Optimizing Energy in Non-Preemptive Mixed-Criticality Scheduling by Exploiting Probabilistic Information

Ashikahmed Bhuiyan, Federico Reghenzani, William Fornaciari, and Zhishan Guo

***Abstract*—The strict requirements on the timing correctness biased the modeling and analysis of real-time systems toward the worst-case performances. Such focus on the worst-case, however, does not provide enough information to effectively steer the resource/energy optimization. In this article, we integrate a probabilistic-based energy prediction strategy with the precise scheduling of mixed-criticality tasks, where the timing correctness must be met for all tasks at all scenarios. The dynamic voltage and frequency scaling (DVFS) is applied to this precise scheduling policy to enable energy minimization. We propose a probabilistic technique to derive an energy-efficient speed (for the processor) that minimizes the average energy consumption, while guaranteeing the (worst-case) timing correctness for all tasks, including LO-criticality ones, under any execution condition. We present a response time analysis for such systems under the non-preemptive fixed-priority scheduling policy. Finally, we conduct an extensive simulation campaign based on randomly generated task sets to verify the effectiveness of our algorithm (with respect to energy savings) and it reports up to 46% energy-saving.**

***Index Terms*—Embedded software, real-time systems, scheduling algorithms.**

I. Introduction

MANY embedded systems have hard real-time constraints that must be satisfied to guarantee the correctness of the system behavior. Missing a time deadline is, in fact, typically considered as a system failure. To ensure the timing correctness of a system, a schedulability test is performed to verify that no task misses any deadline under any condition. The task models are usually characterized by the arrival pattern, (periodic, sporadic, or aperiodic, etc.,) the deadline, and the execution time. In particular, most of the state-of-the-art schedulability analysis considered that a task can execute up to its worst-case execution time (WCET). During the real execution, however, a task rarely needs to execute up to its WCET [48]. For a large class of real-life applications, WCET-based schedulability analysis is often proved to be very pessimistic [35], as the task execution pattern may show great variability [with respect to (w.r.t.) time]. Consequently, designing a system with the assumption that a task will execute up to its WCET, may lead to system over-provisioning, low utilization, high costs, and excessive power/energy consumption [38]. This is in contrast with the requirement of improving performance while maintaining the nonfunctional property at an acceptable level.

To efficiently utilize the non-negligible gap between the WCET and the actual execution time, and to minimize energy consumption, resource over-provisioning, and cost, the mixed-criticality (MC) framework [50] received attention from a wide community. In an MC setup, different software components with different criticality levels are integrated into a common platform. To each task, a criticality level is assigned together with multiple execution time thresholds (at different certification/pessimism levels), as described by Vestal's seminal paper [50]. This value is inspired by the industry standards for safety-critical systems: for instance, the DO-178C avionic software standard [47] sets five levels of criticality: $\{A, B, C, D, E\}$, where A is the criticality level referring to functions that may cause catastrophic failures, while at level E, to functions that do not affect safety. In real-time computing, this value is interpreted as the level of assurance of the WCET. To illustrate this concept, let us consider a dual-criticality system, where the tasks are classified in LO-criticality and HI-criticality. The tasks belonging to the latter category have two values for the WCET: where one value is pessimistic, but safe, while the other value is computed with an analysis that provides a lower level of assurance, and hence less pessimistic. In this case, the less pessimistic value may not correctly (over-)estimate the real WCET. Consequently, the execution time of a task may overrun this value. When this condition happens, i.e., the overrun, we say that a *mode switch* occurs.

Existing MC task-set scheduling strategies aimed at: 1) correctly scheduling all the tasks when the system exhibits less pessimistic behaviors (in this case, the system is said to be in LO-criticality mode) and 2) correctly scheduling the

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. Date of publication October 2, 2020; date of current version October 27, 2020. This work was supported in part by NSF under Grant CNS-1850851, a start-up grant from the University of Central Florida; in part by the EU H2020 Project RECIPE [1], [2], [23] under Grant 801137; and in part by HiPEAC Collaboration Grant 2019. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue. *(Ashikahmed Bhuiyan and Federico Reghenzani contributed equally to this work.) (Corresponding author: Zhishan Guo.)*

Ashikahmed Bhuiyan and Zhishan Guo are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816 USA (e-mail: ashik@knights.ucf.edu; zsguo@ucf.edu).

Federico Reghenzani is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816 USA, and also with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milan, Italy (e-mail: federico.reghenzani@polimi.it).

William Fornaciari is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milan, Italy (e-mail: william.fornaciari@polimi.it).

Digital Object Identifier 10.1109/TCAD.2020.3012231

more important (HI-criticality) tasks under more pessimistic behaviors, while no scheduling guarantee is given to the less important (LO-criticality) tasks (in this case, the system is said to be in HI-criticality mode). The system starts running in LO-criticality mode by scheduling the tasks under optimistic assumptions. When a HI-criticality task violates its assigned execution time threshold, the system switches to the HI-criticality mode, i.e., mode switch, and it drops all the LO-criticality tasks to guarantee the deadlines of HI-criticality tasks. However, discarding (or providing degraded services) to the LO-criticality tasks may result in severe performance loss and it violates the task independence requirements for safety-critical systems [21]. A recent work by Bhuiyan *et al.* [13] handled the *precise scheduling* of MC systems, where full service is provided to all tasks under both the pessimistic and optimistic assumptions. They incorporated the dynamic voltage and frequency scaling (DVFS) scheme to the precise scheduling strategy and derived the energy-aware CPU speed to execute in normal mode. However, the optimized energy consumption is in accordance to the *worst-case* behaviors under the normal mode. The energy consumption should be optimized for the average/expected scenarios instead, and the existing MC task model (with multiple WCETs) does not provide sufficient information to perform such optimization.

Probabilistic Approaches: Probabilistic real-time approaches have been proposed in the last two decades to overcome the problem of the WCET estimation for modern platforms [18], [46]. Most of these approaches are based on the estimation of the distribution tail by exploiting the extreme value theory, a statistical theory to model the probability of extreme events, that in the real-time case it is used for the WCET of the tasks. Unfortunately, the theory is still immature and several challenges yet to be addressed before considering its results reliable [31], [43], [45]. Hence, in this work, we propose to exploit the probabilistic information not to directly estimate the WCET for scheduling analysis purposes, but to use them for the optimization of the system energy consumption. Differently from previous probabilistic energy approaches [42], the focus of this work is the satisfaction of real-time requirements with a *deterministic* approach and on top of that, exploiting the probabilistic information to minimize the expected energy consumption of the system.

Our Contribution: Existing works aimed at minimizing energy consumption at LO-criticality mode, but considered a pessimistic assumption that all the tasks execute up to their WCET, at their respective criticality levels [13]. Since a task rarely needs to execute up to its WCET, we integrate the probabilistic-based prediction strategy and the DVFS scheme to the precise scheduling of MC tasks. To our knowledge, this is the first work that considers the probabilistic information to minimize energy consumption in an MC platform. The key contributions of this article are as follows.

1) We propose an energy-aware scheduling strategy that selects the proper (optimistic) WCETs for tasks and the processor speeds under both (LO- and HI-criticality) modes to minimize the overall average energy consumption. This optimization is performed via a novel probabilistic analysis of the execution time, coupled with a dedicated response time analysis (RTA) which guarantees the timing correctness of all tasks under both the pessimistic and optimistic assumptions.
2) A variant of an existing MC non-preemptive fixed-priority uniprocessor scheduling policy is proposed to integrate the speed changing between different modes and to remove the undesirable dropping of LO-criticality tasks.
3) Based on a randomly generated task sets, we conduct extensive simulation studies which supports the effectiveness of our algorithm (w.r.t. energy consumption).

Organization: The remainder of this article is organized as follows. Section II describes the task model with a solution overview. Section III provides an RTA of our algorithm. Section IV discusses the technique to reduce the average energy consumption. In Sections V and VI, we discuss the simulation results and the related works. Section VII concludes this article with some potential future directions.

II. Problem Formulation and Proposed Solution

A. System Model

We consider a task-set $\overline{\tau} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, running on an uniprocesssor, where each task $\tau_i \in \overline{\tau}$ is an implicit deadline periodic task, i.e., task deadline equals its period. Each task $\tau_i \in \overline{\tau}$ generates an unbounded number of jobs $\tau_{i,j}$, where j can be an arbitrarily large number with $j \geq 1$. We represent τ_i by a 5-tuple $\{T_i, C_i^{\text{LO}}, C_i^{\text{HI}}, pET_i, L_i\}$, where T_i is the interarrival time between two subsequent jobs of τ_i, C_i^{LO} and C_i^{HI}, respectively denote the LO and HI-criticality WCET of τ_i, pET_i the probabilistic profile as described later in Section II-B, and L_i is the criticality level, where $L_i \in \{\text{LO}, \text{HI}\}$. We also define, for convenience of the subsequent notation, the two complementary subsets $\overline{\tau}_{\text{LO}} = \{\tau_i \in \overline{\tau} : L_i = \text{LO}\}$ and $\overline{\tau}_{\text{HI}} = \{\tau_i \in \overline{\tau} : L_i = \text{HI}\}$.

The proposed approach works with non-preemptive fixed-priority schedulers. The necessity of this restriction on scheduling will be detailed later in this article, while the extension to dynamic priority schedulers is left as future work. The scheduling decisions repeat over time and the size of this time interval is called *hyperperiod*. The hyperperiod H can be computed as the least common multiple of the tasks' period: $H = \text{LCM}(T_1, T_2, \ldots, T_n)$.

The system is assumed to be equipped by some sort of power/energy control techniques of the hardware, such as the DVFS. To simplify the theoretical analysis, we normalize the system speed such that it is assumed that the system executes all jobs at speed $s_{\text{HI}} \in (0, 1]$ under HI-criticality mode, while $s_{\text{LO}} \in (0, 1]$ denotes the system speed under LO-criticality mode. Note that the WCET and the probabilistic information of the task model always refers to speed-1 condition, and they can be considered the *amount of work* to be executed. Given the WCET C_i of the ith task, the actual execution time in the system becomes (C_i/s) for under executing speed of s. The period (and the deadline) does not scale according to s and they remain fixed. Since, in our strategy, the speed can change only once per job execution, any overhead to change the DVFS setting is assumed included in the WCET and negligible w.r.t.

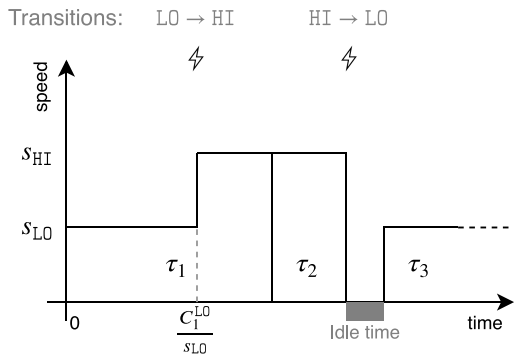

Fig. 1. Diagram of the speed-change mechanism during mode transition to HI-criticality and the switch back to LO-criticality mode after idle.

the probabilistic execution time. The energy consumed by the job of a task depends on s as well as the actual execution time of the job. We use the symbol $\varepsilon(x, s)$ to indicate the energy function that relates the execution time x and the speed s to the energy consumption. Such function is hardware dependent and can be arbitrary: the analysis of the following sections works with any energy model, including nonlinear formulations for $\varepsilon(x, s)$.

Mode Switch Mechanism and Correctness Requirements: Most of the existing papers on MC systems adopt the system mode switch effect (refer to Section VI), i.e., dropping LO-criticality tasks when a HI-criticality task overruns its C_i^{LO}. Instead, we adopted the speed-change mechanism, so that *each* task τ_i (including LO-criticality ones) is guaranteed to be executed under any condition. The system mode switch to HI-criticality causes an increase of the processor frequency to guarantee that all jobs are correctly schedule. The proposed approach is depicted in Fig. 1. Resource/Energy efficiency is achieved in LO-criticality mode by optimizing the trade-off between the minimum system speed and probability of mode-switch. Let s_{LO} (and s_{HI}) denote the processor speed when the system is in LO-criticality (and HI-criticality, respectively) mode. Same as the classical MC setting, the system switches its mode to HI when a HI-criticality task overruns its C_i^{LO}. During the HI-criticality mode, the processor runs at high frequency s_{HI} in order to be able to schedule all the jobs even in the most pessimistic assumptions. The system reverts to LO-criticality at the end of each hyperperiod, or when the system is idle, whichever comes earlier. The values of s_{LO}, s_{HI}, and C_i^{LO} depend on the schedulability of the task set and they impact the system energy consumption. For this reason, the goal of our approach is to select such parameters so that the energy is minimized, according to probabilistic information, while guaranteeing the schedulability of the whole task-set according to deterministic WCET.

B. Probabilistic Model

The probabilistic profile of the execution time of a task τ_i is identified by the symbol pET_i and it is a $3 \times k$ matrix defined as follows:

$$pET_i = \begin{pmatrix} e_1 & e_2 & \cdots & e_k \\ f_i(e_1) & f_i(e_2) & \cdots & f_i(e_k) \\ F_i(e_1) & F_i(e_2) & \cdots & F_i(e_k) \end{pmatrix} \quad (1)$$

where $e_1, e_2, \ldots, e_k$ are execution time values, $f(\cdot)$ is the probabilistic mass function (PMF), and $F(\cdot)$ the cumulative distribution function (CDF). Even if the last two rows are redundant (PMF from the CDF is computable and vice versa), this simplifies the notation in the subsequent sections. The inverse CDF (ICDF) is also defined as $F^{-1}(p) \coloneqq \{e_i : F(e_i) = p\}$.

Example 1: Let consider the following execution profile:

$$pET_1 = \begin{pmatrix} 5 & 7 & 12 & 19 & 20 \\ 0.10 & 0.60 & 0.25 & 0.04 & 0.01 \\ 0.10 & 0.70 & 0.95 & 0.99 & 1 \end{pmatrix}.$$

It represents the statistical distribution of the execution time of the task τ_1. The probability that a task requires 5 unit of execution time is 0.1, for 7 unit of execution time case the probability is 0.6 and so on. The last row represents the CDF, for example, the probability that the execution time is less or equal to 12 is 0.95. The ICDF in this case would be $F^{-1}(0.95) = 12$.

In this article, we considered the probabilistic profile, and the related statistical functions, as discrete. This assumption is in common with most of the other papers on probabilistic approaches [18], since the time is discrete when used to measure the execution time, e.g., as the number of clock cycles. The exact distribution of the task execution time is often unknown and hard to be computed statically. To estimate the f_i and F_i of the tasks, an experimental campaign must be carried out to directly measure the execution time of the task. Since, in this article, we are not interested in the probabilistic-WCET, but on the full probabilistic profile of the execution time, the empirical CDF (ECDF) method is used. This method enables to estimate the values F_i takes by directly measuring n-samples of the execution time

$$F_i(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{x_i < x}$$

where $\mathbf{1}_{x_i<x}$ is the indicator function[1] and $x_1, x_2, \ldots, x_n$ is the set of measured execution time samples. From the CDF it is possible to compute the PMF $f_i(x)$ by the subsequent differences of the CDF: $f_i(x) = F(x) - F(x-1)$. To obtain a precise estimation of the probabilistic profile, the task should experience as much as possible execution conditions during the measurements, i.e., inputs and states. The precision of the ECDF estimation is derived from the Dvoretzky–Kiefer–Wolfowitz inequality [19]: $\epsilon = \sqrt{(\log(2/c))/(2n)}$, where c is an arbitrary low confidence. For example, if $n = 10\,000$ samples are acquired, with a confidence of $c = 10^{-6}$ the maximum absolute error on the CDF is lower than $\epsilon \leq 0.02$. More sophisticated statistical techniques can obtained better results with a smaller number of samples, but this analysis would be out of scope w.r.t. this article goals. It is important to remark that this probabilistic information is used in this work only for the optimization of the energy and it does not impact the schedulability analysis. Consequently, inaccuracies in the probabilistic execution time estimations [44] affect only

[1]The indicator function $\mathbf{1}_A$ has value 1 if the condition A is respected, otherwise 0.

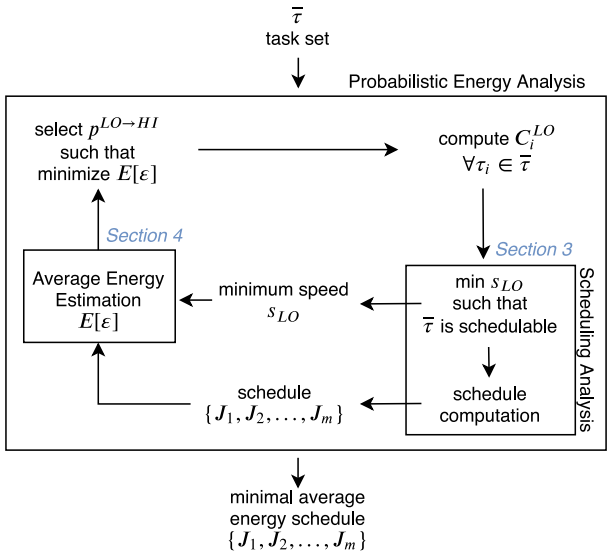

Fig. 2. Overview of the proposed approach, with a focus on the optimization algorithms.

the optimality w.r.t. the energy consumption and it does not affect the satisfaction of the real-time constraints.

WCET in LO*-Criticality Mode:* In most of traditional MC works, the value of WCET in LO-criticality mode, i.e., C_i^{LO}, is assumed to be given or empirically selected according to a defined percentage of the HI-criticality WCET. In this work, instead, we compute the C_i^{LO} so that it minimizes the energy consumption.[2] In fact, there is a tradeoff between the probability to switch to HI-criticality mode and the minimum achievable speed in LO-criticality mode. Large values for C_i^{LO} would delay the activation of the HI-criticality mode, thus reducing probability of this event to happen, but it requires to increase the minimum speed in LO-criticality mode in order to guarantee the schedulability. While, a small value for C_i^{LO} would decrease the minimum speed in LO-criticality mode, but increases the chances of a mode-switch and, if it happens too frequently, it may produce the opposite effect of increasing the energy consumption. The approach to select C_i^{LO} in this work is based on the probabilistic profile, and in particular, by exploiting the ICDF: a value $p^{\mathrm{LO}\to\mathrm{HI}}$, that represents the probability per job to switch from LO-criticality to HI-criticality, is selected according to the optimization problem and used to compute the WCET with the ICDF

$$C_i^{\mathrm{LO}} = F_i^{-1}\left(p^{\mathrm{LO}\to\mathrm{HI}}\right). \tag{2}$$

C. Solution Overview

The flow of the approach proposed in this article to solve the previously defined problem is outlined in Fig. 2. Note that, real-time task scheduling requires a strict timing guarantee. To ensure that the WCET prediction does not compromise the system schedulability, we propose an approach composed of two optimization algorithms that contribute to obtaining the schedule that requires a minimum average energy consumption while guaranteeing that no task will miss the deadline. The probabilistic information is used only for the energy minimization and not for the schedulability test. Our schedulability test remains safe by using the deterministic WCET, i.e., as far as the worst-case estimations are trustworthy, our solution will guarantee worst-case correctness.

1) The outer optimization that selects the WCET of the task in LO-criticality mode, i.e., C_i^{LO}, by choosing the best value for mode switch probability $p^{\mathrm{LO}\to\mathrm{HI}}$ that leads to the minimum average energy. This goal can be formally written with the following optimization problem:

$$\min_{p^{\mathrm{LO}\to\mathrm{HI}}} E[\varepsilon(x, s_{\mathrm{LO}})] \tag{3}$$

where $E[\cdot]$ is the expected value operator over the execution time x and s_{LO} is computed by the inner optimization algorithm from $p^{\mathrm{LO}\to\mathrm{HI}}$. This part is majorly described in Section IV.

2) The inner optimization chooses the minimal s_{LO}, which ensures the task set to be schedulable according to the RTA proposed in Section III, with the C_i^{LO} thresholds computed from $p^{\mathrm{LO}\to\mathrm{HI}}$ according to (2). Once we compute s_{LO} and generate an output schedule, the energy estimation procedure takes this schedule as input. This computes, thanks to the probabilistic information, the average energy, which is, in turn, used in the outer optimization algorithm.

Finally, the optimal values for s_{LO} and $p^{\mathrm{LO}\to\mathrm{HI}}$ are computed, together with the optimal schedule, w.r.t. the average energy minimization, to be applied online.

III. Response Time Analysis

In this section, we discuss the existing RTA for a non-preemptive FP scheduler for both the non-MC and the MC tasks (Section III-A). Then, in Section III-B, we propose the RTA for our algorithm.

A. Existing RTA for Non-MC and MC Tasks

Two state-of-the-art works [3], [39] proposed an RTA of non-preemptive FP scheduling, but under settings that are different from this paper's focus. We will report the main results in the form of equations to make it easier to follow the subsequent RTA for our problem.

First, we describe some commonly used notations in most FP scheduling analysis work. Let hep(i) [or $hp(i)$] denotes the set of tasks with higher or equal (or higher only) priority than τ_i. Similarly, $lep(i)$ [or $lp(i)$] denotes the set of tasks with lower or equal (or lower only) priority than τ_i.

Considering the non-preemptive FP scheduling for the non-MC tasks, Mohan *et al.* [39] calculated the worst-case response time (WCRT) R_i of task τ_i as follows:

$$R_i = B_i + C_i + \sum_{\tau_j \in \mathrm{hep}(i)} N_j \times C_j.$$

Here, B_i is the maximum duration that τ_i can be blocked by lower priority tasks and N_j is the total number of interfering

[2]Although the calculation under such a purpose would technically lead to execution thresholds that have nothing to do with "worst-case," we nevertheless still follow the traditional MC work in the real-time and embedded systems community by calling them WCET under the LO-criticality mode.

jobs of $\tau_j \in \text{hep}(i)$, defined as

$$B_i = \max_{\tau_k \in lp(i)} C_k - 1, \text{ and } N_j = \left\lfloor \frac{R_i - C_i}{T_j} + 1 \right\rfloor. \quad (4)$$

Before moving further into the details, let us introduce the same notations but for the MC case: $lp\chi(i)$ [or $hp\chi(i)$, $lep\chi(i)$, $\text{hep}\chi(i)$, respectively] denote the set of χ-criticality tasks with lower (or higher only, lower or equal, lower or equal) priority than τ_i, where $\chi \in \{\text{LO}, \text{HI}\}$ and other notations carry their usual meaning. We also use B_i^χ, C_i^χ and N_j^χ to denote the maximum blocking time, execution time, and the total number of interfering jobs ($\tau_j \in \text{hep}(i)$) of task τ_i at χ-criticality mode, respectively.

Baek and Lee [3] extended the analysis above to MC task model by considering the adaptive MC (AMC) scheme [4], and derived WCRTs for the following three cases separately.

Case 1: WCRT at LO-criticality mode for any task τ_i is calculated as

$$R_i^{\text{LO}} = B_i^{\text{LO}} + C_i^{\text{LO}} + \sum_{\tau_j \in \text{hep}(i)} N_j^{\text{LO}} \times C_j^{\text{LO}} \quad (5)$$

where

$$B_i^{\text{LO}} = \max_{\tau_k \in lp(i)} C_k^{\text{LO}} - 1, \quad N_j^{\text{LO}} = \left\lfloor \frac{R_i^{\text{LO}} - C_i^{\text{LO}}}{T_j} + 1 \right\rfloor.$$

Case 2: WCRT at HI-criticality mode for any HI-criticality task τ_i is calculated as

$$R_i^{\text{HI}} = B_i^{\text{HI}} + C_i^{\text{HI}} + \sum_{\tau_j \in \text{hepHI}(i)} N_j^{\text{HI}} \times C_j^{\text{HI}} \quad (6)$$

where

$$B_i^{\text{HI}} = \max_{\tau_k \in lp\text{HI}(i)} C_k^{\text{HI}} - 1, \quad N_j^{\text{HI}} = \left\lfloor \frac{R_i^{\text{HI}} - C_i^{\text{HI}}}{T_j} + 1 \right\rfloor.$$

Case 3: WCRT during mode switch of a job released by task τ_i (at LO-criticality mode but finished at HI-criticality mode) is calculated as

$$R_i^{TR} = B_i^{TR} + C_i^{\text{HI}} + \sum_{\tau_j \in \text{hepLO}(i)} N_j^{\text{LO}} \times C_j^{\text{LO}} + \sum_{\tau_j \in \text{hepHI}(i)} N_j^{\text{HI}} \times C_j^{\text{HI}} \quad (7)$$

where

$$B_i^{TR} = \max\left(\max_{\tau_k \in lp\text{LO}(i)} C_k^{\text{LO}}, \max_{\tau_k \in lp\text{HI}(i)} C_k^{\text{HI}} \right) - 1$$
$$N_j^{\text{LO}} = \left\lfloor \frac{R_i^{\text{LO}} - C_i^{\text{LO}}}{T_j} + 1 \right\rfloor$$
$$N_j^{\text{HI}} = \left\lfloor \frac{R_i^{TR} - C_i^{\text{HI}}}{T_j} + 1 \right\rfloor.$$

For any HI-criticality task $\tau_i \in \tau_{\text{HI}}$, regardless of a mode-switch, WCRT is upper bounded by R_i^{TR}. Hence, under the AMC scheme, the following conditions determine the schedulability of a task-set $\overline{\tau}$: 1) $\forall_{\tau_i \in \overline{\tau}_{\text{LO}}}, R_i^{\text{LO}} \leq D_i$ and 2) $\forall_{\tau_i \in \overline{\tau}_{\text{HI}}}, R_i^{TR} \leq D_i$.

B. RTA of Our Algorithm

In this section, we describe the RTA for our scheduling algorithm, which considers the following assumption.

Assumption: We assume that the WCET and the processor speed has a linear relationship [13], i.e., if τ_i starts executing on a processor (with C_i^{LO}) with a degraded speed s_{LO}, it will take $C_i^{\text{LO}}/s_{\text{LO}}$ time units to finish execution.

Recall that under the LO-criticality mode, all the LO- and HI-criticality tasks execute at an energy-conserving speed, i.e., s_{LO}. After a LO- to HI-criticality mode switch, all the LO- and HI-criticality tasks execute at the maximum processor speed s_{HI}.

Deriving R_i^{LO}: Based on such assumption and the RTA analysis in (5), we calculate R_i^{LO} for all the LO- and HI-criticality tasks as follows:

$$R_i^{\text{LO}} = B_i^{\text{LO}} + \frac{C_i^{\text{LO}}}{s_{\text{LO}}} + \sum_{\tau_j \in \text{hep}(i)} N_j^{\text{LO}} \times \frac{C_j^{\text{LO}}}{s_{\text{LO}}} \quad (8)$$

with B_i^{LO} and N_j^{LO} computed as

$$B_i^{\text{LO}} = \max_{\tau_k \in lp(i)} \left(\frac{C_k^{\text{LO}}}{s_{\text{LO}}} \right) - 1$$
$$N_j^{\text{LO}} = \left\lfloor \frac{R_i^{\text{LO}} - \frac{C_i^{\text{LO}}}{s_{\text{LO}}}}{T_j} + 1 \right\rfloor. \quad (9)$$

Deriving R_i^{HI}: Now, we derive R_i^{HI} thanks to (6). Recall that, we do not drop any LO-criticality tasks after a mode switch. Hence, $\text{hepHI}(i)$ in (6) needs to be replaced by $\text{hep}(i)$

$$R_i^{\text{HI}} = B_i^{\text{HI}} + \frac{C_i^{\text{HI}}}{s_{\text{HI}}} + \sum_{\tau_j \in \text{hep}(i)} N_j^{\text{HI}} \times \frac{C_j^{\text{HI}}}{s_{\text{HI}}}. \quad (10)$$

We derive the blocking time, B_i^{HI}, considering two cases.

Case 1: The blocking task, τ_k, initiates the mode-switch. Clearly, $\tau_k \in \overline{\tau}_{\text{HI}}$, and the blocking time become

$$B_i^{(1)} = \max_{\tau_k \in lp\text{HI}(i)} \left(\frac{c_k^{\text{LO}}}{s_{\text{LO}}} + \frac{c_k^{\text{HI}} - c_k^{\text{LO}}}{s_{\text{HI}}} \right) - 1.$$

Case 2: Task τ_k releases and completes execution at HI-criticality mode, and the blocking time becomes

$$B_i^{(2)} = \max_{\tau_k \in lp(i)} \left(\frac{C_k^{\text{HI}}}{s_{\text{HI}}} \right) - 1.$$

We calculate B_i^{HI} and N_j^{HI} as

$$B_i^{\text{HI}} = \max\left(B_i^{(1)}, B_i^{(2)} \right)$$
$$N_j^{\text{HI}} = \left\lfloor \frac{R_i^{\text{HI}} - \frac{C_i^{\text{HI}}}{s_{\text{HI}}}}{T_j} + 1 \right\rfloor. \quad (11)$$

Deriving R_i^{TR}: Now, we calculate R_i^{TR}, i.e., the WCRT of a HI-criticality task τ_i that faces the mode-switch. Note that, R_i^{TR} cannot be deduced from the calculation of WCRT during the stable (LO- or HI-criticality) modes [49]. We instead calculate the WCRT of a HI-criticality task, τ_i, by summing the blocking time, WCET of τ_i and the worst-case interference from other tasks with a higher priority than τ_i. Unlike [3], we do not drop

the LO-criticality tasks after a mode-switch. Hence, the LO-criticality tasks can contribute to the interference (even after a mode-switch). We derive the WCRT of a HI-criticality task τ_i that faces mode-switch considering two cases: 1) τ_i initiates the mode-switch and 2) any other (HI-criticality) task initiates the mode-switch. For both these cases, we assume that the mode-switch happens at time t^*.

Case 1: In this case, τ_i initiates the mode-switch and once τ_i starts execution, it can execute till c_i^{HI}, thanks to the non-preemptivity. According to the assumption, τ_i starts execution at LO-criticality mode. So, a task $\tau_k \in lp(i)$ that blocks τ_i must start and finish execution at LO-criticality mode. We calculate the maximum blocking time $B_i^{(1)}$ as follows:

$$B_i^{(1)} = \max_{\tau_k \in lp(i)} \left(\frac{C_k^{\text{LO}}}{s_{\text{LO}}} \right) - 1.$$

Before the mode-switch, τ_i will execute up to c_i^{LO} at speed s_{LO}, and after that it will execute at s_{HI}. Considering this scenario, we calculate the task execution time $C_i^{(1)}$

$$C_i^{(1)} = \frac{C_i^{\text{LO}}}{s_{\text{LO}}} + \frac{C_i^{\text{HI}} - C_i^{\text{LO}}}{s_{\text{HI}}}. \tag{12}$$

Recall that, τ_i initiates the mode-switch. Hence, τ_i can be interfered by τ_j (where $\tau_j \in \text{hep}(i)$) at LO-criticality mode only. We calculate the maximum interference $I_i^{(1)}$ as follows:

$$I_i^{(1)} = \sum_{\tau_j \in \text{hep}(i)} \left\lfloor \frac{R_i^{\text{LO}} - \frac{C_i^{\text{LO}}}{s_{\text{LO}}}}{T_j} + 1 \right\rfloor \times \frac{C_j^{\text{LO}}}{s_{\text{LO}}}.$$

Finally, we calculate the response time $R_i^{(1)}$

$$R_i^{(1)} = B_i^{(1)} + C_i^{(1)} + I_i^{(1)}. \tag{13}$$

Case 2: In this case, any HI-criticality task (other than τ_i) initiates the mode-switch. We calculate the maximum blocking time considering the following three subcases.

Subcase 2.1: The blocking task τ_k is released at LO-criticality mode and does not initiate a mode-switch. Recall that:

1) τ_i is also released at LO-criticality mode;
2) a HI-criticality task τ_j initiates the mode-switch, where $\tau_j \in \overline{\tau} \setminus \{\tau_i, \tau_k\}$.

From the first assumption and the non-preemptive scheduling policy, it is not possible for τ_k to start execution at LO-criticality mode, while finish at HI-criticality mode. So, τ_k must start and finish execution at LO-criticality mode, Finally, we calculate the maximum blocking time as follows:

$$B_i^{(2)} = \max_{\tau_k \in lp(i)} \left(\frac{c_k^{\text{LO}}}{s_{\text{LO}}} \right) - 1.$$

Subcase 2.2: The blocking task τ_k is released at LO-criticality mode and initiates a mode-switch. Clearly, $\tau_k \in \overline{\tau}_{\text{HI}}$. By the same reasoning as that for case 1 (12), we calculate the maximum blocking time as follows:

$$B_i^{(3)} = \max_{\tau_k \in lp\text{HI}(i)} \left(\frac{c_k^{\text{LO}}}{s_{\text{LO}}} + \frac{c_k^{\text{HI}} - c_k^{\text{LO}}}{s_{\text{HI}}} \right) - 1.$$

Subcase 2.3: The blocking task τ_k is released at HI-criticality mode. This case can be discarded, as τ_k will never block τ_i. This is because, τ_i is released at LO-criticality mode, and will start execution before $\tau_k \in lp(i)$.

To calculate the task execution time, recall that, τ_i is released at LO-criticality mode and does not initiate the mode-switch. Hence, τ_i can start execution only at the HI-criticality mode. Otherwise, τ_i itself invokes a mode-switch or must finish execution before the mode-switch (due to the non-preemptivity), which contradicts our assumption. Hence, the task execution time is

$$c_i^{(2)} = c_i^{\text{HI}}.$$

As τ_i starts execution only at the HI-criticality mode, τ_i can be interfered by $\tau_j \in \text{hep}(i)$ at both LO and HI-criticality modes (Subcase 2.1) or only at HI-criticality mode (Subcase 2.2). Let, the mode-switch takes place at time t^*. The *upper bound* of the interference is then

$$I_i^{(2)} = \sum_{\tau_j \in \text{hep}(i)} \left[\left(\left\lfloor \frac{t^*}{T_j} + 1 \right\rfloor \times \frac{C_j^{\text{LO}}}{s_{\text{LO}}} \right) + \left(\left\lfloor \frac{R_i^{TR} - t^* - \frac{C_i^{\text{HI}}}{s_{\text{HI}}}}{T_j} + 1 \right\rfloor \times \frac{C_j^{\text{HI}}}{s_{\text{HI}}} \right) \right]$$

and the response time $R_i^{(2)}$

$$R_i^{(2)} = \max\left(B_i^{(2)}, B_i^{(3)}\right) + C_i^{(2)} + I_i^{(2)}. \tag{14}$$

Considering all the scenarios (i.e., case 1, Subcases 2.1, 2.2, and 2.3) described above, we calculate R_i^{TR} as

$$R_i^{TR} = \max\left(R_i^{(1)}, R_i^{(2)}\right). \tag{15}$$

Finally, we conclude that, a task-set $\overline{\tau}$ is schedulable by our algorithm if it satisfies the following condition.

1) For each LO-criticality task $\tau_i \in \overline{\tau}$, $\max(R_i^{\text{LO}}, R_i^{\text{HI}}) \leq D_i$.
2) For each HI-criticality task $\tau_i \in \tau_{\text{HI}}$, $\max(R_i^{\text{LO}}, R_i^{\text{HI}}, R_i^{TR}) \leq D_i$; where, R_i^{LO}, R_i^{HI} and R_i^{TR} are calculated using (8), (10), and (15).

C. Computing the Minimum Speed

The role of the RTA in this article, as previously described in Section II-C, is to find the minimum value for s_{LO} given fixed values for the WCETs in LO-criticality mode C_i^{LO}, that are in turn computed from $p^{\text{LO}\rightarrow\text{HI}}$. The minimum speed s_{LO} can be computed by inverting (15). However, due to the complexity of the equation terms (especially due to the presence of max operators), obtaining a closed-form solution for $\min s_{\text{LO}}$ is not possible. In addition, the values for s_{LO} has to be in the interval $(0, s_{\text{HI}})$ from a theoretical standpoint, but in a real system this choice is often limited: the hardware has usually a finite and small number of *operating points*, i.e., DVFS possible settings [37], [51]. This justifies the possibility to explore a finite number of s_{LO} values, that produces a near-optimal solution.

IV. From Probabilistic Execution Time to Average Energy Consumption

According to the output of the previously described RTA and a fixed priority assignment protocol, it is possible to build an ordered schedule of m periodic jobs $\{J_1, J_2, \ldots, J_m\}$ forming a complete hyperperiod. Note that $J_i \in \{\tau_{j,k}\}\ \forall j,k$ where $\tau_{j,k}$ is the definition of job according to Section II-A. The notation change is necessary to highlight the job position in the overall job schedule. Formally, we also assume that each job has the same properties of its parent task, e.g., the WCET C_i^{HI} of the job J_i is the same value C_k^{HI} of the task τ_k such that $J_i = \tau_{k,j}$ for some k, j. Because all the jobs are periodic and the scheduling is fixed-priority non-preemptive, the job schedule is known at design-time and fixed. This restriction allows us to perform the following energy analysis.

A. Determining Mixture Distribution for pET

As part of the minimization problem, we assume in this section to have selected a system-level probability $p^{\text{LO}\to\text{HI}}$, as the probability of a HI-criticality job to overruns its C_i^{LO}. The value C_i^{LO} is computed with the icdf of the pET_i distribution: $C_i^{\text{LO}} = F_{X_i}^{-1}(1 - p^{\text{LO}\to\text{HI}})$, where X_i is the random variable of the execution time distributed according to pET_i. In the task model we assumed that pET_i, C_i^{LO}, and C_i^{HI} have been computed for a processor speed of $s = 1$. According to the linearity assumption of the execution time w.r.t. the processor speed, we define the random variable X_i^{LO} as the execution time in LO-criticality mode of the ith task. The probabilistic profile of its execution time is

$$pET_i^{\text{LO}} = \mathcal{T}_x\left[\begin{pmatrix} s_{\text{LO}} \\ 1 - p^{\text{LO}\to\text{HI}} \\ 1 - p^{\text{LO}\to\text{HI}} \end{pmatrix}^{\odot -1} \odot pET_i\right] \tag{16}$$

where $\odot$ is the symbol of Hadamard product[3] and Hadamard power,[4] x is the column number of C_i^{LO} in pET_i and $\mathcal{T}_x$ is the function that truncated the matrix from $3 \times k$ to $3 \times x$ by removing the upper part. For LO-criticality tasks, $x = k$, thus no truncation occurs. Similarly we define the probabilistic execution time when the task starts in HI-criticality mode pET_i^{HI} and when it starts in LO-criticality but a mode switch occurs to HI-criticality pET_i^{TR}

$$pET_i^{\text{HI}} = \begin{pmatrix} s_{\text{HI}} \\ 1 \\ 1 \end{pmatrix}^{\odot -1} \odot pET_i \tag{17}$$

$$pET_i^{TR} = \mathcal{T}'_x\left[\begin{pmatrix} s_{\text{HI}} \\ p^{\text{LO}\to\text{HI}} \\ p^{\text{LO}\to\text{HI}} \end{pmatrix}^{\odot -1} \odot pET_i + \begin{pmatrix} K & \cdots & K \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \end{pmatrix}\right] \tag{18}$$

where $K = C_i^{\text{LO}}/s_{\text{LO}} - C_i^{\text{LO}}$, x is the column number of C_i^{LO} in pET_i and $\mathcal{T}'_x$ is the function that truncated the matrix from $3 \times k$ to $3 \times (k - x)$ columns by removing the lower part. The

[3]Each element in the ith row of the left matrix is multiplied by the ith number of the right vector; the output has the size of the matrix.

[4]Element-wise power operation; the output has the size of the vector.

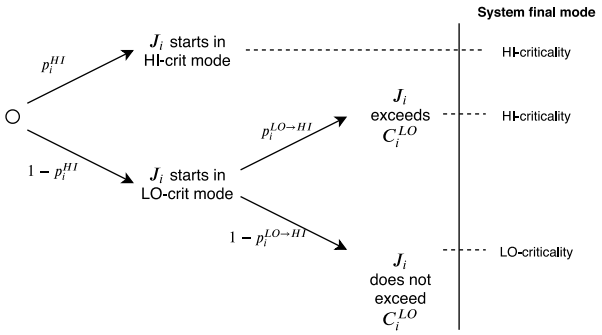

Fig. 3. Probabilistic event tree of an HI-criticality job.

pET_i^{TR} represents only the upper-part of the execution time, i.e., after the transition to HI-criticality mode. This means that it has already executed for a time period of length $C_i^{\text{LO}}/s_{\text{LO}}$ and the amount of work already completed is C_i^{LO}.

Example 2: Given the task of Example 1 and choosing a HI-mode switch probability $p^{\text{LO}\to\text{HI}} = 0.05$ with $s_{\text{LO}} = 0.5$ and $s_{\text{HI}} = 1$, the WCET for the LO-criticality mode will be $C_i^{\text{LO}} = 12$ and the probabilistic execution time in LO-criticality mode is

$$pET_1^{\text{LO}} = \begin{pmatrix} 10 & 14 & 24 \\ 0.11 & 0.63 & 0.26 \\ 0.11 & 0.74 & 1 \end{pmatrix}.$$

The probabilistic execution time when the task starts HI-criticality mode is equivalent to the original distribution because $s_{\text{HI}} = 1$

$$pET_1^{\text{HI}} = pET_1 = \begin{pmatrix} 5 & 7 & 12 & 19 & 20 \\ 0.10 & 0.60 & 0.25 & 0.04 & 0.01 \\ 0.10 & 0.70 & 0.95 & 0.99 & 1 \end{pmatrix}.$$

And the probabilistic execution time after a transition of the task from LO-criticality to HI-criticality is

$$pET_1^{TR} = pET_1 = \begin{pmatrix} 31 & 32 \\ 0.8 & 0.2 \\ 0.8 & 1 \end{pmatrix}.$$

When the transition occurs, the task has already executed 12 amount of work in 24 time units, while 7 and 8 units remain to be executed in HI-criticality mode, for a total of 31 and 32 units of time.

Fig. 3 depicts the probability tree diagram for a generic HI-criticality job J_i. In particular, at the beginning of the execution of the job J_i, the system can be in LO-criticality mode or in HI-criticality mode, respectively, with probability $1 - p_i^{\text{HI}}$ and p_i^{HI}. In the latter case, the job runs in HI-criticality mode and its execution time is distributed according to the pET_i^{HI} distribution. In the first case, the job starts to run in LO-criticality mode and possibly switches to HI-criticality if it overruns its C_i^{LO}. This happens with the already defined probability $p^{\text{LO}\to\text{HI}}$ universally set for all tasks. The task execution time is distributed in the LO-criticality part according to the pET_i^{LO} distribution, while after mode switch it is distributed according to the pET_i^{TR} distribution. Consequently, we can write the probability mass function of the execution time of the ith job

Algorithm 1: Estimation of the Probability for a Job to Start in HI-Criticality Mode (p_i^{HI})

1 $p_1^{\text{HI}} \leftarrow 0$
2 $pET_{\text{conv}} \leftarrow pET_1^*$
3 **for** $j \leftarrow 2$ ***to*** $i-1$ **do**
  ▷ Compute probability for $P(A \cap C)$
4 $P(A) = p_{j-1}^{\text{HI}}$
5 $P(A|C) = 1 - F_{j-1}^*\left(a_j \middle| \bigcup_{k<j}(X_k > C_k^{P_{\text{LO}}})\right)$
6 $P(A \cap C) = P(A)P(C|A)$
  ▷ Compute probability for $P(B \cap C)$
7 $P(B) = P(X_{j-1} \geq C_{i-1}^{\text{LO}}) = 1 - F_j^{\text{LO}}(C_{j-1}^{\text{LO}})$
8 $P(C) = P(\sum_k^{j-1} X_j \geq a_i) = 1 - F_{\text{conv}}^*(a_j)$
9 $P(B \cap C) = P(B)P(C)$
  ▷ Update variables
10 $p_j^{\text{HI}} = P(A \cap C) + P(B \cap C)$
11 $pET_{\text{conv}} \leftarrow \text{convolute}(pET_{\text{conv}}, pET_j^*)$
12 **end**

starting to run in LO-criticality mode as the following *mixture of mass functions*:

$$f_i^{\text{LO}+\text{HI}}(x) = \left(1 - p^{\text{LO}\to\text{HI}}\right) f_i^{\text{LO}}(x) + \left(p^{\text{LO}\to\text{HI}}\right) f_i^{TR}(x) \quad (19)$$

where $f_i^{\text{LO}}(x)$ is the pmf of pET_i^{LO} and $f_i^{TR}(x)$ is the pmf of pET_i^{TR}. This function includes both the case that J_i exceeds its threshold and the case that it finishes before it.

To characterize the execution time distribution of J_i, we need to know the probability that the job starts in HI-criticality mode: p_i^{HI}, which depends on the previously executed jobs. In particular, since the $p^{\text{LO}\to\text{HI}}$ is the same values for all tasks, it depends on the number of previous HI-criticality jobs and the probability of incurring in idle time (that would switch back the system mode to LO-criticality). Informally, the probability p_i^{HI} can be written as

$$p_i^{\text{HI}} = P[(A \cup B) \cap C] \quad (20)$$

where the following events have been considered:

1) *A:* J_{i-1} starts in HI-criticality mode;
2) *B:* J_{i-1} exceeds C_{i-1}^{LO};
3) *C:* system not idle after J_{i-1}.

The events are not independent, but the probability p_i^{HI} can be computed incrementally as follows. Equation (20) can be rewritten as $p_i^{\text{HI}} = P(A \cap C) + P(B \cap C)$, as the events A and B are disjoint (consequently, the probability of the union of events is the sum of their probability). $P(B)$ is always 0 if the job is belonging to a LO-criticality task. The computation of p_i^{HI} is shown in Algorithm 1. The algorithm starts with the fact (line 1) that the first job would run in LO-criticality mode (any HI-criticality mode is reset to LO-criticality at the end of the hyperperiod). From this, it is possible to compute the $pET^{\text{LO}+\text{HI}}$ thanks to (19) and consequently compute the cdf needed for $P(A|C)$ (line 5). Then, $P(B \cap C)$ can be easily computed as B and C are independent: the execution time of each job is independent with the previous ones. To compute C we need the distribution of the sum of execution time, that is incrementally built using the convolution operator (line 11).

Once the value p_i^{HI} is recursively computed, we can compute the final mixture of probability mass distribution of the execution time of the job J_i that takes in account any system mode conditions and switches

$$f_i^*(x) = \left(1 - p_i^{\text{HI}}\right) f_i^{\text{LO}+\text{HI}}(x) + p_i^{\text{HI}} f_i^{\text{HI}}(x). \quad (21)$$

B. Average Energy Consumption

In our assumptions, the energy consumption of a job is a function $\varepsilon(x, s)$, where x is the execution time and s the processor speed. This function can be either linear or superlinear (which is the case for almost all existing energy models). To compute the average energy from the probabilistic information, it is sufficient to apply the energy function to each element of the first rows of pET_i^{LO}, pET_i^{HI}, and pET_i^{TR}, with respectively $s = s_{\text{LO}}$ for the first and $s = s_{\text{HI}}$ for the last twos, obtaining 3 new probabilistic energy profiles pEC_i^{LO}, pEC_i^{HI}, and pEC_i^{TR}. The energy analysis is performed by recomputing (19) and (21) using the new matrices pEC_i. Consequently, the final $f_i^*(x)$ function becomes the statistical distribution of energy. From this distribution we can compute the expected value of the energy $E[\varepsilon] = \sum x \cdot f_i^*(x)$ that is, in turn, used as optimization cost function in (3).

The computational complexity of the energy analysis is mainly dominated by the convolution operator in line 11 of Algorithm 1. The perfect convolution has exponential complexity, however, state-of-the-art approximated solutions exist with complexity $O(m \log m)$ where m is the number of columns of the previous pET matrices. Algorithm 1 is executed for each job, so the overall complexity is $O(n \cdot m \log m)$, where n is the number of jobs. Even if the value of n is not necessarily small, the analysis is performed at design-time, it has no overhead at run-time, and the scheduler is very simple.

C. Complete Example

To better clarify the whole probabilistic algorithm, we describe in this section a toy, but complete, example. Let us consider a simple task-set composed of one HI-criticality task and two LO-criticality tasks $\bar{\tau} = \{\tau_1, \tau_2, \tau_3\}$ with the following characteristics:

$$\tau_1 = \left(15, C_1^{\text{LO}}, 6, pET_1, \text{HI}\right)$$
$$\tau_2 = (30, 5, //, pET_2, \text{LO})$$
$$\tau_3 = (30, 3, //, pET_3, \text{LO})$$

and the following probabilistic profiles[5]:

$$pET_1 = \begin{pmatrix} 3 & 6 \\ 0.95 & 0.05 \\ 0.95 & 1 \end{pmatrix}, \quad pET_2 = \begin{pmatrix} 2 & 5 \\ 0.95 & 0.05 \\ 0.95 & 1 \end{pmatrix}$$
$$pET_3 = \begin{pmatrix} 1 & 3 \\ 0.95 & 0.05 \\ 0.95 & 1 \end{pmatrix}.$$

We set $p^{\text{LO}\to\text{HI}} = 0.05$ and $s_{\text{HI}} = 1$. We compute the LO-criticality WCETs for the HI-criticality task: $C_1^{\text{LO}} = 3$. The minimum speed computed according to Section III is

[5]Please note that this is an abuse of notation for pET_i, because its suffix i refers to the job and not to the task. However, as subsequently presented, the first three jobs J_1, J_2, J_3 correspond to the three tasks τ_1, τ_2, τ_3.

$s_{\text{LO}} = 0.7$. Then, we obtain—similarly to Example 2—the three probabilistic profiles in LO-criticality mode

$$pET_1^{\text{LO}} = \begin{pmatrix} {}^{30}/_{7} \\ 1 \\ 1 \end{pmatrix}, \quad pET_2^{\text{LO}} = \begin{pmatrix} {}^{20}/_{7} & {}^{50}/_{7} \\ 0.95 & 0.05 \\ 0.95 & 1 \end{pmatrix}$$

$$pET_3^{\text{LO}} = \begin{pmatrix} {}^{10}/_{7} & {}^{30}/_{7} \\ 0.95 & 0.05 \\ 0.95 & 1 \end{pmatrix}.$$

Since $s_{\text{HI}} = 1$, the HI-criticality profiles are equivalent to the original ones: $pET_i^{\text{HI}} = pET_i$. Regarding the probabilistic profile for transitions, it is defined for the HI-criticality task only (because LO-criticality tasks cannot generate a mode switch)

$$pET_1^{TR} = \begin{pmatrix} 6 + {}^{30}/_{7} - 3 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} {}^{51}/_{7} \\ 1 \\ 1 \end{pmatrix}.$$

We consider a rate monotonic (RM) scheduler with hyper-period $H = 30$. The schedule is then: $\{J_1, J_2, J_3, J_4\} \in \{\tau_1, \tau_2, \tau_3, \tau_1\}$. The Algorithm 1 can now be used to obtain the final probabilistic time profile. The simplicity of the example allows us to explain step-by-step the solution:

1) $p_1^{\text{HI}} = 0$: the first job starts for sure in LO-mode;
2) $p_2^{\text{HI}} = p^{\text{LO}\to\text{HI}}$: referring to (20), the event B is not possible because J_2 belongs to a LO-criticality task; C is always verified because the arrival times of J_2 and J_3 are the same and the scheduler is work conserving;
3) $p_3^{\text{HI}} = p^{\text{LO}\to\text{HI}}$: the considerations of the previous step are valid also for J_3;
4) $p_4^{\text{HI}} = 0.000125$, as subsequently explained.

To compute p_4^{HI} we should have computed the pET^* and then the convolution of them, to obtain the probability $P(C)$, i.e., there is no idle time after J_3. For the sake of this example, this probability could be easily computed by the following reasoning: J_4 can start in HI-mode only if J_1 switched to HI-mode (with probability 0.05) and there is no idle time after J_3. This happens when both J_2 and J_3 runs until their WCET 5 and 3: the probability of this event is $0.05 \cdot 0.05 = 0.0025$. Then, $p_4^{\text{HI}} = 0.05 \cdot 0.0025 = 0.000125$. Assuming, for simplicity, that $\varepsilon(x, s) = x$, we compute the average energy per job

$$E_1[\cdot] = \sum_x x\big(0.95 f_1^{\text{LO}}(x) + 0.05 f_1^{TR}(x)\big)$$

$$E_2[\cdot] = \sum_x x\big(0.95 f_2^{\text{LO}}(x) + 0.05 f_2^{\text{HI}}(x)\big)$$

$$E_3[\cdot] = \sum_x x\big(0.95 f_3^{\text{LO}}(x) + 0.05 f_3^{\text{HI}}(x)\big)$$

$$f_4^{\text{LO+HI}}(x) = 0.95 f_4^{\text{LO}}(x) + 0.05 f_4^{TR}(x)$$

$$E_4[\cdot] = \sum_x x\big[(1 - 0.000125) f_4^{\text{LO+HI}}(x) + 0.000125 f_4^{\text{HI}}(x)\big].$$

For example, the first term is computed as

$$E_1[\cdot] = {}^{30}/_{7}(0.95 \cdot 1 + 0.05 \cdot 0) + {}^{51}/_{7}(0.95 \cdot 0 + 0.05 \cdot 1) = 4.44.$$

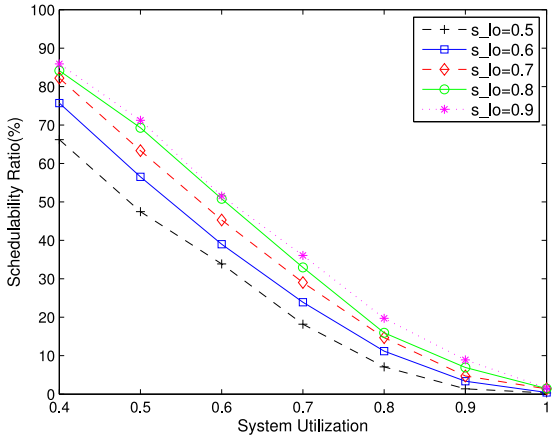

Fig. 4. Schedulability ratio based on the RTA (see Section III-B) with a different s_{LO} values. In this experiment, we use the following fixed parameters: $[U_d, U_u] = [0.02, 0.2]$; $[T_d, T_u] = [5, 50]$; $[Z_d, Z_u] = [1, 4]$; $P = 0.5$.

The total average energy is then the sum of the individual energy components

$$E[\varepsilon] = E_1 + E_2 + E_3 + E_4 = 4.44 + 3.02 + 1.54 + 4.39 = 13.39.$$

V. Evaluation Results

In this section, we report the evaluation result of our algorithm. In Section V-A, we report the *schedulability ratio*, i.e., the ratio of scheduled task-sets over the total number of task-sets, of our algorithm for fixed values of s_{LO}. Section V-B, instead, describes the energy analysis. It shows the benefit on energy consumption of introducing probabilistic information and the choice of $p^{\text{LO}\to\text{HI}}$ and s_{LO}. To simplify the experimental evaluation, without loosing generality, we considered $s_{\text{HI}} = 1$, i.e., the maximum achievable speed in HI-criticality mode is set at the maximum processor speed.

A. Schedulability Ratio

In this section, we report the performance of our algorithm w.r.t. the schedulability ratio. We conduct the experiments on a randomly generated task-set, and use the workload generation model proposed in [24]. We use the following input specifications to generate the workload.

1) U_{bound}: The upper bound of the system utilization.
2) $[T_d, T_u]$: The range of the minimum interarrival period of a task, i.e., $0 < T_d \leq T_i \leq T_u$.
3) $[U_d, U_u]$: The range of the utilization u_i (of τ_i). We use u_i to obtain execution time of τ_i in the LO-criticality mode, i.e., $\forall_{\tau_i \in \tau} : C_i^{\text{LO}} = u_i \times T_i$, where, $0 < U_d \leq u_i \leq U_u \leq 1$.
4) $[Z_d, Z_u]$: The range of the ratio of HI and LO-criticality WCET, here $1 \leq Z_d \leq Z_u$.
5) P: The probability of being a HI-criticality task.

In these experiments, we computed the schedulability ratio of our scheduling algorithm for different values of s_{LO} (ranging from [0.5,0.9]) and U_{bound} (ranging from [0.4, 1.0]). Figs. 4 and 5 report that the schedulability ratio (for any s_{LO} value) decreases with the increase in system utilization, which

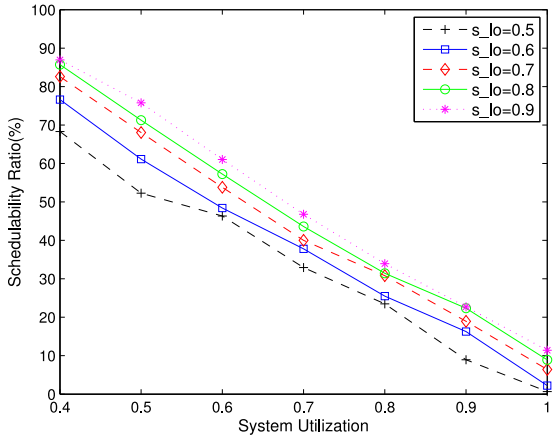

Fig. 5. Schedulability ratio based on the RTA with a different s_{LO} values. In this settings, we use $[Z_d, Z_u] = [1, 8]$, while other parameters remain same as Fig. 4.

matches with our RTA. Most of all task-sets are schedulable when U_{bound} is lower than 0.5, while most of the task-sets become not schedulable when utilization approaches 1. In this experiment, the selection of s_{LO} does not depend on the task-set itself but it has been fixed at different levels. This reduces the schedulability ratio, but it is implicitly solved by the inner optimization problem. As presented in the next section, the optimization problem helps also in improving the scheduling ratio, by selecting the best value for s_{LO} that guarantees schedulability.

B. Energy Optimization

Simulation Parameters: In order to perform the simulation, we selected the NXP i.MX6 SABRE board, for which the energy function and the relation voltage-frequency is known [37]. The speed is assumed proportional to the frequency[6] (see Section II-A). The considered energy function is

$$\varepsilon(x, s) = x\left(aC \cdot V^2 f + P_{\mathrm{leak}}\right)$$
$$aC = 3.4 \cdot 10^{-10}, \quad P_{\mathrm{leak}} = 0.052$$
$$V^2 f = \left(0.95 + 0.0005\frac{f - 396 \cdot 10^6}{10^6}\right)^2 \cdot f$$

where f is the frequency computed as $f = s \cdot \mathrm{max\_freq}$, aC has been experimentally obtained, $V^2 f$ comes from [37] and P_{leak} has been taken from the SoC datasheet. The selected values for exploration are $p^{\mathrm{LO}\to\mathrm{HI}} \in [0.01; 0.50]$ and $s_{\mathrm{LO}} \in [0.5; 0.9]$. The considered fixed priority assignment algorithm is RM.

Results: Fig. 6 depict the exploration of 10 random task-sets by varying $p^{\mathrm{LO}\to\mathrm{HI}}$, i.e., the results of the external optimization algorithm. Fig. 6(a) refers to a total utilization of $U = 0.25$ and Fig. 6(b) refers to a total utilization of $U = 0.50$. The figures show the schedulability and the amount of energy saved

[6]This is not necessarily true in architectures with variable timing instructions. However, in this case, the workload would play a key role in the definition of the energy function, as also in the WCET analysis. In any case, the presence of variable instructions timing would not invalidate our analysis, but it would add excessive verbosity without adding any innovative content.

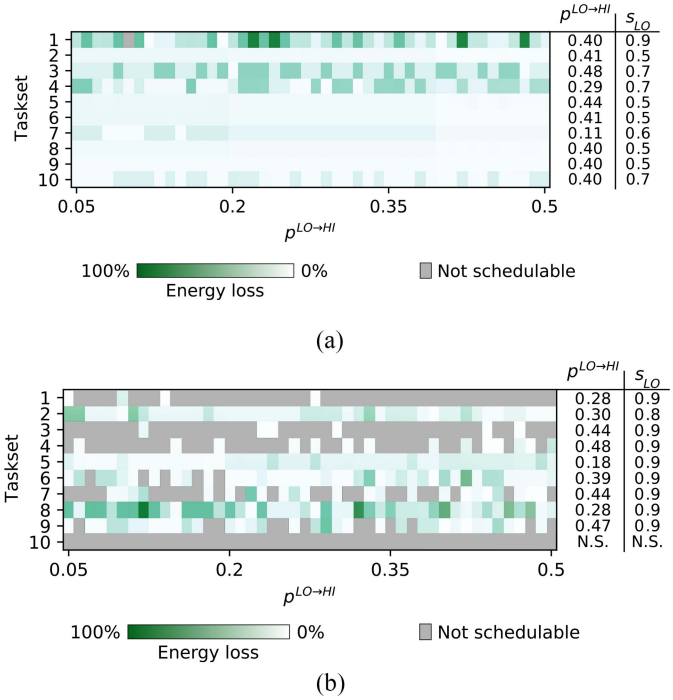

Fig. 6. Energy of the 10 datasets analyzed with our framework, depicted by the energy loss percentage w.r.t. the minimum found for each dataset. The $(p^{\mathrm{LO}\to\mathrm{HI}}, s_{\mathrm{LO}})$ couples with the minimal energy consumption are showed on the right columns. (a) $U = 0.25$. (b) $U = 0.50$.

w.r.t. the minimum energy consumption of each dataset. When utilization is low, the task-sets are schedulable for any value of $p^{\mathrm{LO}\to\mathrm{HI}}$ (with only one exception for task-set 1). Instead, when the utilization increases, it is possible to notice that $p^{\mathrm{LO}\to\mathrm{HI}}$ represents a critical choice to the schedulability: the schedulability for $U = 0.50$ increases to 90% compared to the range of 50%–70% from Figs. 4 and 5. The choice of $p^{\mathrm{LO}\to\mathrm{HI}}$, as expected, also impacts on energy optimization. The optimization function is clearly nonconvex and numerical methods are necessary. Some choices of $p^{\mathrm{LO}\to\mathrm{HI}}$ are very far from the optimal, for example, dataset 1 in the $U = 0.25$ case, the value $p^{\mathrm{LO}\to\mathrm{HI}} = 0.24$ leads to a energy consumption value 1.79 *times* larger than the optimal value at $p^{\mathrm{LO}\to\mathrm{HI}} = 0.4$. For other task-sets, like 6–9 in Fig. 6(a), the saved energy is instead small. No direct relationship can be derived w.r.t. the s_{LO} value: increasing utilization requires a larger s_{LO}, but no general conclusions can be drawn for a fixed utilization value.

Similarly to the previous experiments, we run a simulation with 2500 task-sets by varying the utilization in the range $(0, 1]$. The analysis carries out the optimal value for $p^{\mathrm{LO}\to\mathrm{HI}}$ that corresponds to the minimal energy. Then, this value is compared to the baseline of randomly select $p^{\mathrm{LO}\to\mathrm{HI}}$. The improvements on energy consumption corresponds to 10.0% on average and 46.4% on maximum energy saved values.

Lesson Learnt: Previous works usually assumed given the value of LO-criticality WCET C_i^{LO}, or fixed to a given proportion of the HI-criticality WCET. However, when energy plays a role in the analysis, that choice is critical to reach the optimality. This is especially evident when DVFS mechanism is used to reduce the power consumption during the LO-criticality mode of the system, because the choice of

s_{LO} impacts on both schedulability and energy consumption. Choosing these values to optimize the energy for the average-case (the most common) is nontrivial. Our approach optimized the energy consumption by selecting the best value for $p^{\mathrm{LO}\rightarrow\mathrm{HI}}$, and consequently the value of C_i^{LO}, and the best value of s_{LO}, obtaining a significant value of saved energy.

VI. Related Work

To date, a significant amount of works studied the energy minimization scheduling technique considering both the parallel and sequential real-time tasks, in a non-MC platform, few to mention [11], [12], [15], [17], [26]–[28], [41]. On the other hand, extensive research has been done on real-time scheduling of the MC task model considering both the sequential and parallel workload model (e.g., [4]–[6], [14], [20], [29], and [34]), without considering the energy-awareness.

The majority of the above-mentioned works considered the EDF-VD scheduling policy, while Lee *et al.* [33] proposed the MC-Fluid model. In this model, each task receives a share (dependant on its criticality level) of the available resources. They also proposed MCDP-Fair, an implementable (on a real hardware platform) variant of the MC-Fluid. Considering the dual-criticality platform, Baruah *et al.* [8] proposed MCF, which provided an improved speedup bound of no greater than 1.33. However, all these above-mentioned models received criticism (from Ernst and Di Natale [21] and Esper *et al.* [22]) of being impractical and unable to maintain the run-time robustness. Also, a vast majority of the existing works have another limitation, upon a system mode-switch, no service guarantee is provided to any of the LO-criticality tasks [9], [10], [16], [21], [25]. Some recent works [7], [32], [36] proposed the imprecise MC (IMC) model, that provides degraded service to the LO-criticality tasks even after a mode-switch.

Little work has been done [30], [40], [42] that considered both energy awareness and MC scheduling. All these papers assumed that all LO-criticality tasks are dropped after a mode-switch. The recent work in [13] proposed the technique to provide a full-service guarantee to all LO-criticality tasks (even after a mode switch) and most related to this work. However, it is also based on the pessimistic assumption that all the tasks will execute up to their WCET (at the corresponding system criticality level) and did not consider the probabilistic information to derive the LO-criticality execution time thresholds.

VII. Conclusion

The traditional MC model is criticized as it provides no service guarantee to the LO-criticality tasks in HI-criticality mode. Some recent efforts have been made to fully or partially accommodate LO-criticality tasks after a system mode-switch, but made a pessimistic assumption that all the tasks execute up to their WCET, at their respective criticality levels. In this work, we consider the precise scheduling of MC tasks and integrate the probabilistic-based prediction strategy (of the task execution time) and the DVFS scheme to minimize energy consumption. To our knowledge, no work considers the probabilistic-based prediction strategy to minimize energy consumption in an MC platform. We propose the RTA of our algorithm under a fixed priority non-preemptive scheduler. We also evaluate our algorithm via simulation on randomly generated workloads and report energy saving up to 46% w.r.t. the pessimistic choice of LO-criticality WCET commonly made by previous works.

Applicability to Real Systems: Despite this article is highly theoretical, the deployment of this approach is straightforward. In fact, both scheduling and probabilistic analyses are performed offline and a well-known fixed-priority scheduler is used at run-time, minimizing the impact on software development and integration. The probabilistic profiles are obtained by measuring the execution time of the tasks directly on the real platform, while the WCET is computed with a state-of-the-art static timing analysis tool. Even if our analysis does not take into account scheduling and DVFS-change overheads, the number of frequency changes is upper-bounded to only one time per HI-criticality job. Assuming that the overheads are small compared to the execution time of the tasks, they can be simply added to the tasks WCET, without a significant impact on the solution optimality.

Future Works: This work may trigger several future research directions. Here, we restrict our attention to the uniprocessor platform and dual-criticality levels. We plan to extend our work to adopt the multiprocessor platform and to consider multicriticality levels. Another research direction includes the study of how to apply the analysis to variable priority and/or preemptive schedulers, e.g., Earliest Deadline First. This may include an on-board implementation of our algorithm to measure the actual energy savings and to study online reactive algorithms. Such algorithms can potentially tune the scheduling parameters at run-time in order to reach the optimal energy consumption even in the case when external factors modify the initial assumptions.

References

[1] G. Agosta *et al.*, "Challenges in deeply heterogeneous high performance systems," in *Proc. IEEE DSD*, 2019, pp. 428–435.
[2] G. Agosta *et al.*, "The RECIPE approach to challenges in deeply heterogeneous high performance systems," *Microprocess. Microsyst.*, vol. 77, Sep. 2020, Art. no. 103185.
[3] H. Baek and J. Lee, "Incorporating security constraints into mixed-criticality real-time scheduling," *IEICE Trans. Inf. Syst.*, vol. 100, no. 9, pp. 2068–2080, 2017.
[4] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Proc. IEEE RTSS* , 2011, pp. 34–43.
[5] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic DAG tasks," in *Proc. IEEE RTSS*, 2016, pp. 227–236.
[6] S. Baruah *et al.*, "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems," *J. ACM*, vol. 62, no. 2, pp. 1–33, 2015.
[7] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *Proc. IEEE ECRTS*, 2016, pp. 131–138.
[8] S. Baruah, A. Easwaran, and Z. Guo, "MC-Fluid: Simplified and optimally quantified," in *Proc. IEEE RTSS*, 2015, pp. 327–337.
[9] S. Baruah and Z. Guo, "Mixed-criticality scheduling upon varying-speed processors," in *Proc. IEEE RTSS*, 2013, pp. 68–77.
[10] S. Baruah and Z. Guo, "Scheduling mixed-criticality implicit-deadline sporadic task systems upon a varying-speed processor," in *Proc. IEEE RTSS*, 2014, pp. 31–40.

[11] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient real-time scheduling of DAG tasks," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 5, p. 84, 2018.

[12] A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, N. Guan, and Z. Guo, "Energy-efficient parallel real-time scheduling on clustered multi-core," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 9, pp. 2097–2111, Sep. 2020.

[13] A. Bhuiyan, S. Sruti, Z. Guo, and K. Yang, "Precise scheduling of mixed-criticality tasks by varying processor speed," in *Proc. RTNS*, 2019, pp. 123–132.

[14] A. Bhuiyan, K. Yang, S. Arefin, A. Saifullah, N. Guan, and Z. Guo, "Mixed-criticality multicore scheduling of real-time gang task systems," in *Proc. IEEE RTSS*, 2019, pp. 469–480.

[15] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing CPU energy in real-time systems with discrete speed management," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 8, no. 4, pp. 1–23, 2009.

[16] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surveys*, vol. 50, no. 6, p. 82, 2017.

[17] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Proc. IEEE ISPA*, 2009, pp. 1–12.

[18] R. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *Leibniz Trans. Embedded Syst.*, vol. 6, no. 1, pp. 1–3, 2019.

[19] A. Dvoretzky, J. Kiefer, and J. Wolfowitz, "Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator," *Ann. Math. Stat.*, vol. 27, pp. 642–669, Sep. 1956.

[20] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real Time Syst.*, vol. 50, no. 1, pp. 48–86, 2014.

[21] R. Ernst and M. Di Natale, "Mixed criticality systems—A history of misconceptions?" *IEEE Des. Test*, vol. 33, no. 5, pp. 65–74, Oct. 2016.

[22] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *Proc. RTNS*, 2015, pp. 139–148.

[23] W. Fornaciari *et al.*, "Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems," in *Proc. ACM SAMOS*, 2018, pp. 187–194.

[24] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Improving the scheduling of certifiable mixed-criticality sporadic task systems," Dept. Inf. Technol., Uppsala Univ., Uppsala, Sweden, Rep. 2013-008, 2013.

[25] Z. Guo and S. Baruah, "Mixed-criticality scheduling upon varying-speed multiprocessors," in *Proc. IEEE DASC*, 2014, pp. 237–244.

[26] Z. Guo and S. K. Baruah, "A neurodynamic approach for real-time scheduling via maximizing piecewise linear utility," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 238–248, Feb. 2016.

[27] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms," in *Proc. IEEE RTAS*, 2019, pp. 156–168.

[28] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient multi-core scheduling for real-time DAG tasks," in *Proc. ECRTS*, 2017, pp. 1–22.

[29] Z. Guo, L. Santinelli, and K. Yang, "EDF schedulability analysis on mixed-criticality systems with permitted failure probability," in *Proc. IEEE RTCSA*, 2015, pp. 1–11.

[30] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient DVFS scheduling for mixed-criticality systems," in *Proc. IEEE EMSOFT*, 2014, pp. 1–10.

[31] S. Jiménez Gil, I. Bate, G. Lima, L. Santinelli, A. Gogonel, and L. Cucu-Grosjean, "Open challenges for probabilistic measurement-based worst-case execution time," *IEEE Embedded Syst. Lett.*, vol. 9, no. 3, pp. 69–72, Sep. 2017.

[32] J. Lee, H. S. Chwa, L. T. Phan, I. Shin, and I. Lee, "MC-ADAPT: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5S, pp. 1–21, 2017.

[33] J. Lee *et al.*, "MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors," in *Proc. IEEE RTSS*, 2014, pp. 1–31.

[34] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real Time Syst.*, vol. 53, no. 5, pp. 760–811, 2017.

[35] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Proc. Workshop Lang. Compil. Tools Real Time Syst.*, 1995, pp. 88–98.

[36] D. Liu *et al.*, "EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees," in *Proc. IEEE RTSS*, 2016, pp. 1–32.

[37] G. Massari, F. Terraneo, M. Zanella, and D. Zoni, "Towards fine-grained DVFS in embedded multi-core CPUs," in *Architecture of Computing Systems*. Braunschweig, Germany: Springer, 2018.

[38] A. Maxiaguine, S. Kunzli, and L. Thiele, "Workload characterization model for tasks with variable execution demand," in *Proc. IEEE Design Autom. Test Europe Conf. Exhibit.*, vol. 2, 2004, pp. 1040–1045.

[39] S. Mohan, M. K. Yoon, R. Pellizzoni, and R. Bobba, "Real-time systems security through scheduler constraints," in *Proc. IEEE ECRTS*, 2014, pp. 129–140.

[40] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R. V. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," in *Proc. IEEE RTAS*, 2016, pp. 1–12.

[41] A. Paolillo, J. Goossens, P. M. Hettiarachchi, and N. Fisher, "Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies," in *Proc. IEEE RTCSA*, 2014, pp. 1–10.

[42] F. Reghenzani, G. Massari, and W. Fornaciari, "A probabilistic approach to energy-constrained mixed-criticality systems," in *Proc. IEEE ISLPED*, 2019, pp. 1–6.

[43] F. Reghenzani, G. Massari, and W. Fornaciari, "Probabilistic-WCET reliability: Statistical testing of EVT hypotheses," *Microprocess. Microsyst.*, vol. 77, pp. 103–135, Sep. 2020.

[44] F. Reghenzani, G. Massari, L. Santinelli, and W. Fornaciari, "Statistical power estimation dataset for external validation GoF tests on EVT distribution," *Data Brief*, vol. 25, Aug. 2019, Art. no. 104071.

[45] F. Reghenzani, L. Santinelli, and W. Fornaciari, "Dealing with uncertainty in pWCET estimations," *ACM Trans. Embed. Comput. Syst.*, vol. 19, 2020.

[46] L. Santinelli, Z. Guo, and L. George, "Fault-aware sensitivity analysis for probabilistic real-time systems," in *Proc. IEEE DFT*, 2016, pp. 69–74.

[47] *Software Considerations in Airborne Systems and Equipment Certification*, SC-205 Standard DO-178C, 2011.

[48] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. IEEE DAC*, 1999, pp. 134–139.

[49] K. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority preemptively scheduled systems," in *Proc. IEEE RTSS*, 1992, pp. 100–109.

[50] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. IEEE RTSS*, 2007, pp. 239–243.

[51] D. Zoni, F. Terraneo, and W. Fornaciari, "A DVFS cycle accurate simulation framework with asynchronous NoC design for power-performance optimizations," *J. Signal Process. Syst.*, vol. 83, no. 3, pp. 357–371, Mar. 2015.