Empirical Study of Energy Minimization Issues for Mixed-Criticality Systems with Reliability Constraints

Zheng Li, Xiayu Hua, Chunhui Guo and Shangping Ren
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616
Email: {zli80, xhua, cguo13, ren}@iit.edu

***Abstract*—This paper empirically studies the energy minimization problem on a mixed-criticality system with stringent reliability and deadline constraints. To address this problem, we first analyze the resource demand of a mixed-criticality task set with both reliability and deadline constraints. Based on the analysis, we present our heuristic search based energy minimization algorithm (HSEM), which is used to find a task execution strategy that can not only satisfy task set's reliability and deadline constraints, but also minimize system's total energy consumption. The performance of our proposed HSEM is compared with two baselines, i.e., the largest utilization first (LUF) and the smallest utilization first (SUF) algorithms. The experimental results indicate that the HSEM algorithm outperforms the SUF and LUF algorithms with respect to energy saving.**

***Index Terms*—Energy, Mixed-criticality, Reliability, Deadline**

I. Introduction

When designing real-time and embedded systems, to reduce device cost, and space, weight and power consumption, a common trend is to integrate tasks of different functionalities and of different desired levels of safety on the same hardware platform [1]. The mixed-criticality system design, i.e., tasks of two or more levels of criticality share the same platform, is recently considered by automotive and avionic industries [2]. Examples include unmanned aerial vehicles (UAV), which integrates the HI-criticality functionalities such as flight-control tasks and LO-criticality tasks such as photo capturing tasks [3] on the same platform.

Due to limited resources, LO-criticality and HI-criticality tasks on the same platform may compete for resources and cause tasks to miss their deadlines. However, HI-criticality tasks, such as flight-control tasks in UAV systems, are more crucial for the entire systems. To ensure HI-criticality tasks meet their deadlines, extra pessimism is taken when estimating HI-criticality task's worst case execution time [4].

When all the tasks execute no more than their designed worst case execution time, the system is considered operating under LO-mode. However, if some task executes beyond this limit, the system changes to HI-mode immediately. A resource efficient system should guarantee that both LO-criticality and HI-criticality tasks always meet their deadline under LO-mode; when system changes to HI-mode, HI-criticality tasks are still guaranteed to meet their deadlines [5]. Extensive research study has been conducted to design more resource efficient mixed-criticality systems.

In addition to the schedulability issue, power/energy efficiency and reliability have increasingly become critical issues in designing real-time and embedded systems. As more and more transistors are integrated into a single chip, operation power consumption of the chip has increased exponentially. Dynamic Voltage and Frequency Scaling (DVFS), which dynamically lowering down the supply voltage and working frequency, is widely used for power management. However, existing work [6] has shown that the transient fault rate increases when the supply voltage on an IC chip decreases. In other words, lowering down system's supply voltage can potentially degrade the system's reliability. Hence, how to minimize system's energy consumption without sacrificing the reliability requirement, is another design challenge.

In this paper, we experimentally study how to schedule mixed-criticality tasks so that the system's energy consumption is minimized, but the following constraints are satisfied:

1) schedulability constraint: both HI-criticality and LO-criticality tasks are guaranteed meet their deadlines under LO-mode, and HI-criticality tasks are guaranteed to meet their deadlines under HI-mode;
2) reliability constraint: both HI-criticality and LO-criticality tasks are guaranteed to meet their reliability constraints under LO-mode, and HI-criticality tasks are guaranteed to meet their reliability constraints under HI-mode.

The main contributions of the paper can be highlighted as follows: 1) analyze mixed-criticality task set resource demand under both reliability and schedulability constraints 2) present the heuristic search based energy minimization algorithm for deciding the lowest task execution frequency that satisfies both deadline and reliability requirements 3) empirically evaluate and compare our proposed HSEM algorithm with the two baselines, i.e., LUF and SUF, with respect to energy consump-

The research is supported in part by NSF CAREER 0746643 and NSF CNS 1018731.

tion.

The rest of the paper is organized as follows. We first discuss related work in Section II. Models and the problem formulation are presented in Section III. The theoretical analysis is given in Section IV. In Section V, we present our proposed HSEM scheduling algorithms. Experimental results are illustrated in Section VI. Finally we conclude the work in Section VII.

II. Related Work

The research community has started to study the schedulability issue of mixed-criticality systems in the recent years. Baruah [4] was the first to apply EDF scheduling theory for mixed-criticality system and gave the response time analysis. To determine the EDF schedulability on mixed criticality systems, the EDF-VD algorithm, which assigns HI-criticality tasks a reduced deadline to ensure their schedulability under the worst case scenario, was also proposed by Baruah [7].

Ekberg [5] used the demand-bound function analysis to determine task set's schedulability under EDF algorithm. Ekberg's algorithm has a significant improvement over the EDF-VD, but it has higher time complexity. In order to provide a guaranteed minimum level of service to LO-criticality tasks even when system enters in HI-mode execution, Su [8] considered using elastic task models and proposed a strategy to increase LO-criticality task periods to reduce their competition against HI-criticality tasks, but allow LO-criticality tasks to execute when possible.

Extensive research work has also been published to address the interplay of reliability and energy consumption problem on traditional real-time systems, i.e., all the tasks in the system are at the same criticality level. Zhu et al. [9], [10] proposed a reliability-aware power management scheme which aims to minimize energy consumption while maintaining system's reliability at the same level as if all tasks were executed at the highest processor frequency. Zhao et al. [11] later improved the approach and developed the *shared recovery* (SHR) technique, which to further save more energy cost by reducing the reserved slack time for fault recoveries.

However, not much work has been done in the area of reducing power consumption for mixed-criticality tasks with both reliability and deadline constraints. This paper is to address the above mentioned problem.

III. Models and Problem Formulation

A. Models

Processor Model

The processor is DVFS enabled with a finite set of available frequencies, i.e., $F = \{f_1, ..., f_q\}$. The frequency values in F are in an descending order with $f_1 = f_{max}$ and $f_q = f_{min}$. These frequencies are normalized with respected to f_{max}, i.e., $f_{max} = 1$.

Task Model

Similar to the task models in previous works [12] [13], in a task set Γ, we use a quadruple to define a task: $\tau_i = (C_i, D_i, T_i, L_i)$, where

- $C_i = \{C_i(LO), C_i(HI)\}$ are the task's worst-case execution times,
- D_i is the task's relative deadline,
- T_i is the task's period,
- L_i is the criticality of the task.

Furthermore, for a task τ_i, $C_i(LO)$ and $C_i(HI)$ are the execution times calibrated under maximum frequency f_{max} of the given resource. If τ_i is a HI-criticality task (i.e. $L_i = HI$), $C_i(LO) \leq C_i(HI)$; if it is a LO-criticality task (i.e. $L_i = LO$), $C_i(LO) = C_i(HI)$.

For a task $\tau_i \in \Gamma$, its LO-mode and HI-mode utilization are defined as $u_L(\tau_i) = \frac{C_i(LO)}{T_i}$ and $u_H(\tau_i) = \frac{C_i(HI)}{T_i}$, respectively.

Transient Fault Model

Although both permanent and transient faults may occur during task execution, transient faults are found more frequent than permanent faults. Hence, in this paper, we focus on transient faults and assume the same fault arrival rate model as given in [10]:

$$\lambda(f) = \hat{\lambda}_0 10^{-\hat{d}f} \tag{1}$$

where $\hat{\lambda}_0 = \lambda_0 10^{\frac{d}{1-f_{\min}}}$, $\hat{d} = \frac{d}{1-f_{min}}$, and λ_0 is the average fault arrival rate at f_{max} and $d(> 0)$ is a system-dependent constant.

Fault Recovery Model

Backward fault recovery [10] is adopted in our system to recover failed task instances. More specificly, we assume fault detection is taken at the end of a task instance's execution. If any fault is detected, the failed task instance will be re-executed at the maximum processor's speed, i.e., f_{max}, within the same instance period if time is available for the recovery execution. The fault detection time cost is counted as part of task's worst case execution time.

System Model

The system has two running modes, i.e., LO-mode and HI-mode. Initially, the system runs at the LO-mode. In this mode, each task τ_i can run up to $\frac{C_i(LO)}{f_i}$ time units within each period if its processing frequency is f_i under the LO-mode. If any task τ_i executes beyond $\frac{C_i(LO)}{f_i}$ time units in a period, the system switch to the HI-mode. In the HI-mode, all LO-criticality tasks are removed from the system and every HI-criticality task τ_i's maximum execution time changes to $\frac{C_i(HI)}{f_i'}$ if its processing frequency is f_i' under the HI-mode.

It is worth pointing out that fault recovery time is not counted as tasks' execution time. More specifically, when a task τ_i executes for a_i time to finish but failed, then it take additional b_i time for recovery, if $a_i + b_i \geq \frac{C_i(LO)}{f_i}$ but $a_i < \frac{C_i(LO)}{f_i}$, the system mode will not change. The mode change only can be trigger when $a_i \geq \frac{C_i(LO)}{f_i}$.

Task-Instance-Level Reliability Model

Task-instance-level reliability is defined as the probability of completing a task instance τ_{ij} without incurring errors caused by transient faults [10]. It is denoted as R_i and calculated by:

$$R_i(f_i) = e^{-\lambda(f_i)\cdot\frac{C_i}{f_i}} \tag{2}$$

Task-Level Reliability Model

Task-level reliability is defined as the probability of completing all the instances of τ_i successfully within a hyperperiod H of a given task set [14]. it is denoted as Φ_i. If τ_i has k_i instances in H and all the instances are executed under f_i, and all these instances are allowed to recover at most a_i times if needed, then Φ_i can be calculate as:

$$\Phi_i(f_i, a_i, k_i) = R_i(f_i)^k + \sum_{j=1}^{a_i} \binom{k_i}{j} (1 - R_i(f_i))^j R_i(f_i)^{k_i - j} \quad (3)$$

Energy Model

We adopt the same energy model given in [15]. In particular, if task τ_i is executed under frequency f_i, its energy consumption is represented as:

$$E(f_i, C_i) = P_{ind} \frac{C_i}{f_i} + C_{ef} C_i f_i^{\theta - 1} \quad (4)$$

Where P_{ind}, C_{ef}, and $\theta(\geq 2)$ are system-dependent, but frequency-independent constants.

Though fault recovery also consumes energy, the probability of fault occurrence is really small, hence, the expected energy cost of fault recovery can be ignored. Therefore, for a given task τ_i, if it executes under the working frequency f_i within a hyperperiod H, then the total expected energy cost within a hyperperiod can be calculated as:

$$EC(f_i) = \frac{H}{T_i} \cdot E(f_i, C_i) \quad (5)$$

where H is the length of the hyperperiod, f_i is the working frequency assigned to τ_i.

There is a balanced frequency, i.e., the *energy-efficient frequency* (f_{ee}) — further scaling down the processing frequency below f_{ee} will increase total energy consumption. To simplify the discussion, we assume $f_{min} \geq f_{\text{ee}}$.

Based on these models, we formulate the problem the paper is to investigate below.

B. Problem Formulation

Given a DVFS enabled processor with q different processing frequencies, i.e. $F = \{f_1, \cdots, f_q\}$ and a mixed-criticality task set $\Gamma = \{\Gamma_{LO}, \Gamma_{HI}\}$, develop an algorithm that minimizes system's energy consumption and at the same time satisfies the schedulability constraint and reliability constraint.

For a mixed-criticality system, initially, the system operates under the LO-mode. However, as soon as a task τ_i runs over its $C_i(LO)$, the system changes to the HI-mode to guarantee that all unfinished HI-criticality tasks be completed before their deadlines. Hence, to minimize energy consumption, a straightforward strategy is: when system is under the LO-mode, tasks run at lower frequencies; once the system changes to the HI-mode, all HI-criticality task execution frequencies are switched to the f_{max} so that the deadlines can be met and LO-criticality tasks are removed from the system. However, in order to implement this strategy, we have to answer the following questions:

1) When system operates in the LO-mode, under what frequency should each task $\tau_i \in \Gamma$ operates so that the energy consumption can be reduced?
2) How to schedule the task set to meet the scheduliability constraint?
3) How to guarantee system's reliability constraint is always satisfied?

It is worth pointing out that in this paper, we assume the reliability requirement is set as the task-level reliability when all the tasks instance executes under f_{max} without fault recovery, i.e., $\Phi_i(f_{max}, 0, k_i)$.

Based on the energy model (formula (4)), tasks running under lower frequencies can reduce system's energy consumption. However, running tasks under lower speed takes longer time to complete tasks' execution and also results in lower system reliability. To maintaining system's reliability at the desired level, failed tasks have to be recovered by re-execution, which would need additional time. The additional time cost may cause tasks miss their deadline. Clearly, the three questions are intertwined with each other and cannot be solved independently.

IV. Theoretic Analysis

In this section, we focus on for a given task-frequency assignment, how to schedule the tasks to meet both the schedulability and the reliability constraints.

A. Satisfying Schedulability Constraints

We assume a commonly used mixed-criticality scheduling algorithm, i.e., the EDF-VD algorithm [7], is used in scheduling mixed-criticality task set. With the EDF-VD algorithm, when system operates under the LO-mode, tasks are scheduled using EDF algorithm with reduced deadline, i.e., VD. In other words, to guarantee HI-criticality tasks have enough time to finish their executions when system changes from LO-mode to HI-mode, each HI-criticality task is assigned a reduced deadline when system operates under LO-mode. When system changes to HI-mode, each HI-criticality task's deadline is reset to its original deadline. Ekberg [12] proposed a virtual deadline assignment strategy to schedule a mixed-criticality task set on a mono-speed system. The basic idea is to iterate time points in a task set's hyperperiod and check whether the resource demanded by the task set is always no more than the resource provided by the system.

However, in the problem we are to address, in addition to the schedulability constraint, reliability constraint must also be taken into consideration. Furthermore, tasks can use scaled frequencies to save the energy. Hence, we cannot directly apply this analysis to solve the problem formulated in III-B.

For a given task-frequency assignment, suppose the frequency f_i is assigned to the task τ_i, by looking up the MRT table (see discussion below in Section IV-B), we can obtain the minimum number of recoveries (a_i) needed to satisfy its reliability constraint. The next question is, by executing task τ_i under frequency f_i with a_i recoveries, whether the mixed-criticality task set is schedulable under EDF-VD.

For a given mixed-criticality task set Γ, according to [12], it is schedulable if the following two conditions are satisfied:

$$\forall l \in [0, H] : \sum_{\tau_i \in \Gamma} \text{dbf}_{\text{LO}}(\tau_i, l) \leq l \quad (6)$$

$$\forall l \in [0, H] : \sum_{\tau_i \in \Gamma_H} \text{dbf}_{\text{HI}}(\tau_i, l) \leq l \quad (7)$$

where H is the hyperperiod of Γ, and Γ_H represents the set of the HI-criticality tasks in Γ. For task τ_i, $\text{dbf}_{\text{LO}}(\tau_i, l)$ denotes the the total demanded execution time from time 0 to time l under the LO-mode. $\text{dbf}_{\text{HI}}(\tau_i, l)$ denotes the execution time demand under the HI-mode within a time interval with duration as l.

In the next two sections, we give the calculation of $\text{dbf}_{\text{LO}}(\tau_i, l)$ and $\text{dbf}_{\text{HI}}(\tau_i, l)$, respectively.

1) LO-mode Demand Bound Function:

Before analyzing the task's demand bound function under the LO-mode, we first give the worst case recovery scenario. Given a task set $\Gamma = \{\tau_1, ..., \tau_n\}$ with the frequency assignment as $\{f_1, ..., f_n\}$ and recovery allowance as $\{a_1, ..., a_n\}$, according to [16], the worst case scenario is that the first a_i instances of every task $\tau_i \in \Gamma$ fail and need recoveries. If and only if all the deadlines can be met under this worst case scenario, the whole task set are guaranteed to be schedule under the LO-mode. Our task's demand bound functions in LO-mode is based on such scenario.

For a given periodic task τ_i in the task set Γ, assume its period is T_i, the assigned frequency is f_i, the recovery allowance is a_i and the deadline under the LO-mode, i.e., the virtual deadline, is VD_i. For the first a_i periods, τ_i needs both execution and recovery, therefore the overall needed processor time is $(1 + \frac{f_{\max}}{f_i}) \cdot C_i(LO)$; for all other instance periods, τ_i only needs execution time $C_i(LO)$ without the need for recovery. If $l \leq a_i \cdot T_i$, then there are at most $\lfloor \frac{l - \text{VD}_i}{T_i} \rfloor + 1$ instances in time interval l that need both execution and recovery time; if $l > a_i \cdot T_i$, other than the first a_i instance, there are at most $\lfloor \frac{l - a_i \cdot T_i - \text{VD}_i}{T_i} \rfloor + 1$ instances that only need execution. Combine both cases together, we have the demand bound function of τ_i:

$$\text{dbf}_{LO}(\tau_i, l) = \begin{cases} [\![(\lfloor \frac{l - \text{VD}_i}{T_i} \rfloor + 1) \cdot (1 + \frac{f_{\max}}{f_i}) \cdot C_i(LO)]\!]_0, \\ \qquad \text{if } l \leq a_i \cdot T_i; \\ a_i \cdot (1 + \frac{f_{\max}}{f_i}) \cdot C_i(LO) + \\ [\![\frac{f_{\max}}{f_i} \cdot (\lfloor \frac{l - a_i \cdot T_i - \text{VD}_i}{T_i} \rfloor + 1) \cdot C_i(LO)]\!]_0, \\ \qquad \text{otherwise.} \end{cases} \tag{8}$$

where $[\![x]\!]_0 = \max\{x, 0\}$.

2) HI-mode Demand Bound Function:

When system changes to the HI-mode, all the LO-criticality tasks are removed and all the HI-criticality tasks operate under f_{max}. It is possible that at the mode switch point, there is a carry-over instance which has execution both in LO-mode and HI-mode. Moreover, for a given task τ_i, if the execution of the carry-over instance in HI-mode is shorter than $D_i - \text{VD}_i$, then there is no execution or recovery left for the HI-mode, otherwise τ_i is unschedulable in LO-mode if the mode switch does not happen. Hence, $D_i - \text{VD}_i$ is the shortest length of τ_i's HI-mode portion that makes τ_i can have execution or recovery in HI-mode. In addition, for the carry-over instance, it is possible that part of the execution has already been done in LO-mode at a scaled down frequency f_i ($f_i \leq f_{max}$) and it may already encounter fault(s). As we assume fault detection is taken at the end of each task instance, which means this carry-over instance can only be detected as fault after its remaining part is finished under HI-mode. This would waste the time used to execute the already failed task. In order to save the processor time, for carry-over instance, instead of taking fault detection at its end, we do the fault detection at the mode switch point, i.e., if the task instance is detected with fault, we re-execute the task from the beginning; otherwise, continue to finish its remaining part. Hence, for τ_i, the maximum time demand for its carry-over instance is $C_i(HI)$ which is used to finish its execution.

Based on the shortest length and the maximum time demand of the HI-mode portion of a carry-over instance, we give the task's worst-case scenario of time demand in the HI-mode: 1) the mode switch takes place right before a instance's virtual deadline, and 2) this instance needs to be recovered in the remaining $D_i - VD_i$ time interval.

Under the worst case scenario, the demand bound of a given task τ_i in HI-mode contains two parts. In a time interval l, τ_i's carry-over job need $C_i(HI)$ recovery time in the first $D_i - VD_i$ time interval. For the remaining $l - (D_i - \text{VD}_i)$ parts, there are at most $\lfloor \frac{l - (D_i - \text{VD}_i)}{T_i} \rfloor$ complete instances with time demand of $C_i(HI)$ for each. Therefore, for task τ_i with virtual deadline VD_i, its demand bound function in HI-mode is calculated as:

$$\text{dbf}_{\text{HI}}(\tau_i, l) = [\![(\lfloor \frac{l - (D_i - \text{VD}_i)}{T_i} \rfloor + 1) \cdot C_i(HI)]\!]_0 \tag{9}$$

With both dbf_{LO} and dbf_{HI}, we can then apply (6) and (7) to do the schedulability analysis for a given task set.

B. Reliability Analysis

For a periodic task with a given task-level reliability constraint, the task running under lower frequency needs more recoveries to meet the reliability constraint. According to the reliability model (formula (3)), if the task executes under a frequency f_i, a minimum number of recovery during the task set's hyperperiod, denoted as a_i, should be guaranteed to meet its reliability requirement. Zhao [10] proposed a formula to calculate the value of a_i and record these values in a table called *Minimum Recovery Table* (MRT). In other words, the entry $\text{MRT}_{i,j}$ of MRT is the minimum number of recovery needed for task τ_i executing under frequency f_j. The details on how to obtain MRT entry can be found in [10].

V. Heuristic Search Based Energy Minimization Algorithm

If a frequency f_i is assigned to a task τ_i , the minimum number of recovery allowance a_i to meet the reliability constraint can be obtained by looking up the MRT table. In addition, if the modified deadline of τ_i under the LO-mode is given as VD_i, formula (6) and (7) can be used to check if the schedulability constraint is satisfied. However, for a task τ_i, its modified deadline is not given as a priori. Hence, the next question is, for each task $\tau_i \in \Gamma$, how to determine, if it exists, the modified deadline VD_i which can satisfy the schedulability constraint.

With given $\text{dbf}_{\text{LO}}(\tau_i, l)$ and $\text{dbf}_{\text{HI}}(\tau_i, l)$ for each task τ_i, Ekberg [12] developed a GREEDY algorithm to determine

the virtual deadline assignment strategy to schedule the mixed-criticality task set on a mono-speed system. The general idea is to gradually reduce each HI-criticality task's original deadline until the resource demand of the whole task set is no more than the total resource the system can be provided under both LO-mode and HI-mode.

In our system, there are multiple working frequencies available. For the given frequency assignment to the task set, GREEDY can determine if the task set is schedulable with a given reliability requirement. However, there may be multiple frequency assignments satisfying the schedulability and reliability constraints, then the next question is how to find the one with the minimum energy consumption.

Brute force search may find the best frequency to task assignment, however, it is impractical for large task set. Hence, in this paper, we propose a heuristic search approach, the major steps can be highlighted as follows:

1) initially, all the tasks are running under the f_{max},
2) choose one task to scale down its frequency without violating the reliability and schedulability constraints;
3) re-do step 2) -3) until no task in step 2) is available.

In step 2), there may be more than one task can be scaled down without violating the reliability and schedulability constraints, the question is which task should be chosen? Noticing that, though reducing task's frequency decreases the energy consumption, it demands extra time for task's execution and recovery, i.e., it will increase the resource demand. Hence, we define a metric, i.e., $ED(\tau_i, f_i, f_i')$, which is to measure the energy saving per unit resource cost for a task τ_i by reducing its frequency from f_i to a lower one f_i'.

$$ED(\tau_i, f_i, f_i') = \frac{EC(f_i) - EC(f_i')}{\text{Gap}(F) - \text{Gap}(F')} \tag{10}$$

where $F = \{f_1, ..., f_i, ..., f_{|\Gamma|}\}$ and $F' = \{f_1, ..., f_i', ..., f_{|\Gamma|}\}$. $\text{Gap}(F)$ indicates the smallest gap between the supply bound function and the demand bound function of the task set Γ during a hyperperiod H under the selected frequency set F. $\text{Gap}(F)$ is defined as:

$$\text{Gap}(F) = \min_{l \in [0,H]} \{g(l)\} \tag{11}$$

where $g(l)$ is represented as:

$$g(l) = l - \sum_{\tau_i \in \Gamma} dbf_{LO}(\tau_i, l) \tag{12}$$

When scaling down a task's executing frequency, the expected energy consumption must decrease. Since tasks executing under a lower frequency indicates a longer execution time, and theoretically, its resource demand will increase and hence $\text{Gap}(F') < \text{Gap}(F)$. However, according to the formula (8) and (9), the demand bound calculation is based on the virtual deadline (VD), the GREEDY algorithm [12] we used to find the virtual deadline is a heuristic approach, hence, the found solution is not the optimal. Therefore, it is possible that $\text{Gap}(F') \geq \text{Gap}(F)$.

Based on the above observation, we give our task selection policy (*TSP*) as follows:

- if there exists negative value of ED, choose the one with largest negative ED value;
- otherwise, choose the one with largest ED value.

Based on the defined metric (ED) and the task selection policy, we detail the heuristic search based energy minimization (HSEM) approach in Algorithm 1.

The pseudo-code code of the function *CHECK(*Γ, F, FI, i*)* is given in Algorithm 2. Firstly, we search the MRT table to obtain each task's recovery number with given frequency assignment indices FI. Then we use *GREEDY(*Γ, F, FI, A*)* [12] to test the schedulability of the whole task set with given frequencies.

In Algorithm 1, Line 3 is to find the sub task set I where each task in I can be further scaled down by one level without violating schedulability and reliability constraints. Line 5 is to choose the one based on our proposed task selection policy.

ALGORITHM 1: HSEM (Γ, $F = \{f_{\text{max}}, ..., f_{\text{min}}\}$)

```
1  int[|Γ|] FI = {0, .., 0};
2  while ∃i : FI[i] < |F| do
3      find
       I = {i|CHECK(Γ, F, FI, i) == TRUE ∧ 0 ≤ i ≤ |Γ| − 1}
4      if I is not empty then
5          find τ_k based on TSP
6          FI[k] = FI[k] + 1;
7      end
8      else
9          break;
10     end
11 end
12 return FI;
```

ALGORITHM 2: CHECK (Γ, F, FI, i)

```
1  int[|Γ|] A={0,...,0};
2  FI[i] = FI[i] + 1
3  for (j = 0; j < |Γ|; i++) do
4      A[j] = Obtain from MRT with F[FI[j]] and MRT;
5  end
6  if (GREEDY(Γ, F, FI, A) == SUCCESS) then
7      return TRUE;
8  end
9  return FALSE;
```

VI. Evaluation

In this section, we evaluate the performance of the HSEM algorithm. We compare the normalized energy consumption obtained by the HSEM algorithm with that of other two heuristics: the LUF (Largest Utilization First) and the SUF (Smallest Utilization First). The LUF and SUF heuristics are same with our HSEM algorithm except the policy of choosing which task to scale down its frequency. To be more specifical, LUF chooses the task with largest HI-mode utilization among tasks without violating the reliability and schedulability constraints; while SUF chooses the one with the smallest HI-mode utilization.

A. Experimental Setting

In the following experiments, the mixed-criticality task sets are generated using UUniForm algorithm [17], which

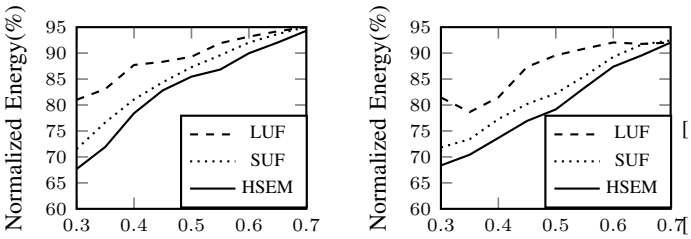
gives an unbiased distribution of utilization values. There are total six tasks in a task set. Among them, three are of HI-criticality and three are of LO-criticality. More specifically, we take the following steps to generate a valid task set: (1) the UUniForm algorithm is used to generate HI-criticality and LO-criticality task sets with HI-mode utilization $U_H(\Gamma_H) = \sum_{\tau_i \in \Gamma_H} u_H(\tau_i)$ and LO-mode utilization $U_L(\Gamma_L) \sum_{\tau_i \in \Gamma_L} u_L(\tau_i)$, respectively; (2) task period T_i is randomly selected from $[100, 2500]$; (3) task execution time $C_i(HI)$ is set as $T_i \cdot u_H(\tau_i)$, $\forall \tau_i \in \Gamma_L : C_i(LO) = C_i(HI)$, and $\forall \tau_i \in \Gamma_H : C_i(LO) = \mu \cdot C_i(HI)$, where μ is a random value within the range of $[0.1, 0.5]$ and (4) the value of $C_i(LO)$ must be a positive integer.

For energy model, we assume $P_{\mathtt{ind}} = 0.1$, $C_{\mathtt{ef}} = 1$, and $\theta = 3$. The available frequencies are set as $F = \{0.4, 0.6, 0.8, 1\}$. For transient fault model, we assume the average fault arrival rate at $f_{\mathtt{max}}$ is $\lambda_0 = 10^{-6}$, and $d = 3$.

To compare the algorithm performance, we normalize energy consumption obtained by the three heuristics to the energy consumption when all tasks run under the highest frequency $f_{\mathtt{max}} = 1$. For each experiment, we generate 100 task sets and the average value is used to evaluate the performance.

B. Experiment Results and Discussions

First, we set $U_H(\Gamma_H) = 0.3$ and change $U_L(\Gamma_L)$ from 0.3 to 0.7 with step 0.05. The experiment results are shown in Fig. 1(a). We have the following observations: (1) as task set utilization increasing, the energy consumption of three approaches increase; (2) the HSEM algorithm can save as much as 19.72% and 6.57% more energy than LUF and SUF, respectively, however, the advantage diminishes with task set total utilization increasing; (3) the energy consumption of three approaches converge when the utilization is high. Second,



(a) $U_L(\Gamma_L)$ (b) $U_H(\Gamma_H)$

Fig. 1. Normalized Energy Consumption (%)

we set $U_L(\Gamma_L) = 0.3$ and change $U_H(\Gamma_H)$ from 0.3 to 0.7 with step 0.05. The experiment results are depicted in Fig. 1(b) which shows similar observations. But the advantages of HSEM over LUF and SUF are a little smaller (19.16% and 5.07%) comparing to when $U_H(\Gamma_H)$ is fixed.

In summary, based on experimental results, the HSEM algorithm can find a task set execution strategy with less energy consumption than that found by SUF and LUF algorithms.

VII. Conclusion

In this paper, we have analyzed the resource demand of a mixed-criticality task set with both deadline and reliability constraints under a given frequency assignment. Based on the resource demand analysis, we propose HSEM algorithm to find a frequency assignment which can minimize system's energy consumption without violating system's reliability and schedulability constraints. We use experiments to compare HSEM with two heuristics (LUF and SUF), Our experimental results indicate that the HSEM algorithm outperforms SUF and LUF algorithms with respect to energy saving.

In the current work, a task's reliability requirement is assumed to be the one running the whole task under f_{max}, our future work is to extend our developed theory and algorithm to address the problem under any given reliability requirement.

References

[1] A. Burns and R. I. Davis, "Mixed criticality systems: A review," Department of Computer Science, University of York, East Lansing, Michigan, Tech. Rep. MCC-1(b), February 2013.

[2] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, Nov 2011, pp. 34–43.

[3] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi, "A research agenda for mixed-criticality systems," in *Cyber-Physical Systems Week*, Apr. 2009.

[4] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, July 2008, pp. 147–155.

[5] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, July 2012, pp. 135–144.

[6] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proceedings of the IEEE/ACM International conference on Computer-aided design*, ser. ICCAD, 2004, pp. 35–40.

[7] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, July 2012, pp. 145–154.

[8] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 147–152.

[9] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 1–27, 2011.

[10] B. Zhao, H. Aydin, and D. Zhu, "Energy management under general task-level reliability constraints," *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, vol. 0, pp. 285–294, 2012.

[11] ——, "Generalized reliability-oriented energy management for real-time embedded applications," in *Proceedings of the Design Automation Conference*, ser. DAC, 2011, pp. 381–386.

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

[13] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*. IEEE, 2011, pp. 34–43.

[14] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD, 2009, pp. 63–70.

[15] D. Zhu and H. Aydin, "Energy management for real-time embedded systems with reliability requirements," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD, 2006, pp. 528 –534.

[16] G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips," in *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*. IEEE, 1995, pp. 110–117.

[17] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.