Lifetime-aware real-time task scheduling on fault-tolerant mixed-criticality embedded systems☆

Kun Cao [a], Guo Xu [a], Junlong Zhou [b], Mingsong Chen [a], Tongquan Wei [a,*], Keqin Li [c,1]

[a] Department of Computer Science and Technology, East China Normal University, Shanghai 200241, China
[b] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China
[c] Department of Computer Science, State University of New York, NY 12561, USA

HIGHLIGHTS

- Conduct the first study on maximizing the lifetime of fault-tolerant mixed-criticality embedded systems.
- Consider both transient faults and thermal cycling incurred permanent faults.
- Present a mixed-integer linear programming (MILP) formulation.
- Propose a novel time-efficient heuristic.
- Carried out experiments on both real-world and synthetic benchmarks.

ARTICLE INFO

ABSTRACT

In recent years, the design of mixed-criticality embedded systems suffering from transient faults has attracted much attention. From the perspective of system users, it is desirable to optimize system lifetime while meeting all design constraints. Existing task scheduling algorithms cannot be utilized to maximize the lifetime of mixed-criticality embedded systems since they do not take into account the impact of providing transient fault tolerance on system lifetime. This paper investigates the problem of prolonging the lifetime of mixed-criticality embedded systems on a uniprocessor equipped with dynamic voltage and frequency scaling (DVFS) technique. The transient faults and thermal cycling incurred permanent faults are simultaneously considered in the system lifetime optimization under the constraints of safety requirements and schedule timeliness. A mixed-integer linear programming (MILP) formulation is first presented to deal with the task scheduling problem. Since the MILP method is a time-consuming solution for large-scale systems, a cross-entropy method based heuristic is then proposed to achieve a better tradeoff between the system lifetime achieved by the derived task schedule and the runtime consumed to generate the task schedule. Experiments based on synthetic and real-world benchmarks are conducted, and simulation results demonstrate that the proposed heuristic improves system lifetime by up to 32.73% with acceptable runtime as compared to benchmarking methods.

1. Introduction

In many safety-related fields such as avionics and automotive industries, tasks of different importance (i.e., criticality levels) usually co-exist [1,2]. For example, the two tasks of music playing and flight control in the flight management systems are obviously with unequal criticality levels and distinct degrees of assurance. However, traditional embedded systems only allow the existence of tasks with a same criticality level, which indicates that the tasks with different criticality levels need to run on multiple separate hardware platforms. As a result, the cost, weight, and power consumption of these hardware platforms will significantly increase or even become unaffordable as the total number of task criticality levels increases. A promising trend in the design of nowadays advanced embedded systems is to integrate multiple functionalities (i.e., tasks) with no less than two criticality levels onto a common and shared computing platform. These emerging embedded systems that permit the co-existence of different task criticality levels are called mixed-criticality (MC) embedded

systems [3,4]. In an MC embedded system, tasks having higher importance should be provided higher criticality levels for the purpose of guaranteeing the safety requirements of these tasks.

In the past few years, there has been a widespread interest in the investigation of MC embedded systems. However, as detailed in [4], most of the existing techniques (e.g., mechanisms in [5–9]) dedicated for MC embedded systems cannot guarantee a dependable system operation because they fail to take fault-tolerance as a design constraint. As the susceptibility of modern processors to soft errors is dramatically increasing with the relentless scaling of feature size and operating voltage [10,11], fault-tolerance management is deemed to be a significant and pressing research issue in MC embedded systems. Recently, several research works [12–16] have devoted to the design of fault-tolerant MC embedded systems. Unfortunately, these existing task scheduling algorithms only consider tolerating transient faults while neglecting tolerance for permanent faults. Directly adopting such algorithms will inevitably accelerate processor wearouts that eventually result in permanent faults occurring earlier and system lifetime decaying drastically. From the perspective of system users, it is desirable to prolong the lifetime of MC embedded systems able to handle both permanent faults and transient faults. However, to the best of our knowledge, there are no works on investigating the lifetime optimization of MC embedded systems that can deal with permanent and transient faults simultaneously.

In this paper, we conduct the first study of prolonging the lifetime of MC embedded systems on a uniprocessor equipped with dynamic voltage and frequency scaling (DVFS) technique. We take into account both transient faults and thermal cycling incurred permanent faults in the system lifetime optimization under the constraints of safety requirements and schedule timeliness. By judiciously determining (1) the task operating frequency, (2) the task re-execution number, and (3) the task execution order, our proposed solution can generate a lifetime-optimum task schedule while satisfying all design requirements. The main contributions of this paper are summarized below.

- We present a mixed-integer linear programming (MILP) formulation to schedule independent real-time tasks for maximizing the lifetime of uniprocessor MC embedded systems.
- We propose a time-efficient solution developed on the cross-entropy method (CEM) to the formulated scheduling problem.
- Experimental results based on synthetic and real-life benchmarks demonstrate that the developed approach prolongs system lifetime by up to 32.73%.

The remainder of the paper is organized as follows. Section 2 surveys the related works on MC embedded systems. We briefly introduce the system architecture and models in Section 3. In Section 4, we describe the problem definition and give an overview of our developed algorithms. Section 5 presents an MILP formulation for the studied task scheduling problem while Section 6 shows the developed CEM-based heuristic. Section 7 numerically investigates the performance achieved by the proposed scheme and Section 8 gives concluding remarks.

2. Related work

Considerable research efforts have been denoted to the design of MC embedded systems. Baruah et al. [5] presented an algorithm of earliest deadline first with virtual deadlines to schedule tasks with any number of defined criticality levels. Huang et al. [6] proposed an effective scheduling algorithm integrating the DVFS technique for optimizing the whole power dissipation of MC embedded systems. Han et al. [7] derived a criticality-aware utilization bound for feasibility tests and developed a novel

scheduling approach to improve task schedulability. From the perspective of probability, Maxim et al. [8] conducted a comprehensive study on fixed priority preemptive task scheduling for MC embedded systems. Davis et al. [9] presented two fixed priority preemptive scheduling schemes integrating context switch costs, and conducted multi-set analyses for each scheduling scheme. However, since all the aforementioned techniques [5–9] fail to consider tolerating transient faults as a design constraint, they are extremely likely to generate unreliable outputs. As a result, a dependable operation for MC embedded systems cannot be guaranteed.

Taking tolerating transient faults as a design constraint, Huang et al. [12] designed a scheduling algorithm that adopts task re-execution technique to handle transient faults. In addition, the authors investigated and quantitatively analyzed the impact of the three techniques of service degradation, task killing, and task re-execution on task schedulability and system safety. Lin et al. [13] developed a two-stage scheduling scheme to ensure the timing requirements of mixed-criticality tasks. At offline stage, a static algorithm is presented to guarantee that most tasks can successfully perform their executions on time. At online stage, a slack-reclaiming algorithm is designed to prevent the system from computing faults without violating task deadline constraints. Zeng et al. [14] explored the joint use of task re-execution and task replication techniques for boosting system safety. Besides, the two techniques of service degradation and task killing are utilized to address the undesirable urgencies where high criticality tasks may not finish their executions at runtime.

Unlike the above works [12–14], Al-bayati et al. [15] modeled four modes to cope with the certification and reliability requirements of MC embedded systems. The proposed model can retain as many low criticality tasks as possible when either overruns or transient faults occur. Zhou et al. [16] presented an effective scheduling solution based on the scheduling scheme of earliest deadlines first with virtual deadlines. Several techniques of period transformation, utilization of idle time, and re-execution are utilized in this algorithm to achieve high reliability and schedulability. However, these task scheduling algorithms themselves [12–16] cannot be directly utilized to optimize the lifetime of MC embedded systems as the impact of tolerating transient faults on system lifetime is not investigated. Furthermore, to our best knowledge, no research works have focused on optimizing the lifetime of MC embedded systems.

3. System architecture and models

This section briefly introduces the system architecture and models including task model, temperature model, fault model, and safety requirement model. Table 1 summarizes the main symbols utilized in this section.

3.1. System architecture and task model

In this work, we consider the typical MC embedded systems that only allow the co-existence of two task criticality levels: one is the high criticality level $\mathcal{H}$ and the other is the low criticality level $\mathcal{L}$ [4,12,16]. Following the works presented in [6,17,18], the target MC embedded system is modeled as a DVFS-enabled uniprocessor Θ supporting Q discrete voltage/frequency pairs $\Psi = \{(v_q, f_q) | q = 1, 2, \ldots, Q\}$. For ease of presentation, we assume that the inequality $v_q < v_{q+1}$ with respect to voltage and the inequality $f_q < f_{q+1}$ with respect to frequency hold for $\forall q \in [1, 2, \ldots, Q - 1]$. As shown in [6,12,16–20], tasks independent with each other are exceedingly common in MC embedded systems such as flight management systems and smart

Table 1
A summary of the main symbols used in Section 3.

Notation	Definition
Γ	The set consisting of N tasks
d, μ	The common deadline and period
$\tau_i \in \Gamma$	The ith task in task set Γ
χ_i	The task τ_i's active factor
c_i	The task τ_i's execution cycles in worst-cases
$\mathcal{H}, \mathcal{L}$	The high criticality level, low criticality level
$\gamma_i \in \{\mathcal{H}, \mathcal{L}\}$	The task τ_i's criticality level
Θ	The DVFS-enabled uniprocessor
(v_q, f_q)	The qth voltage/frequency pair
α, β, δ	The processor dependent non-negative constants
C, R	The thermal capacitance and resistance
$T(t), P(t)$	The temperature and power dissipation at time t
N_{TC}	The number of thermal cycles to failure
$\Delta T, \Delta T_\sigma$	The temperature cycle-range of the whole and part
λ	The permanent failure rate due to thermal cycling
λ_i	The permanent failure rate of executing task τ_i
$\mathcal{A}_q$	The fault rate on average at frequency f_q
$\mathcal{R}_{i,q}$	The task τ_i's reliability at frequency f_q
$\mathcal{C}_\gamma$	The failure probability in one hour at criticality γ
{A,B,C,D,E}	The criticality level set defined in DO-178B

car systems. Therefore, we suppose that a set of N independent tasks, denoted by $\Gamma = \{\tau_i | i = 1, 2, \ldots, N\}$, is to be executed on the uniprocessor Θ in this work. We will study the scheduling of dependent real-time tasks running on MC embedded systems in our future work. A tuple $\tau_i : \{\chi_i, c_i, \gamma_i\}$ is utilized to characterize every task $\tau_i \in \Gamma$. The parameter χ_i represents task active factor, c_i denotes the execution cycles in worst-cases, and $\gamma_i \in \{\mathcal{H}, \mathcal{L}\}$ is the criticality level. All tasks share a common deadline d that equals their common period μ.

3.2. Temperature model

We adopt a simple yet accurate lumped RC thermal model [21], as shown in Fig. 1, to characterize task thermal profiles. This practical thermal model is commonly used in the design of thermal-aware task scheduling algorithms [22,23], and it can be expressed by

$$\frac{T(t)}{R} + C\frac{dT(t)}{dt} = P(t) + \frac{T(0)}{R}. \tag{1}$$

The parameters C and R denote the thermal capacitance and resistance, respectively. $P(t)$ and $T(t)$ represent the power consumption and the temperature at time t, respectively. $T(0)$ refers to the temperature at time 0. Let $T(t_s)$ be the temperature at initial time t_s of executing task τ_i at pair (v_q, f_q) $(1 \le q \le Q)$ and $T(t_e)$ be the ending temperature at time t_e at which task τ_i has just completed its execution. Eqs. (2)–(4) describe the derivation of ending temperature $T(t_e)$ by solving (1), where α, β, and δ are non-negative constants depending on processor architecture [24].

$$\frac{dT(t)}{dt} = \frac{T(0) + \alpha R v_q + (\beta R v_q - 1)T(t) + \chi_i \delta R v_q^2 f_q}{RC} \tag{2}$$

$$\int_{t_s}^{t_e} dt = \int_{T(t_s)}^{T(t_e)} \frac{dT(t)}{\frac{T(0) + \alpha R v_q + \chi_i \delta R v_q^2 f_q}{RC} - (\frac{1 - \beta R v_q}{RC})T(t)} \tag{3}$$

$$T(t_e) = \frac{\alpha R v_q + \chi_i \delta R v_q^2 f_q + T(0)}{1 - \beta R v_q} - \left(\frac{T(0) + \alpha R v_q + \chi_i \delta R v_q^2 f_q}{1 - \beta R v_q}\right.$$
$$\left. - T(t_s)\right)e^{-(\frac{1 - \beta R v_q}{RC})(t_e - t_s)} \tag{4}$$
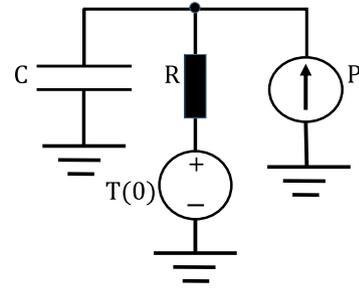
Fig. 1. The lumped RC thermal model [21].

3.3. Fault model

3.3.1. Permanent fault

We concentrate on thermal cycling induced permanent failures. Thermal cycling refers to the wear due to unmatched thermal expansion coefficients of adjacent material layers. The inelastic deformation resulting from temperature variation at runtime eventually leads to permanent failures [25]. Coffin–Manson equation is widely utilized to model the number of thermal cycles to failure, and it is given by [25]

$$N_{TC} = \varphi(\Delta T - \Delta T_\sigma)^{-\psi}. \tag{5}$$

The parameters ΔT and ΔT_σ represent the temperature cycle-range of the whole and part, respectively. The parameter φ is a non-negative material-dependent constant. The parameter ψ, commonly in the range of [1, 9], is an empirically derived Coffin–Manson exponent [25]. Both φ and ψ can be determined by fitting the Coffin-Manson equation from system lifetime measurements [25,26]. Similar to the Coffin–Manson equation settings described in [25,26], the parameters φ and ψ used in (5) are selected as 1 and 6 for on-chip structures in our experiments, respectively. Typically, due to $\Delta T_\sigma \ll \Delta T$, the item ΔT_σ can be dropped from (5), then we have

$$N_{TC} = \varphi(\Delta T)^{-\psi}. \tag{6}$$

It has been shown in [27] that the peak temperature of a task can be reached at its start/ending time instant since the temperature is monotonically increasing or decreasing. Therefore, the number of thermal cycles to failure of processor Θ due to the execution of task τ_i is expressed as

$$N_{TC,i} = \varphi(|T(t_e) - T(t_s)|)^{-\psi}. \tag{7}$$

$T(t_s)$ and $T(t_e)$ are the ending temperature and initial temperature of task τ_i executing at frequency f_q, respectively, both of which can be derived by using (1). According to Miner's rule, the permanent failure rate of running task τ_i due to thermal cycling is readily calculated as [28]

$$\lambda_i = \frac{1}{N_{TC,i}}. \tag{8}$$

3.3.2. Transient fault

Exponential distribution is usually used for modeling soft errors resulting from transient faults. The transient fault rate on average at frequency f_q is given by [29]

$$\mathcal{A}_q = \mathcal{A}_Q 10^{\hbar \frac{f_Q - f_q}{f_Q - f_1}}. \tag{9}$$

$\mathcal{A}_Q$ represents the transient fault rate on average at maximal frequency f_Q. The parameter $\hbar$ is a hardware specific factor and it is larger than 0. Typically, for every task τ_i running at f_q, its

Table 2
The Criticality Levels Specified in DO-178B [12].

γ	A	B	C	D	E
$\mathcal{C}_\gamma$	$(0, 10^{-9})$	$(0, 10^{-7})$	$(0, 10^{-5})$	$[10^{-5}, +\infty)$	$(0, +\infty)$

reliability can be calculated by utilizing the exponential failure law [29], that is,

$$\mathcal{R}_{i,q} = e^{-\mathcal{A}_q c_i / f_q}. \tag{10}$$

We utilize the task re-execution technique [10] to tolerate transient faults. Furthermore, we use this technique to tolerate no more than one transient fault due to that tolerating single-fault is a commonly adopted assumption [16,22,30,31]. Let $\varepsilon_{i,q}$ denote the total execution number of task τ_i executing at frequency f_q, which is derived by our proposed task scheduling strategy to satisfy system safety requirements. At runtime, an acceptance test [32–34] is performed for every execution of task τ_i to check whether task τ_i is successfully executed or not during the current period. Task τ_i will be executed multiple times until a successful execution is achieved. Assume that no soft errors are detected after the $\varepsilon_{i,q}^*$th ($1 \leq \varepsilon_{i,q}^* \leq \varepsilon_{i,q}$) execution, and the previous ($\varepsilon_{i,q}^* - 1$) executions fail. At that moment, the output results generated by this execution are accepted and submitted, and the ($\varepsilon_{i,q}^* + 1$)th$-\varepsilon_{i,q}$th executions of task τ_i are canceled. When soft errors are still detected after the $\varepsilon_{i,q}$th execution of task τ_i, the execution of task τ_i fails during the current period. The time overhead of conducting an acceptance test has been thoroughly investigated in [32–34]. It has been shown in [32–34] that compared to the execution time of a task, the time overhead of conducting an acceptance test is extremely small and even can be safely ignored. In this work, the execution cycles of an acceptance test for task τ_i are assumed to be c_i^*. Therefore, the worst-case execution cycles of task τ_i with consideration of performing an acceptance test are thus updated by $c_i \leftarrow c_i + c_i^*$.

3.4. Safety requirement model

We adopt the DO-178B safety standard [12] to characterize safety requirements. As shown in Table 2, a total of five criticality levels are specified in the DO-178B safety standard. For every criticality level $\gamma \in \{A,B,C,D,E\}$, it is associated with a quantitative failure probability per hour, denoted by $\mathcal{C}_\gamma$, which indicates that the failure probability on average in one hour during a given operation interval cannot be violated by the whole criticality γ tasks. As criticality γ decreases from level A to level E, failure probability $\mathcal{C}_\gamma$ increases strictly. In addition, we can also observe from Table 2 that both the level D and level E tasks have almost no safety requirements whereas the level A, level B and level C tasks have stringent safety requirements with very small (no more than 10^{-5}) average failure probability.

4. Problem definition and solution overview

In this section, we first give a definition of the studied problem and then outline the proposed solution.

4.1. Problem definition

Fig. 2 illustrates the relationship between the system lifetime and the permanent failure rate. From a probabilistic point of view, the integral of the permanent failure rate with respect to the task execution time is equal to 1 in the time interval $[0, \mathbb{T}_{SL}]$, where $\mathbb{T}_{SL}$ is the system lifetime [35]. Let ε_i be the total execution number of task τ_i with neglected operating frequency in one scheduling horizon for the sake of easy presentation. The
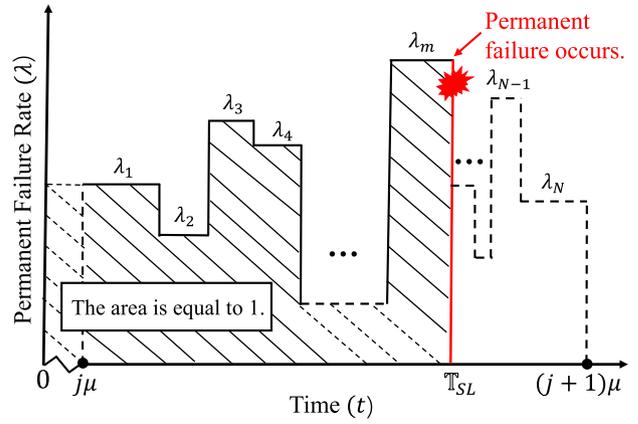

Fig. 2. An illustration of the relationship between system lifetime $\mathbb{T}_{SL}$ and permanent failure rate λ [35]. The processor permanent damage caused by the ε_i executions of task τ_i in one scheduling horizon is given by $\mathcal{D}_i = \lambda_{i,1}\mathbb{T}_i + \lambda_{i,2}\mathbb{T}_i + \cdots + \lambda_{i,\varepsilon_i}\mathbb{T}_i$, where $\lambda_{i,\varepsilon_i}$ is the permanent failure rate of ε_ith execution of task τ_i and $\mathbb{T}_i$ is time consumed by per execution of task τ_i. For simplicity, we use the notation λ_i to denote the sum of $\lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,\varepsilon_i}$, i.e., $\lambda_i = \lambda_{i,1} + \lambda_{i,2} +, \ldots, + \lambda_{i,\varepsilon_i}$. Therefore, the expression $\mathcal{D}_i = \lambda_i\mathbb{T}_i$ holds. The processor permanent damage of each scheduling horizon except for the last one occurring permanent failure is easily calculated as $D = \sum_{i=1}^{N} \lambda_i\mathbb{T}_i$, where N is the number of tasks.

parameter λ_i represents the sum of permanent failure rates of ε_i executions of task τ_i, and $\mathbb{T}_i$ denotes the time consumed by per execution of task τ_i. Therefore, assuming that a permanent failure occurs after the execution of the mth task during the $(j + 1)$th period $[j\mu, (j + 1)\mu]$, we can obtain the following equation about the system lifetime $\mathbb{T}_{SL}$

$$\sum_{i=1}^{N} j\lambda_i\mathbb{T}_i + \sum_{i=1}^{m-1} \lambda_i\mathbb{T}_i + \lambda_m(\mathbb{T}_{SL} - \sum_{i=1}^{m-1} \mathbb{T}_i - j\mu) = 1. \tag{11}$$

By rewriting (11), $\mathbb{T}_{SL}$ is thus expressed as

$$\mathbb{T}_{SL} = \frac{1 - \sum_{i=1}^{m-1} \lambda_i\mathbb{T}_i - \sum_{i=1}^{N} j\lambda_i\mathbb{T}_i}{\lambda_m} + \sum_{i=1}^{m-1} \mathbb{T}_i + j\mu. \tag{12}$$

Formally, the studied problem can be defined below. Given a task set Γ with dual-criticality levels and the uniprocessor Θ supporting DVFS, for each task in task set Γ, determine its (1) operating frequency, (2) execution number, and (3) execution order such that the system lifetime $\mathbb{T}_{SL}$ is maximized while both the safety requirements and the real-time constraints are satisfied.

4.2. Solution overview

The concentration of this paper is to optimize the lifetime of dual-criticality systems while satisfying the constraints of safety requirements and task schedulability. To this end, we first present an MILP-based method for the problem of system lifetime maximization in Section 5. Using existing MILP solvers such as the one developed in [36], the presented MILP-based method can optimally solve the scheduling problem for task set with small number of tasks. However, the MILP solvers do not have the ability to efficiently address task scheduling problems for large-scale systems. Therefore, an effective heuristic is needed to achieve a better tradeoff between the system lifetime attained by the derived task schedule and the runtime consumed to generate the task schedule. Given this, we then propose a time-efficient CEM-based heuristic in Section 6 to overcome the shortcoming of the MILP-based method in terms of runtime.

Table 3
A summary of the main symbols used in Section 5.

Notation	Definition
$\Lambda_{i,q}$	The variable indicating whether τ_i is executed at f_q
ϖ_i	The task τ_i's start execution time
$\varepsilon_{i,q}$	The task τ_i's execution number at f_q
$\lambda_{i,k,q}$	The permanent failure rate of kth execution on f_q
$\mathcal{D}$	The permanent damage of each scheduling horizon
$\mathfrak{R}$	The task maximal execution number
$\mathcal{J}$	A constant number large enough
$\Gamma_{\mathcal{H}}, \Gamma_{\mathcal{L}}$	The set of $\mathcal{H}$ or $\mathcal{L}$ tasks
$\mathcal{C}_{\mathcal{H}}, \mathcal{C}_{\mathcal{L}}$	The safety requirement threshold of $\mathcal{H}$ or $\mathcal{L}$ tasks
$\mathcal{C}(\Gamma_{\mathcal{H}}), \mathcal{C}(\Gamma_{\mathcal{L}})$	The failure probability achieved by $\mathcal{H}$ or $\mathcal{L}$ tasks
$\varpi_{(i,\epsilon)}, \varpi_{(i,\epsilon)}^*$	The smaller or larger of ϖ_i and ϖ_ϵ

5. MILP formulation

This section presents an MILP-based method for dealing with the problem of lifetime maximization under the safety requirements and the task schedulability constraint for fault-tolerant MC embedded systems. Table 3 summarizes the main symbols used in this section. For ease of presentation, the following variables are defined.

$$\Lambda_{i,q} = \begin{cases} 1 & \text{if task } \tau_i \text{ is executed at frequency } f_q \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

$$\varpi_i : \text{the start execution time of task } \tau_i \quad (14)$$

$$\varepsilon_{i,q} : \text{the total execution number of } \tau_i \text{ running at } f_q \quad (15)$$

5.1. Objective

The objective is to maximize the lifetime of fault-tolerant MC embedded systems, which is equivalent to minimize the system permanent damage for every scheduling horizon. The permanent damage caused by task execution in each scheduling horizon is calculated as

$$\mathcal{D} = \sum_{i=1}^{N} \sum_{k=1}^{\varepsilon_{i,q}} \sum_{q=1}^{Q} \lambda_{i,k,q} \Lambda_{i,q} \frac{c_i}{f_q}. \quad (16)$$

The parameter $\lambda_{i,k,q}$ is the permanent failure rate of kth execution on frequency f_q for task τ_i, and it can be derived by using (8). Therefore, the objective function is given by

$$\text{minimize: } \mathcal{D}. \quad (17)$$

5.2. Constraints

The following design constraints should be satisfied.

- *For every task, it has to be exactly executed at only one frequency level.*

$$\sum_{q=1}^{Q} \Lambda_{i,q} = 1, \forall i \in [1, 2, \dots, N]. \quad (18)$$

- *For every task, its execution number at any frequency level cannot exceed the maximal execution number $\mathfrak{R}$ specified by the MC embedded systems.*

$$1 \le \varepsilon_{i,q} \le \mathfrak{R}, \forall i \in [1, 2, \dots, N], q \in [1, 2, \dots, Q]. \quad (19)$$

- *For every task, its finish time cannot exceed the common deadline.*

$$\varpi_i + \sum_{q=1}^{Q} \Lambda_{i,q} \frac{\varepsilon_{i,q} \times c_i}{f_q} \le d, \forall i \in [1, 2, \dots, N]. \quad (20)$$

- *The safety requirements are satisfied.* Let $\Gamma_{\mathcal{H}}$ and $\Gamma_{\mathcal{L}}$ denote the $\mathcal{H}$ task set and the $\mathcal{L}$ task set, respectively. Let $\mathcal{C}_{\mathcal{H}}$ denote the safety requirement threshold of $\mathcal{H}$ tasks, and $\mathcal{C}_{\mathcal{L}}$ represent the safety requirement threshold of $\mathcal{L}$ tasks. Both $\mathcal{C}_{\mathcal{H}}$ and $\mathcal{C}_{\mathcal{L}}$ can be found in Table 2. Let $\mathcal{C}(\Gamma_{\mathcal{H}})$ and $\mathcal{C}(\Gamma_{\mathcal{L}})$ represent the failure probability achieved by $\mathcal{H}$ tasks and the failure probability achieved by $\mathcal{L}$ tasks, respectively. Then, for every scheduling horizon μ (unit: hour), the constraints of safety requirements are summarized below.

$$\mathcal{C}(\Gamma_{\mathcal{H}}) = \sum_{\tau_i \in \Gamma_{\mathcal{H}}} \sum_{q=1}^{Q} \Lambda_{i,q} (1 - \mathcal{R}_{i,q})^{\varepsilon_{i,q}}. \quad (21)$$

$$\mathcal{C}(\Gamma_{\mathcal{L}}) = \sum_{\tau_i \in \Gamma_{\mathcal{L}}} \sum_{q=1}^{Q} \Lambda_{i,q} (1 - \mathcal{R}_{i,q})^{\varepsilon_{i,q}}. \quad (22)$$

$$\mathcal{C}(\Gamma_{\mathcal{H}}) \begin{cases} < \mathcal{C}_{\mathcal{H}} \times \mu & \text{if } \mathcal{H} \in \{A, B, C\} \\ \ge \mathcal{C}_{\mathcal{H}} \times \mu & \text{otherwise.} \end{cases} \quad (23)$$

$$\mathcal{C}(\Gamma_{\mathcal{L}}) \begin{cases} < \mathcal{C}_{\mathcal{L}} \times \mu & \text{if } \mathcal{L} \in \{A, B, C\} \\ \ge \mathcal{C}_{\mathcal{L}} \times \mu & \text{otherwise.} \end{cases} \quad (24)$$

- *For any two tasks on the processor, the overlapping executions are not permitted.* Let the variable ϖ_i and the variable ϖ_ϵ represent the start execution times of two tasks τ_i and τ_ϵ ($1 \le i, \epsilon \le N, i \ne \epsilon$), respectively. Let $\varpi_{(i,\epsilon)}$ and $\varpi_{(i,\epsilon)}^*$ denote the smaller and the larger of start execution time ϖ_i and start execution time ϖ_ϵ, respectively. That is, the two equations $\varpi_{(i,\epsilon)} = \min(\varpi_i, \varpi_\epsilon)$ and $\varpi_{(i,\epsilon)}^* = \max(\varpi_i, \varpi_\epsilon)$ hold. An auxiliary variable $p_{i,\epsilon}$ is introduced to indicate the value of $\varpi_{(i,\epsilon)}$. In the case where $\varpi_i < \varpi_\epsilon$ holds, the variable $p_{i,\epsilon}$ is set to 1; in the case where $\varpi_i > \varpi_\epsilon$ holds, the variable $p_{i,\epsilon}$ is set to 0. $\mathcal{J}$ is a constant large enough and it takes the value of 10000 in the experimental part. The $w_{i,\epsilon}$ and $o_{i,\epsilon}$ are also two auxiliary variables utilized for facilitating the formulation. The constraints of avoiding task overlapping executions are listed below.

$$\varpi_{(i,\epsilon)} \le \varpi_i \quad (25)$$

$$\varpi_{(i,\epsilon)} \le \varpi_\epsilon \quad (26)$$

$$\varpi_{(i,\epsilon)} \ge \varpi_i - \mathcal{J} \times (1 - p_{i,\epsilon}) \quad (27)$$

$$\varpi_{(i,\epsilon)} \ge \varpi_\epsilon - \mathcal{J} \times p_{i,\epsilon} \quad (28)$$

$$p_{i,\epsilon} = 1, 0 \quad (29)$$

$$\varpi_{(i,\epsilon)}^* = \varpi_i + \varpi_\epsilon - \varpi_{(i,\epsilon)} \quad (30)$$

$$w_{i,\epsilon} = \mathcal{J} \times (\varpi_i - \varpi_{(i,\epsilon)}^*) + \varpi_\epsilon \quad (31)$$

$$\varpi_i - w_{i,\epsilon} \ge \sum_{q=1}^{Q} \Lambda_{\epsilon,q} \frac{\varepsilon_{\epsilon,q} \times c_\epsilon}{f_q} \quad (32)$$

$$o_{i,\epsilon} = \mathcal{J} \times (\varpi_\epsilon - \varpi_{(i,\epsilon)}^*) + \varpi_i \quad (33)$$

$$\varpi_\epsilon - o_{i,\epsilon} \ge \sum_{q=1}^{Q} \Lambda_{i,q} \frac{\varepsilon_{i,q} \times c_i}{f_q} \quad (34)$$

Algorithm 1: MILP-Based Method for System Lifetime Maximization

---

Input: 1) Task set Γ, 2) Processor Θ;
Output: Schedule table Ω;
1 Produce schedule table Ω by using the MILP solver developed in [36] to solve the formulated MILP;
2 **Return** Schedule table Ω.

Table 4
A summary of the main symbols utilized in Section 6.

Notation	Definition
ζ^*	The target/optimal variable value
$\mathcal{Z}(x)$	The objective function of the optimization problem
$\mho$	The state space of the optimization problem
$\mathcal{P}_k(x)$	The kth element in probability class $\mathcal{P}_K$
$\mathcal{W}_k$	The expected value corresponding to $\mathcal{P}_k(x)$
$\mathcal{V}$	The threshold or level parameter
$\mathcal{V}^*$	The value of $\mathcal{Z}(x)$ when $x = \zeta^*$
G	The number of candidate samples
$\mathcal{U}_{(\cdot)}$	The indicator function
k	The iteration counter
$\mathcal{V}_k$	The threshold at kth iteration
K	The maximum iteration number
$\mathcal{P}_0$	The initial probability vector
$\mathcal{P}_k$	The probability vector at kth iteration
η	The number of samples with best performance
ϑ	The index set of η best samples
$x_{\rho,i}$	The ith element in sample $\mathcal{X}_\rho$
$\mathcal{P}_{k,i,\varrho}$	The probability of mapping $x_{\rho,i}$ to ϱ at kth iteration
$\mathcal{P}_k^f$	The probability vector of task execution frequency
$\mathcal{P}_k^\varepsilon$	The probability vector of task re-execution number
$\mathcal{P}_k^{eo}$	The probability vector of task execution order

5.3. Algorithm of MILP-based method

Algorithm 1 describes the pseudo-code of MILP-based lifetime optimization method. Inputs to Algorithm 1 are the DVFS-enabled uniprocessor and the task set. By solving the formulated MILP presented in this section, we can obtain an optimal task schedule that maximizes system lifetime under the constraints of safety requirements and task schedulability.

6. CEM-Based heuristic

The MILP-based method generates a lifetime-optimum task schedule with consideration of all design constraints. However, this method cannot efficiently tackle the studied scheduling problem as the size of the problem increases. Given this, we develop a time-efficient CEM-based heuristic to overcome the shortcoming of MILP-based method in terms of runtime. This section first outlines the CEM's theoretic foundation, and then describes in detail the developed CEM-based heuristic. Table 4 summarizes the main symbols used in this section.

6.1. Theoretical foundation

Unlike most of the traditional optimization techniques (e.g., genetic algorithm [37] and particle swarm optimization [38]), CEM has a well-established theoretical foundation for tackling combinatorial or continuous optimization problems. For a deterministic optimization problem to be solved, the CEM first converts it into an associated stochastic optimization problem, and then addresses this converted problem by utilizing an iterative sampling scheme [39,40]. In every round of iteration, a plurality of random samples are produced, and the generated random samples converge in a probabilistic manner to the optimal solution to the original problem. For a detailed introduction to the CEM, we recommend readers to refer to literatures [39,40].

Considering a general combinatorial optimization problem of finding an optimal mapping so that function $\mathcal{Z}(x)$ takes the minimum value when variable $x \in \mho$ is identical to ζ^*, that is,

$$\mathcal{Z}(\zeta^*) = \mathcal{V}^* = \min_{x \in \mho} \mathcal{Z}(x). \tag{35}$$

The above deterministic optimization problem can be converted into the below probability estimation problem [39]

$$\xi(\mathcal{V}) = \mathcal{P}_k(\mathcal{Z}(\mathcal{X}) \leq \mathcal{V}) = \mathcal{W}_k(\mathcal{U}_{\{\mathcal{Z}(\mathcal{X}) \leq \mathcal{V}\}}) \tag{36}$$

The probability $\mathcal{P}_k(x)$ is an element of probability class $\mathcal{P}_K = \{\mathcal{P}_k(x) | k = 1, 2, \ldots, K\}$. $\mathcal{X}$ denotes the sample set containing G random samples $\{\mathcal{X}_\rho | \rho = 1, 2, \ldots, G\}$ that are generated by probability $\mathcal{P}_k(x)$. The parameter $\mathcal{V}$ is a threshold, $\mathcal{P}_k(\mathcal{Z}(\mathcal{X}) \leq \mathcal{V})$ is the probability that $\mathcal{Z}(\mathcal{X})$ is no more than threshold $\mathcal{V}$, and $\mathcal{W}_k(\mathcal{U}_{\{\mathcal{Z}(\mathcal{X}) \leq \mathcal{V}\}})$ represents the expected value of $\mathcal{U}_{\{\mathcal{Z}(\mathcal{X}) \leq \mathcal{V}\}}$. The $\mathcal{U}_{\{\mathcal{Z}(\mathcal{X}) \leq \mathcal{V}\}}$ refers to the indicator function, and it takes the value of 1 when the condition $\mathcal{Z}(x) \leq \mathcal{V}$ holds; otherwise it takes the value of 0. For the probability estimation problem defined in (36), the CEM attempts at finding the maximum $\mathcal{V}_{max}$ that makes $\xi(\mathcal{V}_{max})$ close to 0. Obviously, the probability that $\mathcal{Z}(\mathcal{X})$ is greater than $\mathcal{V}_{max}$ is close to 1 at that moment, which indicates that $\mathcal{V}_{max}$ is the maximum lower bound on $\mathcal{Z}(x)$ for $\forall x \in \mho$ and it is therefore the desired solution to the original problem defined in (35). The following steps describe the main process of the CEM for solving the converted problem defined in (36).

  i. Initialize counter $k \leftarrow 1$ and probability vector $\mathcal{P}_0$;
 ii. Produce G random samples $\mathcal{X} = \{\mathcal{X}_\rho | \rho = 1, 2, \ldots, G\}$ using probability $\mathcal{P}_{k-1}$;
iii. Calculate the performance $\mathcal{Z}(\mathcal{X}) = \{\mathcal{Z}(\mathcal{X}_\rho), | \rho = 1, 2, \ldots, G\}$ achieved by G random samples;
 iv. Choose η best random samples from G random samples according to sample performance $\mathcal{Z}(\mathcal{X})$, and derive threshold $\mathcal{V}_k$ using:

$$\mathcal{V}_k \leftarrow \frac{1}{\eta} \sum_{\iota \in \vartheta} \mathcal{Z}(\mathcal{X}_\iota), \tag{37}$$

 where ϑ denotes the index set of best random samples.
  v. Derive probability vector $\mathcal{P}_k$ of kth round of iteration

$$\mathcal{P}_{k,i,\varrho} = \frac{\sum_{\rho=1}^{G} \mathcal{U}_{\{\mathcal{Z}(\mathcal{X}_\rho) \leq \mathcal{V}_k\}} \mathcal{U}_{\{x_{\rho,i} = \varrho\}}}{\sum_{\rho=1}^{G} \mathcal{U}_{\{\mathcal{Z}(\mathcal{X}_\rho) \leq \mathcal{V}_k\}}}. \tag{38}$$

 The parameter $x_{\rho,i}$ represents the ith element of random sample $\mathcal{X}_\rho$. The parameter $\mathcal{P}_{k,i,k}$ denotes the probability of mapping $x_{\rho,i}$ to ϱ (i.e., $x_{\rho,i} = \varrho$).
 vi. Exit if the predefined stopping criteria are satisfied; else, update $k \leftarrow k + 1$ and return to step ii for next round of iteration.

6.2. Algorithm of CEM-based heuristic

Algorithm 2 describes the pseudo-code of our CEM-based heuristic. The presented heuristic is composed of an initialization stage and an iteration stage. At the initialization stage, line 1 and line 2 first initialize the iteration counter and maximum number of iterations, respectively. Lines 3–9 then initialize three probability vectors: the probability vector of task execution frequency, the probability vector of task re-execution number, and the probability vector of task execution order. At the iteration stage, lines 10–21 iteratively search the sample with optimal system lifetime. To be specific, a total of M samples are first generated according to the current three probability vectors, and G candidate samples are then picked by using the acceptance–rejection technique (lines 11–12). In the cases of finding no candidate samples, lines 13–17 terminate this round of iteration. The calculation of the performance of every candidate sample is performed by line 18. According to the calculation results, the three probability vectors utilized for next round of iteration are hence derived by lines 19–21. Based on the last round of iteration, the lifetime-optimum task schedule is output in line 22.

Algorithm 2: The CEM-Based Heuristic CEMH

Input: 1) Task set Γ, 2) Processor Θ;
Output: Schedule table Ω;

/* Initialization stage */

1 Set iteration counter $k = 1$;
2 Initialize maximum number K of iterations;
3 **for** *each task $\tau_i \in \Gamma$* **do**
4   **for** *each frequency $f_q \in \Psi$* **do**
5   Initialize probability vector $\mathcal{P}_k^f$ of task execution
     frequency: set the probability $\mathcal{P}_{k,i}^{f_q} \in \mathcal{P}_k^f$ of task τ_i
     executed at f_q to $1/Q$;
6   **for** $n = 1 : 1 : \Re$ **do**
7   Initialize probability vector $\mathcal{P}_k^\varepsilon$ of task re-execution
       number: set the probability $\mathcal{P}_{k,i}^{q,n} \in \mathcal{P}_k^\varepsilon$ of n executions
       at f_q for task τ_i to $1/\Re$;
8   **for** $j = 1 : 1 : N$ **do**
9   Initialize probability vector $\mathcal{P}_k^{eo}$ of task execution
         order: set the probability $\mathcal{P}_{k,i}^{q,n,j} \in \mathcal{P}_k^{eo}$ of τ_i
         performed in the jth order to $1/N$;

/* Iteration stage */

10 **while** $k \leq K$ **do**
11   Produce M samples based on $\{\mathcal{P}_k^f, \mathcal{P}_k^\varepsilon, \mathcal{P}_k^{eo}\}$ using importance sampling technique;
12   Choose G candidate samples satisfying (20)–(24) using acceptance–rejection method;
13   **if** $G == 0$ **then**
14   **if** $k == 1$ **then**
15   **Return** false.
16   **else**
17   **break**;
18   Derive the permanent damage achieved by every candidate sample using (16);
19 $k \leftarrow k + 1$;
20   Calculate threshold $\mathcal{V}_{k,1}$ for $\mathcal{P}_k^f$, $\mathcal{V}_{k,2}$ for $\mathcal{P}_k^\varepsilon$, and $\mathcal{V}_{k,3}$ for $\mathcal{P}_k^{eo}$ using (37);
21   Obtain $\{\mathcal{P}_k^f, \mathcal{P}_k^\varepsilon, \mathcal{P}_k^{eo}\}$ using (38);
22 **Return** Schedule table Ω corresponding to the optimal sample generated at the last iteration.

6.3. Time complexity of CEM-based heuristic

The time complexity $O(\cdot)$ of our proposed CEM-based heuristic CEMH is analyzed as follows. The initialization stage in lines 1–9 can be conducted in $O(\Re Q N^2)$ time, where N is the task number, $\Re$ is the task maximal execution number specified by the MC embedded systems, and Q is the total number of processor operating frequencies. The iteration stage in lines 10–21 can be performed in $O(KM\Re Q N^2)$ time, where K denotes the maximum iteration number and M represents the number of samples in each iteration of the CEM-based heuristic CEMH. Therefore, the overall time complexity of our CEM-based heuristic CEMH is $O(KM\Re Q N^2)$. In addition to theoretically analyzing the time complexity, we also compare the measured runtime of various task scheduling algorithms for validating the effectiveness of our CEM-based heuristic CEMH. The comparison results will be demonstrated later in the next experimental part (i.e., Section 7).

Table 5
The parameters of five real-world benchmarks [3,12,20].

Benchmark	Task number	$\mathcal{H}/\mathcal{L}$	$\mathcal{H}$ number	$\mathcal{L}$ number
1	70	A/C	40	30
2	50	A/B	15	35
3	65	C/D	20	45
4	55	B/C	20	35
5	80	A/B	40	40

Table 6
The default parameter settings of tool HotSpot [41].

Parameter	Symbol	Value	Unit
Processor area	–	4.56×4.56	mm^2
Die thickness	–	0.20	mm
Heat spreader side	–	25	mm
Heat sink side	–	66	mm
Thermal resistance	R	1.83	°C/W
Thermal capacitance	C	112.2	mJ/°C
Initial temperature	$T(0)$	45	°C

7. Numerical results

7.1. Simulation setup

Two sets of simulations are carried out: one is conducted on synthetic benchmarks, and the other is conducted on real-world benchmarks. In the first set of simulation experiments, a random task generator implemented in C++ is used to produce multiple synthetic benchmarks (i.e., task sets). For each task in a synthetic benchmark, its worst-case execution cycles incorporating the CPU cycles of an acceptance test are randomly selected from the interval of $[4 \times 10^7, 6 \times 10^8]$, which are generated on the embedded benchmark suite MiBench [27]. As detailed in [23], the active factor of a task captures how intensively functional units of a processor are being utilized by the task, and it is typically in the range of [0.4, 1]. Adopting the same setting in [23], we randomly choose the active factor of a synthetic task from the interval of [0.4, 1].

The task number N of each synthetic benchmark is in the range between 100 and 200, in steps of 10. The common deadline d of N synthetic tasks is deemed to $1.5 \times \sum_{\tau_i \in \Gamma} c_i / f_Q$ [22]. Two criticality levels A and C defined in Table 2 are selected as the system safety requirements. The criticality level of each synthetic task is generated in a random way. In the second set of simulation experiments, five real-world benchmarks, borrowed from [3,12,20], of flight management systems are tested, and their parameters including the total task number, task criticality levels, and the number of high criticality or low criticality tasks, are shown in Table 5. Note that the original flight management system only consists of 26 real-time tasks with two criticality levels B and C. To investigate system schedulability with respect to task number and criticality levels, we make the following two changes to the original tasks. First, we randomly select some original tasks to copy these tasks once. Each copy task is regarded as a new task, but has the same criticality level and execution cycles as the original task. Second, the two criticality levels B and C can be augmented or degraded. For example, criticality levels B and C are augmented to A and B for benchmark 2, and degraded to C and D for benchmark 3. The maximum iteration number K of the proposed heuristic and the maximum execution number $\Re$ of every task are set to 10 [42] and 3 [12], respectively.

We utilize the Alpha 21264 processor as the hardware platform since it is the only publicly available processor integrating complete power and thermal models [43,44]. The voltage (V)/frequency (GHz) pairs supported by the processor are set to (0.7, 1.0), (0.8, 1.25), (0.9, 1.5), (1.0, 1.75), (1.1, 2.0), and (1.2,
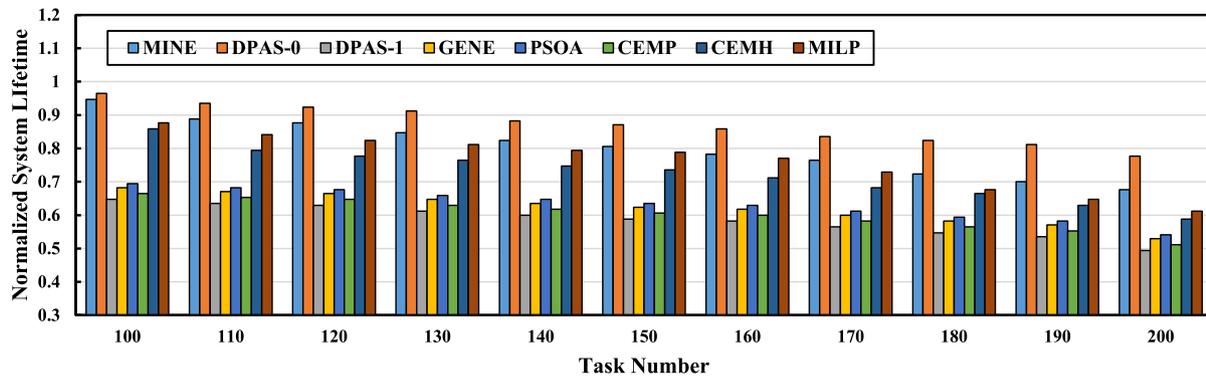
Fig. 3. The normalized system lifetime achieved by various algorithms when executing synthetic benchmarks.
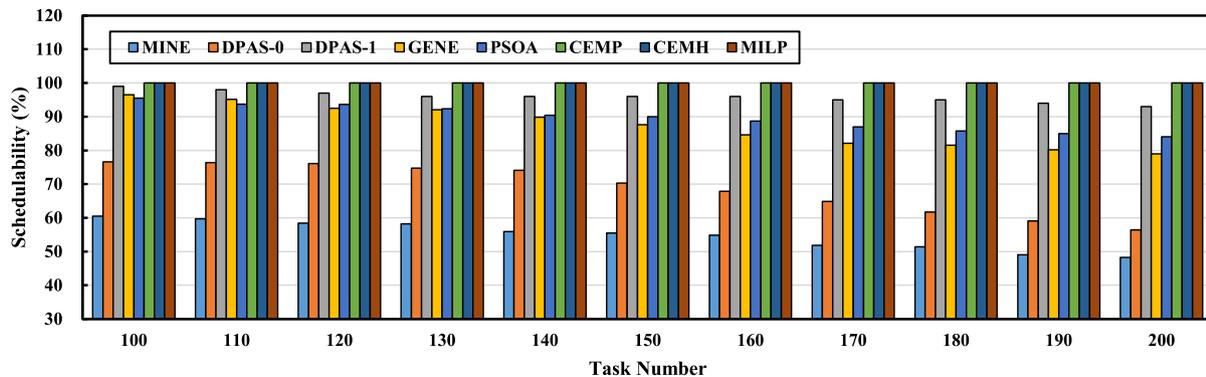

Fig. 4. The schedulability achieved by various algorithms when executing synthetic benchmarks.

2.25), as in [43]. We use the commonly adopted simulation tool HotSpot [41] to capture task thermal profiles. The parameters used in our simulations are listed in Table 6, all of which are the default values of HotSpot. As demonstrated in [32], the fault arrival rate $\mathcal{A}_Q$ and the parameter $\hbar$ are typical in the intervals of $[10^{-8}, 10^{-7}]$ and $[1,10]$, respectively. Therefore, we set the fault arrival rate $\mathcal{A}_Q$ and the parameter $\hbar$ to 10^{-8} and 1, respectively. In both sets of simulations, we compare the system lifetime, schedule feasibility, and measured runtime of our proposed heuristic CEMH with that of MILP, MINE, DPAS, GENE, PSOA, and CEMP. The benchmarking scheme MILP is the MILP-based method for system lifetime maximization, as demonstrated in Algorithm 1. Other mentioned benchmarking algorithms for comparison are described below.

- **MINE** [45] is an MILP based framework to determine the scheduling of real-time tasks with a same criticality level such that the whole energy dissipation can be minimized while the system lifetime constraint and the temperature limit constraint are simultaneously satisfied. The developed framework fails to take into account both transient faults and the heterogeneity of tasks in terms of criticality levels.
- **DPAS** [46] is an algorithm to maximize system lifetime through analyzing the influence of operating temperature on thermal cycling and system lifetime. Its main idea is to minimize the thermal cycling wear by wisely controlling processor operating frequencies. Transient faults and the heterogeneity of tasks in terms of criticality levels are also neglected in this strategy. We utilize the notation **DPAS-0** to denote the original algorithm without tolerating transient faults and the notation **DPAS-1** to represent the improved algorithm using task re-execution technique introduced in Section 3.3.2 to tolerate transient faults. For method **DPAS-1**, it first sets the execution number of every task τ_i to the

task maximal execution number $\Re$ and then runs method **DPAS-0** to genette a task schedule.
- **GENE** is an approach that adopts the well-known genetic algorithm [37] to deal with the system lifetime problem defined in Section 4.1. A detailed introduction to genetic algorithm can be found in [37]. Task re-execution technique is used to tolerate transient faults as our proposed algorithm CEMH. We leverage the available GENE toolbox [47] to call this algorithm for achieving system lifetime maximization.
- **PSOA** is a method that utilizes the traditional particle swarm optimization algorithm [38] to tackle the lifetime optimization problem defined in Section 4.1. A detailed introduction to particle swarm optimization algorithm is given in [38]. Task re-execution technique is adopted to tolerate transient faults as our proposed heuristic CEMH. We utilize the open-source PSOA toolbox [48] to call this algorithm for solving the problem of system lifetime maximization.
- **CEMP**, similar to our proposed heuristic CEMH, is also a CEM-based method to tackle the studied lifetime optimization problem defined in Section 4.1. The only difference between CEMP and CEMH is that the former adopts task replication technique [22] to tolerate transient faults while the latter uses task re-execution technique to tolerate transient faults. The task maximal replica number in CEMP is set to 3, which is equal to the task maximal execution number adopted in our proposed heuristic CEMH. For fault-tolerance algorithms based on task replication technique, they should execute all replicas of any task at runtime, even though the output of first replica is completely correct [22].

The above algorithms are all implemented in C++, and both sets of simulation experiments are carried out on a desktop computer equipped with 32GB memory and Intel i7 Eight-Core 4.9 GHz processor. For each synthetic or real-world benchmark under

Table 7
The normalized runtime consumed to derive the schedule of synthetic benchmarks.

Task number	MINE	DPAS-0	DPAS-1	GENE	PSOA	CEMP	CEMH	MILP	*Sp*
100	5.87	1.00	1.10	1.74	2.29	1.33	1.33	7.21	5.44
110	6.31	1.04	1.16	1.90	2.44	1.41	1.41	7.53	5.32
120	6.49	1.05	1.30	2.03	2.50	1.56	1.56	8.18	5.25
130	6.95	1.07	1.36	2.16	2.64	1.64	1.64	8.68	5.31
140	7.04	1.16	1.54	2.24	2.84	1.89	1.89	9.47	5.01
150	7.61	1.28	1.60	2.58	3.28	1.96	1.96	10.51	5.37
160	7.91	1.33	1.69	2.62	3.40	2.03	2.03	11.92	5.87
170	8.79	1.42	1.88	2.83	4.06	2.28	2.28	13.14	5.77
180	9.98	1.52	1.95	3.04	4.64	2.49	2.49	14.22	5.70
190	10.02	1.59	2.00	3.24	5.03	2.60	2.60	14.97	5.76
200	11.55	1.63	2.12	3.56	5.93	2.90	2.90	15.72	5.41
Avg.	8.05	1.28	1.54	2.54	3.55	1.84	1.84	11.05	5.99

Table 8
The normalized runtime consumed to derive the schedule of real-world benchmarks.

Benchmark	MINE	DPAS-0	DPAS-1	GENE	PSOA	CEMP	CEMH	MILP	*Sp*
1	5.52	1.00	1.16	1.81	2.34	1.40	1.40	6.17	4.41
2	8.50	1.09	1.27	2.17	2.77	1.46	1.46	9.42	6.47
3	9.57	1.13	1.38	2.33	3.11	1.66	1.66	11.67	7.02
4	11.62	1.23	1.67	2.43	3.39	1.77	1.77	13.52	7.64
5	13.18	1.28	1.75	2.65	4.19	1.80	1.80	15.91	8.85
Avg.	9.68	1.14	1.44	2.28	3.16	1.62	1.62	11.34	6.88

test, we produce 1000 benchmark instances for obtaining the simulation data of schedule feasibility and the average simulation data of system lifetime.

7.2. Results for synthetic benchmarks

In the comparative study, the system lifetime is normalized to the range of (0, 1]. Fig. 3 plots the system lifetime achieved by our proposed solution and benchmarking schemes when executing synthetic benchmarks. The results clearly demonstrate that our heuristic CEMH achieves impressive system lifetime improvement. For example, when the task number is set to 100, the system lifetime achieved by CEMH is 32.73%, 25.86%, 23.73%, and 29.20% higher than that of DPAS-1, GENE, PSOA, and CEMP, respectively. Meanwhile, the results shown in the figure also demonstrate that the system lifetime of the proposed heuristic CEMH is second to that of the schemes MINE, DPAS-0, and MILP. Moreover, DPAS-0 is superior to the proposed two algorithms MILP and CEMH in terms of system lifetime. This is because both DAPS-0 and MINE adopt no techniques to tolerate transient faults, and providing transient fault tolerance is antagonistic to the objective of lifetime maximization. Specifically, as indicated in (16) and illustrated in Fig. 2, the permanent damage of processor Θ in each scheduling horizon is calculated as the sum of permanent damages caused by the executions (including re-executions) of all tasks. Once a task triggers re-execution, it certainly incurs the permanent damage that is calculated as the product of task execution time and corresponding permanent failure rate, as both the task execution time and the corresponding permanent failure rate are larger than zero (see (7) and (20)).

Table 7 demonstrates the measured runtime of different schemes when deriving the schedule table of synthetic tasks. The metric *Sp* indicates the speedup in terms of measured runtime attained by the developed heuristic CEMH as compared with the algorithm MILP. The results show that our heuristic CEMH achieves a significant reduction in the runtime consumed to derive task schedule. For example, when the task number is set to 100, CEMH achieves 5.44 times of speedup compared to MILP.
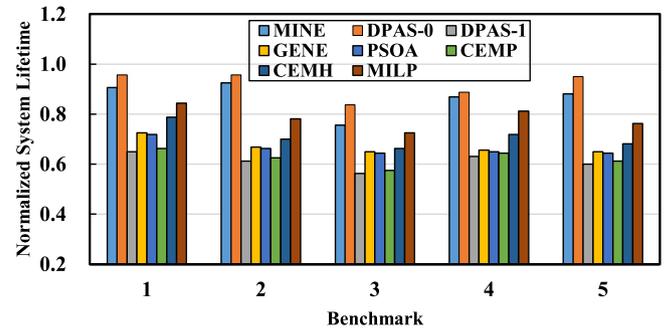

Fig. 5. The normalized system lifetime achieved by various algorithms when executing real-world benchmarks.
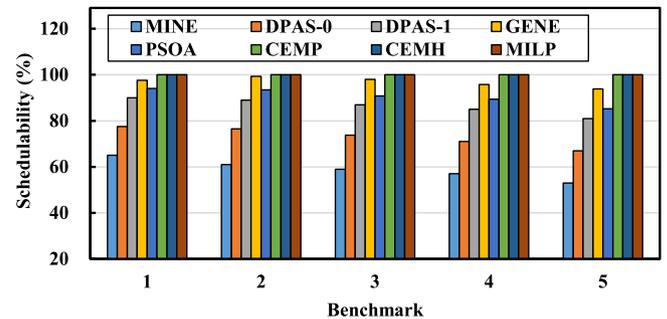

Fig. 6. The schedulability achieved by our proposed solution and benchmarking algorithms for real-world benchmarks.

Fig. 4 exhibits the schedulability achieved by our proposed solution and benchmarking schemes for synthetic benchmarks. For a synthetic benchmark with fixed task number, its schedulability is given by the number of benchmark instances that can be feasibly scheduled under task safety and real-time constraints to the number of tested benchmark instances in total (i.e., 1000 in our experiments). As demonstrated in this figure, the proposed heuristic CEMH achieves 100% schedulability regardless of the number of tasks. On the contrary, the benchmarking schemes MINE, DPAS-0, DPAS-1, GENE, and PSOA cannot guarantee that synthetic tasks can be feasibly scheduled. From the results illustrated in Figs. 3 and 4, and Table 7 we can observe that our proposed CEM-based heuristic achieves a better tradeoff between the system lifetime attained by the derived task schedule and the runtime consumed to generate the task schedule while guaranteeing task safety and real-time constraints.

7.3. Results for real-life benchmarks

Fig. 5 depicts the normalized system lifetime of various scheduling algorithms when executing five real-world benchmarks. Similar to the results presented in Fig. 3, we can clearly see from Fig. 5 that the system lifetime achieved by our proposed algorithm MILP or CEMH is better than that of benchmarking algorithms DPAS-1, GENE, PSOA, and CEMP for comparison. To be specific, the proposed heuristic CEMH can achieve up to 21.15%, 14.29%, 17.78%, 13.86%, and 13.54% system lifetime improvement for benchmarks 1–5, respectively.

The runtime consumed to derive the schedule of real-world tasks using different schemes is listed in Table 8. The table evidently demonstrates that our heuristic CEMH can significantly reduce the scheduling overhead as compared to scheme MILP. To be specific, CEMH achieves 4.41, 6.47, 7.02, 7.64, and 8.85 times of speedup for benchmarks 1–5 as compared to scheme MILP,

respectively. The schedulability achieved by our proposed two algorithms MILP and CENH when executing five real-world benchmarks is presented in Fig. 6. The results illustrate that the proposed schemes MILP and CEMH can always achieve 100% schedulability while other task scheduling algorithms MINE, DPAS-0, DPAS-1, GENE, and PSOA cannot guarantee 100% schedulability. This observation is consistent with the experimental results demonstrated in Fig. 4 and it again validates the effectiveness of the two proposed algorithms MILP and CEMH in terms of schedulability.

8. Conclusion

In this paper, we tackle the lifetime optimization problem for MC embedded systems via designing effective task scheduling schemes. We consider both transient faults and thermal cycling incurred permanent faults. In addition to an MILP-based algorithm, we also present a time-efficient CEM-based heuristic to achieve system lifetime maximization. Task re-execution technique is utilized in the two algorithms to tolerate transient faults. Two sets of simulation experiments based on synthetic task sets produced by a random task generator and real-world benchmarks of flight management systems are conducted to demonstrate the performance of our developed algorithms and benchmarking approaches in terms of system lifetime, schedule feasibility, and measured runtime. The simulation results reveal that our proposed algorithms can drastically prolong system lifetime while meeting the safety requirements and task timeliness constraints.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work.

References

[1] A. Burns, R. Davis, S. Baruah, I. Bate, Robust mixed-criticality systems, IEEE Trans. Comput. (2018) 1–1.
[2] J. Caplan, Z. Al-Bayati, H. Zeng, B. Meyer, Mapping and scheduling mixed-criticality systems with on-demand redundancy, IEEE Trans. Comput. 67 (4) (2018) 582–588.
[3] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, Scheduling of mixed-criticality applications on resource-sharing multicore systems, in: Proceedings of the ACM International Conference on Embedded Software, 2013, pp. 1–15.
[4] A. Burns, R. Davis, A survey of research into mixed criticality systems, ACM Comput. Surv. 50 (6) (2017) 1–35.
[5] S. Baruah, V. Bonifaci, G. Dangelo, H. Li, A. Marchetti-Spaccamela, S. Ster, L. Stougie, Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems, J. ACM 62 (2) (2015) 1–33.
[6] P. Huang, P. Kumar, G. Giannopoulou, L. Thiele, Energy efficient DVFS scheduling for mixed-criticality systems, in: Proceedings of the ACM International Conference on Embedded Software, 2014, pp. 1–10.
[7] J. Han, X. Tao, D. Zhu, H. Aydin, Z. Shao, L. Yang, Multicore mixed-criticality systems: Partitioned scheduling and utilization bound, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (1) (2018) 21–34.
[8] D. Maxim, R. Davis, L. Cucu-Grosjean, A. Easwaran, Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling, in: Proceedings of the ACM International Conference on Real-Time Networks and Systems, 2017, pp. 237–246.
[9] R. Davis, S. Altmeyer, S. Burns, Mixed criticality systems with varying context switch costs, in: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, 2018, pp. 140–151.
[10] S. Kang, H. Yang, S. Kim, I. Bacivarov, S. Ha, L. Thiele, Static mapping of mixed-critical applications for fault-tolerant mpsocs, in: Proceedings of IEEE Design Automation Conference, 2014, pp. 1–6.
[11] K. Cao, J. Zhou, T. Wei, M. Chen, S Hu, K Li, A survey of optimization techniques for thermal-aware 3d processors, J. Syst. Arch. (2019) 1–1.
[12] P. Huang, H. Yang, L. Thiele, On the Scheduling of Fault-Tolerant Mixed-Criticality Systems, Technical Report 351, ETH Zurich, Laboratory TIK, 2013, pp. 1–22.
[13] J. Lin, A. Cheng, D. Steel, M. Wu, N. Sun, Scheduling mixed-criticality real-time tasks in a fault-tolerant system, Int. J. Embedded Real-Time Commun. Syst. 6 (2) (2015) 65–86.
[14] L. Zeng, P. Huang, L. Thiele, Towards the design of fault-tolerant mixed-criticality systems on multicores, in: Proceedings of the ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems, 2016, pp. 1–10.
[15] Z. Al-bayati, J. Caplan, B. Meyer, H. Zeng, A four-mode model for efficient fault-tolerant mixed-criticality systems, in: Proceedings of the IEEE Design, Automation & Test in Europe Conference & Exhibition, 2016, pp. 97–102.
[16] J. Zhou, M. Yin, M. Li, K. Cao, J. Yan, T. Wei, M. Chen, X. Fu, Fault-tolerant task scheduling for mixed-criticality real-time systems, J. Circuits Syst. Comput. 26 (1) (2017) 1–17.
[17] Z. Li, C. Guo, X. Hua, S. Ren, Reliability guaranteed energy minimization on mixed-criticality systems, J. Syst. Softw. 112 (2016) 1–10.
[18] A. Taherin, M. Salehi, A. Ejlali, Reliability-aware energy management in mixed-criticality systems, IEEE Trans. Sustain. Comput. 3 (3) (2018) 195–208.
[19] S. Baruah, H. Li, L. Stougie, Towards the design of certifiable mixed-criticality systems, in: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, 2010, pp. 13–22.
[20] P. Huang, G. Giannopoulou, Service Adaptions for Mixed-Criticality Systems, Technical Report 350, ETH Zurich, Laboratory TIK, 2014, pp. 1–17.
[21] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, D. Tarjan, Temperature-aware microarchitecture: Modeling and implementation, ACM Trans. Archit. Code Optim. 1 (1) (2004) 94–125.
[22] J. Zhou, K. Cao, P. Pong, T. Wei, M. Chen, G. Zhang, J. Yan, Y. Ma, Reliability and temperature constrained task scheduling for makespan minimization on heterogeneous multi-core platforms, J. Syst. Softw. 133 (2017) 1–16.
[23] H. Huang, V. Chaturvedi, G. Quan, J. Fan, M. Qiu, Throughput maximization for periodic real-time systems under the maximal temperature constraint, ACM Trans. Embedded Comput. Syst. 13 (2s) (2014) 1–22.
[24] S. Saha, Y. Lu, J. Deogun, Thermal-constrained energy-aware partitioning for heterogeneous multi-core multiprocessor real-time systems, in: Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2012, pp. 41–50.
[25] T. Rosing, K. Mihic, G. Micheli, Power and reliability management of SoCs, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 15 (4) (2007) 391–403.
[26] B. Kleinjohann, L. Kleinjohann, R. Machado, C. Pereira, P. Thiagarajan, From Model-Driven Design to Resource Management for Distributed Embedded Systems: IFIP TC 10 Working Conference on Distributed and Parallel Embedded Systems, Springer, 2006.
[27] R. Jayaseelan, T. Mitra, Temperature aware task sequencing and voltage scaling, in: Proceedings of the IEEE International Conference on Computer-Aided Design, 2008, pp. 618–623.
[28] Y. Xiang, T. Chantem, R. Dick, X. Hu, L. Shang, System-level reliability modeling for MPSoCs, in: Proceedings of the IEEE/ACM/IFIP Hardware/Software Codesign and System Synthesis, 2010, pp. 297–306.
[29] A. Ejlali, B. Al-Hashimi, P. Eles, Low-energy standby-sparing for hard real-time systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 31 (3) (2012) 329–342.
[30] S. Aminzadeh, A. Ejlali, A comparative study of system-level energy management methods for fault-tolerant hard real-time systems, IEEE Trans. Comput. 60 (9) (2011) 1288–1299.
[31] K. Cao, J. Zhou, P. Cong, L. Li, T. Wei, M. Chen, S. Hu, X. Hu, Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, pp. 1–1.
[32] M. Haque, H. Aydin, D. Zhu, On reliability management of energy-aware real-time systems through task replication, IEEE Trans. Parallel Distrib. Syst. 28 (3) (2017) 813–825.
[33] I. Koren, C. Krishna, Fault-Tolerant Systems, Elsevier, 2010.
[34] D. Pradhan, Fault-Tolerant Computer System Design, Prentice-Hall, 1996.
[35] T. Chantem, Y. Xiang, X. Hu, P. Dick, Enhancing multicore reliability through wear compensation in online assignment and scheduling, in: Proceedings of the IEEE Design, Automation & Test in Europe Conference & Exhibition, 2013, pp. 1373–1378.
[36] S. Lin, B. Schutter, Y. Xi, H. Hellendoorn, Fast model predictive control for urban road networks via MILP, IEEE Trans. Intell. Transp. Syst. 12 (3) (2011) 846–856.
[37] M. Qiu, K.G.Z. Ming, J. Li, Z. Zong, Phase-change memory optimization for green cloud with genetic algorithm, IEEE Trans. Comput. 64 (12) (2015) 3528–3540.
[38] J. Kennedy, Particle swarm optimization, Encycl. Mach. Learn. (2011) 760–766.
[39] R. Rubinstein, D. Kroese, The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning, Springer, 2013.

[40] K. Cao, G. Xu, J. Zhou, T. Wei, M. Chen, S Hu, QoS-adaptive approximate real-time computation for mobility-aware IoT lifetime optimization, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, pp. 1–1.

[41] E3S, Available Online: http://ziyang.eecs.umich.edu/dickrp/e3s/.

[42] X. Zhao, Y. Guo, X. Chen, Z. Feng, S. Hu, Hierarchical cross-entropy optimization for fast on-chip decap budgeting, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 30 (11) (2011) 1610–1620.

[43] J. Zhou, X. Hu, Y. Ma, T. Wei, Balancing lifetime and soft-error reliability to improve system availability, in: Proceedings of the IEEE Asia and South Pacific Design Automation Conference, 2016, pp. 685–690.

[44] G. Quan, V. Chaturvedi, Feasibility analysis for temperature constraint hard real-time periodic tasks, IEEE Trans. Ind. Inf. 6 (3) (2010) 329–339.

[45] W. Liu, J. Yi, M. Li, P. Chen, L. Yang, Energy-efficient application mapping and scheduling for lifetime guaranteed MPSoCs, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2018) 1–1.

[46] Y. Ma, T. Chantem, R. Dick, X. Hu, Improving system-level lifetime reliability of multicore soft real-time systems, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 25 (6) (2017) 1895–1905.

[47] GENE toolbox, Available Online: http://lancet.mit.edu/galib-2.4/.

[48] PSOA toolbox, Available Online: https://github.com/marcnormandin/ParticleSwarmOptimization.

Kun Cao is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, East China Normal University, Shanghai, China. His current research interests are in the areas of Internet of things, cyber physical systems, 3-D ICs, and real-time embedded systems. He was a recipient of the Reviewer Award from Journal of Circuits, Systems, and Computers, in 2016.

Guo Xu is currently pursuing the master degree with the Department of Computer Science and Technology, East China Normal University, Shanghai, China. His current research interest is in the area of power management in mobile devices.

Junlong Zhou received the Ph.D. degree in Computer Science from East China Normal University, Shanghai, China, in 2017. He was a Visiting Scholar with the University of Notre Dame, Notre Dame, IN, USA, during 2014–2015. He is currently an Assistant Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include real-time embedded systems, cloud computing and IoT, and cyber physical systems, where he has published more than 40 refereed papers. Dr. Zhou is an active reviewer of more than 25 international journals. He received the Reviewer Award from Journal of Circuits, Systems, and Computers, in 2016. He has served as publication chairs, publicity chairs, section chairs, and TPC members for numerous conferences. He has been an Associate Editor for the Journal of Circuits, Systems, and Computers, and serves as a Guest Editor for several special issues of ACM Transactions on Cyber–Physical Systems, IET Cyber–Physical Systems: Theory & Applications, and Elsevier Journal of Systems Architecture: EMBEDDED SOFTWARE DESIGN. He is a member of the IEEE.

Mingsong Chen received the B.S. and M.E. degrees from Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006 respectively, and the Ph.D. degree in Computer Engineering from the University of Florida, Gainesville, in 2010. He is currently a full Professor with the Department of Embedded Software and Systems of East China Normal University. His research interests are in the area of design automation of cyber–physical systems, formal verification techniques and mobile cloud computing. He is a member of the IEEE.

Tongquan Wei received his Ph.D. degree in Electrical Engineering from Michigan Technological University in 2009. He is currently an Associate Professor in the Department of Computer Science and Technology at the East China Normal University. His research interests are in the areas of Internet of Things, edge computing, cloud computing, and design automation of intelligent and CPS systems. He serves as a Regional Editor for Journal of Circuits, Systems, and Computers since 2012. He is a member of the IEEE.

Keqin Li is a SUNY Distinguished Professor of computer science in the State University of New York. His current research interests include parallel computing and high performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber–physical systems. He has published over 590 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, and IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.