tasks. The system is schedulable if

\[
dmm_j(N_j^i) \leq K_{i,j}^1, \forall j, 1 \leq j \leq n_i, \forall i,
\]

(2)

where \( dmm_j(N_j^i) \) is the maximum number of deadline misses of task \( \tau_i \) in any \( N_j^i \) consecutive activations. We further assume that tasks will continue running until they finish when their deadlines are missed.

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 [7, 24]. The corresponding discrete-time system dynamics with certain sampling period \( h \) is given by [15]:

\[
x[k+1] = A_dx[k] + B_{d,0}u[k] + B_{d,1}w[k-1].
\]

(3)

where \( B_{d,0} = \int_0^{h-D} e^{As} \cdot Bds \) and \( B_{d,1} = \int_{h-D}^h e^{As} \cdot Bds \). \( D \) 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:

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

(4)

where \( A_{aug} = \begin{bmatrix} A_d & B_{d,1} \\ 0 & 1 \end{bmatrix}, B_{aug} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C_{aug} = [C \ 0] \), with 0 and \( I \) denoting zero matrix and identity matrix of suitable dimensions, respectively. The control law \( u[k] = -Kz[k] \) is calculated by pole place technique [14].

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 \( t_4 \), 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 longer 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 \( t_4 \) satisfies (2, 10) weakly-hard constraint, the system can be proven schedulable with EOC applied to \( t_4 \).

Table 1: Task set of the illustrating example.

| Task name | \( t_1 \) | \( t_2 \) | \( t_3 \) | \( t_4 \) |
|-----------|----------|----------|----------|----------|
| 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\(^1\).

\(^1\)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.
We consider stabilization controller that can bring the system back where \( \varepsilon \) where control input. The above system dynamic can be simplified as \( \xi[k+1] = (A_\xi[k] - B_\xi[k]\xi_k)\xi[k] = \phi[k]\xi[k] \), where \( K_\xi = [K, 0] \).

Control Stability: Assuming we are given the deadline hit/miss pattern of a control task within a hyper-period, thus the delay \( \psi_k \) 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:

\[
\xi[k+N] = \phi[k+N-1] \ldots \phi[k+1] \xi[k] = \\
= \prod_{i=k+1}^{k+N-1} \phi[i] \prod_{j=N-1}^{k} \phi[j] \xi[k] = \Phi_k \xi[k]
\]

The above system under deadline misses is asymptotically stable if the eigenvalues of \( \Phi_k \) are within the unit circle for all \( k \) [30].

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 \( \xi[k] \). We consider the disturbance is rejected if the residual disturbance after \( r \) sampling period satisfies:

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

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

\[
x[k+r] = i_t \xi[k+r] = \Phi_{k+r,k} \xi[k] = \Phi_{k+r,k} \hat{x}[k],
\]

where \( \hat{x} = [I_{n \times n} \ 0_{n \times m}]^T \). Then, the control cost is defined as:

**Definition 3.1.** The control cost metric of a control task \( \tau_t \) is defined as \( J_t = \max(h_k \xi[k]) \), where \( h_k \) satisfies:

\[
||\Phi_{k+r,k} \hat{x}[k]|| \leq \frac{||x[k+r]||}{||x[k]||} \leq J_{rh}, \quad \forall r \geq h_k.
\]  

Approximation of Control Cost: Calculating the precise control cost through event-based simulation is time-consuming. Thus, we try to approximate the control cost by utilizing the deadline miss bound provided by 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, \( \psi[i] \) 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 [9], when \( n = 10 \) and \( k \) changes from 0 to 4. The red line is the approximated control cost.*

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:

\[
J = \sum_{\tau_i \in T_j} \omega_i \frac{J_i}{j_{des}},
\]

where \( \omega_i \) are the given weights, \( T_j \) is the set of control tasks, and \( j_{des} \) is the desired control cost of task \( \tau_i \) assuming no deadline miss occurs. We can formulate our problem as:

(P1) Given \( E, T, T_j \), optimize task assignment, selection of error detection mechanism \( O = \{o_{\tau_1}, \ldots, o_{\tau_{\mid T_j}}\} \) 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 present our optimization algorithm.

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 [31], 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 [19]. The event-based schedulability simulates the execution of tasks within a hyper-period and defines 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 [16, 31]. The state-of-art technique [31] is an improved version of [26]. In [26, 31], 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 [26] nor [31] distinguishes the execution time of different activation classes (i.e. regardless of whether typical or overload activation). In [16], the authors extend the task model with sporadic long execution time overload and the TWCA algorithm is extended based on the result in [26]. In this study, we borrow the task model in [16] 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 [16, 26, 31]. More specifically, we explicitly consider the potentially varying execution time of recovery jobs. Let \( B_i^+(q) \) denotes the maximum time needed to process \( q \) typical activations of task \( \tau_i \) within any busy window, where the transient soft error may occur:

\[
B_i^+(q) = \eta_i^{+(t\pi)}(q) + \sum_{\tau_j \in hp(\tau_i)} \eta_j^{+(t\pi)}(\eta_j^{+(o\pi)}(B_i^+(q))) + \max\{\eta_j^{+(o\pi)}(\eta_j^{+(o\pi)}(B_i^+(q)))|\forall \tau_j \in hp(\tau_i)\}.
\]

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 [31]: 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 \( \tau_i \) have pending jobs.

\[
BW_i = B_i^+(K_i).
\]

where

\[
K_i = \min(q \geq 1 : B_i^+(q) \leq \eta_i^{-(t\pi)}(q + 1)).
\]

The worst-case response time can be calculated as:

\[
r_i = \max_{1 \leq q \leq K_i} \{B_i^+(q) - \delta_i^{-(t\pi)}(q)\},
\]

where \( \delta_i^{-(t\pi)} \) is the event distance function of typical activations.

TWCA assumes that task \( \tau_i \) 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 \( N_i = \{q \in \mathbb{N} | 1 \leq q < K_i \land B_i^+(q) - \delta_i^{-(t\pi)}(q) > d_i\} \). These deadline misses are caused by the overload activations due to transient soft

*In this demo example, for \( k = 2 \), there is an outlier point when the deadline misses are evenly distributed.
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 $r_j$ may impact the k activation of task $r_i$ can be calculated by [16]:

$$\Delta T_{k}^{j,i} = BW_i + \delta_{j}^{(p)}(k) + r_i.$$ 

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

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

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 pattern sometimes has 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.

All tasks are organized into different revival queues, and the corresponding weakly-hard constraints $\nu$ indicates whether current system configuration is schedulable $\nu_k$. Once we have finished the simulation of the whole hyper-period, we can be released are popped from the job queue, and then pushed into the job_queue, and the highest priority job in the job_queue is scheduled to run. Here, $\theta_{ij}$ is the scheduled job at $cur_{time}$ and $\theta_{ij}$ is the next job to release. Then, 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 $c_{0_{ij}}$, and be pushed back to the job_queue. Every time a job $\theta_{kl}$ 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 $\forall r_{ij} \in T, r_{ij} \leq d_{r_{ij}}$ then
3: return true
4: for task $r_i \in T$ do
5: for $j \in \{0, \ldots, HyperPeriod - 1\}$ do
6: event_queue.push($\theta_{ij}$)
7: for $e \in event_queue$ do
8: if InjectError(e) then
9: $\theta_{ij} = event_queue.pop()$, cur_time = $s_{\theta_{ij}}$
10: while cur_time $\leq HyperPeriod$ do
11: while $s_{\theta_{ij}} \leq cur_{time}$ do
12: $\theta_{ij} = event_queue.pop()$
13: job_queue.push($\theta_{ij}$)
14: if job_queue is empty then
15: cur_time = $s_{\theta_{ij}}$
16: else
17: $\theta_{kl} = job_queue.pop(), next = s_{\theta_{kl}}$
18: response = cur_time + $c_{\theta_{kl}}$
19: if response $\leq next$ then
20: $Miss[k][l] = false$
21: cur_time = response
22: else
23: $Miss[k][l] = true$
24: cur_time = max($s_{\theta_{kl}} + d_{\theta_{kl}}, cur_{time})$
25: else
26: $c_{\theta_{kl}} = c_{\theta_{kl}} - (next - cur_{time})$
27: job_queue.push($\theta_{kl}$), 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 [6]. 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. In the first step, 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 shown in Algorithm 2.

4.2.2 Overall Meta-heuristic with SA. Algorithm 3 shows our meta-heuristic optimization. First, the function ObtainInitialSolution(
Algorithm 2: Obtaining Initial Solution

Input: taskset $T$, error coverage requirement $EC_{\text{Threshold}}$

1. $T_{\text{sort}} = \text{utilSort}()$
2. $p = \text{getEC}()$
3. while $p < EC_{\text{Threshold}}$ do
   4. for $t_i \in T_{\text{sort}}$ do
      5. $\tau_i, \alpha_i = \max(1, \tau_i, \alpha_i + 1)$
      6. $p = \text{getEC}()$
   7. return $\text{binPacking}(T_{\text{sort}})$

generates the initial solution as in Algorithm 2. Then, the function $\text{SchedulabilityAnalysis}(\cdot)$ 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 $\text{CalculateCost}(\cdot)$ returns the control cost: a) if TWCA is used, $\text{CalculateCost}(\cdot)$ gets current control cost by approximation as discussed in Section 3.2, or b) if event-based simulation is used, $\text{CalculateCost}(\cdot)$ calculates the exact control cost using the obtained deadline hit/miss pattern.

Algorithm 3: Meta-heuristic Optimization with SA

1. $S_0 = \text{ObtainInitialSolution}()$
2. $S_{\text{best}} = S_{\text{cur}} = S_{\text{new}} = S_0$
3. $is_{\text{sched}} = \text{SchedulabilityAnalysis}(S_0)$
4. $\text{current\_cost} = \text{CalculateCost}()$
5. $\text{current\_cost} = \text{AddPenalty}(\cdot)$
6. $\eta_{\text{best}} = \eta_{\text{cur}} = \eta_{\text{new}} = \text{current\_cost}$
7. while $T > T^*$ do
   8. $k = 1$
   9. while $k \leq \text{iter\_max}$ do
      10. $S_{\text{new}} = \text{RandomMove}(S_{\text{cur}})$
      11. $\eta_{\text{new}} = \text{CalculateCost}(\cdot) + \text{AddPenalty}(\cdot)$
      12. if $\eta_{\text{new}} < \eta_{\text{cur}}$ then
         13. $S_{\text{cur}} = S_{\text{new}}, \eta_{\text{cur}} = \eta_{\text{new}}$
         14. if $S_{\text{new}}.is_{\text{sched}} = \text{true}$ \&\& $S_{\text{new}}.EC \geq EC_{\text{Threshold}} \&\& \eta_{\text{cur}} < \eta_{\text{best}}$ then
            15. $S_{\text{best}} = \min(S_{\text{cur}}, S_{\text{best}})$
            16. $\eta_{\text{best}} = \min(\eta_{\text{cur}}, \eta_{\text{best}})$
         17. else if $\text{AcceptProb}(\eta_{\text{new}} - \eta_{\text{cur}}, T) > \text{rand}()$ then
            18. $S_{\text{cur}} = S_{\text{new}}, \eta_{\text{cur}} = \eta_{\text{new}}$
            19. $k = k + 1$
   20. $T = T \times \text{cooling\_factor}$
21. return $S_{\text{best}}, \eta_{\text{best}}$

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 [9, 21]. 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 can increase dramatically even though we just add the error coverage requirement by 0.1. 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 design choices under different system utilizations.

Comparison of Heuristics 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, with error coverage requirement increases from 0.4 to 0.7. Again, notice 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 can output the best solution as it provides the lowest control cost under different error coverage requirements. While bin-packing based approach meets the error coverage requirement, it cannot optimize the control cost.

![Figure 4: Control cost of different system utilization when error coverage requirement changes from 0.1 to 0.7](image)

Our industrial case is derived from the WATERS 2019 Challenge [10], 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 the taskset schedulable, we scale the WCET of each task by a scaling factor. For our study, 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.](image)

![Figure 5: Control cost obtained by different heuristics.](image)

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. Notice 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 between error coverage and control cost for the WATERS 2019 Challenge example enabled by our approach.](image)

**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.
