IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 16, 2021, accepted November 3, 2021, date of publication November 16, 2021,
date of current version November 29, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3128397

Fully Adaptive Stochastic Handling of Soft-Errors in Real-Time Systems

HYUNG-CHAN AN[1], (Member, IEEE), AND HOESEOK YANG[2], (Member, IEEE)
[1]Department of Computer Science, Yonsei University, Seoul 03722, South Korea
[2]Department of Electrical and Computer Engineering, Ajou University, Suwon 16499, South Korea

Corresponding author: Hyung-Chan An (hyung-chan.an@yonsei.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1C1C1008934 and No. NRF-2019R1F1A1064209). This research was supported by the Yonsei Signature Research Cluster Program of 2021 (2021-22-0001).

ABSTRACT In the design of real-time systems, it is becoming increasingly important to take soft-error tolerance into account. While hardening techniques such as error detection and error correction enable us to build systems that can better tolerate soft-errors, they are inevitably accompanied with execution time overhead. In order to mitigate the impact of the increased execution time, Chen *et al.* proposed to identify the (m, k)-*constraints* of real-time tasks, which demand that at least m jobs out of k consecutive task invocations must be fault-free, and to design hardening policies based on this constraint. In this paper, we propose a new method to design hardening policies that are *adaptive* and *stochastic*. At the heart of our method is a new linear program (LP) formulation that finds an adaptive stochastic policy optimizing the CPU utilization. At the design time, we first identify the task set information of a given system, verify the system schedulability, and solve LPs to find an optimal policy. This policy is represented as a look-up table that specifies the stochastic hardening decisions as a function of the past execution history of the system. At the run time, hardening decisions are made simply by looking up this table. The proposed method finds hardening policies that adaptively reacts to the execution history of the system, allowing improvement in the CPU utilization. The method also deviates from the previous approaches' viewpoint that reliability must be assessed in an all-or-nothing manner, by devising the notion of stochastic hardening policies. We evaluated the effectiveness of the proposed method using various task sets. In a set of 2,050 benchmarks, the system's CPU utilization was improved by 2.80%-7.16% on average under different configurations. The improvement was by as high as 18.45% in the best benchmark.

INDEX TERMS Optimization, real-time systems, processor scheduling, software reliability, fault tolerance.

I. INTRODUCTION

Real-Time systems often operate under non-functional requirements. Such typical requirements include limited power budget [1], [2], worst-case temperature constraints [3], [4], and fault-tolerance to soft-errors [5], [6]. Since most of the techniques addressing these requirements, if not all, tend to increase the execution times of tasks, they often result in increased overall resource utilization, sometimes even affecting the schedulability of the entire system. For this reason, a large volume of literature already proposes to co-optimize resource utilization/schedulability and non-functional design requirements, exploring the trade-off thereof.

The associate editor coordinating the review of this manuscript and approving it for publication was Laxmisha Rai.

Amongst these non-functional requirements, it is becoming increasingly important to take *soft-error* tolerance into account in the design of real-time systems [7]–[10]. Soft-errors, also known as *transient faults*, refer to errors that do not permanently affect the system, but result in a temporary unintended behavior in software. Detection and correction of soft-errors are usually achieved through replicating the task execution on redundant hardware resource and/or executing the task a multiple number of times on a single processing element [8]–[10]. The enhanced fault-tolerance by these *hardening* techniques comes at the cost of enlarged execution times, and as such, indiscriminate application of hardening techniques will likely result in wasteful use of resource.

In order to mitigate the inefficiency caused by the naive hardening approaches, Chen *et al.* [7] proposed policies

that make hardening decisions while guaranteeing a relieved fault-tolerance constraint called the (m, k)-*firm guarantee* by Ramanathan [11]. This constraint demands that at least m jobs among k consecutive task invocations are fault-free: i.e., faults in up to $k - m$ jobs are tolerated. Chen *et al.* first present the *static hardening policy*, which can be encoded by a (0, 1)-vector called an (m, k)-*pattern*. In this pattern, 1's indicate the job instances executed with error correction and 0's indicate the instances executed without error correction. A static hardening policy simply follows what is indicated by this pattern repeatedly. Naturally, the pattern must be of length k and contain m number of 1's. They then present the *dynamic hardening policy*, which "lazily" follows an (m, k)-pattern—when the policy sees a 0, it repeatedly executes jobs with error *detection* only, and moves on to the next position of the pattern only after a fault is detected. Chen *et al.* [7] show that this lazy policy also satisfies the (m, k)-constraint and is schedulable if the static counterpart is schedulable.

Chen *et al.*'s policies, especially the dynamic policy, successfully reduce the system's overall resource utilization compared to a fully (or indiscriminately) hardened system. However, it is noteworthy that their dynamic hardening policies are much firmly tied to the static counterparts and have almost identical behaviors as the static counterparts. In fact, the only deviation from the static counterpart allowed to the dynamic policy is that it can postpone its movement to the next position within the (m, k)-pattern. However, this level of similarity is not necessary. Once we establish the schedulability of the system using the static counterpart, we can explore many other hardening policies whose behaviors significantly deviate from that of the static one. This will allow a hardening policy to more flexibly adapt to the history of previous job instances, such as what the hardening decisions were and whether or not faults occurred.

Another aspect of Chen *et al.*'s policies [7] that leaves room for improvement is its "all-or-nothing" assessment of reliability. In their model, the system is deemed reliable if the imposed (m, k)-constraint is *always* satisfied by the chosen policy. In other words, a system is evaluated to be either completely reliable or just unreliable, whilst reliability is often quantified by probabilities in more realistic models [10]. It was this simplified reliability model that forced the optimization of resource usage in their work to be more conservative than necessary.

In this paper, we propose a new method to design hardening policies that are both adaptive and stochastic. Our method will yield a policy that monitors the history of $k - 1$ previous job instances and makes the hardening decision on the k-th instance in an "adaptive" manner. In order to fully optimize the resource usage while meeting a probabilistic reliability target, the hardening decision will be also stochastic.

We will automatize the design of an optimal adaptive stochastic policy by formulating it as a linear program (LP). Due to the combined complexity of adaptiveness and stochasticity, it is nearly impossible to manually design adaptive stochastic policies. We write the constraints of the LP so that a feasible solution to the LP can encode an adaptive stochastic policy that meets the given quantitative reliability target. In fact, a solution can be directly interpreted as a "look-up table" that specifies the hardening decision probability distribution as a function of the previous history. While we could optimize any linear objective function with this LP, in this paper, we choose to focus on minimizing the CPU utilization.

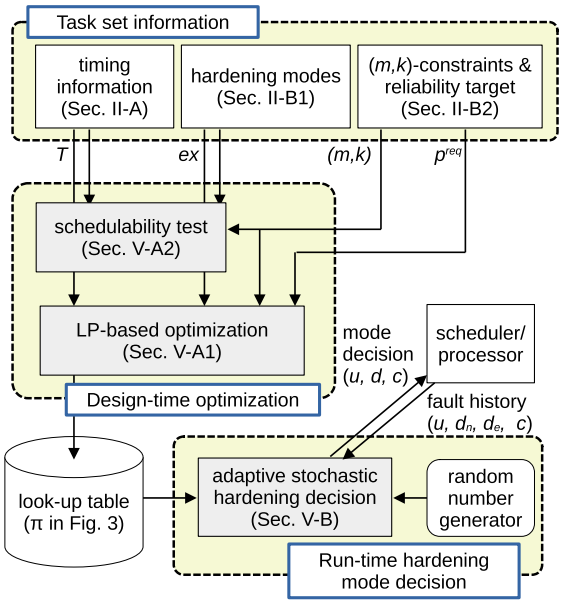


FIGURE 1. Overview of the proposed approach.

Fig. 1 provides an overview of our approach. At the *design time*, we begin with identifying the task set information such as the periods (T), execution times (ex), (m, k)-constraints, and reliability targets (p^{req}) of the tasks in a given system. Section II describes our system model in further details. We then verify the system schedulability under static policies and solve LPs to find an optimal adaptive stochastic policy. Section V-A1 presents this LP, and Section V-A2 shows how the schedulability of the policy found by this LP is guaranteed. At the *run time*, hardening decisions are made in an adaptive and stochastic manner by looking up a table. This look-up table is defined by the variables of the LP (Fig. 3). The run-time hardening mode decision takes as input the previous fault history and random numbers so that the decisions can be adaptive and stochastic. Section V-B presents the full details of this run-time algorithm. (See Algorithm 1.)

Before we present these technical details, Section III briefly sketches the proposed approach in comparison with Chen *et al.*'s [7], and Section IV presents some motivational examples to illustrate how adaptive and stochastic policies can better optimize resource utilization. After the proposed approach is presented in Section V, we experimentally validate our approach in Section VI. Finally, concluding remarks are given in Section VII.

TABLE 1. Comparison of the proposed method with previous works.

Previous work	target system	workload	soft-error hardening technique	hardening policy	error types	(m,k)-constraint	reliability target	design-time optimization algorithm
Zhou et al. [12]	cyber physical cloud system (multiprocessor with DVFS)	DAG with a single deadline	re-execution	static or dynamic	soft-error, hard-error, security	no	MTTF w.r.t. hard-errors	heuristics
Kim and Yang [13]	satellite on-board computer (uniprocessor)	periodic multi-task set	re-execution	static	soft-error, hard-error	yes	quantitative w.r.t. soft-errors	ad hoc
Chen et al. [7]	real-time control application (uniprocessor)	periodic multi-task set	independent	static or dynamic	soft-error	yes	all-or-nothing w.r.t. (m,k)-constraints	N/A
Proposed	real-time control application (uniprocessor)	periodic multi-task set	independent	adaptive and stochastic	soft-error	yes	quantitative w.r.t. (m,k)-constraints	linear programming

A. RELATED WORK

One of the particularly remarkable related work is due to Santini *et al.* [14]. Most of the existing studies on soft-error hardening, including this paper, assume that the underlying operating system is fault-free, and concentrate on application-level hardening decisions. Therefore, in order to adopt these studies in an actual system, we need to have the operating system hardened as well. Santini *et al.* [14] proposed to harden real-time operating systems via duplicated kernel data structures. This nicely complements the existing literature and this paper.

In Table 1, we focus on previous studies that aim at improving reliability through a systematic deployment of soft-error hardening techniques in real-time systems, and compare them with the proposed method while highlighting its difference from the others.

Zhou *et al.* [12] proposed a hardening optimization method for multiprocessor cyber-physical cloud systems that support dynamic voltage and frequency scaling (DVFS) [15]. In addition to soft-errors, they also consider hard-errors[1] and security. DVFS is known to create a trade-off between soft-errors and hard-errors [16], which Zhou *et al.* [12] optimizes using heuristics. The biggest difference between this result and the others in Table 1 lies in the workload model. Zhou *et al.* [12] considers workloads represented by single directed acyclic graphs (DAGs), whereas the other papers consider periodic multi-task sets. Due to this difference in the model, the challenges in their hardening optimization have completely different flavors from those in the other papers.

Kim and Yang [13] presented how Chen *et al.*'s static method [7] can be applied to a satellite on-board computer system with re-execution-based hardening technology. Similarly to Zhou *et al.* [12], they consider hard-errors, optimizing the mean time to failure (MTTF) [17] subject to a given soft-error rate constraint. This optimization uses an ad hoc method based on domain-specific knowledge. The method for example considers the ambient temperature in making its hardening decisions.

On the other hand, Chen *et al.* [7]'s model is very similar to the proposed method's. Aside from the differences already discussed in this section, one more distinction lies in the optimization methods. The proposed method uses linear programming as the underlying optimization engine.

All these results in Table 1, except for Zhou *et al.* [12], target uniprocessor systems. However, it is now becoming more plausible to run real-time workloads on multiprocessor systems. For example, Ali *et al.* [18] considered scheduling in cloud datacenters and Khan *et al.* [19] studied the migration-aware scheduling of mixed workloads of periodic and aperiodic tasks. It will be interesting and necessary to apply the proposed approach to multiprocessor systems, which we leave as a future direction. We will further discuss this topic in Section VII.

B. OUR CONTRIBUTION

We summarize the contribution of the proposed method as follows:

- *The proposed method yields hardening policies that are adaptive and stochastic.* This enables us to improve the CPU utilization compared to Chen *et al.*'s [7] methods.
- *We allow quantitative reliability targets to be specified.* This is in contrast to the reliability targets of the other works.[2]
- *The proposed method uses LP as the underlying optimization method.* The optimization procedure is automated with an LP that explores the huge design space of adaptive stochastic policies.

[1] Hard-errors refer to permanent failures of a system, as opposed to soft-errors.

[2] The MTTF target considered by Zhou *et al.* [12] is defined by hard-errors, i.e., permanent failures of the system. Kim and Yang [13] treats individual soft-errors, not (m,k)-violations, as reliability violations. Both Chen *et al.* [7] and the proposed method defines reliability violation as (m,k)-violations, although they differ in that Chen *et al.* is based on all-or-nothing assessment whereas the proposed method allows quantitative assessment.

II. SYSTEM MODEL

In this section, we review the model of Chen *et al.* [7] and define our extension to the model.

A. REAL-TIME TASK SETS

We consider a system that consists of n periodic *tasks*, denoted by $\tau_1, \ldots, \tau_n$. Each periodic task τ_i is associated with its invocation period T_i and considered to have the implicit deadline. That is, the relative deadline of task τ_i is equal to T_i. As in Chen *et al.* [7], all tasks are independent and scheduled under the preemptive rate-monotonic (RM) policy where static fixed priorities are assigned according to the period of the task, i.e., a shorter period results in a higher task priority.

B. SOFT-ERROR HANDLING

1) TASK HARDENING

Both Chen *et al.* [7] and the proposed method are not specific to any soft-error handling techniques. Without loss of generality in the choice of hardening techniques, a task is assumed to be executed in one of the following three modes: *unreliable* (u), *error-detecting* (d), and *error-correcting* (c). Let ex_i^u, ex_i^d, and ex_i^c such that $ex_i^u \leq ex_i^d \leq ex_i^c$ respectively denote the execution time of τ_i's job when executed in these three modes. Let f_i^u and f_i^d denote the probability that a fault occurs during an unreliable and error-detecting execution of the job, respectively.

Chen *et al.* [7] consider two possible ways of executing a job in the error-correcting mode: *reliable execution (RE)* and *detection and recovery (DR)*. DR is an opportunistic way. In DR, when a job is designated to be executed in the error-correcting mode, we run the job with error detection first, check whether a fault occurred, and re-run it with error correction only if a fault occurred. Since some hardening techniques require significantly more time correcting errors than detecting them, this could be effective compared to RE, a straightforward way. In RE, if a job is to be executed in the error-correcting mode, we run it with error correction right away. Since both Chen *et al.*'s approach and the proposed method are not specific to a particular soft-error handling technique, we can handle DR simply by re-defining ex_i^c accordingly. Therefore, in what follows, we will not describe the proposed method separately for DR and RE, in favor of simplicity of presentation.

2) (m, k)-CONSTRAINT AND RELIABILITY TARGET

For each task τ_i, an (m_i, k_i)-constraint and a reliability target p_i^{req} are specified. The (m_i, k_i)-constraint is that at least m_i of k_i consecutive jobs must be fault-free (either because they were executed in the error-correcting mode or we were just fortunate that no fault occurred). Consider the probability that this (m_i, k_i)-constraint is violated at any given job, and our goal is to keep this probability as low as p_i^{req}, the reliability target.

TABLE 2. Notation for system parameters.

Symbol	Description
n	Number of tasks
(m_i, k_i)	(m, k)-constraint of τ_i
f_i^u	Fault probability of τ_i (unreliable mode)
f_i^d	Fault probability of τ_i (error-detecting mode)
ex_i^u	Execution time of τ_i (unreliable mode)
ex_i^d	Execution time of τ_i (error-detecting mode)
ex_i^c	Execution time of τ_i (error-correcting mode)
p_i^{req}	Reliability target of τ_i
T_i	Invocation period of τ_i

We will incorporate both (m_i, k_i) and p_i^{req} into our LP formulation to ensure that the designed policy meets this reliability target.

Table 2 summarizes the notation for system parameters.

III. OVERVIEW OF THE PROPOSED APPROACH

In this section, we present an overview of our hardening policy in comparison with Chen *et al.* [7].

A. CHEN et al.'s HARDENING POLICIES

First, let us briefly review the hardening policies proposed by Chen *et al.* [7]. An (m, k)-*pattern* is defined as a k-dimensional (0, 1)-vector containing exactly m number of 1's. Given an (m, k)-pattern, Chen *et al.*'s static policy iterates through the pattern one position at a time, and executes each job in the error-correcting mode if the "current" position of the pattern contains a 1 and unreliable/error-detecting if 0. The policy returns to the beginning of the pattern when it reaches the end.

Their dynamic policy is very similar to the static one except that, when the pattern says 0, the dynamic policy executes the job in the error-detecting mode while advancing to the next position only if the error-detecting execution suffers a fault. If no fault occurs, it stays at the same position of the pattern. Chen *et al.* [7] show that this dynamic policy satisfies the (m, k)-constraint as well. Its schedulability is established via the corresponding static policy that interprets 0 as error-detecting. If this *static counterpart* is schedulable, the dynamic policy also is schedulable.

Although both policies of Chen *et al.* are not tied to a particular (m, k)-pattern, they considered two static patterns borrowed from Quan and Hu [20]: R- and E-patterns. In an R-pattern, error-correcting executions happen in a row at the end of the pattern, whereas in an E-pattern they are distributed as evenly as possible. For instance, when $(m, k) = (3, 10)$, the corresponding R- and E-pattern respectively are $(0, 0, 0, 0, 0, 0, 0, 1, 1, 1)$ and $(0, 0, 0, 1, 0, 0, 1, 0, 0, 1)$.

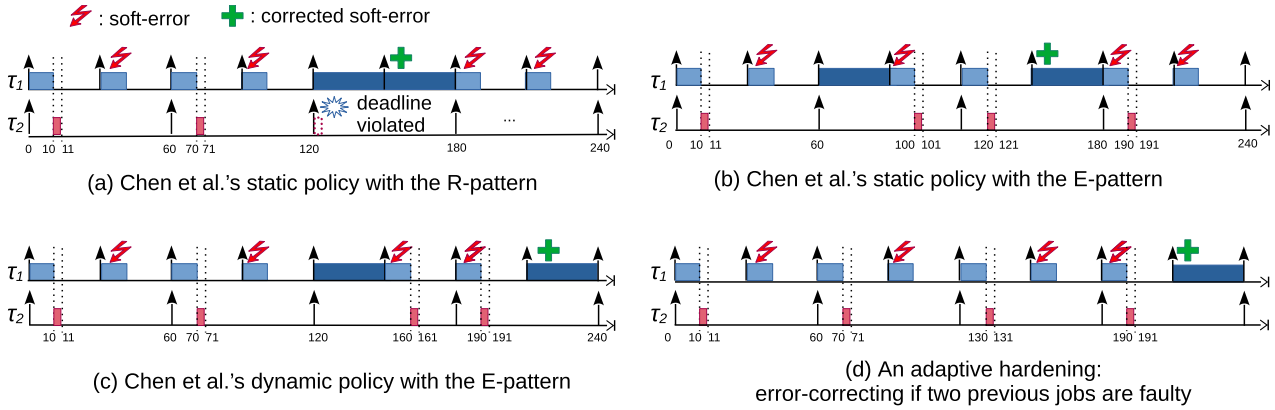


FIGURE 2. Hardening decisions of the first motivational example task set, consisting of two tasks τ_1 and τ_2. Real-time requirements are satisfied in all cases except for (a), whilst the CPU utilization differs. The error-correcting mode is triggered twice in (b) & (c), and only once in (d). See Section IV for further discussion.

B. PROPOSED APPROACH

Our approach aims at designing a fully adaptive, stochastic hardening policy. An adaptive stochastic policy can be considered simply as a look-up table: for each job of τ_i, the "trace" of the last $k_i - 1$ jobs determines (the probability distribution of) the hardening mode of the current job. The trace of a past individual job can be one of the following four values:

- "the job was executed in the unreliable mode",
- "the job was executed in the error-detecting mode and no fault occurred",
- "the job was executed in the error-detecting mode and a fault occurred", and
- "the job was executed in the error-correcting mode".

For each combination of $k_i - 1$ traces, the look-up table specifies a probability distribution over $\{u, d, c\}$. The run-time hardening policy is as simple as choosing one of these three execution modes according to the probability distribution set by the table. All the optimization effort is made during the design time, when solving an LP to determine this look-up table.

Since we aim at finding a stochastic policy, it is sensible to consider its *steady state*. Roughly speaking, a steady state is a state where the probability distribution of the policy's "behavior" does not change over time. For example, imagine a very simple stochastic policy which begins in the unreliable mode and switches back and forth between the unreliable and error-correcting modes with probability $1/10$ for each job. Let X_i be the probability that the i-th job is executed in the unreliable mode, and we have that $X_1 = 1, X_2 = 9/10, X_3 = (9/10)^2 + (1/10)^2 = 82/100$, and so on. However, rather than this transient behavior, it is often the "asymptotic" or steady-state efficiency of real-time systems that one is more interested in, since these systems tend to operate for a very long time. (In the previous example, we have $\lim_{i\to\infty} X_i = 1/2$.) As such, our LP will describe the steady-state behavior of a hardening policy.

TABLE 3. System parameters (adaptiveness example, see Table 2).

	k_i	m_i	f_i^u	f_i^d	ex_i^u	ex_i^d	ex_i^c	p_i^{req}	T_i
τ_1	6	2	0.3	0.3	10	10	30	0	30
τ_2	1	1	0.3	0.3	0.5	0.5	1	0	60

As in Chen *et al.* [7], the proposed approach can be used with an arbitrary (m, k)-pattern. In fact, we will use the same method as Chen *et al.* in ensuring the schedulability of the hardening policy, which is letting the static counterpart guarantee the schedulability of our policy. To this end, we will add constraints to the LP that ensures any feasible solution yields a hardening policy that is schedulable as long as the static counterpart is. One of the nice implications of this approach is that we can optimize one task at a time, without having to simultaneously optimize all tasks. This makes our approach scalable even for systems with a large number of tasks.

IV. MOTIVATIONAL EXAMPLES

In this section, we present motivational examples that highlight the effectiveness of adaptive and stochastic hardening policies.

A. AN ADAPTIVE POLICY DEVIATING FROM THE STATIC COUNTERPART

Consider a system with two tasks. Table 3 shows the parameters we chose for this simple system to illustrate the power of adaptiveness. Since $(m_2, k_2) = (1, 1)$, we have no other choice for τ_2 than always executing in the error-correcting mode. The question therefore narrows down to the hardening policy of τ_1. It is easy to observe that the system is not schedulable if the R-pattern is adopted. Fig. 2(a) shows that the third job of τ_2 misses its deadline when both tasks start at time offset $t = 0$. We thus adopt the E-pattern in our example. Using the results of Chen *et al.* [7, Lemma 1] based on [21],

TABLE 4. System parameters (stochasticity example).

	k_i	m_i	f_i^u	f_i^d	ex_i^u	ex_i^d	ex_i^c	p_i^{req}	T_i
τ_1	3	2	0.3	0.3	3	10	10	0.07	10

it is easy to prove that the system is indeed schedulable when the E-pattern is used.

Fig. 2(b) shows the execution of the two tasks when the static hardening policy of Chen *et al.* [7] is used. The total CPU utilization is $\frac{10+10+30+10+10+30}{6\cdot 30} + \frac{1}{60} \approx 57.2\%$. Fig. 2(c), on the other hand, shows the execution when their dynamic hardening policy is used. An easy calculation shows that the expected CPU utilization of this policy is 50.4%.[3] Both policies satisfy the (m_1, k_1)-constraint. For the scenario presented in Fig. 2 where soft-errors occur in the 2nd, 4th, 6th, 7th, and 8th jobs of τ_1, we can observe that both policies ensure that at least two out of any six consecutive jobs of τ_1 are fault-free. The dynamic policy is schedulable as long as its static counterpart is. (See [7].)

However, as we noted earlier, the dynamic policy is not the only policy whose schedulability can be automatically guaranteed from that of the static one. Following is a simple example of such an adaptive (non-stochastic) policy. For each job of τ_1, we consider just the last two jobs. If both suffered faults, we execute the current job in the error-correcting mode. Otherwise, we execute it in the error-detecting mode. Note that this policy satisfies the (m_1, k_1)-constraint. Moreover, we can show that its schedulability automatically follows from that of the static counterpart. This policy achieves the expected total CPU utilization of 39.3%.[4]

This example demonstrates that an adaptive hardening policy that is allowed to deviate from the static counterpart can bring significant improvement in efficiency. Fig. 2(d) depicts how this policy would work for the same scenario as in Fig. 2(c). In contrast to Chen *et al.*'s dynamic policy which chooses the error-correcting mode at the 5th and 8th jobs of τ_1, the adaptive policy triggers the error-correcting mode only once at the 8th job.

B. NECESSITY OF STOCHASTIC POLICIES

Now we present an example to show how hardening decisions can be made stochastically. We keep this example as simple as possible in the interest of presentation. In fact, our example consists of a single task whose parameters are shown in Table 4. The system is trivially schedulable.

[3]Note that the expected number of error-detecting jobs until a fault is $\frac{1}{0.3}$, including the last faulty job itself. We have that the expected total CPU utilization is $\frac{\frac{1}{0.3}\cdot 10+30+\frac{1}{0.3}\cdot 10+30}{(\frac{1}{0.3}+1+\frac{1}{0.3}+1)\cdot 30} + \frac{1}{60} \approx 0.504$.

[4]The expected number of error-detecting jobs until two consecutive faults is $\frac{1}{0.3}(\frac{1}{0.3}+1) = \frac{130}{9}$, including the last two faults. The expected total CPU utilization is therefore $\frac{\frac{130}{9}\cdot 10+30+\frac{130}{9}\cdot 10+30}{(\frac{130}{9}+1+\frac{130}{9}+1)\cdot 30} + \frac{1}{60} \approx 0.393$.

TABLE 5. An adaptive stochastic hardening policy.

Last two traces	prob. dist. for current mode		
	u	d	c
(u, u)	0	0	1
(u, c)	1	0	0
(c, u)	5/11	0	6/11

Imagine a policy that executes one job in every three in the error-correcting mode and the others in unreliable. While this policy has the total CPU utilization of $\frac{3+3+10}{10\cdot 3} = 53.3\%$, the probability that a job violates the (m_1, k_1)-constraint is $(f_1^u)^2 = 0.09 > p_1^{req}$. This policy is therefore unacceptable. On the other hand, if we execute two jobs in every three in the error-correcting mode and the remaining in unreliable, we "overshoot" the reliability target by making the probability zero, at the expense of a higher total CPU utilization of 76.7%.

Now consider the following (non-adaptive) stochastic hardening policy: for each job, execute it in the error-correcting mode with probability 1/2 and the unreliable mode with probability 1/2, regardless what happened in the past. The probability that a job violates the (m_1, k_1)-constraint is now $\binom{3}{3}(\frac{f_1^u}{2})^3 + \binom{3}{2}(\frac{f_1^u}{2})^2(1-\frac{f_1^u}{2}) = 0.06075 < p_1^{req}$, meeting the reliability target. The average total CPU utilization is 65%. This example shows how a system could benefit from a stochastic hardening policy.

C. A FINAL EXAMPLE

Finally, we will use the previous example (Table 4) to illustrate how an LP solution looks, before we rigorously present our LP in Section V-A. The main goal of this part is to provide an intuitive overview. Therefore, we will use a simplified form of solutions that slightly differs from the actual form of the LP solutions but better exposes the intuition.

Table 5 summarizes an adaptive stochastic hardening policy for the system. For example, if the second-to-last job was executed in the error-correcting mode and the last was in unreliable, in order to determine the execution mode of the "current" job, we look up the last row of the table. This row instructs us to choose the unreliable mode with probability 5/11 and the error-correcting mode with 6/11. Note that the table shows only three combinations of the last two traces, since the other traces will never be needed.

When the policy is at the steady state, the probability that the last two traces of an arbitrary job are (u, u) is calculated to be 5/27. (We omit the detailed calculation.) The probability that the last two traces are (u, c) is 11/27, and (c, u) is 11/27. Now, for example, in order for the "last two traces" of the *next* job to be (u, u), it must be the case that the last two traces of the current job was (c, u) and we decide to execute the current job in the unreliable mode. The probability that this happens is $(11/27) \cdot (5/11)$, which is indeed equal to 5/27,

$$\text{Minimize} \quad \sum_{t_{-(k-1)},\ldots,t_{-1}\in T, a_0\in A} ex^{a_0}\pi_{t_{-(k-1)},\ldots,t_{-1},a_0},$$

$$\text{subject to} \quad \pi_{t_{-(k-1)},\ldots,t_{-1},a_0} = 0, \qquad \text{for all forbidden } \pi_{t_{-(k-1)},\ldots,t_{-1},a_0}, \tag{a}$$

$$\begin{aligned}
&\sum_{a_0\in A} \pi_{t_{-(k-1)},\ldots,t_{-2},u,a_0} = \sum_{t_{-k}\in T} \pi_{t_{-k},\ldots,t_{-2},u}, && \forall t_{-(k-1)},\ldots,t_{-2}\in T,\\
&\sum_{a_0\in A} \pi_{t_{-(k-1)},\ldots,t_{-2},d_e,a_0} = f^d \sum_{t_{-k}\in T} \pi_{t_{-k},\ldots,t_{-2},d}, && \forall t_{-(k-1)},\ldots,t_{-2}\in T,\\
&\sum_{a_0\in A} \pi_{t_{-(k-1)},\ldots,t_{-2},d_n,a_0} = (1-f^d) \sum_{t_{-k}\in T} \pi_{t_{-k},\ldots,t_{-2},d}, && \forall t_{-(k-1)},\ldots,t_{-2}\in T, \qquad \text{(b)}\\
&\sum_{a_0\in A} \pi_{t_{-(k-1)},\ldots,t_{-2},c,a_0} = \sum_{t_{-k}\in T} \pi_{t_{-k},\ldots,t_{-2},c}, && \forall t_{-(k-1)},\ldots,t_{-2}\in T,
\end{aligned}$$

$$\sum_{t_{-(k-1)},\ldots,t_{-1}\in T, a_0\in A} F(t_{-(k-1)},\ldots,t_{-1},a_0)\pi_{t_{-(k-1)},\ldots,t_{-1},a_0} \le p^{req}, \tag{c}$$

$$\sum_{t_{-(k-1)},\ldots,t_{-1}\in T, a_0\in A} \pi_{t_{-(k-1)},\ldots,t_{-1},a_0} = 1,$$

$$\pi_{t_{-(k-1)},\ldots,t_{-1},a_0} \ge 0, \qquad \forall t_{-(k-1)},\ldots,t_{-1}\in T, a_0\in A.$$

FIGURE 3. The LP to design hardening policies (solved at the design time). See Table 2 for the system parameter notation.

consistent with the fact that this is a steady state. Repeating similar arguments verify that the given probability distribution is indeed a steady state.

A job is deemed to *violate* the (m, k)-constraint if, out of the last k jobs including the current one, strictly more than $k-m$ are faulty. The probability that any given job at a steady state violates the (m, k)-constraint can now be calculated as follows: $(5/27)\cdot(0.3)^2 + (11/27)\cdot(0.3)^2 + (11/27)\cdot(5/11)\cdot(0.3)^2 = 0.07$, showing that the reliability target is met.

V. PROPOSED METHOD

In this section, we propose a method to design an adaptive stochastic hardening policy. First, we formulate the search for an efficient hardening policy as an LP, which is to be solved at the design time. (Section V-A). The constraints of this LP ensure that the produced policy meets the reliability target and guarantees schedulability. At the run time, the execution mode decision is made stochastically as per the probability distribution table produced by the LP. Section V-B describes this run-time process.

A. DESIGN-TIME OPTIMIZATION

We present the LP in Fig. 3.

- Part (a) of the LP is the constraints that guarantee schedulability. In Section V-A2, we will describe in details which variables are "forbidden" and how this ensures schedulability.
- Part (b) is the steady-state constraints. Section V-A1 explains how a solution to the LP can be interpreted as a hardening policy in its steady state.
- Part (c) encodes the reliability target. We will explain this part at the end of Section V-A1.

Recall that we can optimize one task at a time. In what follows, we will focus on a single given task and omit the subscript i unless necessary.

1) LP VARIABLES AND STOCHASTIC CONSTRAINTS

As the hardening policy needs to act adaptively according to the (m, k)-constraint, it will consider the "trace" of the last $k-1$ jobs to determine its next "action". The trace of each individual job is represented by an element of the set $T := \{u, d_n, d_e, c\}$, which encodes the decision made for that job and its outcome. If the trace of a job is u, this means it was executed in the unreliable mode. If a job was executed in the error-detecting mode, this is represented by d_n (if no fault occurred) or d_e (if a fault occurred). If the trace of a job is c, this means that it was executed in the error-correcting mode. The decision made by the policy for the current job is denoted by u, d, and c, each representing the unreliable, error-detecting, and error-correcting modes. Let $A := \{u, d, c\}$ be the set of these three.

Consider a time point in a steady state. Let $\pi_{t_{-(k-1)},\ldots,t_{-1},a_0}$ be the marginal probability that the last $k-1$ traces are $t_{-(k-1)},\ldots,t_{-1}$ and the policy chooses a_0 as the decision for the current job. They become the variables of the LP. Since π is a joint probability distribution of $(t_{-(k-1)},\ldots,t_{-1},a_0)$, it must be a stochastic vector. This is ensured by the last two constraints of the LP.

In order to ensure that feasible solutions encode steady states, we write LP constraints that say the marginal probability distribution of the last traces of an arbitrary job must be equal to that of the previous job. Note that the marginal probability that the last $k-1$ traces are $t_{-(k-1)},\ldots,t_{-1}$ is $\sum_{a_0\in A}\pi_{t_{-(k-1)},\ldots,t_{-1},a_0}$. Therefore, we must have the

following constraints satisfied in order for π to be a steady state.

Case 1: If $t_{-1} = u$, the decision for the *previous* job must have been u. The $(k-1)$-st job before the *previous* job does not affect the last $k-1$ traces of the *current* job. Hence, we must have $\sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-2},u,a_0} = \sum_{t_{-k} \in T} \pi_{t_{-k},\ldots,t_{-2},u}$ for all $t_{-(k-1)}, \ldots, t_{-2} \in T$. In other words, the constraint states that the marginal probability that the last $k-1$ traces are $t_{-(k-1)}, \ldots, t_{-1}$ must be consistent, whether it is calculated using the probability distribution of the last traces of the current job (the left-hand side) or of the previous job (the right-hand side).

Case 2: If $t_{-1} = d_e$, the decision for the previous job must have been d. Since an error-detecting execution suffers a fault with probability f^d, we have $\sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-2},d_e,a_0} = f^d \sum_{t_{-k} \in T} \pi_{t_{-k},\ldots,t_{-2},d}$ for all $t_{-(k-1)}, \ldots, t_{-2} \in T$.

The other two constraints of part (b) follow from analogous arguments.

Finally, let us consider the reliability target. Recall that a job is deemed to violate the (m,k)-constraint if, out of the last k jobs including the current one, strictly more than $k-m$ are faulty. Let $F(t_{-(k-1)}, \ldots, t_{-1}, a_0)$ denote the conditional probability that a job violates the (m,k)-constraint, conditioned on the event that the last $k-1$ traces are $t_{-(k-1)}, \ldots, t_{-1}$ and the decision for the current job is a_0. To obtain a closed-form formula for F, let U be the number of u's from $t_{-(k-1)}, \ldots, t_{-1}$ and E be the number of d_e's. That is, we have $U := |\{j \mid t_{-j} = u, 1 \le j \le k-1\}|$ and $E := |\{j \mid t_{-j} = d_e, 1 \le j \le k-1\}|$. For each u in the trace, the job suffers a fault with probability f^u. If $a_0 \in \{u, d\}$, the current job suffers a fault with probability f^{a_0}. Suppose that a fault did occur. Then the current job violates the (m,k)-constraint if at least $k-m-E$ of the E number of u's suffer faults. If the current job is not faulty, the current job violates the constraint if at least $k-m-E+1$ are faulty. We thus have $F(t_{-(k-1)}, \ldots, t_{-1}, a_0) := f_{a_0} \sum_{i=\max(0,k-m-E)}^{U} \binom{U}{i} (f^u)^i (1-f^u)^{U-i} + (1-f_{a_0}) \sum_{i=\max(0,k-m-E+1)}^{U} \binom{U}{i} (f^u)^i (1-f^u)^{U-i}$. If $a_0 = c$, we have $F(t_{-(k-1)}, \ldots, t_{-1}, a_0) := \sum_{i=\max(0,k-m-E+1)}^{U} \binom{U}{i} (f^u)^i (1-f^u)^{U-i}$. With this definition of F, the unconditional probability that a job violates the (m,k)-constraint can be written as $\sum_{t_{-(k-1)},\ldots,t_{-1} \in T, a_0 \in A} F(t_{-(k-1)}, \ldots, t_{-1}, a_0)\, \pi_{t_{-(k-1)},\ldots,t_{-1},a_0}$.

This explains how part (c) of the LP encodes the reliability target. We remark that this is an *inequality* constraint. If the LP discovers that an optimal solution does not have to cause as frequent (m,k)-constraint violations as allowed by the reliability target, it can choose to produce a policy that violates the constraint less frequently than p^{req}.

2) SCHEDULABILITY CONSTRAINTS

Let $\Phi = (\phi_0, \ldots, \phi_{k-1}) \in \{0,1\}^k$ be an (m,k)-pattern. We will write the LP constraints which ensure that the policy output by the LP is schedulable as long as Chen *et al.*'s static counterpart is. These constraints are in a very simple form, which declares some variables as *forbidden* and requires them be zero.

Chen *et al.* [7] used the following lemma to verify system schedulability. We will base our analysis on the same lemma. Let $\Psi_i(\ell)$ be the maximum total execution time of ℓ consecutive jobs of τ_i. For simplicity, let $\Psi_i(0) := 0$.

Lemma 1 ([7], [21]): Suppose that the tasks of the system are indexed in the order of priority: $T_1 \le \cdots \le T_n$. The entire system is schedulable under the preemptive RM scheduling if, for all i,

$$\exists t \in (0, T_i] \quad \Psi_i(1) + \sum_{j=1}^{i-1} \Psi_j(\lceil \frac{t}{T_j} \rceil) \le t.$$

Intuitively speaking, our goal now is forbidding an appropriate set of variables to ensure that $\Psi_i(\ell)$ of our policy is no greater than that of the static counterpart, for all i and ℓ.

Let $\chi(\ell)$ be the maximum number of 1's in a consecutive length-ℓ subsequence of the infinitely repeated version of Φ, i.e., $\chi(\ell) := \max_{s \in \mathbb{N}} \sum_{j=0}^{\ell-1} \phi_{(s+j) \mod k}$. If we can show that the maximum number of error-correcting executions in ℓ consecutive jobs under our policy is no more than $\chi(\ell)$, we will have the desired conclusion since $0 < ex_i^u \le ex_i^d \le ex_i^c$. (See the proof of Lemma 2.) This motivates the following definition of ω, but the definition does not involve infinitely repeated versions anymore for a technical reason.

For $t_{-(k-1)}, \ldots, t_{-1} \in T$, $a_0 \in A$, and $\ell \in \{1, \ldots, k\}$, let $\omega_{t_{-(k-1)},\ldots,t_{-1},a_0}(\ell)$ be the maximum number of error-correcting executions in a consecutive length-ℓ subsequence of the sequence $(t_{-(k-1)}, \ldots, t_{-1}, a_0)$, not its infinitely repeated version. That is, $\omega_{t_{-(k-1)},\ldots,t_{-1},a_0}(\ell) := \max_{s:-(k-1) \le s \le 1-\ell} |\{j \mid t_{s+j} = c, 0 \le j \le \ell-1\}|$, where $t_0 := a_0$ for notational convenience. It is now simple to describe the forbidden variables. We declare $\pi_{t_{-(k-1)},\ldots,t_{-1},a_0}$ forbidden if and only if there exists some $\ell \in \{1, \ldots, k\}$ such that $\chi(\ell) < \omega(\ell)$.[5]

We now show that our LP produces a schedulable policy.

Lemma 2: Suppose that a given system is schedulable under Chen et al.'s static policy. If we design the hardening policy of every task by solving the given LP, the system remains schedulable.

Proof: Consider an arbitrary task τ_i. Let $\Psi_i^p(\ell)$ (and $\Psi_i^s(\ell)$) be the maximum total execution time of ℓ consecutive jobs of τ_i under the hardening policy produced by the LP (and Chen et al's static policy, respectively). As was also observed in [7], we have $\Psi_i^s(\ell) = \lfloor \ell/k_i \rfloor \Psi_i^s(k_i) + \Psi_i^s(\ell \mod k_i)$. This is because Ψ^s is defined by a static policy of period k_i, and we do not necessarily have the same equality for Ψ_i^p.

We claim that $\Psi_i^p(\ell) \le \Psi_i^s(\ell)$ for all $\ell \in \mathbb{N}$. Note that this together with Lemma 1 yields the desired conclusion.

When the hardening policy produced by the LP is followed, for any $\ell' \in \{1, \ldots, k_i\}$ consecutive jobs, the number of jobs executed in the error-correcting mode is at most $\omega(\ell')$.

[5] Later in the proof of Lemma 2, we use a slightly weaker condition than this. We present a stronger version here because it does not affect the space of feasible policies.

(*Proof.* Consider these ℓ' jobs and the preceding $k_i - \ell'$ jobs. Let a_0 be the decision made for the last of these k_i jobs and $t_{-(k_i-1)}, \ldots, t_{-1}$ be the trace of the first $k_i - 1$ jobs. We then have that $\pi_{t_{-(k_i-1)},\ldots,t_{-1},a_0}$ is not forbidden.[6] Therefore, the number of error-correcting executions during the last ℓ' jobs is at most $\omega(\ell')$.) We have

$$\begin{aligned}\Psi_i^p(\ell) &\le (k_i - \omega(\ell'))ex_i^d + \omega(\ell')ex_i^c \\ &\le (k_i - \chi(\ell'))ex_i^d + \chi(\ell')ex_i^c \\ &= \Phi_i^s(\ell'),\end{aligned}$$

where the first inequality follows from $0 < ex_i^u \le ex_i^d \le ex_i^c$ and the second from $\chi(\ell') \ge \omega(\ell')$ since $\pi_{t_{-(k_i-1)},\ldots,t_{-1},a_0}$ is not forbidden. Since this inequality holds for any ℓ' consecutive jobs, we have $\Psi_i^p(\ell) \le \lfloor \ell/k_i \rfloor \Psi_i^p(k_i) + \Psi_i^p(\ell \mod k_i) \le \lfloor \ell/k_i \rfloor \Psi_i^s(k_i) + \Psi_i^s(\ell \mod k_i) = \Psi_i^s(\ell)$ for any $\ell \in \mathbb{N}$, completing the proof. □

B. RUN-TIME HARDENING DECISION

Once we solved the LPs at the design time, the run-time procedure for hardening decision is simple. Without any "optimization" computation, we can simply follow the stochastic policy dictated by the LP solution. The run-time procedure is easily derived from the following observation.

With a slight abuse of notation, let $t_{-(k-1)}, \ldots, t_{-1}$ denote the *event* that the $(k-1)$-st, ..., first trace from the last are $t_{-(k-1)}, \ldots, t_{-1}$, respectively. Similarly, let a_0 denote the event that the decision for the current job is a_0. From the definition of conditional probabilities, we have

$$\begin{aligned}\Pr[a_0 \mid t_{-(k-1)}, \ldots, t_{-1}] &= \frac{\Pr[t_{-(k-1)}, \ldots, t_{-1}, a_0]}{\Pr[t_{-(k-1)}, \ldots, t_{-1}]} \\ &= \frac{\pi_{t_{-(k-1)},\ldots,t_{-1},a_0}}{\sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-1},a_0}}.\end{aligned}$$

This shows that the following run-time procedure ensures that the marginal probability distribution indeed becomes π.

1) PER-JOB PROCEDURE

Consider an arbitrary job. Let $t_{-(k-1)}, \ldots, t_{-1}$ be the trace of the last $k-1$ jobs. For the current job, we execute it

- in the unreliable mode with probability
$$\pi_{t_{-(k-1)},\ldots,t_{-1},u} \Big/ \sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-1},a_0},$$
- in the error-detecting mode with probability
$$\pi_{t_{-(k-1)},\ldots,t_{-1},d} \Big/ \sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-1},a_0}, \text{ and}$$
- in the error-correcting mode with probability
$$\pi_{t_{-(k-1)},\ldots,t_{-1},c} \Big/ \sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-1},a_0}.$$

[6]Technically speaking, we need to argue here that probability zero events never happen in our policy, but this will be clear from Section V-B.

Algorithm 1 Run-Time Procedure to Make Hardening Decisions for a Single Task

```
initialize the last traces: choose t_{-(k-1)}, ..., t_{-1} with probability Σ_{a0∈A} π_{t_{-(k-1)},...,t_{-1},a0}
for each job do
    sample a random number R ~ U[0, 1)
    if R < π_{t_{-(k-1)},...,t_{-1},u} / Σ_{a0∈A} π_{t_{-(k-1)},...,t_{-1},a0} then
        execute the current job in the unreliable mode
        t_0 ← u
    else if R < (π_{t_{-(k-1)},...,t_{-1},u} + π_{t_{-(k-1)},...,t_{-1},d}) / Σ_{a0∈A} π_{t_{-(k-1)},...,t_{-1},a0} then
        execute the current job in the error-detecting mode
        if the execution was fault-free then
            t_0 ← d_n
        else
            t_0 ← d_e
        end if
    else
        execute the current job in the error-correcting mode
        t_0 ← c
    end if
    (t_{-(k-1)}, ..., t_{-1}) ← (t_{-(k-2)}, ..., t_0)
end for    {until the system halts}
```

2) INITIALIZING PROCEDURE

It still needs to be specified how the "last traces" are defined for the first job. In fact, a careful initialization is crucial to ensure that the system is jumpstarted into the steady state from the beginning. When the system starts, we define the last traces $t_{-(k-1)}, \ldots, t_{-1}$ by sampling them from the probability distribution $\{\sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-1},a_0}\}_{t_{-(k-1)},\ldots,t_{-1}}$. That is, the probability that the $(k-1)$ last traces are initialized as $t_{-(k-1)}, \ldots, t_{-1}$ is $\sum_{a_0 \in A} \pi_{t_{-(k-1)},\ldots,t_{-1},a_0}$. This stochastic initialization removes transient behavior from the system and enables us to immediately achieve the steady-state optimum computed by the LP.

Algorithm 1 summarizes the run-time procedure described so far.

VI. EVALUATION

In this section, we evaluate the effectiveness of the proposed method using various task sets.

A. SYNTHETIC BENCHMARKS

1) GENERATION OF BENCHMARKS

First, we applied the proposed method to a set of benchmarks synthesized using the procedure of Chen *et al.* [7]. Although we randomly synthesized these benchmarks, we used the same set of parameters and algorithms as [7]. We briefly describe them in what follows. Further discussions on the generation of the benchmarks can be found in Chen *et al.* [7].

- Each benchmark was defined by two parameters: the sum of the nominal "peak utilizations"[7] of all jobs, and

[7]The peak utilization of a task is defined as its utilization when every job of the task runs in the error-correcting mode.

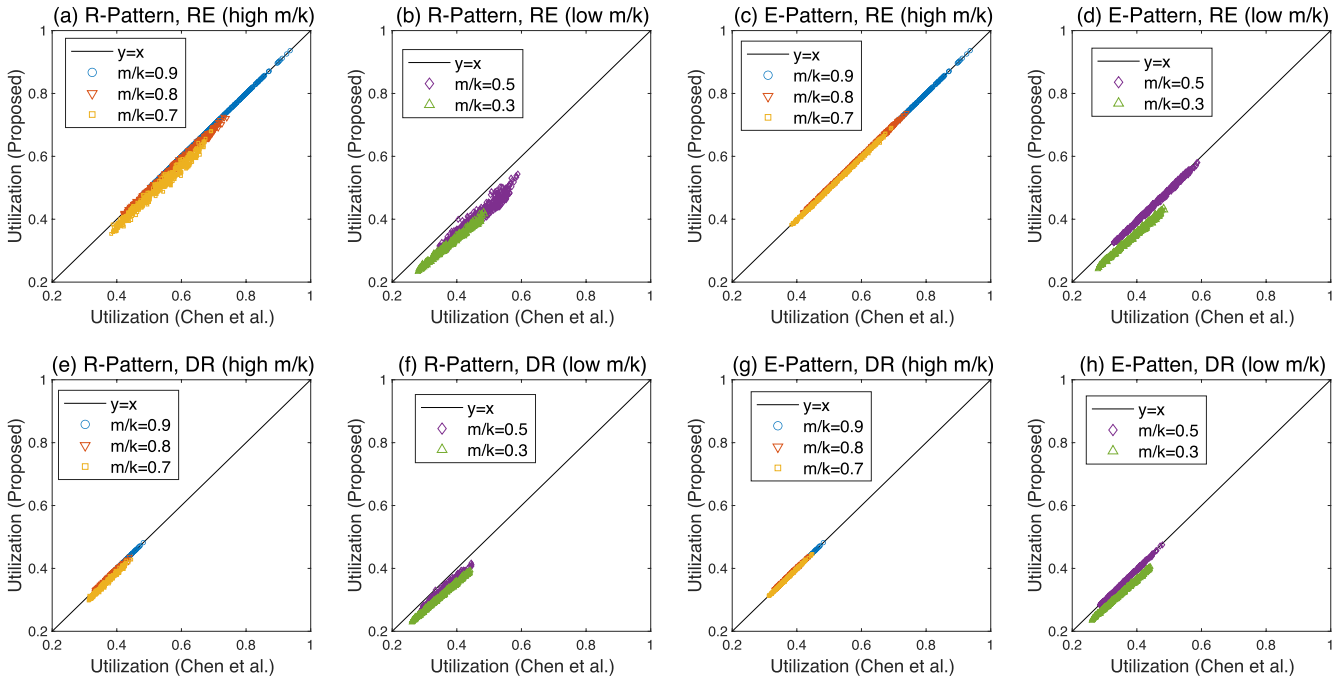


FIGURE 4. CPU utilization of the proposed method compared to Chen *et al.*'s dynamic policy.

the ratio m/k. The former was varied from 60% to 100%, in the increment of 1%. The latter was chosen as one of the following three values: 0.7, 0.8, and 0.9. For each of the $41 \cdot 3 = 123$ combinations of the total peak utilization and m/k, we generated 10 benchmarks. Each benchmark in turn consisted of 10 tasks.

- Once the total peak utilization was specified, the peak utilization U_i of each job τ_i was determined by using the UUniFast [22] method. The task periods T_i were generated so that they were distributed in three buckets [1, 10], [10, 100], and [100, 1000] as evenly as possible. As Chen *et al.* indicated, this "exponential" distribution was suggested by Davis *et al.* [23]. Since our model considers implicit deadlines, the relative deadline of τ_i is set equal to T_i.
- The error-correcting execution time of τ_i was set as $ex_i^c := T_i U_i$, in accordance with the definition of the peak utilization. The other execution times were set as $ex_i^u := ex_i^c/3$ and $ex_i^d := 1.21 \cdot ex_i^u$. Chen *et al.* selected these parameters to emulate the behavior of existing hardening techniques such as SWIFT+PROFiT [24]. They set the fault probabilities as $f_i^u = f_i^d = 0.3$.
- In order to determine the (m_i, k_i)-constraint, we first chose k_i uniformly at random from [3, 10] and then set m_i according to the desired value of m/k, rounded to the nearest integer.
- The only parameter that was not explicitly discussed by Chen *et al.* [7] is the reliability target. Since their model had no quantitative reliability targets, for fair comparison, we set $p_i^{req} := 0$ for all jobs.

2) SIMULATED UTILIZATIONS OF SYNTHETIC BENCHMARKS

For each benchmark, we first checked its schedulability (see Lemma 1). For each schedulable benchmark, we simulated and measured the performance of Chen *et al.*'s dynamic policy and the proposed policy twice, once with R-patterns and then with E-patterns. The simulation was performed for $10^5 \cdot \max_i T_i$ units of time. For the sake of generality, the release offset r_i of each task τ_i was randomly selected from $[0, T_i)$ in each simulation. The whole evaluation was first performed for the *reliable execution (RE)* technique, and then repeated for *detection and recovery (DR)*. The latter evaluation involved accordingly modifying ex^c.

Fig. 4(a) shows the simulation results of the schedulable benchmarks with R-patterns and RE. Each point corresponds to a single benchmark, whose x-coordinate is the utilization achieved by Chen *et al.*'s dynamic policy [7] and the y-coordinate is by the proposed method. Interestingly, if we focus on the points with $m/k = 0.9$ (marked as blue circles in Fig. 4(a)), we can see that there was no improvement. (Since faults were randomly injected, small fluctuation appeared.) This is because, when $m/k = 0.9$, each task τ_i can tolerate up to only one fault in every k_i consecutive frames. When this is the case, there is nothing much better to do than immediately starting error-correction as soon as we detect a fault and continuing until this fault goes beyond the time horizon of k_i frames. This is what Chen *et al.*'s dynamic policy already does. Hence, for this special case, there cannot be any improvement. Since we observed some improvements for $m/k = 0.8$ and 0.7, this suggested that the relation between the performance and m/k needs to be more carefully studied.

To this end, we added two more choices of m/k to our test set. In total, m/k were chosen as one of the following five values: {0.3, 0.5, 0.7, 0.8, 0.9}. See Fig. 4(b) for the results on these additional benchmarks.[8]

Fig. 4(c)-(d) show the results with E-patterns. Since E-patterns distribute 0's more evenly, schedulability could be guaranteed for a more limited set of policies than R-patterns, and improvements tended to be smaller. Fig. 4(a)-(d) confirmed that, under RE, the proposed method improved the total CPU utilization, and that the improvement became greater when m/k was smaller. The improvement for R-patterns was by 7.16% on average, whereas 2.80% for E-patterns. In the best benchmark, the improvement was by as high as 18.45%.

Fig. 4(e)-(h) shows the results when the evaluation was repeated for DR. The improvement was by 6.41% for R-patterns on average and 3.39% for E-patterns. The largest improvement for a benchmark was 13.35%. On one hand, improvements tended to be smaller than RE because the opportunistic behavior of DR reduces ex_i^c in expectation. This implies that lowering the number of error-correcting executions became relatively less "profitable". On the other hand, the set of schedulable benchmarks were quite different between RE and DR. Therefore, even though we used the same set of benchmarks for RE and DR, a direct comparison can be misleading. Out of the 2,050 benchmarks,[9] 2,014 were schedulable with R-patterns under RE (and 2,019 with E-patterns). On the contrary, only 1,067 (and respectively 1,141) benchmarks were schedulable with R-patterns (and E-patterns) under DR. Such poor performance of DR in schedulability was reported also by Chen *et al.* [7].

a: EDF SCHEDULING

Finally, we measured the performance of the proposed policy when EDF scheduling [25] was used in conjunction with it. While the proposed method assumes RM scheduling (see Section V-A2), the optimality of EDF scheduling[10] implies that the actual implementation of our system can also use EDF without harming the system schedulability. Fig. 5 shows the evaluated performance of the proposed policy under EDF scheduling. As can be expected from the work-conserving nature of both EDF and RM, the results showed almost equal improvements in both scheduling methods. While the average improvement was by 4.93%, the largest improvement in a single benchmark was by 18.33%.

[8]We note that the CPU utilization was generally lower both under Chen *et al.*'s dynamic policy and under the proposed method when m/k is smaller. This is because a smaller value of m/k implies that more jobs can be executed without error correction, resulting in lower CPU usage.

[9]We had 41 points of peak utilization and five points of m/k ratio. For each of these $41 \cdot 5$ settings, we generated 10 benchmarks. The total number of benchmarks was therefore $41 \cdot 5 \cdot 10 = 2,050$.

[10]This can be proven from a typical exchange argument and mathematical induction used to show the optimality of greedy algorithms. We refer the interested readers to Jackson [26], although the settings slightly differ in their details.

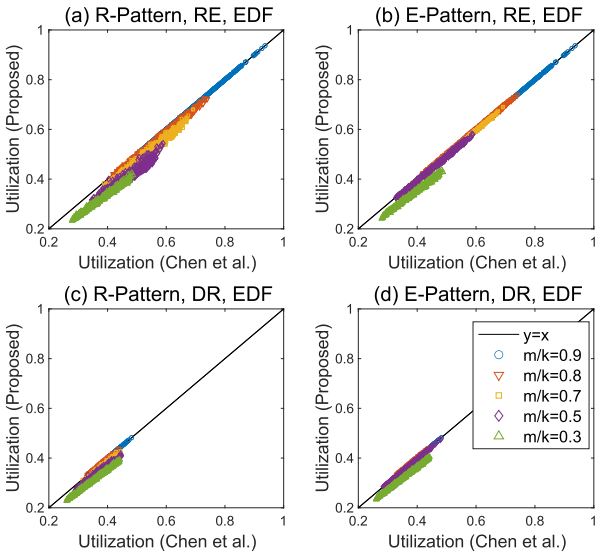


FIGURE 5. CPU utilization of the proposed method under EDF scheduling.

B. ADDITIONAL EVALUATIONS

While we synthesized an extensive set of benchmarks using Chen *et al.*'s method, there remain aspects that are not adequately covered by this set. In this section, we complement the previous section by presenting two additional evaluations.

In the rest of this section, fixing the total nominal peak utilization and m/k at the midpoints (80% and 0.7, respectively), we focus on ten benchmarks among those generated in Section VI-A1. All evaluations were performed with R-patterns and RE.

1) HARDENING WITH RE-EXECUTION

The proposed method is independent from specific hardening techniques adopted by the system, but benchmarks from Section VI-A1 assumed a particular hardening technique. Kang *et al.* [10] proposed an error-correction technique that is based on re-executions. It can be modeled by scaling down ex_i^u and ex_i^d so that $3 \cdot ex_i^d = ex_i^c$ (where the *re-execution degree* is 2). We evaluated the proposed method with these modified benchmarks.

Fig. 6 shows the results, confirming that our approach yielded improvements under this varied setting too, as was expected. On average, the improvement over Chen *et al.*'s dynamic policy was by 6.80%, which was slightly larger than the improvement for these ten benchmarks in Section VI-A (5.76%). This is due to the fact that ex_i^c was relatively larger than the other two execution times in the new benchmarks, and therefore saving error-correcting executions had higher impact on the CPU utilization.

2) QUANTITATIVE RELIABILITY TARGETS

Another important aspect that was missing from the benchmark of Section VI-A1 is the quantitative reliability target. For each task in the ten benchmarks, we randomly set its

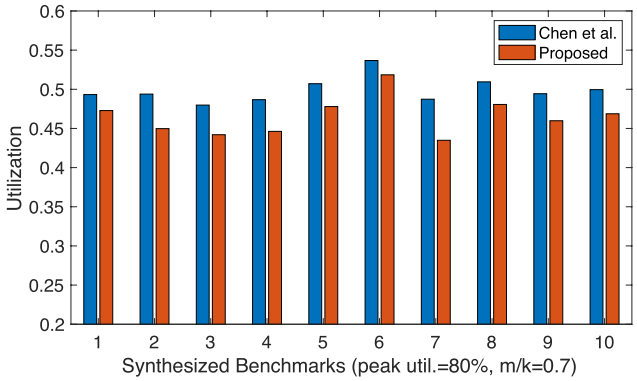


FIGURE 6. CPU utilization of re-execution hardening benchmarks.

TABLE 6. Rates of (m_i, k_i)-constraint violations by reliability targets.

Target (p^{req})	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Average	0.974×10^{-2}	0.969×10^{-3}	0.941×10^{-4}	0.928×10^{-5}
Median	1.000×10^{-2}	1.000×10^{-3}	0.997×10^{-4}	0.998×10^{-5}
Min.	0.578×10^{-2}	0.018×10^{-3}	0.010×10^{-4}	0.000×10^{-5}
Max.	1.019×10^{-2}	1.354×10^{-3}	1.170×10^{-4}	1.800×10^{-5}
Std. Dev.	0.087×10^{-2}	0.158×10^{-3}	0.238×10^{-4}	0.325×10^{-5}

reliability target as one of the following values: $p_i^{req} \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. We repeated this random generation five times, obtaining 50 benchmarks in total.

We simulated and monitored the (m, k)-constraint violation rates of the proposed method for these benchmarks. Since faults were randomly injected and the reliability target could be as small as 10^{-5}, a longer simulation was necessary. The simulation was performed for $10^6 \cdot \max_i T_i$ units of time. The 50 benchmarks contained 500 tasks in total, which had different reliability targets. We classified them according to their reliability targets and calculated the statistics shown in Table 6. The results confirmed that the proposed method met the reliability target. As our evaluations were based on randomly injected faults, fluctuations were naturally observed in all cases, especially those with small reliability targets.

C. CASE STUDY

Finally, as a case study, we evaluated the proposed method using the parameters profiled by Chen *et al.* [7] from an existing real-time control application. They used a robot [27] on LEGO Mindstorms NXT with an ARM7 microprocessor and profiled three periodic real-time control tasks with fault injection. For completeness' sake, we show the entire set of parameters in Table 7. We fixed τ_1 in the error-correcting mode, as in [7]. We verified that the system was schedulable, and all simulations were performed for $10^7 \cdot \max_i T_i$ units of time.

For a comparison with Chen *et al.*'s dynamic policy, we first evaluated the proposed method for $p_i^{req} = 0$. Table 8 shows that the proposed method successfully improved the CPU utilization by 8.548%. In addition to this, we also evaluated the proposed method with $p_2^{req} = p_3^{req} = 10^{-5}$. The results confirm that the proposed method met the reliability targets. The improvement in the total CPU utilization did not significantly differ from that when $p^{req} = 0$, which was as expected from the very small value of $p^{req} = 10^{-5}$.

TABLE 7. System parameters [7] (case study).

	Task Name	k_i	m_i	f_i^u	f_i^d	ex_i^u	ex_i^d	ex_i^c	p_i^{req}	T_i
τ_1	Balance	1	1	n/a	n/a	n/a	n/a	435	0	4000
τ_2	Path	10	3	0.3	0.3	99.267	102.598	291.139	$0, 10^{-5}$	1000
τ_3	Distance	5	3	0.3	0.3	99.933	103.93	173.217	$0, 10^{-5}$	3000

TABLE 8. Evaluation results (case study).

	Chen et al.'s Dynamic	Proposed ($p_i^{req} = 0$)	Proposed ($p_{2,3}^{req} = 10^{-5}$)
CPU Utilization	27.464%	25.116% (8.548% improv.)	25.112% (8.565% improv.)
(m, k)-const. viol. rate of τ_2	0	0	1.090×10^{-5}
(m, k)-const. viol. rate of τ_3	0	0	0.960×10^{-5}

STATISTICS ON THE LP SOLUTION TIME

We conclude this section by reporting the computation times required to solve the LPs in all the evaluations of Section VI. We solved all LPs on a Naver Cloud Platform Server [28] with 32vCPUs and 128GB of RAM, using the barrier method of IBM ILOG CPLEX 20.1.0. We selected the barrier method as it is known to perform well on LPs with sparse columns [29]. On average, solving one LP took 56.685 seconds, with the standard deviation of 216.926 seconds. The minimum running time was 0.006 seconds, whereas the maximum was 1,335.811 seconds. The median was 0.229 seconds. This shows that solving most of the LPs required very short amount of time, and even the LP that took the longest time was solved in less than 25 minutes.

In the proposed method, all the optimization happens at the design time, and therefore we can afford a significant amount of time on optimization. Nevertheless, the above statistics show that the computational load was already manageable in our evaluations.

VII. CONCLUDING REMARKS

In this paper, we proposed a method to design an adaptive stochastic hardening policy for real-time systems. We allow the behavior of the policy to significantly deviate from the static counterpart. We use an LP to explore the huge space of adaptive stochastic policies whose schedulabilities are guaranteed. The proposed approach lets quantitative reliability targets be specified, departing from the previous

all-or-nothing viewpoint. We demonstrated that the proposed method brings improvements in CPU utilization while meeting reliability targets through an extensive evaluation. We note that the run-time decision process can be easily implemented by table lookups, as all the optimization efforts are made at design-time while solving the LP.

We will conclude this paper with some discussion on practical considerations and/or future directions of research.

While all the computational effort for optimization occurs at the design time, it may still be desirable to reduce the design-time computational load in some use cases. Depending on the platform and/or error-detecting/correcting technology used, ex^u may be close to ex^d. In this case, the unreliable mode may become ''less useful'' because unreliable and error-detecting execution times are close, but an unreliable execution has the obvious disadvantage that a fault cannot be detected. We could then instruct the LP not to use the less useful mode at all, by declaring any variables whose trace contains u as forbidden. This would shrink the size of the LP and help reducing the computational load. An additional heuristic we have is the following. If two tasks in a system have the same parameters (m, k), p^{req}, and the ratio $ex^u : ex^d : ex^c$, their LP optimal solutions must be the same, so we do not need to solve the LP a multiple number of times.

Another practical consideration is related to steady states. It may be the case that, starting from one trace, one cannot arrive at another trace in some LP solution. If we want every trace with nonzero LP variable value to be eventually observed, we may intermittently re-initialize the policy with small probability.

Finally, while the proposed method focuses on the hardening policy on a uniprocessor, it would be necessary and interesting to extend the method to multiprocessor systems so that the hardening policy can be co-optimized with processor mapping. This is because today's real-time workloads are becoming more likely to be processed on top of multiprocessor systems. In particular, the hardening decision can be co-optimized with the energy dissipation of multiprocessor systems through task-to-processor mapping and DVFS [18]. For that, the system model used in this work would need to be extended to capture the trade-off among frequency, reliability, and schedulability. Another important aspect that the hardening decision can be co-optimized with is the task migration between processors [19] as it would remarkably affect the schedulability and reliability.

We believe further investigation on these future topics would be of great interest.

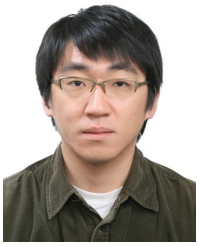
REFERENCES

[1] P. Pillai and K. G. Shin, ''Real-time dynamic voltage scaling for low-power embedded operating systems,'' in *Proc. 18th ACM Symp. Operating Syst. Princ.*, Oct. 2001, pp. 89–102.
[2] V. Devadas and H. Aydin, ''On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications,'' *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 31–44, Jan. 2012.
[3] V. Chaturvedi, H. Huang, S. Ren, and G. Quan, ''On the fundamentals of leakage aware real-time DVS scheduling for peak temperature minimization,'' *J. Syst. Archit.*, vol. 58, no. 10, pp. 387–397, Nov. 2012.
[4] D. Rai, H. Yang, I. Bacivarov, J.-J. Chen, and L. Thiele, ''Worst-case temperature analysis for real-time systems,'' in *Proc. Design, Autom. Test Eur.*, Mar. 2011, pp. 1–6.
[5] G. Chen, N. Guan, K. Huang, and W. Yi, ''Fault-tolerant real-time tasks scheduling with dynamic fault handling,'' *J. Syst. Archit.*, vol. 102, Jan. 2020, Art. no. 101688.
[6] A. Burns, R. Davis, and S. Punnekkat, ''Feasibility analysis of fault-tolerant real-time task sets,'' in *Proc. 8th Euromicro Workshop Real-Time Syst.*, Jun. 1996, pp. 29–33.
[7] K.-H. Chen, B. Bönninghoff, J.-J. Chen, and P. Marwedel, ''Compensate or ignore? Meeting control robustness requirements through adaptive soft-error handling,'' in *Proc. 17th ACM SIGPLAN/SIGBED Conf. Lang., Compil., Tools, Theory for Embedded Syst.*, New York, NY, USA, Jun. 2016, pp. 82–91, doi: 10.1145/2907950.2907952.
[8] P. Pop, V. Izosimov, P. Eles, and Z. Peng, ''Design optimization of time-and cost-constrained fault-tolerant embedded systems with checkpointing and replication,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 389–402, Mar. 2009.
[9] C. Bolchini and A. Miele, ''Reliability-driven system-level synthesis for mixed-critical embedded systems,'' *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2489–2502, Dec. 2013.
[10] S.-H. Kang, H. Yang, S. Kim, I. Bacivarov, S. Ha, and L. Thiele, ''Static mapping of mixed-critical applications for fault-tolerant MPSoCs,'' in *Proc. 51st Annu. Design Autom. Conf. Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.
[11] P. Ramanathan, ''Overload management in real-time control applications using (m, k)-firm guarantee,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 6, pp. 549–559, Jun. 1999.
[12] J. Zhou, J. Sun, M. Zhang, and Y. Ma, ''Dependable scheduling for real-time workflows on cyber–physical cloud systems,'' *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7820–7829, Nov. 2021.
[13] B. Kim and H. Yang, ''Reliability optimization of real-time satellite embedded system under temperature variations,'' *IEEE Access*, vol. 8, pp. 224549–224564, 2020.
[14] T. Santini, C. Borchert, C. Dietrich, H. Schirmeier, M. Hoffmann, O. Spinczyk, D. Lohmann, F. R. Wagner, and P. Rech, ''Effectiveness of software-based hardening for radiation-induced soft errors in real-time operating systems,'' in *Proc. Int. Conf. Archit. Comput. Syst.* Springer, 2017, pp. 3–15. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-54999-6_1
[15] S. Herbert and D. Marculescu, ''Analysis of dynamic voltage/frequency scaling in chip-multiprocessors,'' in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2007, pp. 38–43.
[16] J. Zhou, J. Sun, X. Zhou, T. Wei, M. Chen, S. Hu, and X. S. Hu, ''Resource management for improving soft-error and lifetime reliability of real-time MPSoCs,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 12, pp. 2215–2228, Dec. 2019.
[17] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang, ''System-level reliability modeling for MPSoCs,'' in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES/ISSS)*, 2010, pp. 297–306.
[18] H. Ali, M. S. Qureshi, M. B. Qureshi, A. A. Khan, M. Zakarya, and M. Fayaz, ''An energy and performance aware scheduler for real-time tasks in cloud datacentres,'' *IEEE Access*, vol. 8, pp. 161288–161303, 2020.
[19] A. A. Khan, A. Ali, M. Zakarya, R. Khan, M. Khan, I. U. Rahman, and M. A. A. Rahman, ''A migration aware scheduling technique for real-time aperiodic tasks over multiprocessor systems,'' *IEEE Access*, vol. 7, pp. 27859–27873, 2019.
[20] G. Quan and X. Hu, ''Enhanced fixed-priority scheduling with (m,k)-firm guarantee,'' in *Proc. 21st IEEE Real-Time Syst. Symp.*, Nov. 2000, pp. 79–88.
[21] A. K. Mok and D. Chen, ''A multiframe model for real-time tasks,'' *IEEE Trans. Softw. Eng.*, vol. 23, no. 10, pp. 635–645, Oct. 1997.
[22] E. Bini and G. C. Buttazzo, ''Measuring the performance of schedulability tests,'' *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, May 2005. [Online]. Available: https://doi.org/10.1007/s11241-005-0507-9
[23] R. I. Davis, A. Zabos, and A. Burns, ''Efficient exact schedulability tests for fixed priority real-time systems,'' *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1261–1276, Sep. 2008.
[24] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, ''Software-controlled fault tolerance,'' *ACM Trans. Archit. Code Optim.*, vol. 2, no. 4, pp. 366–396, Dec. 2005. [Online]. Available: https://doi.org/10.1145/1113841.1113843
[25] C. L. Liu and J. W. Layland, ''Scheduling algorithms for multiprogramming in a hard-real-time environment,'' *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973. [Online]. Available: https://doi.org/10.1145/321738.321743

[26] J. R. Jackson, "Scheduling a production line to minimize maximum tardiness," Univ. California, Los Angeles, CA, USA, Manage. Sci. Res. Project Res. Rep. 43, 1955.
[27] Y. Yamamoto. *NXTway-GS (Self-Balancing Two-Wheeled Robot) Controller Design*. Accessed: Sep. 13, 2021. [Online]. Available: https://www.mathworks.com/matlabcentral/fileex change/19147-nxtway-gs-self-balancing-two-wheeled-robot-controller-design
[28] NAVER Cloud Corp. *NAVER Cloud Platform*. Accessed: Oct. 14, 2021. [Online]. Available: https://www.ncloud.com/
[29] IBM Corporation. *Introducing the Barrier Optimizer—IBM Documentation*. Accessed: Oct. 14, 2021. [Online]. Available: https://www.ibm.com/docs/en/icos/20.1.0?topic=optimizer-introducing-barrier

HYUNG-CHAN AN (Member, IEEE) received the B.S. degree in computer science and engineering from Seoul National University, Seoul, South Korea, in 2006, and the Ph.D. degree in computer science from Cornell University, Ithaca, NY, USA, in 2012.

From 2012 to 2016, he was a Postdoctoral Researcher at the École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. He is currently an Assistant Professor with the Department of Computer Science, Yonsei University, Seoul. His research interests include combinatorial optimization and its application to engineering problems.



HOESEOK YANG (Member, IEEE) received the B.S. degree in computer science and engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2003 and 2010, respectively.

He was a Postdoctoral Researcher with D-ITET, ETH Zürich, Zürich, Switzerland, from 2010 to 2014. He joined Ajou University, Suwon, South Korea, as an Assistant Professor, in 2014, where he is currently an Associate Professor. His current research interests include HW/SW codesign, reliability- and temperature-aware optimization/analysis of multiprocessor system-on-chip (MPSoC), embedded systems design with non-volatile memories, and deep learning for embedded systems.