On the Scheduling of Energy-Aware Fault-Tolerant Mixed-Criticality Multicore Systems with Service Guarantee Exploration

Sepideh Safari, Mohsen Ansari, Ghazal Ershadi, and Shaahin Hessabi, *member*, *IEEE*

Abstract— Advancement of Cyber-Physical Systems has attracted attention to Mixed-Criticality Systems (MCSs), both in research and in industrial designs. As multicore platforms are becoming the dominant trend in MCSs, joint energy and reliability management is a crucial issue. In addition, providing guaranteed service level for low-criticality tasks in critical mode is of great importance. To address these problems, we propose “LETR-MC” scheme that simultaneously supports certification, energy management, fault-tolerance, and guaranteed service level in mixed-criticality multicore systems. In this paper, we exploit task-replication to not only satisfy reliability requirements, but also to improve the QoS of low-criticality tasks in overrun situation. Our proposed LETR-MC scheme determines the number of replicas, and reduces the execution time overlap between the primary tasks and replicas. Moreover, instead of ignoring low-criticality tasks or selectively executing them without any guaranteed service level in overrun mode, it mathematically explores the minimum achievable service guarantee for each low-criticality task in different execution modes, i.e. normal, fault-occurrence, overrun and critical operation modes. We develop novel unified demand bound functions (DBF), along with a DVFS method based on the proposed DBF analysis. Our experimental results show that LETR-MC provides up to 59% (24% on average) energy saving, and significantly improves the service levels of low-criticality tasks compared to the state-of-the-art schemes.

Index Terms—Task Replication, Energy Management, Guaranteed Service Level, DBF, Multicores, Mixed-Criticality Systems.

1 INTRODUCTION

WITH the advancement of Cyber Physical Systems, Mixed-Criticality Systems (MCSs) have recently become the subject of an important research area as the next generation of complex embedded and cyber physical systems [1]. MCSs integrate components with different levels of criticality onto a common platform to reduce cost, space, weight, heat generation and power consumption of the system [5], [6]. The advent of MCSs poses significant new challenges on the system design since applications can interfere with each other on common resources. This point shows the importance of task scheduling and certification on the shared platform of MCSs.

In addition to the certification issue, simultaneous energy and reliability management is another crucial aspect of designing MCSs. As the number of cores on a single chip continues to increase [8], [15], the chip power/energy consumption will increase exponentially. Dynamic Power Management (DPM) [35] and Dynamic Voltage and Frequency Scaling (DVFS) [9] are two popular techniques for energy management. However, scaling the supply voltage in DVFS can potentially degrade the system reliability due to the increasing transient fault rate in the current ever-decreasing technology feature sizes [7], [10]. Transient faults are usually mitigated through exploiting re-execution [16], [27], [28] or replication [2], [11]. The former imposes time overhead, i.e. affects the most critical parameter in MCSs, and the latter imposes power consumption overhead. Therefore, choosing a suitable fault-tolerant technique for MCSs while reducing overall energy consumption is essential.

Another growing difficulty in the scheduling of MCSs is the quality-of-service (QoS) of low-criticality (LC) tasks in overrun situations. LC tasks have one designer-specified Worst-Case Execution Time (WCET) [16], [17], while high-criticality (HC) tasks have two instances of WCETs: W^{LO} which is estimated by system designers, and W^{HI} which is more pessimistic and estimated by certification authorities [6], [17], [18]. The system starts in normal mode, and whenever an HC task exceeds its W^{LO}, the system switches to the overrun mode. HC tasks must be schedulable in both the normal and overrun modes, but the schedulability of LC tasks in overrun mode depends on the chosen scheduling scenario. Some scheduling algorithms discard all LC tasks [6], [16], [17], while the others guarantee a minimum service level for LC tasks. Overall, presenting a scheduling algorithm that simultaneously supports timeliness, energy management, fault-tolerance and guaranteed service level for LC tasks is becoming increasingly challenging in the design of MCSs as technology advances to multicores.

In this paper, we exploit task replication to achieve the reliability target. The task replication approach schedules multiple copies (i.e., one as primary and others as replica) of a task on different cores to tolerate a certain number of faults [10], [15], [17]. Our proposed LETR-MC scheme determines the proper number of replicas for each HC task to achieve the given reliability target. Then, it conservatively maps the task set into cores and schedules them to reduce the execution time overlap between the primary tasks and

- *The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran 14588, Iran (e-mail: ssafari@ce.sharif.edu; mansari@ce.sharif.edu; ershadi@ce.sharif.edu; hessabi@sharif.edu).*

their replicas in an energy-saving manner while preserving certification constraints. In order to conquer service abruption problem in overrun situations, by addressing fault and overrun as two independent events, we consider different execution modes including normal, fault-occurrence, overrun and critical. Also, we modify the elastic mixed-criticality task model to let LC tasks have different guaranteed relaxed periods in each execution mode. In order to check the schedulability of the task set in each execution mode, we develop unified demand bound function (DBF) analysis. Also, we propose a DBF-based DVFS technique that assigns the proper task frequency, which considers timeliness and reliability constraints as well as QoS. By considering various execution modes and proposing the unified DBF schedulability test, the multicore system can tolerate fault occurrence, overrun and even both of them in distinct cores at the same time. The main contributions of this work are:

- Proposing a novel Low Energy Task Replication mechanism in Mixed-Criticality systems (called LETR-MC scheme) to support certification (timeliness), energy management, fault tolerance, and guaranteed service level simultaneously in multicores.
- Adapting task replication, not only as a fault tolerant technique but also to improve LC tasks' QoS in the overrun situation.
- Considering different execution modes including normal, fault-occurrence, overrun, and critical (distinction between overrun and fault occurrence) and mathematically guarantying an acceptable service level for LC tasks in each mode, instead of killing or selectively executing them.
- Developing a new unified DBF-based schedulability test, under deadline, energy reduction, reliability requirements, and guaranteed relaxed periods considerations in different execution modes. Meanwhile, we propose a DBF-based DVFS technique, and exploit DVFS along with DPM to save energy under timeliness and reliability constraints.

The remainder of this paper is organized as follows. In Section 2, we review related work. Section 3 presents models and assumptions. In Section 4, we present our LETR-MC scheme in details. The experimental results are presented and discussed in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

Mixed-criticality systems were first introduced by Vestal [4]. The primitive research proposed different scheduling algorithms. The proposed scheduling algorithms are classified according to their policy to deal with LC tasks in the overrun situation. Some scheduling algorithms discard all LC tasks after entering the overrun mode, e.g. EDF-VD (Earliest Deadline First with Virtual Deadline) [6], [16], [17], [21], [24]. Other ones degrade the QoS of LC tasks to mitigate the effect of service interruption [18], [23], [25], [26]. Su *et al.* [18] have proposed ER-EDF scheduling in a single-core system, which increases the period of LC tasks in the overrun mode to reduce their execution frequency and competition with HC tasks. Su *et al.* [25] have improved the ER-EDF algorithm by focusing on the online behavior of the single-core system. Su *et. al*. [26] have applied the ER-EDF scheduling to multicores. The reference [23] combines ER-EDF scheduling with virtual deadlines for the dual-criticality single-core system, and guarantees LC tasks service level in the overrun mode. The mentioned works do not consider fault-tolerance or energy management.

Other recent work explore the scheduling problem in the context of fault-tolerant MCSs without considering energy management. Works presented in [16], [27], [28], [29] use re-execution as their fault-tolerance technique. The references [16] and [26] wisely select the droppable LC tasks in the overrun mode. The references [28] and [29] immediately drop all LC tasks once either a transient fault or an overrun occurs. All of the above-mentioned references have considered a two-mode operation. Hence, the system switches to critical mode due to fault or overrun, and they do not guarantee an acceptable service level for LC tasks. The references [30] and [31] provide analysis techniques to bound the effects of task killing and service degradation on the safety and schedulability of the system. The reference [30] presents a method to convert the fault-tolerance problem into a standard scheduling problem in a single-core MCS. The reference [20] addresses fault occurrence and overrun with separate modes in a single-core and multiprocessors. However, it selectively chooses LC tasks to continue their execution in each mode.

Few works like [9], [32], [33], and [34] cope with the energy management problem in MCSs, but they do not consider reliability requirements. Huang *et al.* [9] have proposed an optimal solution based on DVFS with the EDF-VD scheduling to minimize dynamic energy consumption for single-cores in normal mode, where tasks of the same criticality level share the same frequency. The reference [32] have extended the work in [9] to multicores, and HC tasks share the same frequency in overrun mode. Legout *et al.* [34] have developed an optimal solution for static energy reduction by applying DPM technique for single-core MCSs. Volp *et al.* [34] have considered an energy budget for multicore MCSs, and focus on energy utilization of HC tasks at the expense of sacrificing LC tasks. The reference [35] applies DVFS only on LC tasks with the cost of degrading their service level and only in the low-criticality mode of the system. Lia *et. al* in [36] and [37] have reduced the energy consumption of a single-core fault-tolerant MCS through resource demand analysis. However, they consider two-mode model, and drop all LC tasks in overrun situations; also the re-execution energy is not taken into account. All the mentioned works have considered two operating modes and have discarded all LC tasks or have selectively executed them in the overrun mode.

In this paper, we address the problem of scheduling mixed-criticality tasks on multicores, and reduce energy consumption while satisfying timeliness and reliability requirements, and at the same time, guarantee the acceptable service level for LC tasks in each mode.

Table 1. Timing parameters for tasks in different models

ζ_i	WCET	Basic MC task model		VD model		E-MC		E²MC		MEMC (proposed model)	
HC	W_i^{LO}, W_i^{HI}	T_i	D_i	T_i	$D_i^{LO} \leq T_i$	T_i	D_i	T_i	$D_i^{LO} < T_i$	T_i	$D_i^{LO} \leq T_i$
LC	W_i^{LO}	T_i	D_i	T_i	D_i	T_i, T_i^{max}	D_i	T_i, T_i^{LO}, T_i^{HI}	D_i	T_i, T_i^{NR}, T_i^{OV}, T_i^{FO}, T_i^{CR}	$D_i^X<T_i^X$

3 Models and Preliminaries

In this section, we introduce the models and preliminaries which are used throughout the rest of the paper.

3.1 System and Application Model

In this paper, we consider an MCS with two different criticality levels, which are denoted as high-criticality and low-criticality levels. Also, we may choose any two criticalities out of the five criticality levels in DO-178B standard [21]. There are n mixed-criticality tasks $\Gamma=\{\tau_1,\tau_2, ..., \tau_n\}$ that will be executed on a multicore platform. The cores are identical and DVFS enabled with a finite set of available frequencies, i.e. $F=\{f_1,...,f_q\}$ where $f_1=f_{max}$ and $f_q=f_{min}$. These frequencies are normalized with respect to f_{max}, i.e. $f_{max}=1$. Tasks are independent, and do not share any resources other than the core. Also, similar to [49], [50], [51], and [52] we assume that the context switching overhead is negligible.

Each task τ_i has hextuple of parameters $\{\zeta_i, W_i^{LO}, W_i^{HI}, T_i, D_i, L_i\}$:

- ζ_i € $\{LC, HC\}$ denotes the criticality level of τ_i.
- W_i^{LO} is the designer-specified WCET for τ_i.
- W_i^{HI} is the CAs-specified WCET for τ_i.
- T_i is the period of τ_i (minimum inter-arrival time).
- D_i is the deadline for the task. We assume $D_i \leq T_i$.
- L_i is the task's safety level according to DO-178B.

It should be noted that in dual-criticality systems, if $\zeta_i=LC$, then $W_i^{HI}=W_i^{LO}$, otherwise $W_i^{LO} \leq W_i^{HI}$. Each task τ_i generates a sequence of jobs (or task instances) with the period of T_i. Table 1 shows different task models. In conventional (basic) mixed-criticality task model, both of the LC and HC tasks have one desired period T_i. The Elastic Mixed-Criticality (E-MC) task model [18] defines a maximum period T_i^{max} for LC tasks in addition to their desired period. When a task is executed with the desired period, it has a maximum level of QoS. However, maximum period represents the minimum level of QoS. The period of LC tasks can be extended by 2 to 5 times [35]. In the E-MC model, the utilization of the system is defined as follows (which will also be used in this paper):

- Low-level utilization of HC tasks:

$$U(HC,L)=\sum_{\tau_i \in \Gamma}^{\zeta_i=\zeta_i^{HC}} u_i^{LO},\ u_i^{LO}=\frac{W_i^{LO}}{T_i} \tag{1}$$

- High-level utilization of HC tasks:

$$U(HC,H)=\sum_{\tau_i \in \Gamma}^{\zeta_i=\zeta_i^{HC}} u_i^{HI},\ u_i^{HI}=\frac{W_i^{HI}}{T_i} \tag{2}$$

- Low-level (desired) utilization of LC tasks:

$$U(LC,L)=\sum_{\tau_i \in \Gamma}^{\zeta_i=\zeta_i^{LC}} u_i^{LO},\ u_i^{LO}=\frac{W_i^{LO}}{T_i} \tag{3}$$

- Minimum utilization of LC tasks:

$$U(LC,min)=\sum_{\tau_i \in \Gamma}^{\zeta_i=\zeta_i^{LC}} u_i^{min},\ u_i^{min}=\frac{W_i^{LO}}{T_i^{max}} \tag{4}$$

The Extended E-MC (E²MC) task model [23] assumes that an LC task can have a pair of small and large periods T_i^{LO} and T_i^{HI} which represent its service guarantee in the normal and overrun execution modes, respectively.

Our Modified E-MC (MEMC) task model: We present MEMC task model, where HC tasks have one period T_i, while LC tasks, in addition to their desired period T_i, have four other periods. Each of these periods reveals QoS of LC tasks according to a specific execution mode of HC tasks. T_i^{NR} and T_i^{OV} are the periods of LC tasks in the normal and overrun operation of HC tasks and fault-free scenario, respectively. T_i^{FO} and T_i^{CR} are the periods of LC tasks with the normal and overrun operation of HC tasks and fault occurrence, respectively. These periods are bigger than or equal to the desired period, and show the minimum achievable release frequency for jobs of the LC tasks in each mode. Details of the binary search method to find the proper periods are clarified in Section 4.3. Table 2 shows the notation used for variables throughout this paper.

3.2 Fault Model and Reliability Analysis

As mixed-criticality embedded systems often control safety-critical applications, tolerating faults and achieving high reliability levels are of great importance; i.e., faults must be detected, and appropriate recovery tasks must be successfully completed before the deadlines. In MCSs, each criticality level has an important property, which is known as Probability of Failure per Hour (PFH). PFH represents the maximum probability of failure to which each task of that level must adapt. The avionics DO-178B standard defines five criticality levels from A with highest, to E with lowest criticality levels. Safety requirements of each criticality level are shown in Table 3 [2]. In this paper, we assume that LC tasks are chosen from D or E levels without any fault-tolerant provisions, and HC tasks are chosen from A, B, or C levels. Hence, each task τ_i from HC task set must be guaranteed to be schedulable, even in presence of faults, to achieve a failure rate of at most $PFH_i=PFH(\zeta^{HI})$. Faults can be categorized into transient

Table 2. Adopted notations

Notation	Description
$\Gamma=\{\tau_1,\tau_2, ..., \tau_n\}$	A set of n E³MC tasks
$M=\{m_1, ...,m_c\}$	A set of c identical cores
$F=\{f_{max}, ..., f_{min}\}$	Core frequency levels
$\zeta= \{HC, LC\}$	Task's criticality levels
$W^{\varsigma}\ \vert \varsigma=\{LO, HI\}$	LO- and High-level WCET of task τ_i
$X=\{NR, OV, FO, CR\}$	System operation modes
Γ_{m_k}	Set of tasks on the core m_k including HC, LC and replica tasks
$\Gamma_{m_k}(\zeta)$	Set of tasks with criticality level ζ on the core m_k
$\Gamma_{m_k}(B)$	Set of replica tasks on the core m_k
PFH(ζ)	probability of failure per hour of tasks with criticality level ζ

Table 3. DO178B safety requirements [7]

ζ	A	B	C	D	E
PFH	$< 10^{-9}$	$< 10^{-7}$	$< 10^{-5}$	$\geq 10^{-5}$	-

and permanent faults. Transient faults are found more frequently than permanent faults [36], [39]. Hence, in this paper, we focus on transient faults. Nevertheless, we try to provide provisions to tolerate permanent faults as well. Transient faults are typically modeled using a Poisson distribution with an average arrival rate λ, which depends on the core frequency [36], [38]. The fault rate at frequency f_i is modeled as [39]:

$$\lambda(f_i) = \lambda_0 10^{\frac{d(1-f_i)}{1-f_{\min}}} \tag{5}$$

where $\lambda_0 = 10^{-4}$ (fault/ms) is the transient fault rate at f_{max}, exponent d is a sensitivity factor parameter with typical values in the range 2-6 [14], [15]. We choose d=2 similar to [15]. The reliability of a task is defined as the probability of executing the task successfully, in the absence of transient faults [38]. The reliability of task τ_i running at frequency f_i can be expressed as [39]:

$$R_i(f_i) = e^{-\lambda(f_i)\frac{W_i}{f_i}} \tag{6}$$

where $\lambda(f_i)$ is given by Eq. 5, and W_i is the execution time of the task τ_i. In our proposed method, similar to [2], [11], [12], [15], and [39], the fault detection takes place at the end of the completion of each task instance. If a task instance completes earlier than its WCET, the fault detection mechanism takes place as soon as the completion of the task instance; otherwise, faults are detected at task's low and high level WCETs. Therefore, at the end of execution of each task instance, an acceptance test (or, sanity check) [45], [46] is conducted to check for the occurrence of transient faults. If the acceptance test indicates a fault occurrence, the faulty output is discarded and the task copy (replica task) will be executed to determine the correct output. Otherwise, task copies will be cancelled as soon as successful completion of the corresponding primary tasks [38].

It should be noted that acceptance tests are not completely accurate. Sometimes a fault may remain undetected or the acceptance test may diagnose a correct outcome as a faulty one [11]. Therefore, the probability that the acceptance test will perform incorrectly is considered as a factor in the reliability computation of the task, which is known as the coverage factor of the acceptance test. Thus, the reliability of a task instance can be expressed as:

$$R_i(f_i) = (1-\alpha) \times e^{-\lambda(f_i)\frac{W_i}{f_i}} \tag{7}$$

where a is the probability of making an incorrect decision during the acceptance test. The reliability of HC tasks depends on their WCETs as follows:

$$R_i^{LO}(f_i) = (1-\alpha) \times e^{-\lambda(f_i)\frac{W_i^{LO}}{f_i}}, \quad R_i^{HI}(f_i) = (1-\alpha) \times e^{-\lambda(f_i)\frac{W_i^{HI}}{f_i}} \tag{8}$$

Consequently, the probability of failure (PoF) of the task τ_i based on its W^{LO} and W^{HI} are as follows:

$$PoF_i^{HI}(f_i) = 1 - R_i^{HI}(f_i) \tag{9}$$

$$PoF_i^{LO}(f_i) = 1 - R_i^{LO}(f_i) \tag{10}$$

In multicore platforms, task replication is likely to become a quite viable option for reliability management. By scheduling multiple copies of the same task on multiple cores, the likelihood of successfully completing at least one of them (i.e., without encountering transient faults) increases significantly. If the PoF^{HI} of an HC task (when executed with W^{HI}) meets the PoF_{target}, it means that the mentioned task does not need any replicas. Otherwise, the scheduler should determine the proper number of replicas of task τ_i to achieve the reliability targets. Note that each replica task, being a periodic task itself, generates a sequence of instances on the core where it is assigned. Hence, we define two extremes for the number of replica tasks. The minimum number of required replica tasks for each HC task (r_{lower_bound}) is the case when all the replicas will be executed with W^{LO}, and can be expressed as:

$$\begin{gathered} PoF_{target} \geq PoF_i^{HI}(f_i).PoF_i^{LO}(f_i)^r \\ r_{i_{lower\_bound}} \geq \left\lfloor \frac{\log(PoF_{target} / PoF_i^{HI}(f_i))}{\log(PoF_i^{LO}(f_i))} \right\rfloor \end{gathered} \tag{11}$$

The maximum number of required replicas (r_{upper_bound}) is the case when all the replicas will be executed with W^{HI}.

$$\begin{gathered} PoF_{target} \geq PoF_i^{HI}(f_i).PoF_i^{HI}(f_i)^r \\ r_{i_{upper\_bound}} \geq \left\lfloor \frac{\log(PoF_{target} / PoF_i^{HI}(f_i))}{\log(PoF_i^{HI}(f_i))} \right\rfloor \end{gathered} \tag{12}$$

Therefore, we define upper and lower bounds for the number of required replicas. Hence, we define the parameter K, where $K \in [r_{lower\_bound}, r_{upper\_bound}]$. When K=0, all the replica tasks will be executed with W^{LO}; i.e., r is the minimum number of required replicas. Otherwise, increasing the value of K, increases the number of replica tasks that will be executed with W^{HI}. Therefore, by giving a certain PoF_{target}, we can find the minimum number of replicas r for each task to achieve its reliability target as follows:

$$PoF_{target} \geq PoF_i^{HI}(f_i).PoF_i^{HI}(f_i)^K.PoF_i^{LO}(f_i)^{r-K} \tag{13}$$

$$r_i \geq \left\lfloor \frac{\log((PoF_{target} / PoF_i^{HI}(f_i)^{K+1}).PoF_i^{LO}(f_i)^K)}{\log(PoF_i^{LO}(f_i))} \right\rfloor \tag{14}$$

Therefore, in task replication technique, it is sufficient to have at least one task copy execution that passes the acceptance test. Hence, the execution will be unsuccessful only if all copies of a task encounter faults. Therefore, in task replication, if a primary task and its replica(s) have an overlap execution part, as soon as a primary task completes successfully, the remaining parts of its replica(s) will be abounded to avoid further energy consumption. Also, we consider that each core is capable of detecting faults [36], [37]. In order to detect faults, processing cores typically employ a low-cost hardware checker like Argus [47]. Argus provides low-cost, comprehensive, low power and high accuracy fault detection. It uses run-time checking of control flow, computation, data flow, and memory invariants. Meixner et al. [47] have shown that checking these four invariants is sufficient for detecting all possible single errors, and many multiple-error scenarios. Argus adds less than 17% to the core area (and less than

11% to the total chip area, including caches) and increases runtime execution by 3.2-3.9% on average. Argus does not change the clock cycle time, and is applicable to many embedded applications as well as multicore chips. We consider the overhead of fault detection as a part of the task's WCET [15], [36], [37].

3.3 Power and Energy Consumption Model

Power Model: System-level power model consists of static and dynamic components [9], [36], [37]. The static power, P_{static}, is consumed even when no computation is carried out [38]. The dynamic power $P_{dynamic}$ includes a frequency-independent (P_{ind}), and a frequency-dependent (P_{dep}) power consumption factor. P_{ind} is consumed by the peripheral modules such as I/O in the active mode. Hence, the total power consumption of each core can be written as:

$$P_{total} = P_{static} + P_{dynamic} = I_{sub}V_i + C_{eff}V_i^2 f_i + P_{ind} \tag{15}$$

where C_{eff}, V_i, and f_i are the effective switched capacitance, supply voltage, and operating frequency of the core during the execution of task τ_i, respectively. Also, we assume that P_{ind} is equal to 0.1 [36]. When DVFS is used, each task τ_i is executed at a voltage V_i, which is less than the maximum supply voltage V_{max}. By considering a near-linear relationship between voltage and frequency [13], [15], [39] when a task τ_i is executed at the scaled voltage $V_i=\rho_i.V_{max}$, the operational frequency is $f_i=\rho_i.f_{max}$, where f_i is the operational frequency corresponding to V_i, and f_{max} is the maximum operating frequency corresponding to V_{max}. Therefore, the total power consumption, which is consumed to execute a task τ_i is given by:

$$P_{total} = I_{sub}\rho_i V_{max} + C_{eff}(\rho_i V_{max})^2 \rho_i f_{max} \tag{16}$$

$$P_{total} = \rho_i P_{static}^{max} + \rho_i^3 P_{dynamic}^{max} \tag{17}$$

In this equation, ρ_i is in the range of ρ_{min} to ρ_{max}=1 and $\rho_{min}=V_{min}/V_{max}$. In this paper, in contrast to most of the previous works that consider P_{static}=0 [9], [12], [36], [37], we assume that maximum static power is constant and is a portion of the maximum dynamic power, i.e. $P_{static}^{max} = \varphi.P_{dynamic}^{max}$, [11], [35], [67] where, φ=0.2 [11], [35], [38]. Hence, the total power consumption can be re-written as follows:

$$P_{total} = \rho_i \varphi P_{dynamic}^{max} + \rho_i^3 P_{dynamic}^{max} \tag{18}$$

By scaling *V-f*, in addition to power, the execution time of the task, and its energy consumption will change.

Energy Model: Scaling down the voltage and frequency levels increases the execution time of the task $W_i'=W_i/\rho_i$. The energy consumption of j^{th} job (single job) of a task (τ_i) is as follows:

$$E_{ij} = P_{dynamic}^{max}(\varphi\rho_{ij} + \rho_{ij}^3) \times \frac{W_{ij}}{\rho_{ij}} \tag{19}$$

Also, the normalized energy consumption *NE* of a single job of a task τ_i is:

$$NE_{ij} = (\varphi + \rho_{ij}^2) \times W_{ij} \tag{20}$$

Since the tasks are periodic and they may have multiple jobs in any time interval, the total normalized energy consumption of a task in any time interval of a given length is the summation of all its jobs' energies h_{τ_i} in this duration.

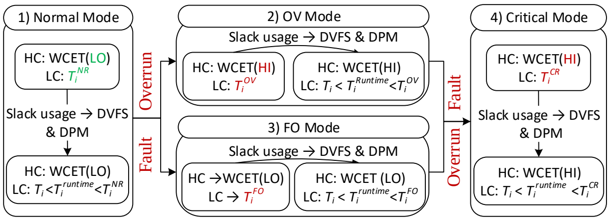


Fig. 1. Overview of system execution model (The execution model of each core).

$$NE_{\tau_i} = \sum_{j=1}^{h_{\tau_i}} (\varphi + \rho_{ij}^2) \times W_{ij} \tag{21}$$

The normalized total energy consumption of each core is the sum of energy consumptions of tasks on that core (m_k: Set of tasks on the core m_k including HC, LC and replica tasks) as follows:

$$NE_{m_k} = \sum_{\forall \tau_i \in \Gamma_{m_k}} \sum_{j=1}^{h_{\tau_i}} (\varphi + \rho_{ij}^2) \times W_{ij} \tag{22}$$

Finally, the normalized total energy consumption of the whole system is the sum of energy consumptions of tasks on all the cores as follows:

$$NE_{system} = \sum_{k=1}^{c} NE_{m_k} = \sum_{k=1}^{c} \sum_{\forall \tau_i \in \Gamma_{m_k}} \sum_{j=1}^{h_{\tau_i}} (\varphi + \rho_{ij}^2) \times W_{ij} \tag{23}$$

3.4 System Operational Model

The overview of the execution model for each core is shown in Fig. 1. Similar to [20], we distinguish between the execution time overrun of tasks and fault occurrence. Therefore, we guarantee an achievable service level for LC tasks in each execution mode to improve their overall QoS, and also save more energy. The behavior of our execution model in each mode is defined by the following epochs:

Epoch 1: The system starts with a normal mode (called NR). Each core stays in NR mode until either overrun or fault occurs. Hence, in this mode, all tasks are executed once with W^{LO}, and LC tasks are executed with T_i^{NR} period. T_i^{NR} represents the highest service level of LC tasks.

Epoch 2: If any HC task exceeds its low-level WCET, its designated core switches to the overrun mode (called OV), where HC tasks (on that core) are safely executed once, and must meet their deadlines assuming high-level WCETs. LC tasks of that core will be executed with T_i^{OV} period to guarantee their service levels. If an LC task does not complete in its low-level WCET, it will be terminated.

Epoch 3: If any HC task signals completion after running for its W^{LO} but encounters a fault, the system switches to the fault-occurrence mode (called FO), where the scheduler guarantees the execution of the sufficient number of replicas of a faulty HC task to satisfy its reliability requirement. Hence, the scheduler executes its replicas on the cores. Only the cores which host the replicas of the faulty tasks switch to the FO mode. In this mode, HC tasks and replicas are still executed with W^{LO}. Also, the execution of the required replicas for HC tasks must be guaranteed to finish before the deadline. LC tasks will be executed with T_i^{FO} period.

Epoch 4: When a core is in OV mode, the fault can occur; similarly, when it is in FO mode, overrun can happen, and both of these conditions enter the designated core into the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2907846, IEEE Transactions on Parallel and Distributed Systems

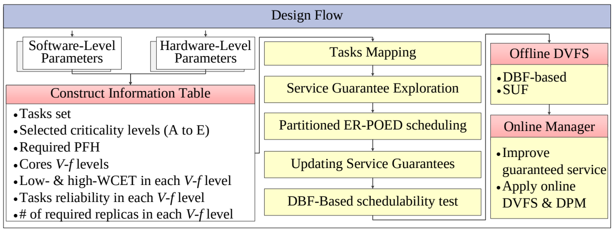

Design Flow
Software-Level Parameters
Hardware-Level Parameters
Construct Information Table
•Tasks set
•Selected criticality levels (A to E)
•Required PFH
•Cores V-f levels
•Low- & high-WCET in each V-f level
•Tasks reliability in each V-f level
•# of required replicas in each V-f level
Tasks Mapping
Service Guarantee Exploration
Partitioned ER-POED scheduling
Updating Service Guarantees
DBF-Based schedulability test
Offline DVFS
•DBF-based
•SUF
Online Manager
•Improve guaranteed service
•Apply online DVFS & DPM

Fig. 2. System design flow.

critical mode (called CR). In this state, HC tasks are executed with W^{HI}, and replicas are executed completely to tolerate faults, and LC tasks are executed with their T_i^{CR} period which is the minimum guaranteed service level that can be maintained.

Fig. 1 shows the various operating modes that each core can operate independently of the others. As it is clear in Fig. 1, in all epochs, the scheduler exploits the released dynamic slacks to improve the periods of LC tasks at runtime ($T_i^{runtime}$), which is between their desired periods and the guaranteed ones. Also, we consider a local switching [31], i.e. any core switches from one mode to another one, independent of other cores. Therefore, it is not necessary to reduce the QoS of LC tasks on all cores at once. Also, in our proposed operational model, the system switches back from each of the OV, FO, or CR modes to the normal mode at the end of each hyper period.

4 PROPOSED METHOD

In this section, we explain each part of our proposed scheme in details.

4.1 System Overview

Fig. 2 illustrates the overall design flow of the proposed LETR-MC scheme. The scheduler receives different inputs from hardware and software levels, and schedules tasks in the offline and online phases. In the offline phase, the scheduler initially creates Information Table (IT), which will be used during system operation in different modes to find the best *V-f* level for each task to satisfy its reliability and timing constraints. IT consists of the task set, criticality and required PFH levels for tasks, *V-f* (voltage-frequency) levels, the W^{LO} and W^{HI} of tasks, reliability, and the number of required replicas in each *V-f* levels. After mapping all tasks, the scheduler calls the service guarantee exploration function for computing scaling factors to apply to LC tasks' periods. We adapt the ER-POED scheduling for our proposed method. The offline scheduler applies partitioned scheduling to all tasks on each core. After scheduling all tasks, the service guarantee exploration is called again to update scaling factors, while checking for DBF-based schedulability. The last section of the offline phase is to apply DVFS by using static slack reclamation. Eventually, the final offline schedule is sent to the online manager for applying DVFS and DPM, or improving the service guarantees of LC tasks by using dynamic slacks.

4.2 Task Mapping

Algorithm 1 shows the pseudo-code of our proposed task mapping method. In line 1, IT is constructed and sent to the task mapping mechanism, where HC tasks Γ(HC) and their corresponding replicas Γ(B) are initially mapped to cores, and then LC tasks Γ(LC) are allocated. HC and replica tasks are sorted in decreasing order of utilization at the maximum *V-f* level, respectively (in lines 2-3). In each iteration, the first task in the queue is selected for mapping onto the cores (line 5). Since task replication has the potential to tolerate permanent faults in addition to transient faults, the scheduler tries to avoid assigning replicas of the same task on the same core. Therefore, in line 6, set of cores containing the primary or replica versions of the selected task is removed from the core set *M*, and the remaining cores are put into the temporary core set *TC*. However, if after removing the mentioned cores, the *TC* becomes empty, all cores are returned to the *TC*. Selection of cores

Algorithm 1: The task mapping mechasim of LETR-MC scheme

Inputs: *Γ*, *PFH*, *M*, *V-f* levels.
Output: The task mappimg on each core.

```
BEGIN:
1.  IT=Construct(Γ,PFH, V-f levels);
2.  Γ(HB)= Γ(HC)+Γ(B);                          //# of replicas at $f_{max}$
3.  Γ(HB).sort();        //Sort HC and replica tasks w.r.t utilization
4.  for all tasks in Γ(HB) do
5.      $τ_i$ = Γ(HB).select();    //Select the task with largest utilizaton
6.      TC=M-{∀ $m_k$:$τ_i$∈$m_k$};
7.      TTC=TC.sort();            //Sort cores based on WFD or FFD
8.      sc=TTC.select();          //Select the core sc
9.      if sc is empty then
10.         TC=M;
11.         goto line 7;
12.     end if
13.     if sc.schedulability_test() then                //Based on Eq. 24
14.         sc.map($τ_i$);
15.     else
16.         TC.remove(sc);        //Remove sc from TC for the task $τ_i$
17.         if TC is empty then
18.             TC={∀ $m_k$:$τ_i$∈$m_k$};
19.             TTC=TC.sort();    //Sort cores based on WFD or FFD
20.             sc=TTC.select();    //Select the core sc
21.             if sc is empty then
22.                 return infeasible;
23.             else if sc.schedulability_test() then        //w.r.t Eq. 24
24.                 sc.map($τ_i$);
25.             else
26.                 TC.remove(sc);
27.                 goto line 19;
28.             end if
29.         end if
30.         goto line 7;
31.     end if
32. end for                          //Have all HC & B tasks mapped
33. Γ(LC).sort();                       // Sort LC tasks w.r.t utilization
34. for all tasks in Γ(LC) do
35.     $τ_i$ = Γ(LC).select();    //Select the task with largest utilizaton
36.     TC=M;
37.     TTC=TC.sort();            //Sort cores based on WFD or FFD
38.     sc=TC.select();           // Select core sc
39.     if sc.schedulability_test() then                //Based on Eq. 25
40.         sc.map($τ_i$);
41.     else
42.         TC.remove(sc);        //Remove sc from TC for the task $τ_i$
43.         if TC is empty then
44.             return infeasible;
45.         end if
46.         goto line 37;
47.     end if
48. end for
END
```

for mapping is based on Worst-Fit Decreasing (WFD) and First-Fit Decreasing (FFD) bin packing. In the Worst-Fit Decreasing bin packing cores are sorted in decreasing order by utilization (line 7), then the core with the lowest utilization among others is selected for mapping (line 8), i.e., WFD allocates the selected task into the core with the greatest capacity available, in which it can be feasibly allocated. In the First-Fit-Decreasing bin packing, cores are sorted in decreasing order by utilization (line 7), then the selected task is allocated into the core with the lowest capacity available (largest utilization), in which it can be feasibly allocated in line 8. WFD is the best from the energy-awareness perspective, due to its load-balancing behavior [48]. If there is a core for mapping, the schedulability condition is checked in line 13 as follows:

$$U_{\tau_i \in \Gamma_{m_k}(HC)}(HC,H) + U_{\tau_i \in \Gamma_{m_k}(B)}(B,L) \leq 1 \tag{24}$$

If the selected core *sc* passes the schedulability test, the selected task is mapped to the core *sc* in line 14 and the next task is chosen for mapping. However, if the selected core cannot satisfy the schedulability constraint, it is removed from the core set and the algorithm goes to line 7. Removing the cores which contain the primary or replica versions of the selected task, and also removing the cores which cannot satisfy the schedulability constraint may make *TC* empty. In this case, the cores containing the primary or replica versions of the selected task are added to the *TC*, and selection of a core is performed based on this set in lines 17-29. After mapping all HC and replica tasks, the scheduler sorts LC tasks in a queue, based on their utilization in decreasing order in line 33, and selects the first task in the queue for mapping as shown in line 35. The core selection (*sc*) is also based on WFD and FFD lines 37-38. In each iteration of LC task mapping the scheduler checks the total utilization of each core in lines 39-40 as follows:

$$U_{\tau_i \in \Gamma_{m_k}(HC)}(HC,H) + U_{\tau_i \in \Gamma_{m_k}(B)}(B,L) + U_{\tau_i \in \Gamma_{m_k}(LC)}(LC,L) \leq U\_bound \tag{25}$$

where *U_bound*$\underline{\leq}$1.5. If the schedulability test is passed, the selected task is mapped to the chosen core. Otherwise, the core *sc* is removed from *TC*, and the algorithm iterates in lines 42-47.

After mapping all tasks, due to the challenges associated with task migration including increased sensitivity to implementation complexity, tight power budgets, requirements on execution predictability, the lack of virtual memory support in many low-end MPSoCs, and high runtime overhead [56], [58], migration of task instances from a core to another one is not permitted; i.e., at runtime each task is permitted to execute only on its designated core. After mapping all tasks, the service guarantee exploration function is called for finding the proper scaling factors for periods of LC tasks in each core to extend their periods and reduce their utilization.

4.3 Service Guarantee and Period Assignment

As it is mentioned in Section 2, most mixed-criticality scheduling algorithms discard all LC tasks after entering the overrun mode, or selectively execute them [6], [9], [15-16], [19-20], [23], [26-28], [31-36]. In order to guarantee an acceptable service level for LC tasks, we consider different execution modes (NR, FO, OV, CR), and define four uniform scaling factors {*a*, *β*, *θ*, *γ*} for each LC task in each execution mode. These scaling factors indicate the period of LC tasks in each execution mode, and also represent how frequently LC tasks release their instances to guarantee the timeliness and QoS level. Similar scaling factors of LC tasks that run on the same core are equal to each other. Also, the relationships between these scaling factors and the desired periods T_i of LC tasks in each operation mode are as follows:

$$T_i^{NR} = \alpha.T_i, \;\; T_i^{OV} = \beta.T_i, \;\; T_i^{FO} = \theta.T_i, \;\; T_i^{CR} = \gamma.T_i \tag{26}$$

The total utilization of a core after mapping LC tasks may be bigger than one. Therefore, T_i^{NR}, T_i^{OV}, T_i^{FO}, and T_i^{CR} may have a bigger value than T_i. Differences between T_i and the other periods of LC tasks are illustrated in the example in Appendix.

Based on the schedulability conditions, we first derive lower and upper bounds of these uniform scaling factors in each operation mode. The feasible periods, which satisfy scheduling constraints, will be found between the lower and upper bounds.

Epoch 1: Each core's utilization in NR mode is defined as follows:

$$U_{\Gamma_{m_k}}^{NR} = U_{\Gamma_{m_k}(HC)}(HC,L) + U_{\Gamma_{m_k}(LC)}(LC,L) \leq 1 \tag{27}$$

By replacing the desired period of LC tasks with $T_i^{NR}=\alpha.T_i$ in their utilization formula, we have:

$$U_{\Gamma_{m_k}(LC)}(LC,L) = \sum_{\forall \tau_i \in \Gamma_{m_k}(LC)} \frac{W_i^{LO}}{\alpha T_i} = \frac{1}{\alpha} \sum_{\forall \tau_i \in \Gamma_{m_k}(LC)} \frac{W_i^{LO}}{T_i} \tag{28}$$

It should be noted that T_i is the original period of LC tasks that determines when the tasks are generated. T_i^{NR} is the period of LC tasks in the normal operation mode of the system. Difference between T_i and T_i^{NR} for LC tasks is illustrated in Appendix.

To ensure the schedulability of tasks in NR mode, by replacing Eq. 28 in Eq. 27, we need to have:

$$U_{\Gamma_{m_k}}^{NR} = U_{\Gamma_{m_k}(HC)}(HC,L) + \frac{1}{\alpha} U_{\Gamma_{m_k}(LC)}(LC,MC) \leq 1 \tag{29}$$

Hence, the lower bound for *a* can be found as:

$$\alpha_{LB} = \frac{U_{\Gamma_{m_k}(LC)}(LC,MC)}{1 - U_{\Gamma_{m_k}(HC)}(HC,L)} \tag{30}$$

Here, the utilization which arises from execution overlap of HC tasks and corresponding replicas is ignored. The point is that utilization-based approach does not consider scheduling of tasks (the start time and end time of tasks). Therefore, considering the utilization of all replicas in normal mode is pessimistic because in the actual case in the normal mode, replicas do not need to completely proceed, and only their overlap time with primary tasks will be executed. Therefore, at first, the service guarantee unit computes the scaling factors based on the utilization of tasks. Then, after scheduling the tasks, if it is necessary, the computed scaling factors will be updated through demand bound function analysis, as explained in Section 4.4.

Epoch 2: The utilization bound at OV mode is as follows:

$$U_{\Gamma_{m_k}}^{OV} = U_{\Gamma_{m_k}(HC)}(HC,L) + U_{\Gamma_{m_k}(LC)}(LC,L) \leq 1 \tag{31}$$

In the OV mode, LC tasks are executed with $T_i^{OV}=\beta.T_i$. By replacing the desired period of LC tasks with T_i^{OV}, in their utilization formula (similar to computation of *a*) the lower bound for β can be found as:

$$\beta_{LB} = \frac{U_{\Gamma_{m_k}(LC)}(LC,MC)}{1-U_{\Gamma_{m_k}(HC)}(HC,H)} \tag{32}$$

Epoch 3: In the FO mode, replicas are executed completely due to fault occurrence in primary HC tasks. The utilization bound at this point is as follows:

$$U^{FO}_{\Gamma_{m_k}} = U_{\Gamma_{m_k}(HC)}(HC,L) + U_{\Gamma_{m_k}(B)}(B,L) + U_{\Gamma_{m_k}(LC)}(LC,L) \le 1 \tag{33}$$

In the fault-occurrence mode, the execution period of all the LC tasks on core m_k is T_i^{FO} where $T_i^{FO}=\theta.T_i$. Hence, the lower bound for θ can be found as:

$$\theta_{LB} = \frac{U_{\Gamma_{m_k}(LC)}(LC,MC)}{1-(U_{\Gamma_{m_k}(HC)}(HC,L)+U_{\Gamma_{m_k}(B)}(B,L))} \tag{34}$$

Epoch 4: In the CR mode, the utilization bound of each core is as follows:

$$U^{FO}_{\Gamma_{m_k}} = U_{\Gamma_{m_k}(HC)}(HC,H) + U_{\Gamma_{m_k}(B)}(B,L) + U_{\Gamma_{m_k}(LC)}(LC,L) \le 1 \tag{35}$$

In the FO mode, LC tasks execute with $T_i^{CR}=\gamma.T_i$. Hence, the lower bound for γ can be found as:

$$\gamma_{LB} = \frac{U_{\Gamma_{m_k}(LC)}(LC,MC)}{1-(U_{\Gamma_{m_k}(HC)}(HC,H)+U_{\Gamma_{m_k}(B)}(B,L))} \tag{36}$$

We now define the upper bounds of these scaling factors. γ_{LB} is the factor that makes the task set schedulable in a worst-case scenario [18], [23]. Suppose that, $T_i \underline{<} T_i^{FO} \underline{<} T_i^{CR}$ and $T_i \underline{<} T_i^{OV} \underline{<} T_i^{CR}$. Therefore, the upper bound for β and θ are set as follows:

$$\beta_{UB} = \theta_{UB} = \gamma_{LB} \tag{37}$$

Also, assume that $T_i^{NR} \underline{<} T_i^{FO}$ and $T_i^{NR} \underline{<} T_i^{OV}$. However, T_i^{FO} and T_i^{OV} are not equal, hence we choose the maximum of the two, as the upper bound for α:

$$\alpha_{UB} = \max\{\theta_{LB}, \beta_{LB}\} \tag{38}$$

Hence, the lower and upper bounds are guaranteed in the offline phase by considering task utilization. The pseudo code for finding proper scaling factors is described in Algorithm 2. The scaling factors are defined iteratively through binary search, according to [23], and e shows the tolerable error. The code iterates for each core to find proper scaling factors for all LC tasks on that core. In line 9 of the algorithm, the scheduler checks whether the computed α, β, and θ are feasible based on the proposed scheduling (the effect of scheduling algorithm on scaling factors are described in Section 4.5). It should be noted that if the lower-bounds of these factors become less than 1, the algorithm sets them to 1. After finding the proper scaling factors, the scheduler updates Eq. 23 to set proper periods for LC tasks, and tune job's release distances as follows:

$$T_i^{NR} = \frac{l}{\left\lfloor \frac{l}{\alpha.T_i} \right\rfloor},\quad T_i^{OV} = \frac{l}{\left\lfloor \frac{l}{\beta.T_i} \right\rfloor},\quad T_i^{FO} = \frac{l}{\left\lfloor \frac{l}{\theta.T_i} \right\rfloor},\quad T_i^{CR} = \frac{l}{\left\lfloor \frac{l}{\gamma.T_i} \right\rfloor} \tag{39}$$

4.4 Unified Demand Bound Functions Analysis

Demand Bound Functions (DBF) present an approach to analyze the schedulability of real-time workloads [22]. A mixed-criticality task set is schedulable if the maximum execution demand of all tasks is less than the resource supply in any time interval of a given length in each operation mode. The demand bound of a task in a given interval is defined as the sum of execution times of all jobs of tasks, which have both arrival times and deadlines in this interval. DBF based test is effective but has high computation complexity, and can be applied to general mixed-criticality task sets. In the following, we develop a new DBF computation in each operation mode by considering timing, reliability, QoS, and energy reduction. We compute DBF locally for each core, but we should also have a global glance at all cores because of replicas existence since execution or cancelation of replicas may have an effect on DBF computations of other cores. Also, in order to improve the accuracy of demand bound analysis, we use the unified DBF approach similar to [22], [40], which considers time intervals crossing the mode switch point (transition from one mode to another one) to link the system behaviors in different execution modes. We now derive task's resource demands in all operation modes (epochs) to determine whether a given task set is schedulable under our proposed method.

Algorithm 2: Find feasible scaling factors $SF=\{\alpha, \beta, \theta, \gamma\}$

Inputs: Γ, tasks to cores mapping, M, α_{LB}, α_{UB}, β_{LB}, β_{UB}, θ_{LB}, θ_{UB}, γ_{LB}, and e=0.001. Execution modes X.

Output: Finding the best scaling factors.

```
BEGIN
1. for i = 1 to M do                                  // loop over all cores
2.  while ($\beta_{UB}$ – $\beta_{LB}$ > e) do
3.    $\beta = (\beta_{UB} + \beta_{LB})/2$;
4.    while ($\theta_{UB}$ – $\theta_{LB}$ > e) do
5.       $\theta = (\theta_{UB} + \theta_{LB})/2$;
6.       $\alpha_{UB\_new} = \max\{\beta, \theta\}$;
7.       while ($\alpha_{UB\_new}$ – $\alpha_{LB}$ > e) do
8.          $\alpha = (\alpha_{UB\_new} + \alpha_{LB})/2$;
9.          If ($\Gamma$ is schedulable in all execution modes X) then
10.            $\beta_{UB} = \beta$, $\theta_{UB} = \theta$, $\alpha_{UB} = \alpha$;
11.         else
12.            $\beta_{LB} = \beta$, $\theta_{LB} = \theta$, $\alpha_{LB} = \alpha$;
13.         end if
14.      end while
15.   end while
16.  end while
17. end for
END
```

Epoch 1: In the NR mode, the DVFS technique is applied to HC and LC tasks. Although replicas are canceled at the end of the successful completion of their corresponding primary tasks, their execution time overlaps with corresponding primary tasks should be considered in DBF computations. The overlap time of each replica job (*overlap_time$_j$*) is the difference between the completion time of the primary HC job and the start time of its corresponding replica job. The resource demands of tasks which are executed on core m_k are computed as follows:

$$DBF^{NR}_{\tau_i \in \Gamma_{m_k}(HC)}(\tau_i, l) = \max\{\left\lfloor \frac{l+T_i-D_i}{T_i} \right\rfloor, 0\}.\frac{f_{max}}{f_i}.W_i^{LO} \tag{40}$$

$$DBF^{NR}_{\tau_i \in \Gamma_{m_k}(B)}(\tau_i, l) = \sum_{j=1}^{\left\lfloor \frac{l+T_i-D_i}{T_i} \right\rfloor} overlap\_time_j \tag{41}$$

$$DBF^{NR}_{\tau_i \in \Gamma_{m_k}(LC)}(\tau_i, l) = \max\{\left\lfloor \frac{l+T_i-D_i}{T_i^{NR}} \right\rfloor, 0\}.\frac{f_{max}}{f_i}.W_i^{LO} \tag{42}$$

A task set is schedulable in NR mode on core m_k if and only if for $\forall l \underline{>} 0$:

$$DBF^{NR}_{\tau_i\in\Gamma_{m_k}}(\tau_i,l)=\sum_{\tau_i\in\Gamma_{m_k}(HC)} DBF^{NR}_{\Gamma_{m_k}(HC)}(\tau_i,l)+\sum_{\tau_i\in\Gamma_{m_k}(B)} DBF^{NR}_{\Gamma_{m_k}(B)}(\tau_i,l)+\sum_{\tau_i\in\Gamma_{m_k}(LC)} DBF^{NR}_{\Gamma_{m_k}(LC)}(\tau_i,l)\le l \quad (43)$$

Epoch 2: When an overrun occurs, three types of jobs may exist, i.e. normal, overrun, and crossover [22], [23]. Normal jobs have both release time and deadline before the mode switch point; these jobs execute with W^{LO}. Overrun jobs are released after the mode switch point. Jobs which are released before the mode switch point but have later deadlines are candidates for crossover jobs and their finish time (when they signal their completions) indicates their role. Assume that the switch point to the overrun mode is t^o ($t^s\underline{\le}t^o\underline{\le}\ t^e$) where the length of time interval [t^s, t^e] is l=t^e-t^s and the time that the core spends in overrun mode is q=t^e-t^o. In this case, we define x to show which tasks have crossover candidate jobs as follows:

$$crossover\_candidate=\begin{cases} x=0, & if \quad t^o \bmod T_i=0 \\ x=1, & if \quad t^o \bmod T_i\ne 0 \end{cases} \quad (44)$$

Now, if there is a task that has a crossover candidate job, i.e. x=1, we initially specify its job number j that has release time before the mode switch point and the deadline after this point, and then check whether it is a crossover. Here, if a crossover candidate job j signals completion before the mode switch point t^o, it is counted as a normal job (although its deadline is after t^o); if the released job starts after t^o it is an overrun job. Otherwise, it is a crossover job:

$$crossover=\begin{cases} y=0, & if \quad finish\_time_j<t^o \\ y=1, & if \quad start\_time_j<t^o<finish\_time_j \\ y=0, & if \quad t^o<start\_time_j \end{cases} \quad (45)$$

If $x.y$=1, task τ_i has a crossover job and the scheduler saves its executed time before mode switch point in Exc_i. Note that, $n_i(l)=\lfloor(l+T_i-D_i)/T_i\rfloor$ is the number of jobs of the task τ_i in interval l. Also, $n_i(w)$ and $n_i(q)$ are the number of jobs of each task in normal and overrun modes, respectively. Therefore, $n_i(w)$+$n_i(q)$=$n_i(l)$. We can find the number of overrun jobs for each task in duration q as follows:

$$n_i(q)=\begin{cases} \left\lfloor\dfrac{(t^e-t^o)+T_i-D_i}{T_i}\right\rfloor+1, & if \quad x.y=1 \\ \left\lfloor\dfrac{(t^e-t^o)+T_i-D_i}{T_i}\right\rfloor, & if \quad x.y=0 \end{cases} \quad (46)$$

Here, crossover job is considered as an overrun job. The crossover job has different execution time and frequency in the NR and OV modes. Hence, If $x.y$=1 the DBF requirement for crossover job (called n_{OV}) is computed as follows:

$$n_{OV}=(Exc_i.\frac{f_{max}}{f_i})+(W_i^{HI}-Exc_i) \quad (47)$$

The DBF of HC and replica tasks in OV mode are as follows:

$$DBF^{OV}_{\tau_i\in\Gamma_{m_k}(HC)}(\tau_i,l)=\begin{cases} n_i(w).\dfrac{f_{max}}{f_{i_{NR}}}.W_i^{LO}+n_i(q).W_i^{HI}, & if \quad x.y=0 \\ n_i(w).\dfrac{f_{max}}{f_{i_{NR}}}.W_i^{LO}+n_{OV}+(n_i(q)-1).W_i^{HI}, & if \quad x.y=1 \end{cases} \quad (48)$$

$$DBF^{OV}_{\tau_i\in\Gamma_{m_k}(B)}(\tau_i,l)=\sum_{j=1}^{j=\left\lfloor\frac{l+T_i-D_i}{T_i}\right\rfloor}(overlap\_time_j) \quad (49)$$

Jobs of LC tasks are initially released based on T_i^{NR} period, and their frequency for applying DVFS is f_{NR}. After overrun, jobs of LC tasks on core m_k are released according to T_i^{OV}, and the frequency scaling factor changes to f_{OV}, which may be different from their normal frequencies due to various slack reclamation in these two modes.

$$DBF^{OV}_{\tau_i\in\Gamma_{m_k}(LC)}(\tau_i,l)=(\left\lfloor\frac{(t^o-t^s)+T_i^{NR}-D_i}{T_i^{NR}}\right\rfloor.\frac{f_{max}}{f_{i_{NR}}}+\left\lfloor\frac{(t^e-t^o)+T_i^{OV}-D_i}{T_i^{OV}}\right\rfloor.\frac{f_{max}}{f_{i_{OV}}}).W_i^{LO} \quad (50)$$

The DBF of each core in Epoch 2 is computed as follows:

$$DBF^{OV}_{\tau_i\in\Gamma_{m_k}}(\tau_i,l)=\sum_{\tau_i\in\Gamma_{m_k}(HC)} DBF^{OV}_{\Gamma_{m_k}(HC)}(\tau_i,l)+\sum_{\tau_i\in\Gamma_{m_k}(B)} DBF^{OV}_{\Gamma_{m_k}(B)}(\tau_i,l)+\sum_{\tau_i\in\Gamma_{m_k}(LC)} DBF^{OV}_{\Gamma_{m_k}(LC)}(\tau_i,l)\le l \quad (51)$$

Epoch 3: When a job of an HC task τ_i encounters a fault at time t^f (where t^s<t^f< t^e), the replica(s) of the faulty job will be executed completely. Hence, the cores which contain its replicas are activated for updating their DBF computation. In this condition, all previous jobs of a faulty task are executed correctly and their replicas are dropped, and only their execution time overlap is calculated in DBF (Eq. 48).

$$DBF^{FO}_{\tau_i\in\Gamma_{m_k}(HC)}(\tau_i,l)=\max\{\left\lfloor\frac{l+T_i-D_i}{T_i}\right\rfloor,0\}.\frac{f_{max}}{f_i}.W_i^{LO} \quad (52)$$

$$DBF^{FO}_{\tau_i\in\Gamma_{m_k}(B)}(\tau_i,l)=\sum_{j=1}^{\left\lfloor\frac{(t^f-t^s)+T_i-D_i}{T_i}\right\rfloor} overlap\_time_j+\sum_{j=1}^{\left\lfloor\frac{(t^e-t^f)+T_i-D_i}{T_i}\right\rfloor} W_j^{LO} \quad (53)$$

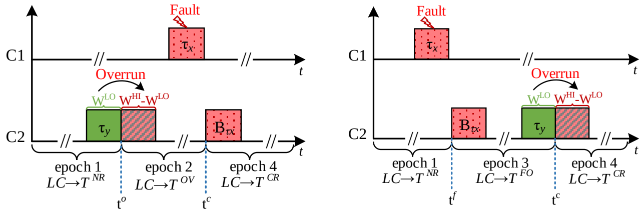
$$DBF^{FO}_{\tau_i\in\Gamma_{m_k}(LC)}(\tau_i,l)=(\left\lfloor\frac{(t^f-t^s)+T_i^{NR}-D_i}{T_i^{NR}}\right\rfloor.\frac{f_{max}}{f_{i_{NR}}}+\left\lfloor\frac{(t^f-t^e)+T_i^{FO}-D_i}{T_i^{FO}}\right\rfloor.\frac{f_{max}}{f_{i_{FO}}}).W_i^{LO} \quad (54)$$

A task set is schedulable in the FO mode on core m_k if and only if for $\forall l\underline{>}$ 0:

$$DBF^{FO}_{\tau_i\in\Gamma_{m_k}}(\tau_i,l)=\sum_{\tau_i\in\Gamma_{m_k}(HC)} DBF^{FO}_{\Gamma_{m_k}(HC)}(\tau_i,l)+\sum_{\tau_i\in\Gamma_{m_k}(B)} DBF^{FO}_{\Gamma_{m_k}(B)}(\tau_i,l)+\sum_{\tau_i\in\Gamma_{m_k}(LC)} DBF^{FO}_{\Gamma_{m_k}(LC)}(\tau_i,l)\le l \quad (55)$$

In DBF based approach, according to the online conditions, the scheduler can provide better service levels for LC tasks, and execute them more frequently than T_i^{OV} or T_i^{FO}. Similar to [35], the scheduler introduces a set of early release points between the desired period and T_i^{OV} or T_i^{FO} and chooses one of the early release points that passes DBF test as a new release point of LC tasks ($T_i^{runtime}$ in Fig. 1).

Epoch 4: As soon as a core experiences both the fault and overrun situations at time t^c, all LC tasks on the core should be executed with T_i^{CR}. The scheduler considers two consecutive mode changes, i.e. from the NR to the OV and then to the CR (Fig. 3 (a)), and from the NR to the FO and then to the CR (Fig. 3 (b)). For example, in Fig. 3 (a) one of the jobs of the HC task τ_y on C2 overruns at the time t^o and C2 switches to OV mode. Also, one of the jobs of τ_x encounters a fault in C1 and its replica in C2 ($B_{\tau x}$) should be executed. Therefore, C2 enters the critical mode at switch point time t^c and all remaining LC jobs on this core are executed with



(a) (b)

Fig. 3. Overview of Epoch 4. a) Mode switch from NR to FO to CR, b) Mode switch from NR to OV to CR.

T_i^{CR} period. Hence, DBF computations from switch point time t^c to the end of the interval are as follows:

$$DBF^{CR}_{\tau_i \in \Gamma_{m_k}(HC)}(\tau_i, t^e - t^c) = \max\{\left\lfloor \frac{(t^e - t^c) + T_i - D_i}{T_i} \right\rfloor, 0\}.W_i^{HI} \tag{56}$$

$$DBF^{CR}_{\tau_i \in \Gamma_{m_k}(B)}(\tau_i, t^e - t^c) = \max\{\left\lfloor \frac{(t^e - t^c) + T_i - D_i}{T_i} \right\rfloor, 0\}.W_i^{LO} \tag{57}$$

$$DBF^{CR}_{\tau_i \in \Gamma_{m_k}(LC)}(\tau_i, t^e - t^c) = \max\{\left\lfloor \frac{(t^e - t^c) + T_i^{CR} - D_i}{T_i^{CR}} \right\rfloor, 0\}.\frac{f_{max}}{f_i}.W_i^{LO} \tag{58}$$

A task set is schedulable in the CR mode on core m_k if:

$$DBF^{CR}_{\tau_i \in \Gamma_{m_k}}(\tau_i, l) = \sum_{\tau_i \in \Gamma_{m_k}(HC)} DBF^{CR}_{\Gamma_{m_k}(HC)}(\tau_i, l) + \sum_{\tau_i \in \Gamma_{m_k}(B)} DBF^{CR}_{\Gamma_{m_k}(B)}(\tau_i, l) + \sum_{\tau_i \in \Gamma_{m_k}(LC)} DBF^{CR}_{\Gamma_{m_k}(LC)}(\tau_i, l) \le (t^e - t^c) \tag{59}$$

4.5 Scheduling Algorithm

Algorithm 3: The task scheduling of our LETR-MC scheme

Inputs: *Γ*, *PFH*, Scaling factors, *M*, Available V-f levels ,different execution modes *X*.

Output: The task scheduling on each core.

```
BEGIN:
1.  Call(Algorithm 1);                                   //Task mapping
2.  Call(Algorithm 2);                                   //Compute SF
3.  for each core $m_k$ in $M$ do
4.      while (all tasks in $\Gamma_{m_k}$ are not scheduled) do
--      //Event: A job of $\tau_i$ ($J_{ij}$) is released at time $t$ on the core $m_k$
5.          if ($Q_{ASAP} \neq \emptyset$ & $Q_{ALAP} = \emptyset$) then
6.              if $m_k$ is busy then                    // $J_{mn}$ is executed
7.                  if $J_{ij}$ is HC or LC then          //ASAP tasks
8.                      if $J_{mn}$ is LC or HC
9.                          if ($d(J_{ij}) > d(J_{mn})$) then //d: deadline of the job
10.                             ER-EDF($J_{ij}$);
11.                         else
12.                             ER-EDF($J_{mn}$);
13.                         end if
14.                     else // $J_{mn}$ is replica
15.                         ER-EDF($J_{ij}$);//shift $J_{mn}$ toward its release time
16.                     end if
17.                 end if
18.             else // core $m_k$ is idle
19.                 ER-EDF($J_{ij}$);
20.             end if
21.         elseif ($Q_{ASAP} = \emptyset$ & $Q_{ALAP} \neq \emptyset$) then
22.             Schedule $J_{ij}$ in free time slots from $d(J_{ij})$ to $r(J_{ij})$ ;
23.         else // $Q_{ASAP} \neq \emptyset$ & $Q_{ALAP} \neq \emptyset$
24.             $J_{ij}^S$= $Q_{ASAP}$.select();          //Sort jobs based on their d
25.             $J_{xy}^L$= $Q_{ALAP}$.select(); // Select the first J with earliest d
26.             if d($J_{ij}^S$)< d($J_{xy}^L$) then
27.                 goto line 6;
28.             else
29.                 Schedule $J_{xy}^L$ in free time slots from d($J_{xy}^L$) to r($J_{xy}^L$);
30.             end if
31.         end if
32.     end while
33. end for
34. Update.SF;                                   // Update α, β, and θ
35. Check.DBF;          // for each core in each execution mode X
36. Apply offline DVFS;           //SUF and DBF-based heuristics
END
```

Our proposed LETR-MC scheme consists of an offline and an online phase. Algorithm 3 shows the pseudo code of the task scheduling mechanism of LETR-MC. Algorithm 1 is called for task mapping in line 1. Then, in line 2, Algorithm 2 is called to compute the scaling factors of LC task periods. Afterward, tasks on each core should be scheduled based on our proposed algorithm. We assume that each type of tasks has a preference to indicate how the jobs of its tasks will be executed. For this purpose, we adapt the partitioned ER-POED (ER-Preference-Oriented Earliest-Deadline) scheduling algorithm on multicore fault-tolerant MCSs [25]. The scheduler divides tasks into two different groups, i.e. as soon as possible (ASAP) and as late as possible (ALAP) tasks. The scheduler tries to execute ASAP tasks before the ALAP ones. In this work, we consider HC and LC tasks as ASAP, and replicas as ALAP, respectively. The algorithm iterates over all cores (line 3) and goes on until all the tasks on each core are scheduled. The scheduler checks ASAP (Q_{ASAP}) and ALAP (Q_{ALAP}) queues on each event. If the Q_{ALAP} is empty while Q_{ASAP} is not, the scheduler selects the task with the earliest deadline in Q_{ASAP} for scheduling. The preemption scenarios, when the core is busy due to the execution of another job (J_{mn}), are shown in lines 6-17. If J_{mn} is an HC or an LC job, ER-EDF chooses one of them for execution. However, if J_{mn} is a replica task, the replica is shifted back toward its release time to produce free time slots for executing J_{ij}. If there are more than one task with the same deadline in ASAP queue, the execution order of individual tasks with the same preferences (ASAP tasks) is distinguished with ER-EDF. If Q_{ASAP} is empty while Q_{ALAP} is not, a replica with the earliest deadline is chosen for scheduling. The scheduler defines how much the execution of replica task can be delayed while still meeting its deadlines by executing at maximum frequency. Here, the scheduler finds slack times (equal to WCET of the replica task) from the deadline of selected replica task toward its release time, and schedules the replica in these free time slots (lines 21-22). If there are tasks in both of the Q_{ASAP} and Q_{ALAP}, if the deadline of an ASAP job (J_{ij}^S) is smaller than ALAP one (J_{xy}^L), the algorithm schedules J_{ij}^S in lines 23-27. Otherwise, the J_{xy}^L is scheduled in line 29. After scheduling all tasks based on the worst-case scenario, the scheduler updates scaling factors of LC tasks. For example, in order to update a, the scheduler cancels the overrun part of all HC tasks, i.e. W_i^{HI}-W_i^{LO}, and drops the non-overlapping part of the replica tasks. Hence, LC tasks will be scheduled in these released slacks. During this process, the scheduling of the HC tasks is left untouched, i.e. HC jobs are executed exactly based on the times that the scheduler determines in the worst-case scenario. For updating β, only the released slacks from the cancellation of non-overlapping part of replicas are used to schedule LC tasks more frequently. For updating θ, only the released slacks from the cancellation of the overrun parts of all HC tasks (W_i^{HI}-W_i^{LO}) are used to schedule LC tasks more frequently.

By scheduling HC tasks as soon as possible and their corresponding replicas as late as possible, it is sufficient to complete only one copy of each task successfully. So, if one replica completes and no fault is detected, we can cancel other versions of that task immediately to avoid further energy consumption. Also, in this paper we exploit partitioned approach, i.e. each task is statically assigned to a single core, and migration is not allowed due to its non-negligible overheads among cores [56].

The offline DVFS is applied to the final schedule of normal execution mode based on our proposed method in Section 4-6. After applying DVFS, the scheduler updates execution time overlap between primary tasks and their corresponding replicas, and obtains new scaling factors if necessary.

4.6 Energy Minimization Problem

We define the energy minimization problem for a set of periodic mixed-criticality tasks on multicore platforms that exploits task replication technique (task-level reliability targets). The number of required replicas and the frequency of tasks should be properly determined, while the timing constraints of HC tasks and replicas are met, and acceptable service levels for LC tasks are simultaneously guaranteed.

1) *Formal Problem Modeling*: The goal of reliability-aware energy minimization problem on mixed-criticality multicores is to minimize the total energy consumption in all operation modes (called X in Table 2). LC tasks are executed with W^{LO}/f_i in all execution modes. However, HC tasks are executed with W_i^{LO}/f_i in NR and FO modes, and with W_i^{HI} in OV and CR modes. The objective can be written as:

$$\text{Minimize:} \quad \sum_{k=1}^{c} \sum_{\forall \tau_i \in \Gamma_{m_k}} \sum_{j=1}^{h_{\tau_i}} (\varphi + (\frac{f_{ij}}{f_{\max}})^2) \times W_{ij}^{X} \tag{60}$$
$$\forall X \in \{NR, OV, FO, CR\}$$

Constraints of the energy minimization problem are as follows. Task's frequency levels in each mode are in the range of the minimum and maximum core frequencies:

$$\text{s.t.} \quad \forall \tau_i, \;\; f_i \in [f_{\min}, f_{\max}] \tag{61}$$

The WCET of a task τ_i at frequency level f_i in criticality level X should not exceed the task timing constraint (D_i).

$$\text{s.t.} \quad \forall \tau_i, \; \forall X, \; \forall f_i, \; \frac{W_i^X}{f_i} \le D_i \tag{62}$$

Total DBF of each core in each execution mode should be less than the given interval l:

$$\text{s.t.} \quad \forall X, \; \forall M, \; \forall l \ge 0, \; DBF_{\Gamma_{m_k}}^{X} \le l \tag{63}$$

Each task's reliability should meet the reliability target according to PFH in DO-178B standard, as follows:

$$\text{s.t.} \quad \forall \tau_i, \; R_{\tau_i} \ge R_{target} \tag{64}$$

Frequency scaling affects the WCET of a task and consequently its reliability and PoF. Therefore, the required number of replicas for each task (r_i) to meet its reliability target in frequency level f_i should be considered in energy minimization. Also, the scheduler should decide about using slacks for energy reduction or improving the QoS of LC tasks. In addition, the task set should be schedulable in each execution mode with computed scaling factors (α, β, θ, γ) even after applying DVFS.

$$\text{s.t.} \quad \forall \alpha, \beta, \theta, \gamma: \; 1 \le \alpha \le \beta, \theta \le \gamma \tag{65}$$

Optimally solving the energy minimization of multicore systems without considering the reliability requirement is NP-hard [9], [11]. Therefore, solving this problem in fault-tolerant mixed-criticality multicore systems is also NP-hard; hence, we develop a heuristic to manage energy consumption.

2) *Proposed Heuristics:* Offline DVFS is applied to the normal execution mode of the system. In the offline phase, the scheduler uses static slacks to apply DVFS for energy reduction. At first, f_{max} is assigned to all the tasks as their execution frequency. Then, selection of tasks for reducing the frequency and using the static slack in offline phase is done according to Smallest-Utilization-First (SUF), and the proposed DBF-based heuristics. Although executing a task under a lower frequency reduces energy consumption, the task takes longer time to complete, which may violate the timing constraints. Furthermore, lowering task's execution frequency increases the system transient fault rate, which in turn increases the probability of executing all replicas, or increments the number of required replicas. The scheduler should consider timing constraints, reliability and QoS analysis simultaneous in energy management because a decision made in any one affects the others. Algorithm 4 shows the pseudo-code of offline DFVS and the interplay of energy, reliability, and schedulablity analysis in energy management.

In SUF, HC and LC tasks are sorted in a queue, based on their utilization in increasing order in line 2. The scheduler selects the first task in the queue to scale down its frequency, in line 4. First, the scheduler extracts the amount of static slack times and finds the minimum frequency that the selected task can use it, in lines 5-6. If the schedulability constraints are met in DBF analysis of the normal mode, and if the reliability target is met, this frequency is assigned to the selected task in lines 8-9. If timing constraints in DBF analysis are not satisfied, the frequency is scaled one level up in lines 10-11. However, if the reliability target is not met, the scheduler checks whether more replicas can be properly mapped and scheduled. If it can, this frequency is set in lines 12-16. Otherwise, it scales frequency one level up in line 18. The scheduler repeats the previous steps until no task's execution frequency can be scaled down, and updates the static slack in each iteration.

In the DBF-based heuristic, there might be more than one task that can be scaled down; the scheduler chooses the task which has more impact on total energy reduction. After extracting the amount of static slack times (line 24), while there is a task in a Γ(HL) and there is static slack, the algorithm iterates among all the tasks, except replicas, on each core. In each iteration, it selects one of the tasks (line 27). The scheduler receives the WCET of the task and the amount of static slacks, and finds task's minimum acceptable execution frequency based on the QoS, reliability and timing constraints in equations 60-65 (line 28).

If the reliability of the selected task decreases after lowering down the frequency level, the scheduler tries to increase the number of replicas. If new replicas can be properly mapped and scheduled on multicores, the scheduler sets the execution frequency of the selected task, otherwise, it scales up the frequency level of the selected task, and algorithm goes on (lines 30-37). If lowering down the frequency violates the schedulability in DBF analysis, the scheduler scales the frequency one level up and iterates the above steps in lines 38-39. If the reliability and certification constraints are met, the scheduler assigns the computed frequency to the selected task, and computes the energy efficiency factor (*EEF*) for each task based on the minimum acceptable execution frequency (lines 41-42) as

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2907846, IEEE Transactions on Parallel and Distributed Systems

follows:

$$EEF_{\tau_i} = \frac{NE_{\tau_i}(before\_DVFS) - NE_{\tau_i}(after\_DVFS)}{NE_{\tau_i}(before\_DVFS)} \qquad (66)$$

After computing the *EEF* for all the tasks, the scheduler finds the task with maximum *EEF*, which decreases energy consumption the most, in line 46. Afterward, the scheduler sets the frequency of the selected task in line 47, updates static slacks in line 48, removes the selected task from the for loop in line 49, returns the frequency of all remaining tasks to the maximum value in line 50, and repeats the above steps for all remaining tasks. The above steps are repeated until the *EEFs* of all tasks become zero, or there are no more static slacks. During applying offline DVFS special care should be taken. One point is that the released slack from cancellation of replicas in the normal mode cannot be used for applying DVFS to HC tasks, because replicas may require to be executed, and only LC tasks can occupy their places. Whenever a fault or an overrun occurs, LC tasks can be extended and replicas are executed. The other point is that all the available static slacks cannot be used for applying DVFS to HC tasks. Regarding the overrun occurrence, only slack *S* (*S*=Available slack-(W_i^{HI}-W_i^{LO})) can be used for applying DVFS to HC tasks because by applying DVFS, each task τ_i is executed up to W_i^{LO}/f_i, and overrun is detected at the end of this time. Therefore, by reserving some of the available static slack time for the overrun occasion, if an overrun occurs after applying DVFS, there is enough time for executing the remaining parts of the job. It should be noted that after applying DVFS, the execution time overlap between the primary tasks and their corresponding replicas will be updated and considered in DBF analysis, i.e., applying DVFS and computing the execution time overlap between primary tasks and replicas are performed simultaneously.

Algorithm 4: Offline DVFS, interplay of energy, reliability, and certification

```
Inputs: The task scheduling on each core, Γ, PFH, Scaling factors, M, Available V-f levels.
Output: Acceptable V-f level for each task.
BEGIN:
-- SUF hueristic
1.  Γ(HL)= Γ(HC)+Γ(LC);                    // HC & LC tasks set
2.  Γ(HL).sort();      //Sort HC and replica tasks w.r.t utilization
3.  for all tasks in Γ(HL) do
4.    τi = Γ(HL).select();  //Select the task with smallest utilization
5.    SS← Extract_StaticSlack();
6.    $f_{\tau_i}$← Determine_min_freq(Wi, SS);
7.    while (freq is not assigned) do
8.      if ($R_{\tau_i(f_i)} \geq R_{target}$) and (DBF≤l)
9.        assign $f_{\tau_i}$ to τi;
10.     elseif (DBF>l)
11.       $f_{\tau_i}$= UP_Scale($f_{\tau_i}$);
12.     elseif ($R_{\tau_i(f_i)} < R_{target}$) and (DBF≤l)
13.       increase the # of replicas;
14.       Update_DBF;
15.       if (DBF≤l)
16.         assign $f_{\tau_i}$ to τi;
17.       else
18.         $f_{\tau_i}$= UP_Scale($f_{\tau_i}$);
19.       end if
20.     end if
21.   end while
22. end for
-- DBF-based hueristic
23. Γ(HL)= Γ(HC)+Γ(LC);                    // HC & LC tasks set
24. SS← Extract_StaticSlack();
25. While ((Γ(HL)≠ ∅) or (SS≠ ∅)) do
26.   for all tasks in Γ(HL) do
27.     τi = Γ(HL).select();
28.     $f_{\tau_i}$← Determine_min_freq(Wi, SS);
29.     while (freq is not assigned) do
30.       if ($R_{\tau_i}(f_{\tau_i})$<$R_{target}$) and (DBF≤l)
31.         increase the # of replicas;
32.         Update_DBF;
33.         if (DBF≤l)
34.           assign $f_{\tau_i}$ to τi;
35.         else
36.           $f_{\tau_i}$= UP_Scale($f_{\tau_i}$);
37.         end if
38.       elseif (DBF>l)
39.         $f_{\tau_i}$= UP_Scale($f_{\tau_i}$);
40.       elseif ($R_{\tau_i(f_i)} \geq R_{target}$) and (DBF≤l)
41.         assign $f_{\tau_i}$ to τi;
42.         Compute_EFF$\tau_i$  // w.r.t Eq. 66
43.       end if
44.     end while
45.   end for
46.   j=determine_max_EEF(Γ(HL));
47.   $f_{\tau_j}$=set_freq();
48.   SS← Update_StaticSlack();
49.   Γ(HL)= Γ(HL)-{$\tau_j$};
50.   $f_{\Gamma(HL)}$= set_freq($f_{max}$);
51. end While
END
```

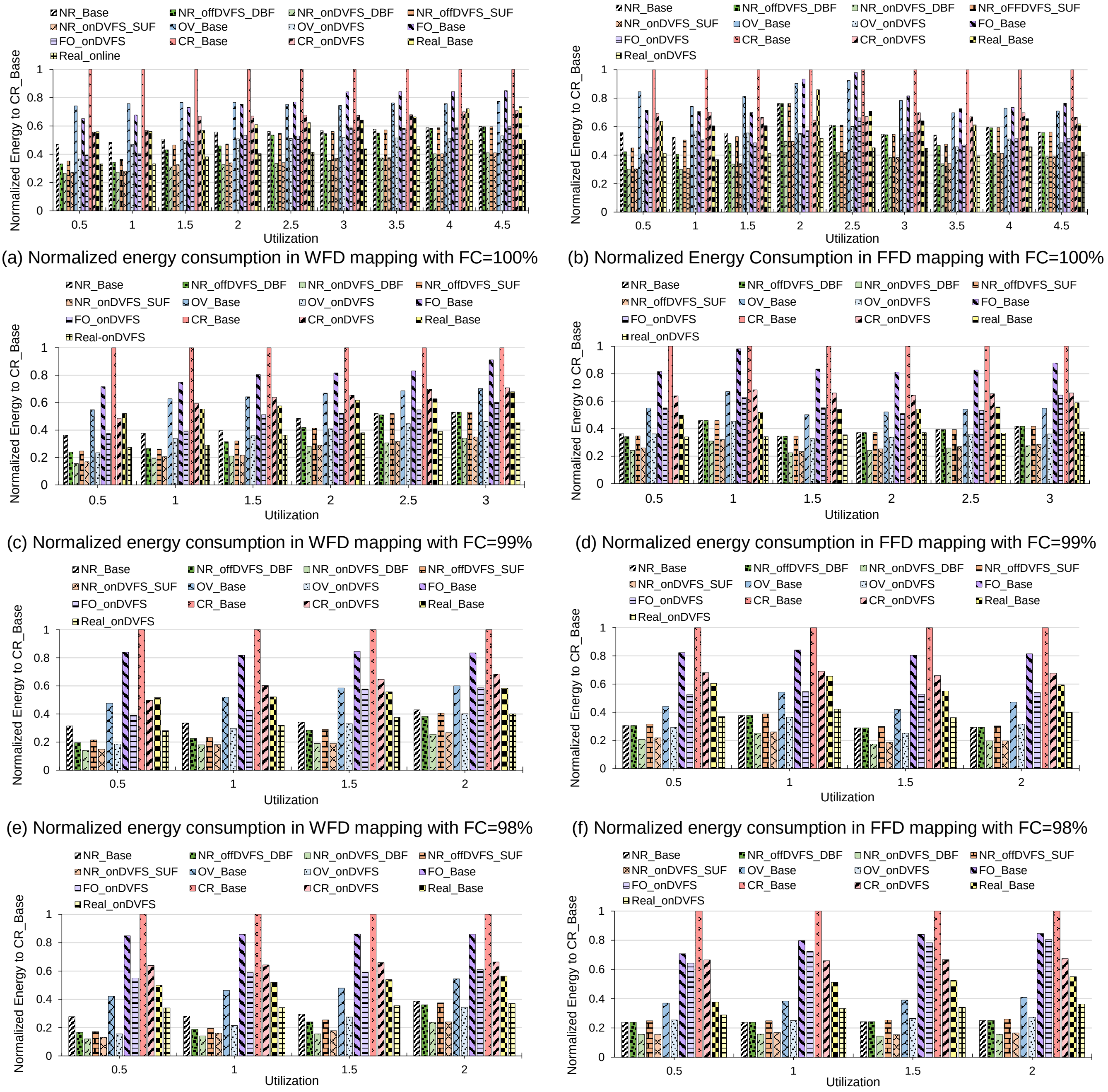
Our offline approach can be used in conjunction with an online energy manager which uses dynamic slacks to apply DVFS and DPM during runtime for further energy savings. Dynamic slacks are released due to replica cancellation or early completion of tasks. In runtime, initially the amount of released dynamic slack is determined. If the idle time of the core is longer than Δ_{th}, it is beneficial for the system to go into sleep mode to reduce energy consumption. Otherwise, online lightweight job-level DVFS is used for energy saving. It exploits greedy slack assignment, i.e. all the released slack from the current job will be used to further lower down the *V-f* level of the next job whenever it is possible to do so. In order to reduce the overhead of DVFS computations, inserting more replicas is not permitted. Therefore, the V-f level is lowered down to a level where no more replicas need to be inserted. Otherwise, the slack will be transferred to the next jobs or it may even not be used. DVFS transition overhead may be significant or negligible based on how often the DVFS is applied. In modern microprocessors, changing the DVFS setting is rather frequent in response to rapid changes in the application behavior. The DVFS transition delay overhead in high-end Intel Core2 Duo E6850 is between 9 to 62 us, for embedded Samsung Exynos 4210 processor based on ARM Cortex-A9 core is about 11 to 18 us, and this overhead for the TI MSP430 microcontroller that is used for ultralow-power embedded systems is about 10 to 145 us [66]. Also, by using ultra-fast voltage regulators, where V_{dd} switching is moved into the sub-micron regime like the Intel Haswell CPU, switching between voltage levels takes place in less than 1us [65]. Therefore, since the online DVFS overhead is in the order of micro second (us), and the order of execution of tasks in the systems is millisecond (ms) [2], [15], the overhead of applying DVFS is negligible and can be considered as a part of the task's WCET. Energy consumption of the memory unit is out of the scope of this paper. However, there are data recompuatation techniques [59-64] that can be used for energy reduction of the memory. Finally, in order to explain how our proposed method works, there is an illustrative example which is presented in the appendix.

5 Results and Discussion

In this section, we perform extensive simulations to present the effectiveness of our proposed LETR-MC method from the perspective of energy saving, reliability and QoS in different operation modes.

5.1 Experimental Setup

Due to lack of benchmark packages for MCSs, similar to [18], [20], [22-23], [25-28], [30-31], [33-37], [40-43], [67], we evaluate our proposed scheme using synthetic task sets. The UUnifast algorithm is used to generate utilization for n tasks $\Gamma=\{\tau_1, \tau_2, \ldots, \tau_n\}$ with total utilization equal to U [28]. The UUnifast algorithm is proposed by bini and buttazzo [44] to generate utilizations of a task set to study uniprocessor scheduling, which has the lowest complexity among all task generation algorithms. The generated utilizations for HC and LC tasks correspond to their high-level WCETs (i.e. u_i(HC,H)) and low-level WCETs (i.e. u_i(LC,L)),



(a) Normalized energy consumption in WFD mapping with FC=100%

(b) Normalized Energy Consumption in FFD mapping with FC=100%

(c) Normalized energy consumption in WFD mapping with FC=99%

(d) Normalized energy consumption in FFD mapping with FC=99%

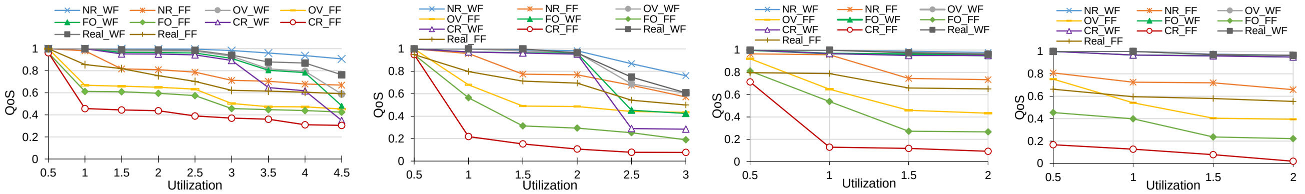
(e) Normalized energy consumption in WFD mapping with FC=98%

(f) Normalized energy consumption in FFD mapping with FC=98%

(g) Normalized energy consumption in WFD mapping with FC=95%

(h) Normalized energy consumption in FFD mapping with FC=95%

Fig. 4: Normalized energy consumption of different operation modes based on WFD and FFD mappings, with different fault coverage values.



(a) Fault coverage=100% (b) Fault coverage=99% (c) Fault coverage=98% (d) Fault coverage=95%

Fig. 5. QoS of LETR scheme in different operation modes and different fault coverage values.

respectively. We define the P^{HI} factor which is the probability that the generated task is HC. The periods of tasks are randomly selected from the set T={10, 20, 40, 50, 100, 200, 400, 500, 1000} ms [20]. Hence, the W_i^{HI} for an HC task is computed according to $W_i^{HI}=T_i.u_i(\text{HC,H})$, and for LC tasks $W_i^{HI}=W_i^{LO}$. However, for HC tasks $W_i^{LO}=\mu.W_i^{HI}$, where μ is a random value in the range of [0.3, 0.5]. The PFH level of all HC tasks is selected from levels A, B, and C in DO-178B standard. Hence, the target reliability and the number of required replicas for each HC task is computed based on the selected PFH level. We consider a multicore platform, and the available frequencies for each core are set as F={0.6, 0.7, 0.8, 0.9, 1}. In the online phase, tasks are executed with actual execution time which varies between 70 to 100 percent of their WCET.

5.2 Experimental Results and Discussions

We evaluated the energy consumption, reliability and QoS of our proposed method in different operation modes in a quad-core platform based on WFD and FFD mapping, and the results are shown in Fig. 4 and Fig. 5, respectively. Utilization U was varied from 0.5 to 4.5 with steps of 0.5, and in each utilization point, 50 task sets were synthetically generated. The results are reported as the averages of 100 repetitions of the experiment at each utilization point. The P^{HI} factor is equal to 0.5. HC, and LC tasks are selected from levels B and D, respectively. Therefore, the average energy consumption and QoS are computed with preserving the target reliability in all experiments. Also, in Fig. 4 and Fig. 5 the energy consumption and QoS of our proposed method for different fault coverage values (FC) are reported. In these figures, the energy consumption and QoS of each operation mode are computed based on the worst-case scenario of that mode, i.e. the energy consumption of OV and FO modes are reported where all the HC tasks overrun, or all HC tasks encounter faults and all replicas are executed completely with maximum frequency, respectively. In addition to the worst-case scenario, the energy consumption is analyzed based on the actual case (Real_WF and Real_FF). In the actual-case, faults are injected into the system with Poisson distribution, and HC tasks overrun randomly. In Fig. 4, the normalized energy consumption before applying offline DVFS (X_Base, where X represents the operation mode including NR, OV, FO, and CR), after applying offline DVFS technique based on DBF and SUF (X_OffDVFS), and after applying online DVFS (X_OnDVFS) are shown. By increasing the utilization, the energy consumption is increased, and the DBF-based DVFS lowers the energy more than the SUF one. Since transient faults and overrun are rare in nature, LETR-MC achieves further energy reduction at runtime beyond what is achieved through the offline part of LETR-MC at design-time.

Also, decreasing the fault coverage value reduces the reliability of the system according to equations 6-8. Hence, the schedulability of the task sets will be decreased, because more replica tasks are needed to schedule to achieve the given reliability target. Therefore, the lower fault coverage value leads to lower schedulability. Fig 5. represents the QoS of our proposed LETR-MC method in different operation modes with WFD and FFD mappings and different fault coverage values. The QoS is computed based on the fraction of the number of remaining schedulable LC jobs in each operation mode to the total original number of jobs based on the desired period of LC tasks. As it is clear, by increasing the utilization, QoS is decreased to keep the system schedulable. Also, due to the load balanced mapping in the WFD, it out performs FFD in all operation modes, i.e. WFD can preserve higher percentage of LC jobs in each operation mode in comparison with FFD mapping. Meanwhile, the normal operation mode with WFD mapping (NR_WF) has the highest QoS, and the critical operation mode with FFD mapping (CR_FF) has the lowest. The results of actual-case scenario (Real_WF and Real_FF) is close to the QoS of Normal mode (NR_WF and NR_FF). Also, lower fault coverage leads to lower schedulability and also lower QoS in both the WFD and FFD mapping, because replica tasks reserve the execution time of the cores to satisfy the given reliability target.

Fig. 6 illustrates both energy consumption and QoS of the LETR-MC method in different operation modes with WFD and FFD mappings and fault coverage 100%. In this figure, the energy consumption is normalized to that of the scenario where all tasks are executed at f_{max}, HC tasks are executed with W^{HI} and the jobs of LC tasks are released based on their desired period. This figure shows that in each utilization point, the energy consumption of the FFD mapping is close to WFD one. However, the QoS of FFD in that utilization point is less than the WFD one. Hence, FFD mapping consumes less energy at the expense of lowering down the number of executable LC jobs.

We compared our proposed method from the QoS point of

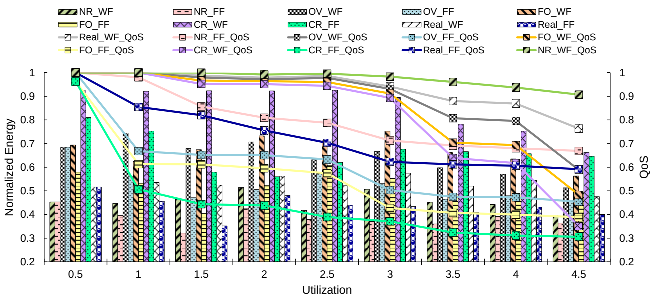


Fig. 6. Analyzing the energy consumption and QoS of LETR scheme in different operation modes.

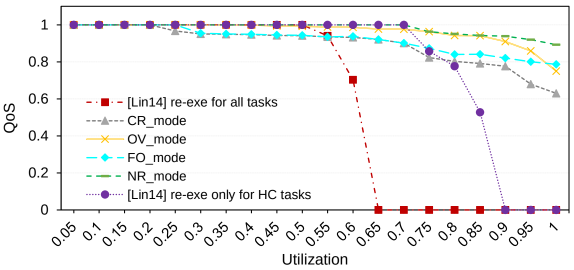


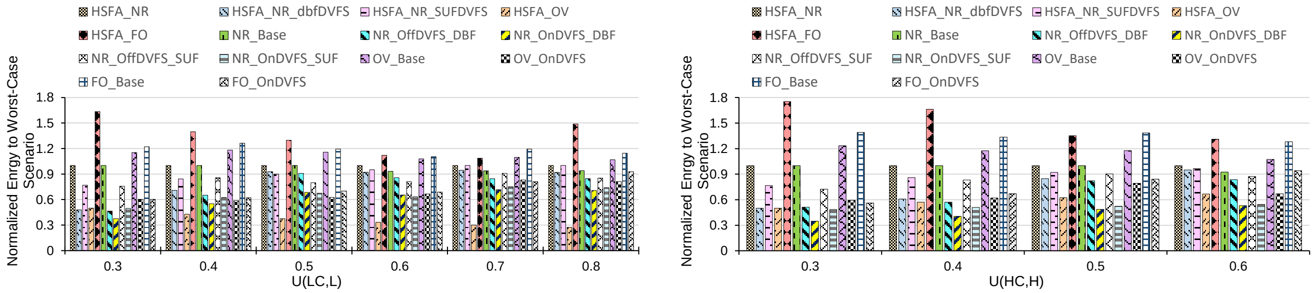
Fig. 7. Comparing the QoS of LETR scheme and [27].

view with [27] in Fig. 7. The reference [27] proposes an offline algorithm which improves the QoS of EDF-VD algorithm by trying to save as much LC tasks as possible in overrun mode. Without considering any PFH level for tasks, it assigns one re-execution to each HC and LC task. Therefore, all HC and LC tasks and their re-executions must be schedulable in normal mode. However, in overrun mode, HC tasks and their re-executions must be schedulable, and for LC tasks the algorithm tries to find schedulable set of LC tasks based on the scaling factor in EDF-VD scheduler. Afterward, it tries to reserve re-executions for LC tasks that are schedulable in overrun mode. In this set of experiments, the utilization of a single-core platform is varied from 0.05 to 1 with steps of 0.05. In each utilization point 50 task sets are synthetically generated and the experiments are repeated 100 times. The P^{HI} factor is equal to 0.4. In the first scenario, we consider that all HC and LC tasks need one re-execution. Therefore, the proposed method in [27] finds the number of LC jobs that can be schedulable in overrun mode. As it is shown in Fig. 7, this method is not schedulable after utilization point 0.65. In the second scenario, we modified their proposed method considering that LC tasks do not need re-execution. Hence, the schedulability of their method is improved. However, LETR-MC is schedulable in all utilization points, also LETR-MC can preserve higher number of jobs than [27] in overrun mode. It should be noted that the reference [27] considers two operation modes, i.e. normal and overrun. However, in this set of experiments, we reported the QoS of LETR-MC in FO and CR mode in addition to NR and OV modes. By considering different operation modes and proposing MEMC task model, the LETR-MC method can provide higher guaranteed QoS.

We compared the energy consumption of LETR-MC with HSFA algorithm in [36], [37] (Fig. 8). In these set of experiments, each task set has 10 tasks and the experiments are repeated 100 times and the average of these repetitions are reported. In Fig. 8 (a), the high utilization of HC tasks is equal to 0.3 (U(HC,H)=0.3), and the utilization of LC tasks H(LC,L) varies from 0.3 to 0.7 with steps of 0.1 in a single core platform. By increasing the utilization of LC tasks, all methods consume more energy, since in higher utilizations there is less slack time to use for energy saving. In Fig 8 (a) and (b), the LETR-MC method consumes lower energy than HSFA, since it lowers down the number of LC jobs to make the system schedulable. Also, the DBF-based DVFS lowers the energy more than the SUF one. It should be noted that HSFA does not have an online manager, and dynamic slacks are not used for energy saving. However, the LETR-MC method can reduce the energy the most in the online phase. The faulty mode of HSFA algorithm has the highest energy consumption. However, in the FO and OV modes of LETR-MC method, due to the lower number of LC jobs, the energy consumption is less than the worst-case scenario where all HC tasks are executed with W^{HI} and all replicas are executed completely at f_{max}. HSFA uses EDF-VD algorithm and it drops all LC tasks after entering the overrun mode. Hence, this method consumes the least energy in the OV mode. However, LETR-MC executes guaranteed service level of LC jobs in OV mode, which leads in more energy consumption than HSFA. However, by applying the online DVFS, we can mitigate the energy consumption.

6. Conclusion

In this paper, we proposed the LETR-MC scheme that concurrently considers certification, fault-tolerance, energy reduction, and QoS. We used task replication to tolerate fault, and improve QoS of LC tasks. The number of required replicas for each HC task is computed through the proposed formulas. Then, our proposed scheduling algorithm reduces the execution time overlap between the primary tasks and replicas to save more energy by dropping the remaining parts of replicas at the end of correct execution of their primaries. Also, through the presented service guarantee exploration algorithm, we theoretically guarantee an acceptable service level for LC tasks in different operation modes of the system, i.e. normal, overrun, fault-occurrence, and critical. In order to check the schedulability of the proposed method, we analyzed the resource demands of mixed-criticality tasks with the deadline and reliability constraints, energy reduction and QoS guarantee. Finally, we showed that energy consumption can be reduced in the offline and online phases by exploiting static and dynamic slacks, respectively, while the preserving the guaranteed service level for LC tasks.



(a) Comparing energy consumption under various U(LC,L)

(b) Comparing energy consumption under various U(HC,H)

Fig. 8. Comparing the energy consumption of LETR method and HSFA in [37].

REFERENCES

[1] P. Marwedel, "Embedded system design: Embedded systems foundations of cyber-physical systems," *Springer Science & Business Media, 2nd ed.*, 2010.

[2] M. Ansari, S. Safari, A. Y. Khaksar, M. Salehi, and A. Ejlali, "Peak Power Management to Meet Thermal Design Power in Fault-Tolerant Embedded Systems," *IEEE TPDS*, 2018.

[3] "RTCA/DO-178B, Software Considerations in Airborne Systems and Equipment Certification," 1992.

[4] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degree of execution time assurance," *RTSS*, 2007.

[5] B. Hu, K. Huang, P. Huang, L. Thiele and A. Knoll, "On-the-fly fast overrun budgeting for mixed-criticality systems," *Int'l Conf. Embedded Software (EMSOFT)*, 2016.

[6] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougi, "Scheduling real-time mixed-criticality jobs," *IEEE TC*, vol. 61, no. 8, pp. 1140-1152, 2012.

[7] J. Henkel, V. Narayanan, S. Parameswaran, and J. Teich, "Run-time adaption for highly-complex multi-core systems," *Int'l Conf. Hardware/Software Codesign and Syst. Synthesis, (CODES+ISSS)*, 2013.

[8] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on Multi/Many-Core Systems," *DAC*, 2013.

[9] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient DVFS scheduling for mixed-criticality systems," *Int'l Conf. Embedded Software (EMSOFT)*, 2014.

[10] R. Melhem, D. Mosse. and E. Elnozahy, "The interplay of power management and fault recovery in real-Time systems," *IEEE TC*, vol. 53, no. 2, pp. 217-231, 2004.

[11] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE TPDS*, vol. 28, no. 3, pp. 813-825, 2017.

[12] M. A. Haque, H. Aydin, and D. Zhu, "Energy-aware task replication to manage reliability for periodic real-time applications on multicore platforms," International Green Computing Conference, 2013.

[13] F. R. Poursafaei, S. Safari, M. Ansari, M. Salehi, and A. Ejlali, "Offline replication and online energy management for hard real-time multicore systems," *IEEE CSI Symposium on Real-Time and Embedded Syst. and Technologies (RTEST)*, 2015.

[14] R. Sridharan, and R. Mahapatra, Reliability aware power managemnt for dual-processor real-time embedded systems," in Proc. Of the IEEE Design Automation Confrence, 2010.

[15] M. Salehi, A. Ejlali, and B. M. Al-Hashimi, "Two-phase low-energy N-modular redundancy for hard real-time multi-core systems," *IEEE TPDS*, vol. 27, no. 5, pp. 1497-1510, 2015.

[16] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," *RTAS*, 2010.

[17] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. M. Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," *ECRTS*, 2012.

[18] H. Su, and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," *DATE*, 2013.

[19] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," *RTSS*, 2009.

[20] Z. Al-bayati, J. Caplan, B. H. Meyer, and H. Zeng, "A four-mode model for efficient fault-tolerant mixed-criticality systems," *DATE*, 2016.

[21] S. Baruah, and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," *ECRTS*, 2008.

[22] P. Ekberg, and W. Yi., "Bounding and shaping the demand of mixed-criticality sporadic tasks," *ECRTS*, 2012.

[23] H. Su, N. Guan, and D. Zhu, "Service guarantee exploration for mixed-criticality systems," *RTCSA*, 2014.

[24] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems," *Journal of the ACM*, vol. 62, no. 2, pp. 1-33, 2015.

[25] H. Su, D. Zhu, and S. Brandt, "An Elastic Mixed-Criticality Task Model and Early-Release EDF Scheduling Algorithms," *ACM TODAES*, vol. 22, no. 2, pp. 1-28, 2016.

[26] H. Su, D. Zhu, and D. Mosse, "Scheduling algorithms for elastic mixed-criticality tasks in multicore systems," *RTCSA*, 2013.

[27] J. Lin, A. M. K. Cheng, D. Steel, and M. Yu-Chi Wu "Scheduling mixed-criticality real-time tasks in a fault-tolerant system," *RTSS*, 2014.

[28] R. M. Pathan, "Fault-tolerant and real-time scheduling for mixed-criticality systems," *Real-Time Syst.*, vol. 50, no. 4, pp. 509-547, 2014 .

[29] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, "Mixed criticality scheduling in fault-tolerant distributed real-time systems, " *Int'l Conf. on Embedded Syst. (ICES)*, 2014.

[30] P. Huang, H. Yang, and L. Thiele, "On the scheduling of fault-tolerant mixed-criticality systems," *Technical report, Computer Engineering and Networks Laboratory, ETH Zurich*, 2014.

[31] L. Zeng, P. Huang, and L. Thiele, "Towards the design of fault-tolerant mixed-criticality systems on multicores," *Int'l Conf. Compilers, Architectures, and Synthesis of Embedded Syst. (CASES)*, 2016.

[32] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele and R. V. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," *RTAS*, 2016.

[33] V. Legout, M. Jan and L. Pautet, "Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses," *Workshop on Real-Time Mixed Criticality Syst. (ReTiMiCS)*, 2013.

[34] M. Völp, M. Hähnel, and A. Lackorzynski, "Has energy surpassed timeliness? Scheduling energy-constrained mixed-criticality systems," *RTAS*, 2014.

[35] A. Taherin, M. Salehi, and A. Ejlali, "Stretch: Exploiting service level degradation for energy management in mixed-criticality systems," *IEEE CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, 2015.

[36] Z. Lia, C. Guo, X. Hua, and S. Ren, "Reliability guaranteed energy minimization on mixed-criticality systems," *Journal of Syst. and Software*, vol. 112, pp. 1-10, 2016.

[37] Z. Lia, X. Hua, C. Guo, and S. Ren "Empirical study of energy minimization issues for mixed-criticality systems with reliability constraints," *Workshop on Low-Power Dependable Computing (LPDC), Int'l Green Computing Conf. (IGCC)*, 2014.

[38] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," *ACM TECS*, vol. 10, no. 2, pp. 1-27, 2011.

[39] A. Ejlali, B.M. Al-Hashimi, and P. Eles, "A standby-sparing technique with low energy-overhead for fault-tolerant hard real-Time systems," *Int'l Conf. Hardware-Software Codesign and Syst. Synthesis (CODES+ISSS)*, pp. 193-202, 2009.

[40] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," *RTSS*, 2017.

[41] T. Zhang, N. Guan, Q. Deng, and W. Yi, "On the analysis of edf-vd scheduled mixed-criticality real-time systems," *Int'l Symposium on Industrial Embedded syst.*, 2014.

[42] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and Efficient Scheduling of Certifiable Mixedcriticality Sporadic Task Systems," *RTSS*, 2011.

[43] H. Li and S. Baruah, "Global Mixed-Criticality Scheduling on Multiprocessors," *ECRTS*, 2012.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2907846, IEEE Transactions on Parallel and Distributed Systems

[44] E. Bini, and G. Buttazzo, "Measuring the Performance of Schedulability Tests," *Real-Time Syst.*, vol 30, pp 129-154, 2005.

[45] I. Koren and C. M. Krishna, Fault-tolerant systems. Morgan Kaufmann, 2010.

[46] D. Pradhan, Fault Tolerant Computer System Design. Prentice Hall, 1996.

[47] A. Meixner, M. E. Bauer and D. Sorin, "Argus: Low-Cost, Comprehensive Error Detection in Simple Cores," *IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, 2007.

[48] A. Elewi, M. Shalan, M. Awadalla, and E. M. Saad, "Energy-Efficient Task Allocation Techniques for Asymmetric Multiprocessor Embedded Systems," *ACM Trans. on Embedded Computing Syst.*, vol. 13, no. 2s, 2014.

[49] M. Holenderski, "Real-time system overheads: a literature overview," Computer Science Report 08/26, Eindhoven University of Technology, 2008.

[50] J. Anderson, V. Bud, and U. C. Devi, "An EDF-Based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems," *ECRTS*, 2005.

[51] U. C. Devi, and J. Anderson, "Tardiness Bounds for Global EDF Scheduling on a Multiprocessor," *RTSS*, 2005.

[52] H. Cho, B. Rabindran, and E. D. Jensen, "An Optimal Real-Time Scheduling Algorithm for Multiprocessors," *RTSS*, 2006.

[53] A. Burns, "Preemptive priority based scheduling: An appropriate engineering approach," In S. Son, ed., Advances in Real-Time Systems, pp. 225–248. Prentice-Hall, 1994.

[54] F. M. David, J. C. Carlyle, and R. H. Campbell, "Context Switch Overheads for Linux on ARM Platforms," *ACM workshop on Experimental computer science*, 2007.

[55] D. Tsafrif, "The Context-Switch Overhead Inflicted by Hardware Interrupts (and the Enigma of Do-Nothing Loops)," *ACM Workshop on Experimental Computer Science,* 2007.

[56] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An Emperical Comparison of Global, Partitioned and Clustered Multiprocessor EDF Schedulers," *RTSS*, 2010.

[57] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. H. Anderson, and S. K. Baruah, "A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms". *In the Hadnbook of Scheduling: Algorithms, Models and Performance Analysis*, 2014.

[58] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali, "Supporting Task Migration in Multi-Processor Systems-on-Chip: A Feasible Study," *DATE*, 2006.

[59] H. Koc, S. Tosun, O. Ozturk, and M. Kandemir, "Reducing Memory Requirements through Task Recomputation in Embedded Multi-CPU Systems," *IEEE Comp. Society Annual Symp. on Emerging VLSI Technologies and Architectures (ISVLSI)*, 2006.

[60] H. Koc, O. Ozturk, M. Kandemir, S.H.K. Narayanan, and E. Ercanli, "Minimizing Energy Consumption of Banked Memories Using Data Recomputation," *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2006.

[61] S. Tosun, M. Kandemir, and H. Koc, "Using Task Recomputation during Application Mapping in Parallel Embedded Architectures Using Task Recomputation during Application Mapping in Parallel Embedded Architectures," *Int'l Conf. on Comp. Design & Conf. on Computing in Nanotechnology (CDES)*, 2006.

[62] H. Koc, M. Kandemir, E. Ercanli and, O. Ozturk, "Reducing Off-Chip Memory Access Costs Using Data Recomputation in Embedded Chip Multi-processors," *ACM/IEEE Design Automation Conf.*, 2007.

[63] H. Koc, M. Kandemir, and E. Ercanli, "Exploiting Large On-Chip Memory Space through Data Recomputation," *IEEE Int'l SOC Conf.*, 2010.

[64] B. Nimer, and H. Koc, "Improving Reliability through Task Recomputation in Heterogeneous Multi-Core Embedded Systems," *Int'l Conf. on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, 2013.

[65] V. M. van Santen, H. Amrouch, N. Parihar, S. Mahapatra, and J. Henkel, "Aging-aware voltage scaling," *DATE*, 2016.

[66] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang "Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32 , no. 5, pp. 695-708, 2013.

[67] A. Taherin, M. Salehi, and A. Ejlali, "Reliability-Aware Energy Management in Mixed-Criticality Systems," *IEEE Trans. on Sustainable Computing (TSC)*, vol. 3, no. 3 pp. 195-208, 2018.

Sepideh Safari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. She is currently working toward the PhD degree in computer engineering at Sharif University of Technology. Her research interests include, energy management in fault-tolerant mixed-criticality systems, and multicore systems with emphasis on reliability.

Mohsen Ansari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. He is currently working toward the PhD degree in computer engineering at Sharif University. His research interests include low-power design of embedded systems, peak power management in embedded systems, and multi-/many-core systems with a focus on dependability/reliability.

Ghazal Ershadi is currently working toward the B.Sc. degree in computer engineering at Sharif University of Technology. Her research interests include low-power design of cyber physical systems, energy management in fault-tolerant multicore embedded systems.

Shaahin Hessabi received the BS and MS degrees in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1986 and 1990, respectively, and the PhD degree in electrical and computer engineering from the University of Waterloo, Ontario, Canada. He joined Sharif University of Technology, in 1996. Since 2007, he has been an associate professor in the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. He has published more than 100 refereed papers in the related areas. His research interests include cyber-physical systems, reconfigurable and heterogeneous architectures, network-on-chip, and system-on-chip. He has served as the program chair, general chair, and program committee member of various conferences, like DATE, NOCS, NoCArch, and CADS.