Design Automation for Embedded Systems
https://doi.org/10.1007/s10617-019-09232-3

An energy-efficient time-triggered scheduling algorithm for mixed-criticality systems

Lalatendu Behera[1] · **Purandar Bhaduri**[1]

Received: 14 December 2018 / Accepted: 22 November 2019

Abstract
Real-time safety-critical systems are getting more complicated due to the introduction of mixed-criticality systems. The increasing use of mixed-criticality systems has motivated the real-time systems research community to investigate various non-functional aspects of these systems. Energy consumption minimization is one such aspect which is just beginning to be explored. In this paper, we propose a time-triggered dynamic voltage and frequency scaling (DVFS) algorithm for uniprocessor mixed-criticality systems. We show that our algorithm outperforms the predominant existing algorithm which uses DVFS for mixed-criticality systems with respect to minimization of energy consumption. In addition, ours is the first energy-efficient time-triggered algorithm for mixed-criticality systems. We prove an optimality result for the proposed algorithm with respect to energy consumption. Then we extend our algorithm for tasks with dependency constraints.

Keywords Real-time systems · Mixed-criticality systems · Energy-efficient computing · Time-triggered scheduling · EDF-VD · TT-Merge

1 Introduction

In recent years, real-time researchers have focused on the field of platform integration and multi-functionalities, e.g., Unmanned Aerial Vehicle (UAV) [26], Integrated Modular Avionics (IMA) for aerospace and AUTomotive Open System ARchitecture (AUTOSAR) for the automotive industry. A *mixed-criticality real-time system* (MCRTS) [8,15,27] is one that has two or more distinct levels of criticality, such as, safety-critical, mission-critical, non-critical, etc. Typical names of the criticality levels used in industry are Automotive Safety and Integrity Levels (ASIL) and Safety Integrity Level (SIL), etc. For example, the functionalities of UAV's may be classified into two categories, viz., *mission-critical* and *flight-critical*.

✉ Lalatendu Behera
lalatendu@iitg.ac.in

Purandar Bhaduri
pbhaduri@iitg.ac.in

[1] Department of CSE, IIT Guwahati, Guwahati, India

Published online: 05 December 2019

– *Mission-critical* (LO-criticality) functionalities include capturing images from the ground and transmitting those to the base station, etc.
– *Flight-critical* (HI-criticality) functionalities include safe operations while performing the mission.

The design parameters for safety-critical functionalities (e.g., flight-critical) as estimated by certification authorities (CAs) are much more pessimistic than that for mission-critical ones. The gaps between the CAs and the system designers are likely to increase in the future as pointed out in [20]. The challenges in scheduling such mixed-criticality systems are to find a single scheduling policy so that the requirements of both the system designers and the CAs are met.

Apart from scheduling, researchers are beginning to look at various other aspects of mixed-criticality systems such as energy consumption minimization. In the energy consumption minimization problem, task executions are slowed down by using dynamic voltage and frequency scaling (DVFS) and/or dynamic power management (DPM) such that the system energy consumption is minimized without affecting the mixed-criticality schedulability requirement. The energy consumption minimization problem is as hard as the mixed-criticality scheduling problem which has been proved to be NP-hard in the strong sense [8]. The work closest to ours is [21], where the proposed method is based on EDF-VD [9,10] and successful only if the task set is schedulable by EDF-VD. We try to find a method which can schedule a larger number of task sets compared to [21] as well as minimize the energy consumption.

1.1 Contributions

Our work integrates the mixed-criticality energy-efficient problem with the time-triggered scheduling algorithm discussed in [13], hereafter abbreviated as TT-Merge. As in [21], we consider minimizing the energy consumption only in the LO-criticality scenarios as the probability of occurrence of a HI-criticality scenario is extremely low. The energy optimization problem for HI-criticality scenarios is part of future work.

This paper has the following contributions. We show that our algorithm outperforms the predominant existing algorithm using DVFS with respect to minimization of energy consumption for mixed-criticality systems. We also prove the optimality of the proposed algorithm with respect to energy consumption minimization for schedules produced by the TT-Merge algorithm. Ours is the first energy-efficient time-triggered algorithm for scheduling of mixed-criticality systems. Extending our algorithm to multiprocessor mixed-criticality systems is part of future work.

The rest of the paper is organized as follows: Sect. 1.2 is focused on related work. The system model and definitions are described in Sect. 2. The TT-Merge algorithm for scheduling mixed-criticality systems is revisited in Sect. 3. In Sect. 4, we formulate the energy consumption minimization problem for the TT-Merge algorithm. Section 5 presents a new algorithm which finds the processor frequency and the execution time to be considered by the DVFS scheme for each job. We extend the proposed algorithm for dependent jobs in Sect. 6. Section 7 presents the evaluation of the proposed algorithm. Section 8 concludes the paper.

1.2 Related work

Vestal [27] identified and formalized the mixed-criticality concept in his seminal work. Baruah and Vestal [12] presented a thorough study of feasibility and schedulability for

multi-criticality real-time systems using Audsley's algorithm [3] when implemented upon preemptive uniprocessor platforms. Vestal's mixed-criticality model is criticized by different researchers in the field [18,19,28]. Later in 2018, Baruah [7] established the scope, promise and limitations of the mixed-criticality theory.

The papers [1,2,4,5,21,22,24] have looked at energy-efficient scheduling of mixed-criticality systems. Out of these only [1,21,24] are for uniprocessor systems. The work by Huang et al. [21] and Narayana et al. [24] on energy-efficient scheduling of mixed-criticality systems for uniprocessor systems are the ones with which our work is comparable.

In [21], Huang et al. studied the energy consumption minimization in uniprocessor mixed-criticality systems using the DVFS technique based on continuous frequency levels. They found the processor frequencies $f_{\text{LO}}^{\text{LO}}$ of LO-criticality tasks and $f_{\text{HI}}^{\text{LO}}$ of HI-criticality tasks which can be used in the EDF-VD algorithm [9] to schedule the given task set successfully and which result in minimum energy consumption in the LO-criticality scenario with respect to EDF-VD. They also proved that the energy consumed in the system is optimal for the EDF-VD algorithm. In [1], Ali et al. proposed an algorithm, hereafter abbreviated by PMC, which is based on EDF-VD and claimed that it consumes less energy than the algorithm in [21] based on experimental evidence, but without a proof. Both the algorithms discussed above use the EDF-VD algorithm to find an energy-efficient schedule if and only if EDF-VD finds a schedule for the task set. In other words, both the algorithms do not work if EDF-VD fails to schedule a task set. In this work, we show that the TT-Merge algorithm successfully schedules a bigger set of task sets than the EDF-VD algorithm. We also show that the schedule constructed by our method integrated with TT-Merge consumes less energy than the energy-efficient EDF-VD algorithm. We also compare our energy-efficient algorithm with PMC in Sect. 7 with randomly generated task sets. Moreover, we prove that our algorithm is optimal with respect to energy consumption for our time-triggered scheduling algorithm TT-Merge.

Narayana et al. [24] proposed a method based on a more generalized system model to reduce energy consumption in multicore mixed-criticality systems. Since the search space for optimality condition is huge, Narayana et al. considered 3 separate processor frequency variables as in [21]. The optimal processor frequencies computed in [24] when restricted to the unicore systems, turns out to be the same as in [21], i.e., the one which is optimal for EDF-VD. We show that our algorithm consumes less energy as compared to all the energy-efficient algorithm based on EDF-VD.

2 System model

2.1 Mixed-criticality task model

A mixed-criticality (MC) periodic task system $\mathscr{T}$ consists of a number of tasks $\tau_1, \ldots, \tau_n$. A task τ_i is characterized by a 4-tuple $(\chi_i, c_i(\text{LO}), c_i(\text{HI}), p_i)$, where

- $\chi_i \in \{LO, HI\}$ denotes the *criticality* level.
- $C_i(\text{LO}) \in \mathbb{N}^+$ denotes the LO-criticality *worst-case execution time*.
- $C_i(\text{HI}) \in \mathbb{N}^+$ denotes the HI-criticality *worst-case execution time*.
- $p_i \in \mathbb{N}^+$ denotes the *period*.

We assume that $C_i(\text{LO}) \leq C_i(\text{HI})$ for all tasks τ_i and the deadline of each task is the same as its period. Hence this work considers only implicit-deadline periodic tasks. Each of these tasks may generate an unbounded number of dual-criticality jobs, either of LO-

criticality or HI-criticality. A job j_{ik} of task τ_i is characterized by a 5-tuple of parameters: $j_{ik} = (a_{ik}, d_{ik}, \chi_i, C_i(\text{LO}), C_i(\text{HI}))$, where

- $a_{ik} \in \mathbb{N}$ denotes the *arrival time*, $a_{ik} \geq 0$.
- $d_{ik} \in \mathbb{N}^+$ denotes the *relative deadline*, $d_{ik} = p_i$.

We assume that the system is *preemptive*. Generally, a job in the task set is available for execution at time a_{ik} and should finish its execution before $a_{ik}+d_{ik}$. The job j_{ik} must execute for c_i amount of time which is the actual execution time between a_{ik} and $a_{ik} + d_{ik}$, but this can be known only at the time of execution. The collection of actual execution time c_i of the jobs in a task set at run time is called a ***scenario***. Scenarios in our model can be of two types, i.e., *LO-criticality scenarios* and *HI-criticality scenarios*. When each job j_{ik} of a task set executes c_i units of time and signals completion before its $C_i(\text{LO})$ units of execution time, it is called a LO-criticality scenario. If any job j_{ik} of a task set executes c_i units of time and does not signal its completion after it completes the $C_i(\text{LO})$ units of execution time, then this is called a HI-criticality scenario. Each mixed-criticality instance needs to be scheduled by a scheduling strategy where both kinds of scenarios (LO and HI) can be scheduled. Now we define a schedulability condition for a mixed-criticality task set.

Definition 1 A scheduling strategy is *feasible or correct* if and only if the following conditions are true:

1. If all the jobs finish their execution within $C_i(\text{LO})$ units of time then they meet their deadlines.
2. If any job does not declare its completion after executing its $C_i(\text{LO})$ units of execution time, then only the HI-criticality jobs must finish their $C_i(\text{HI})$ units of execution time on or before their deadlines.

Here we focus on **time-triggered schedules** [11] of the MC task set. Time-triggered schedules are computed offline and cannot be changed online. Since we consider dual-criticality mixed-criticality task sets, two time-triggered scheduling tables $\mathcal{S}_{\text{LO}}$ (to be used in a LO-criticality scenario) and $\mathcal{S}_{\text{HI}}$ (to be used in a HI-criticality scenario) are constructed for a given task set. These tables are used to schedule the task set at run time. The length of the tables is the hyperperiod, i.e., the length of the least common multiple of the periods of the task set. The rules to use the tables $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ at run time, (i.e., the *scheduler*) are as follows:

- The criticality level indicator Γ is initialized to LO.
- While (Γ = LO), at each time instant t the job available at time t in the table $\mathcal{S}_{\text{LO}}$ will execute.
- If a job executes for more than its LO-criticality WCET without signaling completion, then Γ is changed to HI.
- While (Γ = HI), at each time instant t the job available at time t in the table $\mathcal{S}_{\text{HI}}$ will execute.

The scheduler starts execution with the system in LO-criticality scenario, i.e., Γ = LO and remains in the LO-criticality scenario until a job executes for more than its LO-criticality WCET without signaling completion. In this situation, the criticality level indicator Γ changes to HI, i.e., Γ = HI. Once the criticality level indicator changes to HI, it remains in the HI-criticality scenario forever and dispatches jobs according to the table $\mathcal{S}_{\text{HI}}$.

Definition 2 A dual-criticality MC task set is said to be **time-triggered schedulable** [11] if it is possible to construct the two schedules $\mathcal{S}_{\text{HI}}$ and $\mathcal{S}_{\text{LO}}$ for $\mathcal{T}$, such that the run-time scheduler algorithm described above schedules $\mathcal{T}$ in a correct manner.

2.2 Power model and DVFS

Here we consider the state-of-the-art power model [16,25,29]:

$$P(f) = P_s + P_d(f) = P_s + \beta \cdot f^{\alpha} \tag{1}$$

where f is the processor frequency, $P(f)$ is the power consumption at frequency f, P_s is the static power consumption due to leakage current, and P_d denotes the frequency-dependent active power. The quantity β is a circuit dependent positive constant and $\alpha \geq 2$ depends on the hardware. Since $\alpha \geq 2$, the power consumption is a convex increasing function of the process frequency.

We have used DVFS instead of DPM because this will allow the system to reserve lower execution times for the HI-criticality tasks in a HI-criticality scenario and more slack could be explored to minimize the expected energy consumption. Moreover, the DPM strategy is used to shut down (or put in sleep mode) the processor during the processor idle time to lower the energy consumption. But, frequent switches between sleep and wake up mode will increase the energy consumption. There is considerable amount of work to be done to decide the time for which to put the processor in sleep mode. We also need to consider various system power states, e.g., standby, hibernate, deep sleep and shutdown. We assume that the processor is work conservative. Hence the problem of energy efficiency with respect to the time-triggered paradigm and the DPM method becomes a very complex optimization problem. Since all these considerations are out of scope of the current work, we keep these as part of future investigation. Hence our choice of DVFS over DPM for minimizing energy consumption. Since we use DVFS, we can only reduce dynamic power P_d and hence we ignore static power P_s. We also assume that the system runs at a base frequency f_b, $f_{\min} \leq f_b \leq f_{\max}$, where $f_{\min}$ and $f_{\max}$ are the minimum and maximum processor frequencies. Without loss of generality, we assume the frequency $f_{\max}$ to be 1.

3 The TT-Merge algorithm

In this section, we review the TT-Merge algorithm proposed by us in [13] and show that it can schedule more instances (i.e., a strict superset) than the EDF-VD algorithm. Note that EDFVD schedules the more general model of sporadic tasks while TT-Merge schedules only periodic tasks. Since most of the energy-efficient algorithms for mixed-criticality systems in the literature are based on the EDF-VD algorithm, we decided to show the dominance of TT-Merge (the dominant time-triggered scheduling algorithm for mixed-criticality systems) over EDF-VD.

Here we briefly review the TT-Merge algorithm from [13] which constructs two scheduling tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$. If the jobs of a task set are dispatched according to these tables, then no job will miss its deadline. The length of the tables is the hyperperiod P of the tasks set. First, the TT-Merge algorithm constructs two temporary tables, $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$. Second, these two tables are merged to construct the table $\mathcal{S}_{\text{LO}}$. Then, the table $\mathcal{S}_{\text{HI}}$ is constructed from $\mathcal{S}_{\text{LO}}$. We develop our energy consumption minimization strategy based on the TT-Merge algorithm.

3.1 Construction of tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$

This is a preprocessing phase. The tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are constructed using only LO-criticality and HI-criticality jobs, respectively. The LO-criticality and only HI-criticality jobs in the

Table 1 Task set for Example 1

Task	Arrival time	Period	Criticality	C_i(LO)	C_i(HI)
τ_1	0	14	HI	3	5
τ_2	0	14	HI	1	2
τ_3	0	7	LO	3	3
τ_4	0	14	HI	3	7

tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$, respectively are ordered according to the EDF algorithm [23]. The jobs in both the tables are then moved as close to their deadline as possible. Following this, the initial C_i(LO) units of allocations of each HI-criticality job in the table $\mathcal{T}_{\text{HI}}$ are retained and the remaining C_i(HI) $-$ C_i(LO) units of allocation are unallocated.

For example, consider the task set in Table 1. The hyperperiod of the task set is 14. There is one LO-criticality task τ_3 with period 7 which means two jobs (j_{31} and j_{32}) of τ_3 will be executed between 0 and 14 with arrival times 0 and 7, respectively. Now we can schedule j_{31} and j_{32} as close to their deadline as possible in the interval [4, 7] and [11, 14] in table $\mathcal{T}_{\text{LO}}$. Similarly, all the HI-criticality jobs (j_{11}, j_{21} and j_{41}) are scheduled in the interval [0, 5], [5, 7] and [7, 14] in table $\mathcal{T}_{\text{HI}}$. All the Hi-criticality jobs will be executed once between 0 and 14, because the periods of all the jobs are 14. But we need to retain the initial C_i(LO) units of execution time and unallocate the remaining C_i(HI) $-$ C_i(LO) units of execution time from table $\mathcal{T}_{\text{HI}}$. The final table $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are given in Fig. 8.

3.2 Construction of tables $\mathcal{S}_{\text{LO}}$ and $\mathcal{S}_{\text{HI}}$

The tables $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are then merged to construct the table $\mathcal{S}_{\text{LO}}$, starting from time instant 0 going up to P. At each time instant t, four situations can occur: (1) $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are both empty (2) $\mathcal{T}_{\text{LO}}$ is empty but $\mathcal{T}_{\text{HI}}$ is non-empty (3) $\mathcal{T}_{\text{LO}}$ is non-empty but $\mathcal{T}_{\text{HI}}$ is empty and (4) $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ are both non-empty. In case of situation 4, the algorithm declares failure and in all other cases, a job is allocated at the time slot t, if ready; see [13] for the details. In Example 2, we have explained the construction of table $\mathcal{S}_{\text{LO}}$. Once the table $\mathcal{S}_{\text{LO}}$ is constructed, the algorithm starts the construction of table $\mathcal{S}_{\text{HI}}$. Example 2 shows how the TT-Merge algorithm works along with the proposed energy efficient algorithm. For further details about TT-Merge, we refer the reader to [13].

3.3 Dominance of TT-Merge over EDF-VD

Example 1 Consider the MC task set of 4 tasks given in Table 1. To be schedulable under EDF-VD, it must satisfy the following condition [9]:

$$x U_{\text{LO}}^{\text{LO}}(\mathcal{T}) + U_{\text{HI}}^{\text{HI}}(\mathcal{T}) \leq 1 \tag{2}$$

where $U_{\chi_1}^{\chi_2} = \sum_{\tau_i \in \mathcal{T}_{\chi_1}} \frac{C_i(\chi_2)}{p_i}$ and $x = \frac{U_{\text{HI}}^{\text{LO}}}{1-U_{\text{LO}}^{\text{LO}}}$. For the given task set in Table 1, $x = 0.875$, $U_{\text{LO}}^{\text{LO}} = 0.375$ and $U_{\text{HI}}^{\text{HI}} = 1$. Since the left-hand side of (2) is greater than 1, this task set is not schedulable under EDF-VD.

But the task set is schedulable under TT-Merge where the resulting tables are given in Fig. 1. Here we can see that each HI-criticality job can finish its execution at the time instant

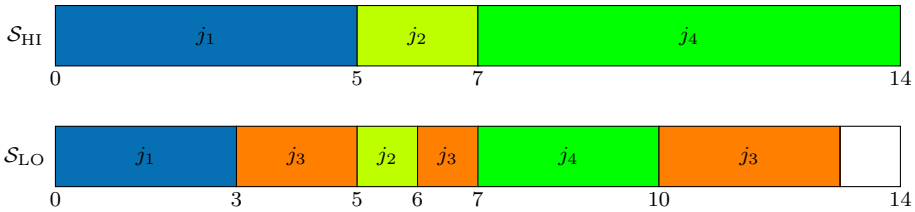


Fig. 1 Tables constructed by the TT-Merge algorithm

where a scenario change occurs in the table $\mathcal{S}_{\mathrm{HI}}$. On the other hand, if a scenario change does not occur, then all jobs can finish their LO-criticality execution in the table $\mathcal{S}_{\mathrm{LO}}$.

Lemma 1 *If a task set is schedulable under EDF-VD, then the task set is schedulable under TT-merge.*

Proof We need to show that if the EDF-VD algorithm finds a mixed-criticality schedule for a task set, then the TT-Merge algorithm will not encounter a situation where at time slot t, tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ are non-empty, for any t.

We know that C_i (LO) units of execution time are allocated to each job j_{ik} for constructing the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$. Each job in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ is allocated as close to its deadline as possible. That means no job can be executed after its allocation time in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ without affecting the schedule of any other job and still meet its deadline. The TT-Merge algorithm declares failure if it finds a non-empty slot at any time t in both the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$. This means the two jobs which are found at time instant t in the tables $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ respectively cannot be scheduled with all other remaining jobs from this point, because all the jobs to the right have already been moved as far to the right as possible.

Suppose a task set satisfies the preprocessing phase of EDF-VD and therefore has a mixed-criticality schedule.

Let j_l and j_h be two jobs in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ respectively found at time t during the construction of table $\mathcal{S}_{\mathrm{LO}}$ by the TT-Merge algorithm, which means all job segments in the interval $[0, t-1]$ from table $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ have already been assigned in $\mathcal{S}_{\mathrm{LO}}$. But, we know that these jobs can be scheduled by EDF-VD as the task set satisfies the required schedulability condition. The EDF-VD algorithm executes or dispatches the waiting jobs with earliest scheduling deadline. Without loss of generality, we can say that the processor is not idle in the interval $[0, t]$, i.e., there is no such time when a task is not available. This is because j_l and j_h are found at time t in $\mathcal{T}_{\mathrm{LO}}$ and $\mathcal{T}_{\mathrm{HI}}$ respectively and all other jobs whose deadlines are less than that of j_l and j_h are scheduled before time t. Hence there is no idle time before t, i.e., between 0 and t where either j_l or j_h could be scheduled. Both the algorithm schedules all the LO-criticality jobs whose deadline is less than or equal to the deadline of job j_l in the interval $[0, t]$. This is because both the algorithms schedule the LO-criticality jobs in the EDF order. We know that EDF-VD schedules the HI-criticality jobs in the EDF order according to the LO-scenario deadline, i.e., $\hat{p}_i$ $(= x \cdot p_i)$ [9]. We know that the LO-scenario deadline is computed using the factor $\frac{U_{\mathrm{HI}}^{\mathrm{LO}}}{1-U_{\mathrm{LO}}^{\mathrm{LO}}}$ for each HI-criticality job. That means the deadlines of HI-criticality job is modified with equal proportion which does not change the original EDF order, i.e., the EDF order with actual deadlines. So the EDF order of the HI-criticality jobs according to the LO-scenario deadline is the same as the EDF order of the HI-criticality jobs according to the actual deadlines. So all the HI-criticality jobs whose deadline is less than or equal to

the deadline of job j_h are allocated between 0 and t by both the algorithms. We know that the LO-criticality execution times of each job is scheduled in the interval $[0, t]$ in a work conservative way by both the algorithms. According to our algorithm, there is no time to schedule j_l and j_h before time t, because there is no empty space in the interval $[0, t]$ after scheduling LO-criticality execution time of each job whose deadline is less than or equal to the deadlines of j_l and j_h. EDF-VD also schedules all the LO-criticality jobs whose deadline is less than or equal to that of job j_l and all the HI-criticality jobs whose LO-scenario deadline is less than or equal to that of job j_h. EDF-VD cannot schedule a HI-criticality job whose LO-scenario deadline is greater than that of j_h in the interval $[0, t]$, because the jobs whose deadline is less than that of j_l and j_h has already filled the interval $[0, t]$. Moreover, both j_l and j_h are close to their deadlines. Since our algorithm does not find a place to schedule j_h or j_l in the time interval $[0, t]$, EDF-VD cannot schedule j_h or j_l in that time interval. Hence either j_h or j_l will miss its deadline as they are close to their deadline at time t. Therefore EDF-VD cannot schedule the task set. This is a contradiction. Hence all task sets which can be scheduled by the EDF-VD algorithm can also be scheduled by our algorithm. □

Theorem 1 *The collection of task sets schedulable under EDF-VD is a strict subset of TT-Merge, i.e.,* $\mathscr{T}^{EDF\text{-}VD} \subsetneq \mathscr{T}^{TT\text{-}Merge}$ *where* $\mathscr{T}^{EDF\text{-}VD}$ *and* $\mathscr{T}^{TT\text{-}Merge}$ *are the set of task sets schedulable by the EDF-VD and TT-Merge algorithms, respectively.*

Proof By Lemma 1, at least $\mathscr{T}^{\text{EDF-VD}} \subseteq \mathscr{T}^{\text{TT-Merge}}$ is satisfied. By Example 1, some task sets are not schedulable by EDF-VD, but schedulable by the TT-Merge algorithm. Hence, $\mathscr{T}^{\text{EDF-VD}} \subsetneq \mathscr{T}^{\text{TT-Merge}}$. □

4 Problem definition

The algorithm presented in [21,24] finds optimal processor frequencies for both LO-criticality and HI-criticality jobs in a LO-criticality scenario with respect to EDF-VD [9] to get an energy efficient schedule.

Here we propose an energy-efficient adaptation of the TT-Merge algorithm (Algorithm 3 in Sect. 5) which gives more energy efficient schedules than the energy-efficient EDF-VD algorithm discussed in [21,24] and we show that the energy consumption of the energy-efficient TT-Merge algorithm is less than that of the energy-efficient EDF-VD algorithm and PMC. We also prove an optimality result with respect to energy consumption for our algorithm.

Now we formally present our energy consumption minimization problem. Our objective is to minimize the system energy consumption by slowing down the tasks in the LO-criticality scenario while ensuring that they do not miss their deadlines using DVFS method. Without loss of generality, we calculate the energy consumption minimization up to the hyperperiod P of the task set. The idea is to find the energy-efficient LO-criticality WCET $\tilde{C}_{ik}(\text{LO})$ and the corresponding frequency f_{ik}^{LO} for each job j_{ik} of the task set in the hyperperiod which will minimize the energy consumption in the LO-criticality scenario.

Definition 3 We denote by $\tilde{C}_{ik}(\text{LO})$ the worst-case execution time at LO-criticality level of the job j_{ik} of task τ_i after DVFS using the processor frequency f_{ik}^{LO}, i.e.,

$$\tilde{C}_{ik}(\text{LO}) = C_i(\text{LO}) \cdot \frac{f_b}{f_{ik}^{\text{LO}}} \tag{3}$$

and by $\tilde{C}_{ik}(\text{HI})$ as the worst-case execution time at HI-criticality level of the job j_{ik} of task τ_i after DVFS, i.e.,

$$\tilde{C}_{ik}(\text{HI}) = \tilde{C}_{ik}(\text{LO}) + (C_i(\text{HI}) - C_i(\text{LO})) \tag{4}$$

Since we assume that in HI-criticality scenario every job runs at the base frequency f_b, i.e., DVFS is not used in HI-criticality scenario.

Problem 1 Energy Consumption Minimization for TT-Merge (***ECMTTM***) Given a dual-criticality task set schedulable under TT-Merge, decide offline the frequency f_{ik}^{LO} for each job j_{ik} of the task set in the hyperperiod P of the task set such that

- each j_{ik} of τ_i whether it is of LO-criticality or HI-criticality should execute for $\tilde{C}_{ik}(\text{LO})$ units of execution time instead of $C_i(\text{LO})$ with the frequency $f_{ik}^{\text{LO}} \leq f_b$ in a LO-criticality scenario, and
- each HI-criticality job j_{ik} of task τ_i should execute for $\tilde{C}_{ik}(\text{LO})$ units of execution time instead of $C_i(\text{LO})$ with the frequency $f_{ik}^{\text{LO}} \leq f_b$ and $C_i(\text{HI}) - C_i(\text{LO})$ units of execution time with the frequency f_b in a HI-criticality scenario, respectively

and further the LO-criticality scenario system energy consumption is minimized without affecting the schedulability of the system under TT-Merge.

We want to minimize the normalized energy consumption over a hyperperiod (i.e., the energy consumed per unit time) in the LO-criticality scenario by varying the processor frequency. The normalized energy consumption in a hyperperiod is:

$$\frac{1}{P} \cdot \sum_{\tau_i \in \mathcal{T} \wedge 1 \leq k \leq N_i} C_i(\text{LO}) f_b \cdot \frac{1}{f_{ik}^{\text{LO}}} \cdot \beta \cdot (f_{ik}^{\text{LO}})^{\alpha} \tag{5}$$

where N_i is the number of times a job of task τ_i will run in a hyperperiod, i.e., $N_i = \frac{P}{p_i}$.

The energy consumption minimization is constrained by:

- Bounds for the frequency for each job:

$$f_{ik}^{\text{LO}} \in [f_{min}, f_{max}]. \tag{6}$$

- Construction of temporary tables:

> It should be possible to construct $\tilde{\mathcal{T}}_{\text{LO}}$ and $\tilde{\mathcal{T}}_{\text{HI}}$ with $\tilde{C}_{ik}(\text{LO})$ and $\tilde{C}_{ik}(\text{HI})$ units of execution time of job j_{ik} as inputs using the TT-Merge algorithm of [13] reviewed in Sect. 3.1, without missing any deadline. (7)

- Construction of time-triggered table:

> It should be possible to construct $\tilde{\mathcal{S}}_{\text{LO}}$ using TT-Merge while ensuring the failure situation (4) in Sect. 3.2 does not occur, at any time t (i.e., the situation where $\tilde{\mathcal{T}}_{\text{LO}}[t]$ contains a LO-criticality job and $\tilde{\mathcal{T}}_{\text{HI}}[t]$ contains a HI-criticality job at time t). (8)

Finally, our energy consumption minimization problem is

$$\begin{aligned} &\textbf{minimize} \quad (5) \\ &\textbf{s.t.} \quad (6), (7), (8) \end{aligned} \tag{9}$$

Recall that the table $\tilde{\mathcal{S}}_{\text{HI}}$ is constructed using the table $\tilde{\mathcal{S}}_{\text{LO}}$. Also, the TT-Merge algorithm guarantees that if the table $\tilde{\mathcal{S}}_{\text{LO}}$ can be constructed, then so can the table $\tilde{\mathcal{S}}_{\text{HI}}$ [13].

5 The proposed algorithm

In this section, we propose an algorithm to find a processor frequency f_{ik}^{LO} and an energy-efficient LO-criticality WCET $\tilde{C}_{ik}(\mathrm{LO})$ for each job j_{ik} in the LO-criticality scenario. We also find $\tilde{C}_{ik}(\mathrm{HI})$ of each HI-criticality job using $\tilde{C}_{ik}(\mathrm{LO})$. Then we use $\tilde{C}_{ik}(\mathrm{LO})$ and $\tilde{C}_{ik}(\mathrm{HI})$ to construct the tables $\tilde{\mathcal{S}}_{\mathrm{LO}}$ and $\tilde{\mathcal{S}}_{\mathrm{HI}}$ using the TT-Merge algorithm. The key ideas behind our algorithm are as follows.

- We find a processor frequency for each job to run in the LO-criticality scenario.
- We find a LO-criticality worst-case execution time $\tilde{C}_{ik}(\mathrm{LO})$ of each job which gives a minimized energy consumption schedule in LO-criticality scenario using the processor frequency f_{ik}^{LO}.
- On a scenario change from LO-criticality to HI-criticality, all HI-criticality jobs run at the base frequency f_b.
- We achieve energy efficiency by reducing the idle time of the processor as much as possible.

Algorithm 1 Construct-LO-table($\mathscr{T}$)

Input : A task set $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, where j_{ik} is a job of $\tau_i =< a_{ik}, d_i, \chi_i, C_i(\mathrm{LO}), C_i(\mathrm{HI}) >$.
Output : Temporary table E_{LO}
Assume the earliest arrival time is 0.

1: Let P denote the hyperperiod of the task set $\mathscr{T}$: $P := lcm(p_1, p_2, \ldots, p_n)$.
2: The length of table E_{LO} is set to P;
3: Let L be the set of LO-criticality tasks of the task set;
4: Let O be the EDF order of the tasks of L on the time-line using $C_i(\mathrm{LO})$ units of execution for each job j_{ik};
5: **if** (any job cannot be scheduled) **then**
6: Declare failure;
7: Starting from the rightmost job segment of the EDF order of L, move each segment of a job j_{ik} as close to its deadline as possible in table E_{LO}.

We recall Algorithm 1 from [13] that constructs the table E_{LO} which includes only the LO-criticality tasks. This algorithm chooses the LO-criticality tasks from the task set and orders them in EDF order [23]. Then, all the job segments of the EDF schedule are moved as close to their deadline as possible so that no job misses its deadline in table E_{LO}.

Note that, if the arrival times of the jobs are not the same, then the jobs may execute in more than one segment, in general. If the arrival times of all the jobs are the same then, the jobs will execute in one segment.

Algorithm 2 Construct-HI-table($\mathscr{T}$)

Input : A task set $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, where job j_{ik} of $\tau_i =< a_{ik}, d_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}) >$.
Output : Temporary table E_{HI}

1: Let P denote the hyperperiod of the task set $\mathscr{T}$: $P := lcm(p_1, p_2, \ldots, p_n)$.
2: The length of table E_{HI} is set to P;
3: Let H be the set of HI-critical tasks of the task set;
4: Let O be the EDF order of the tasks of H on the time-line using $C_i(\text{HI})$ units of execution for job j_{ik} ;
5: **if** (any job cannot be scheduled) **then**
6: Declare failure;
7: Starting from the rightmost job segment of the EDF order of H, move each segment of a job j_{ik} as close to its deadline as possible in table E_{HI}.
8: **for** $i := 1$ to m **do**
9: Allocate $C_i(\text{LO})$ units of execution to job j_{ik} from its starting time in table E_{HI} and leave the rest unallocated;

Algorithm 2, also from [13], constructs the table E_{HI} which contains only the HI-criticality tasks. This algorithm chooses the HI-criticality tasks from the task set and orders them in EDF order. Then, all the job segments of the EDF schedule are moved as close to their deadline as possible so that no job misses its deadline in table E_{HI}. Then, out of the total allocation so far, the algorithm allocates $C_i(\text{LO})$ units of execution of job j_{ik} in table E_{HI} from the beginning of its slot and leaves the rest of the execution time of j_{ik} unallocated in table E_{HI}.

Algorithm 3 is the same as the main algorithm in [13] except it calls the function SetFrequency(E_{LO},E_{HI},E_{FINAL}) at the end. Function SetFrequency(E_{LO},E_{HI},E_{FINAL}) in Algorithm 4, the actual contribution of the paper, finds the energy efficient LO-criticality WCET $\tilde{C}_{ik}(\text{LO})$ after DVFS and the processor frequency f_{ik}^{LO} for each job j_{ik} which will be used by the TT-Merge algorithm to construct the scheduling tables $\tilde{\mathcal{S}}_{\text{LO}}$ and $\tilde{\mathcal{S}}_{\text{HI}}$. In step 1, all the job segments in table E_{FINAL} are moved to the right as close to their finishing time in table E_χ as possible, where χ is the criticality level of the job. After each job j_{ik} is moved to its right, the completion time of each job in the table E_{FINAL} is called its *finishing time* d_{ik}^{Δ}. In step 3, the function initializes $\tilde{C}_{ik}(\text{LO})$ to the value of $C_i(\text{LO})$. In step 6, the function finds all the empty intervals in the table E_{FINAL}. All the jobs found before the start of the first empty interval in the table E_{FINAL} have not been moved. So these jobs cannot be slowed down by lowering the processor frequency as they have no idle time between their arrival times and finishing times. In step 9, the function removes the set of jobs J' which cannot be slowed down from the set of jobs J, namely the jobs before the first empty interval, where J is the set of all jobs and J' is the set of all jobs which cannot be slowed down. In step 10, the possible expansion per unit of execution time r_{LO} for each job in J is calculated using the following formula.

$$r_{\text{LO}} = \frac{P - EMPTY_1^S}{\sum_{\forall j_{ik} \in J} C_{ik}(\text{LO})} \tag{10}$$

where $EMPTY_1^S$ is the start time of the first empty interval in table E_{FINAL}.

Algorithm 3 EE-TT-MERGE($\mathscr{T}$, table E_{LO}, table E_{HI})

Input : Table E_{LO}, table E_{HI}, $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, where job j_{ik} of $\tau_i =< a_{ik}, d_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}) >$.
Output : $\tilde{C}_{ik}(\text{LO})$, $\tilde{C}_{ik}(\text{HI})$, f_{ik}^{LO}.

1: Copy table E_{LO} and E_{HI} to $\tilde{E}_{\text{LO}}$ and $\tilde{E}_{\text{HI}}$, respectively;
2: Let P denote the hyperperiod of the task set $\mathscr{T}$: $P := lcm(p_1, p_2, \ldots, p_n)$.
3: The length of table E_{FINAL} is set to P;
4: $t := 0$;
5: **while** $(t \leq P)$ **do**
6: **if** $(\tilde{E}_{\text{LO}}[t] = NULL \ \& \ \tilde{E}_{\text{HI}}[t] = NULL)$ **then**
7: Search the tables $\tilde{E}_{\text{LO}}$ and $\tilde{E}_{\text{HI}}$ simultaneously from the beginning to find the first available job at time t;
8: Let k be the first occurrence of a job j_{ik} in $\tilde{E}_{\text{LO}}$ or $\tilde{E}_{\text{HI}}$;
9: **if** (Both LO-criticality & HI-criticality job are found) **then**
10: $E_{\text{FINAL}}[t] := \tilde{E}_{\text{LO}}[k]$;
11: $\tilde{E}_{\text{LO}}[k] := NULL$;
12: **else if** (LO-criticality job is found) **then**
13: $E_{\text{FINAL}}[t] := \tilde{E}_{\text{LO}}[k]$;
14: $\tilde{E}_{\text{LO}}[k] := NULL$;
15: **else if** (HI-criticality job is found) **then**
16: $E_{\text{FINAL}}[t] := \tilde{E}_{\text{HI}}[k]$;
17: $\tilde{E}_{\text{HI}}[k] := NULL$;
18: **else if** (NO job is found) **then**
19: $E_{\text{FINAL}}[t] := NULL$
20: $t := t + 1$;
21: **else if** $(\tilde{E}_{\text{LO}}[t] = NULL \ \& \ \tilde{E}_{\text{HI}}[t] \neq NULL)$ **then**
22: $E_{\text{FINAL}}[t] := \tilde{E}_{\text{HI}}[t]$;
23: $\tilde{E}_{\text{HI}}[t] := NULL$;
24: $t := t + 1$;
25: **else if** $(\tilde{E}_{\text{LO}}[t] \neq NULL \ \& \ \tilde{E}_{\text{HI}}[t] = NULL)$ **then**
26: $E_{\text{FINAL}}[t] := \tilde{E}_{\text{LO}}[t]$;
27: $\tilde{E}_{\text{LO}}[t] := NULL$;
28: $t := t + 1$;
29: **else if** $(\tilde{E}_{\text{LO}}[t] \neq NULL \ \& \ \tilde{E}_{\text{HI}}[t] \neq NULL)$ **then**
30: Declare failure;
31: SetFrequency(E_{LO},E_{HI},E_{FINAL});

Algorithm 4 Function SetFrequency(E_{LO},E_{HI},E_{FINAL},$\mathscr{T}$)

Input : E_{LO}, E_{HI}, E_{FINAL} and a task set $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, where $j_{ik} = (a_{ik}, d_i, \chi_i, C_i(\text{LO}), C_i(\text{HI}))$ is the k$^{\text{th}}$ job of τ_i.
Output : $\tilde{C}_{ik}(\text{LO})$, $\tilde{C}_{ik}(\text{HI})$, f_{ik}^{LO}.
/* $\tilde{C}_{ik}(\text{LO})$ and $\tilde{C}_{ik}(\text{HI})$ are the LO-criticality and HI-criticality execution time of each job after DVFS, respectively and f_{ik}^{LO} is the LO-criticality processor frequency of each job in DVFS */

1: Starting from the rightmost job segment of the table E_{FINAL}, move each segment of a job j_{ik} as close to its finishing time in E_χ as possible, where χ is the criticality of j_{ik};
2: Let J be the set of all jobs in table E_{FINAL};
3: **for** each job in J **do**
4: $\tilde{C}_{ik}(\text{LO}) := C_i(\text{LO})$;
5: Find all the empty intervals in the table E_{FINAL};
6: Let *EMPTY* be the set of empty intervals in the table E_{FINAL};
7: Let J' be the set of jobs before the first empty interval;
8: $J := J \setminus J'$;

9: $r_{\text{LO}} := \frac{P - EMPTY_1^S}{\sum_{\forall j_{ik} \in J} C_{ik}(\text{LO})}$; /* $EMPTY_1^S$ is the beginning of the first empty interval */
10: **if** $(\frac{f_b}{r_{\text{LO}}} \leq f_{\min})$ **then**
11: **for** each job in J **do**
12: $\tilde{C}_{ik}(\text{LO}) := C_{ik}(\text{LO}) \cdot \frac{f_b}{f_{\min}}$;
13: **else**
14: **for** each job in J **do**
15: $\tilde{C}_{ik}(\text{LO}) := C_{ik}(\text{LO}) \cdot r_{\text{LO}}$;
16: Find the finishing time (d_{ik}^{Δ}) of each job in J;
17: Sort the jobs in table E_{FINAL} according to their finishing times in non-decreasing order;
18: **for** each job j_{ik} in increasing order of the corresponding d_{ik}^{Δ} **do**
19: **if** $(\tilde{C}_{ik}(\text{LO}) + \sum_{j_l \in J \wedge d_l^{\Delta} < d_{ik}^{\Delta}} \tilde{C}_l(\text{LO}) \geq d_{ik}^{\Delta})$ **then**
20: $t^{\Delta} :=$ CheckEmptySpace(E_{FINAL},d_{ik}^{Δ});
21: /* compute the amount of time δ by which j_{ik} misses its finishing time */
22: $\delta := d_{ik}^{\Delta} - \sum_{j_l \in J \wedge d_l^{\Delta} \leq d_{ik}^{\Delta}} \tilde{C}_l(\text{LO})$;
23: **for** each job $j_l \in J \wedge a_l \geq t^{\Delta} \wedge d_l^{\Delta} \leq d_{ik}^{\Delta}$ **do**
24: /* subtract a total of δ amount of execution time in equal parts from jobs whose arrival time is greater than or equal to t^{Δ} and finishing time is less than or equal to d_{ik}^{Δ} */
25: $\tilde{C}_l(\text{LO}) := \tilde{C}_l(\text{LO}) - \frac{\delta}{\sum_{j_l \in J \wedge a_l \geq t^{\Delta} \wedge d_l^{\Delta} \leq d_{ik}^{\Delta}} C_l(\text{LO})} \cdot C_l(\text{LO})$;
26: **for** each job $j_l \in J \wedge d_l^{\Delta} > d_{ik}^{\Delta}$ **do**
27: $\tilde{C}_l(\text{LO}) := \tilde{C}_l(\text{LO}) + \frac{\delta}{\sum_{j_l \in J \wedge d_l^{\Delta} > d_{ik}^{\Delta}} C_l(\text{LO})} \cdot C_l(\text{LO})$;
28: Goto step 18;
29: **for** each job $j_{ik} \in J$ **do**
30: $f_{ik}^{\text{LO}} = \max\{f_{\min}, \frac{C_{ik}(\text{LO})}{\tilde{C}_{ik}(\text{LO})} \cdot f_b\}$;
31: $\tilde{C}_{ik}(\text{HI}) := (C_i(\text{HI}) - C_i(\text{LO})) + \tilde{C}_{ik}(\text{LO})$;

Algorithm 5 CheckEmptySpace(E_{FINAL},d^{Δ})

Output : Time instant t^{Δ} of an empty space

1: $\delta := d^{\Delta} - \sum_{j_l \in J \wedge d_l^{\Delta} \leq d^{\Delta}} \tilde{C}_l(\text{LO})$;
2: **for** each job $j_l \in J \wedge d_l^{\Delta} \leq d^{\Delta}$ **do**
3: $\tilde{C}_l(\text{LO}) := \tilde{C}_l(\text{LO}) - \frac{\delta}{\sum_{j_l \in J \wedge d_l^{\Delta} \leq d^{\Delta}} C_l(\text{LO})} \cdot C_l(\text{LO})$;
4: Simulate each job $j_l \in J \wedge d_l^{\Delta} \leq d^{\Delta}$ using the EDF algorithm considering d_l^{Δ} as the deadline.
5: **if** (an empty space is found) **then**
6: return the end time (t^{Δ}) of the empty space;
7: **else**
8: return 0;

The fraction r_{LO} is used as follows to compute the processor frequency for each job. If the processor frequency $\frac{f_b}{r_{\text{LO}}}$ is less than or equal to $f_{\min}$, then $\tilde{C}_{ik}(\text{LO})$ of each job $j_{ik} \in J$ is updated using the following formula:

$$\tilde{C}_{ik}(\text{LO}) = C_{ik}(\text{LO}) \cdot \frac{f_b}{f_{\min}} \tag{11}$$

On the other hand, if $\frac{f_b}{r_{\text{LO}}} > f_{\min}$, then $\frac{f_b}{r_{\text{LO}}}$ is the lowest possible frequency or each job. So $\tilde{C}_{ik}(\text{LO})$ of each job $j_{ik} \in J$ is updated using the following formula:

$$\tilde{C}_{ik}(\text{LO}) = C_{ik}(\text{LO}) \cdot r_{\text{LO}} \tag{12}$$

In step 20, the function finds the finishing time d_{ik}^{Δ} of each job j_{ik} in J from the table E_{FINAL}. Then all the jobs are sorted according to their finishing time in non-decreasing order. In step 22, the function checks that each job must meet its finishing time d_{ik}^{Δ} using $\tilde{C}_{ik}(\text{LO})$ units of execution time. The schedulability of each job can be verified by the following condition:

$$\tilde{C}_{ik}(\text{LO}) + \sum_{j_l \in J \wedge d_l^{\Delta} < d_{ik}^{\Delta}} \tilde{C}_l(\text{LO}) \leq d_{ik}^{\Delta} \tag{13}$$

If no job execution crosses its finishing time d_{ik}^{Δ}, then the total energy consumption found for the LO-criticality scenario is the minimum, as proved below in Lemma 3. On the other hand, if a job j_{ik} crosses its finishing time in the above process, then the algorithm finds processor frequencies for each job which are as close to $\max(f_{\min}, \frac{f_b}{r_{\text{LO}}})$ as possible. Suppose job j_{ik} misses its deadline with δ time remaining to be executed. In this case, the algorithm calls the function CheckEmptySpace(). This function reduces $\tilde{C}_l(\text{LO})$ of all the jobs whose finishing time is less than d_{ik}^{Δ} by equal proportion (given in Step 3 of the function CheckEmptySpace()). Then it simulates all the jobs whose deadlines are less than d_{ik}^{Δ} using the EDF algorithm, where the finishing time of each job in the table E_{FINAL} is considered to be the deadline. If an empty space is found, then it returns the time (t^{Δ}) of that empty space. Otherwise, it returns the initial time, i.e., 0. Note that $t^{\Delta} \neq 0$ means the jobs present before t^{Δ} in simulation are the only ready jobs before time t^{Δ} and the arrival time of all the remaining jobs are greater or equal to t^{Δ}. Then $\tilde{C}_l(\text{LO})$ of all the jobs j_l whose $a_l \geq t^{\Delta}$ and $d_l \leq d_{ik}^{\Delta}$ is updated using the following formula:

$$\tilde{C}_l(\text{LO}) := \tilde{C}_l(\text{LO}) - \frac{\delta}{\sum_{j_l \in J \wedge a_l \geq t^{\Delta} \wedge d_l^{\Delta} \leq d_{ik}^{\Delta}} \tilde{C}_l(\text{LO})} \cdot \tilde{C}_l(\text{LO}) \tag{14}$$

Here the extra amount of execution time δ is subtracted in equal proportion from each job j_l whose $a_l \geq t^{\Delta}$ and $d_l \leq d_{ik}^{\Delta}$ to accommodate j_{ik} between its arrival time a_{ik} and finishing time d_{ik}^{Δ}. The process continues until all the jobs whose finishing times lie between t^{Δ} and d_{ik}^{Δ} meet their finishing times. Then the extra amount of workload δ is distributed in equal proportion among all the jobs whose finishing time is greater than d_{ik}^{Δ} using the following formula:

$$\tilde{C}_l(\text{LO}) := \tilde{C}_l(\text{LO}) + \frac{\delta}{\sum_{j_l \in J \wedge d_l^{\Delta} > d_{ik}^{\Delta}} \tilde{C}_l(\text{LO})} \cdot \tilde{C}_l(\text{LO}) \tag{15}$$

The above process continues until all the jobs of the table E_{FINAL} meet their finishing times with the computed $\tilde{C}_{ik}(\text{LO})$. Then the processor frequency for each job is calculated using the following formula:

$$f_{ik}^{\mathrm{LO}} = \max\left\{f_{\min}, \frac{C_{ik}(\mathrm{LO})}{\tilde{C}_{ik}(\mathrm{LO})} \cdot f_b\right\} \tag{16}$$

Finally, $\tilde{C}_{ik}(\mathrm{HI})$ for each HI-criticality job j_{ik} is computed using the following formula:

$$\tilde{C}_{ik}(\mathrm{HI}) := (C_i(\mathrm{HI}) - C_i(\mathrm{LO})) + \tilde{C}_{ik}(\mathrm{LO}) \tag{17}$$

Example 2 Consider the following task set. Here we assume that the base frequency f_b of the processor is the same as $f_{\max}$, i.e., 1 and the minimum frequency $f_{\min}$ is 0.2, where $\beta = 1$ and $\alpha = 2.5$. Let us first find tables E_{LO} and E_{HI} in which the LO-criticality and HI-criticality jobs are allocated respectively.

Task	Period	Criticality	$C_i(\mathrm{LO})$	$C_i(\mathrm{HI})$
τ_1	8	HI	2	5
τ_2	12	LO	1	1
τ_3	16	LO	2	2

- The hyperperiod of the task set is 48, and hence so is the length of the tables E_{LO} and E_{HI}.
- According to Algorithm 1, we choose the LO-criticality jobs and allocate them in E_{LO} in EDF order. Then, each segment of the jobs in EDF order is shifted as close to its deadline as possible according to its $C_i(\mathrm{LO})$ units of execution. The resulting table E_{LO} is given in Fig. 2.
- According to Algorithm 2, we choose the HI-criticality jobs in order to allocate them in E_{HI} in EDF order. Then each segment of the jobs in EDF order is shifted as close to its deadline as possible according to its $C_i(\mathrm{HI})$ units of execution. We know that j_{11} whose arrival time and deadline are 0 and 8 respectively, is scheduled in the interval [0, 5] by the EDF algorithm. After shifting to the right as close to its deadline as possible, j_{11} will be scheduled in the interval [3, 8] as the deadline of j_{11} is 8. Similarly, the other arrivals are allocated in the intervals [11, 16], [19, 24], [27, 32], [35, 40] and [43, 48] respectively.
- Then we allocate $C_i(\mathrm{LO})$ units of execution of each arrival of j_{ik} and leave the $(C_i(\mathrm{HI}) - C_i(\mathrm{LO}))$ units of execution unallocated. Here the first arrival of j_{11} has been allocated its $C_i(\mathrm{LO})$ units of execution time in the interval [3, 8]. Then we empty the occurrence of j_{11} in the interval [5, 8] which leaves the interval [3, 5] in table E_{HI}. We repeat the same process for the other arrivals of task j_{11}. The resulting table E_{HI} after this modification is given in Fig. 3.
- Finally, we construct the table E_{FINAL} from these two temporary tables.

We construct the table E_{FINAL} according to Algorithm 3.

- We start from time $t = 0$.
- At $t = 0$, both E_{LO} and E_{HI} are empty. Then we search both the tables E_{LO} and E_{HI} starting from time $t = 0$. We find j_{21} in table E_{LO} and j_{11} in table E_{HI} available at time $t = 0$. So we allocate the LO-criticality job from E_{LO}, i.e., j_{21}. We empty the interval [11, 12] in E_{LO} from where the first occurrence of j_{21} was found.
  At $t = 1$, both E_{LO} and E_{HI} are empty. Then we search both the tables E_{LO} and E_{HI} starting from time $t = 1$. We find j_{31} in table E_{LO} and j_{11} in table E_{HI} available at time $t = 1$. So we allocate the LO-criticality job from E_{LO}, i.e., j_{31}. We empty the interval

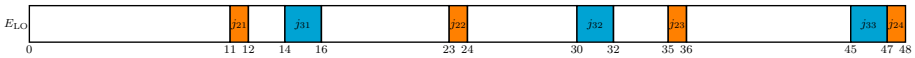


Fig. 2 Table E_{LO}

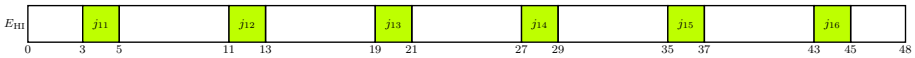


Fig. 3 Table E_{HI}

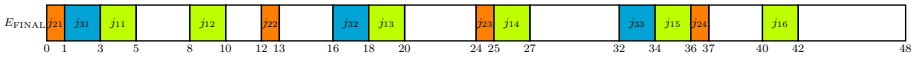


Fig. 4 Table E_{FINAL}

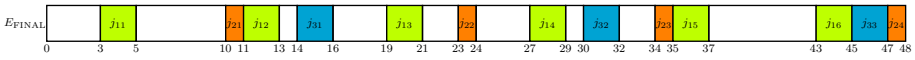


Fig. 5 Table E_{FINAL} after the moving the job segments to their right

[14, 15] in E_{LO} from where the first occurrence of j_{31} was found. We continue in this way till all the jobs have been allocated in table E_{FINAL}.

- The resulting table E_{FINAL} is given in Fig. 4.
- Then the SetFrequency() function is called, where all the jobs, beginning from the right end of E_{FINAL}, are moved as close to their finishing time in table E_χ as possible, where χ is the criticality level of a job. We start with j_{16} present in the interval [40, 42] in E_{FINAL} and move it to the interval [43, 45] which is the finishing time of j_{16} in E_{HI}. We continue until we move all the jobs to their right. Then the resulting table E_{FINAL} is given in Fig. 5.
- Here $EMPTY_1^S$ and P are 0 and 48, respectively. The total LO-criticality execution time between $EMPTY_1^S$ and P in table E_{FINAL} is 22.
- $r_{\text{LO}} = \frac{P - EMPTY_1^S}{\sum_{\forall j_{ik} \in J} C_{ik}(\text{LO})} = \frac{48-0}{22} = 2.181$, where $f_{\min} < \frac{f_b}{r_{\text{LO}}}$. This means the lowest possible processor frequency for each job in J is 0.458.
- We find the $\tilde{C}_{ik}(\text{LO})$ for each job of tasks $\langle \tau_1, \tau_2, \tau_3 \rangle$ to be $\langle 4.363, 2.183, 4.363 \rangle$.
- Now we use the relation 13 to check the schedulability of each job with the $\tilde{C}_{ik}(\text{LO})$ units of execution time. We show this in the table given below.
- There is a finishing time miss for job j_{15} by 0.084 units in Table 2 which is shown in bold. Then the 0.084 units of time is reduced from all the jobs whose finishing time lies in [0, 37]. As a result, the processor frequencies of those jobs are increased.
- On the other hand, the 0.084 units of extra time is added to the execution times of all those jobs whose finishing time is greater than 37. As a result, the processor frequencies of such jobs are reduced. This is shown in Table 3.
- Since we do not find any finishing time miss, the processor frequency for each job is the final one.
- The total normalized energy consumption by the TT-Merge energy-efficient algorithm using the above processor frequencies is 0.09.
- The processor frequencies $f_{\text{LO}}^{\text{LO}}$ and $f_{\text{HI}}^{\text{LO}}$ for the task set of this example calculated by the EDF-VD energy-efficient algorithm are 0.54 and 0.65, respectively.
- The normalized energy consumption computed by the EDF-VD energy-efficient algorithm is 0.21 which is greater than the TT-Merge energy-efficient algorithm.

Table 2 Table showing initial processor frequency allotment

Job	Arrival time	Deadline	Exec Assgn	Rem. exec	Total time elapsed	Frequency alloted
j_{11}	0	5	4.363	0	4.363	0.458
j_{21}	0	11	2.181	0	6.544	0.458
j_{12}	8	13	4.363	0	12.363	0.458
j_{31}	0	16	4.363	0	15.27	0.458
j_{13}	16	21	4.363	0	20.363	0.458
j_{22}	12	24	2.181	0	21.814	0.458
j_{14}	24	29	4.363	0	28.363	0.458
j_{32}	16	32	4.363	0	30.54	0.458
j_{23}	24	35	2.181	0	32.721	0.458
$\boldsymbol{j_{15}}$	**32**	**37**	**4.363**	**0.084**	**37.084**	**0.458**
j_{16}	40	45	4.363	4.363	0	0.458
j_{33}	32	47	4.363	4.363	0	0.458
j_{24}	36	48	2.181	2.181	0	0.458

Since j_{12} is not ready at time instant 8, the already ready job j_{31} is executed between time instant 6.544 and 8.

Table 3 Table showing final processor frequency allotment

Job	Arrival time	Deadline	Exec Assgn	Rem. exec	Total time elapsed	Frequency alloted
j_{11}	0	5	4.353	0	4.353	0.459
j_{21}	0	11	2.176	0	6.53	0.459
j_{12}	8	13	4.353	0	12.353	0.459
j_{31}	0	16	4.353	0	15.236	0.459
j_{13}	16	21	4.353	0	20.353	0.459
j_{22}	12	24	2.176	0	21.765	0.459
j_{14}	24	29	4.353	0	28.353	0.459
j_{32}	16	32	4.353	0	30.471	0.459
j_{23}	24	35	2.176	0	32.647	0.459
j_{15}	32	37	4.353	0	37	0.459
j_{16}	40	45	4.4	0	45.8	0.45
j_{33}	32	47	4.4	0	41.4	0.45
j_{24}	36	48	2.2	0	48	0.45

5.1 Energy-efficient EDF-VD versus Energy-efficient TT-Merge

First, we prove that the schedule constructed by the energy-efficient TT-Merge algorithm using $\tilde{C}(\text{LO})$ units of execution time does not have any idle time. Then we prove that the energy-efficient TT-Merge algorithm finds the lowest possible energy consumption among all frequency assignments to jobs that lead to schedulability by the TT-Merge algorithm. Finally, we prove the dominance of Energy-efficient TT-Merge over Energy-efficient EDF-VD. Recall that r_{LO} is $\frac{P-EMPTY_1^S}{\sum_{\forall j_{ik} \in J} C_{ik}(\text{LO})}$, where P is the hyperperiod and $EMPTY_1^S$ is the start time of the first empty interval. We use the following result in our proof.

Lemma 2 *The optimal processor frequency to minimize the total energy consumption while meeting all the finishing times d^{Δ} is constant and equal to $f^{\mathrm{LO}} = \max\{f_{\min}, U_{\mathrm{LO}} \cdot f_b\}$, where $U_{\mathrm{LO}} = \frac{\sum_{\forall j_{ik}} C_{ik}(\mathrm{LO})}{P}$. Moreover, when used along with this speed f^{LO} which fully utilize the processor can be used to obtain a feasible schedule.*

Proof We know that the function SetFrequency() assigns $\tilde{C}_{ik}(\mathrm{LO})$ units of execution time to each job j_{ik} between its arrival time a_{ik} and its finishing time d_{ik}^{Δ} with processor frequencies $f^{\mathrm{LO}} = U_{\mathrm{LO}} \cdot f_b$. If all the jobs complete execution before their finishing time d_{ik}^{Δ}, then we say that each job is stretched with equal proportion. We need to show that

$$\frac{\sum_{\forall j_{ik}} \tilde{C}_{ik}(\mathrm{LO})}{P} = 1 \tag{18}$$

We know that

$$\tilde{C}_{ik}(\mathrm{LO}) = C_{ik}(\mathrm{LO}) \cdot \frac{f_b}{f^{\mathrm{LO}}} = \frac{C_{ik}(\mathrm{LO})}{U_{\mathrm{LO}}} \tag{19}$$

Without loss of generality, we assume $f_b = 1$. Now we replace $\tilde{C}_{ik}(\mathrm{LO})$ of the left hand side in Eq. 18 with Eq. 19. So clearly $\frac{\sum \tilde{C}_{ik}(\mathrm{LO})}{P} = 1$. if $f_{\min} > U_{\mathrm{LO}} \cdot f_b$, then we choose the minimum processor frequency available. □

Fact 1 If all jobs in the hyperperiod for a given task can be scheduled using the same processor frequency $\frac{f_b}{r_{\mathrm{LO}}}$ for each job then the function SetFrequency() will assign this frequency to each job. This is because lines 23-35 in the function SetFrequency() are never executed.

Lemma 3 *A task set $\mathscr{T}$ with all tasks scheduled with the same processor frequency $\frac{f_b}{r_{\mathrm{LO}}}$ without violating any finishing time d^{Δ} in the LO-criticality scenario and guaranteeing the execution of $(C(\mathrm{HI}) - C(\mathrm{LO}))$ time units of each HI-criticality job results in the minimum energy consumption among possible frequency assignments under the TT-Merge algorithm.*

Proof We try to assign a uniform processor frequency to each job and utilize the total hyperperiod as much as possible. Without loss of generality, $EMPTY_1^S$ is assumed to be 0. Then the lowest possible processor frequency $\frac{f_b}{r_{\mathrm{LO}}}$ is the same as the LO-criticality utilization (U_{LO}) of the task set $\mathscr{T}$, where $f_b = f_{\max} = 1$. If $\frac{f_b}{r_{\mathrm{LO}}} > f_{\min}$ and no job misses its finishing time d^{Δ}, then the energy-efficient TT-Merge algorithm generates a schedule with no idle time and each job is expanded with equal proportion using the processor frequency $\frac{f_b}{r_{\mathrm{LO}}}$. In this way, we use the total hyperperiod available to run the task set $\mathscr{T}$, i.e., the total effective task utilization is 1. We can see this from the equation given below:

$$\frac{\sum_{\forall j_{ik}} \frac{C_{ik}(\mathrm{LO})}{\frac{f_b}{r_{\mathrm{LO}}}}}{P} = \frac{U_{\mathrm{LO}}}{\frac{f_b}{r_{\mathrm{LO}}}} = 1 \tag{20}$$

From the function SetFrequency(), it is clear that we choose $f_{\min}$ as the lowest possible processor frequency, if $\frac{f_b}{r_{\mathrm{LO}}} \leq f_{\min}$. It can be seen that the maximum processor frequency between $f_{\min}$ and $\frac{f_b}{r_{\mathrm{LO}}}$ will always result in a total effective task utilization which is not greater than 1. This follows from Lemma 2. □

If a job misses its finishing time d_{ik}^{Δ} with processor frequency $\frac{f_b}{r_{\mathrm{LO}}}$, then the lowest possible energy solution does not exist. So our algorithm searches for the processor frequencies for each job which are as close to $\frac{f_b}{r_{\mathrm{LO}}}$ as possible.

Lemma 4 *In the proposed energy-efficient TT-Merge algorithm, i.e., Algorithm 3, the time-triggered schedule generated with* $\tilde{C}_{ik}(\mathrm{LO})$ *units of execution time does not contain any idle time interval* $[t, t']$ *where a job with processor frequency* $f_{ik}^{\mathrm{LO}} > f_{\min}$ *is available.*

Proof This is easy to see from the algorithm. Without loss of generality, we consider $EMPTY_1^S$ to be 0. Initially, the total hyperperiod is distributed among all the jobs. If all the jobs satisfy condition 13 with the assigned execution times, then there will not be any idle time in the schedule. Here idle slots are the time slots where either no job is available or a job cannot be stretched because the assigned processor frequency is less than $f_{\min}$. Suppose a job j_{ik} misses its finishing time (d_{ik}^{Δ}) by δ amount of times in table E_{FINAL}. Then we call the function CheckEmptySpace() which finds empty spaces before d_{ik}^{Δ} in table E_{FINAL}, if any. First, the function subtracts the extra amount of the assigned execution time δ by which job j_{ik} misses its finishing time d_{ik}^{Δ} is subtracted in equal proportion from each job whose finishing time is less than d_{ik}^{Δ}. Then it simulates to find an empty space from time instant 0 to d_{ik}^{Δ} using EDF algorithm. If an empty space is not found, then we finalize the processor frequency for each job. Otherwise, we subtract the extra amount of the assigned execution time δ in equal proportion from each job j_l whose $a_l \geq t^{\Delta}$ and $d_l \leq d_{ik}^{\Delta}$. Then job j_{ik} meets its finishing time exactly along with all the jobs j_l whose $a_l \geq t^{\Delta}$ and $d_l \leq d_{ik}^{\Delta}$. from the above discussion, it is clear that the jobs which are allocated before t^{Δ} will not be assigned new processor frequencies, whereas the jobs between t^{Δ} and d_{ik}^{Δ} are. There will not be any idle time in the schedule after d_{ik}^{Δ} as the extra amount of execution time δ is distributed in equal proportion among the jobs whose finishing time is greater than d_{ik}^{Δ}. From the above arguments, it is clear that the schedule generated using our algorithm does not have any idle time. □

Lemma 5 *The frequency assigned by the function SetFrequency() to each job of task set* $\mathscr{T}$ *when the condition of Lemma 3 is not satisfied (i.e., the task set is not scheduled by energy-efficient TT-Merge with the frequency assignment of* $\frac{f_b}{r_{\mathrm{LO}}}$ *for all tasks) results in minimum energy consumption.*

Proof Initially, the function SetFrequency() assigns a single processor frequency f^{LO} to each job, where $f^{\mathrm{LO}} = \frac{f_b}{r_{\mathrm{LO}}}$. If no job misses its finishing time d_{ik}^{Δ}, then each job gets the minimum frequency f^{LO}. This is clear from Lemma 3. So our energy consumption function becomes as follows:

$$\sum_{\tau_i \in \mathscr{T} \wedge 1 \leq k \leq N_i} C_i(\mathrm{LO}) f_b \cdot \frac{1}{f^{\mathrm{LO}}} \cdot (f^{\mathrm{LO}})^{\alpha} \tag{21}$$

Then the extra amount of the assigned execution time δ by which job j_{ik} misses its finishing time d_{ik}^{Δ} is subtracted in equal proportion from each job whose finishing time is less than d_{ik}^{Δ}. Then we call the function CheckEmptySpace() which finds empty spaces before d_{ik}^{Δ} in the table E_{FINAL}, if any. If an empty space is not found we finalize the processor frequency for each job. Otherwise, we subtract the extra amount of the assigned execution time δ in equal proportion from each job j_l whose $a_l \geq t^{\Delta}$ and $d_l \leq d_{ik}^{\Delta}$. If the function CheckEmptySpace() finds an empty space at t^{Δ}, then our algorithm does not change the assigned processor frequency to the jobs which are scheduled before t^{Δ}. In other words, the frequency of each job j_l whose $a_l \geq t^{\Delta}$ and $d_l \leq d_{ik}^{\Delta}$ is increased by $x = \frac{K}{\Psi - \delta} - f^{\mathrm{LO}}$ and the frequency of each job whose finishing time lies after d_{ik}^{Δ} is decreased by $y = f^{\mathrm{LO}} - \frac{L}{\Omega + \delta}$, where

$$K = \sum_{j_r \in J \wedge a_r \geq t^{\Delta} \wedge d_r^{\Delta} \leq d_{ik}^{\Delta}} C_r(\mathrm{LO}),$$

$$L = \sum_{j_s \in J \wedge d_s^{\Delta} > d_{ik}^{\Delta}} C_s(\mathrm{LO}),$$

$$\Psi = \sum_{j_r \in J \wedge a_r \geq t^{\Delta} \wedge d_r^{\Delta} \leq d_{ik}^{\Delta}} \tilde{C}_r(\mathrm{LO}) \quad \text{and}$$

$$\Omega = \sum_{j_s \in J \wedge d_s^{\Delta} > d_{ik}^{\Delta}} \tilde{C}_s(\mathrm{LO}).$$

In other words, K is the sum of execution times of all jobs j_r whose $a_r \geq t^{\Delta}$ and $d_r^{\Delta} \leq d_{ik}^{\Delta}$ and L is the sum of execution times of all jobs whose finishing times lie after d_{ik}^{Δ}. Similarly, Ψ is the sum of execution times after DVFS of all jobs whose $a_r \geq t^{\Delta}$ and $d_r^{\Delta} \leq d_{ik}^{\Delta}$ and Ω is the sum of execution times after DVFS of all jobs whose finishing times lie after d_{ik}^{Δ}. Now our energy consumption function is modified as follows:

$$\sum_{j_q \in J \wedge a_q \leq t^{\Delta} \wedge d_q \leq t^{\Delta}} C_q(\mathrm{LO}) \cdot (f^{\mathrm{LO}})^{\alpha-1} + \sum_{j_r \in J \wedge a_r \geq t^{\Delta} \wedge d_r^{\Delta} \leq d_{ik}^{\Delta}} C_r(\mathrm{LO}) \cdot (f^{\mathrm{LO}} + x)^{\alpha-1} + \sum_{j_s \in J \wedge d_s^{\Delta} > d_{ik}^{\Delta}} C_s(\mathrm{LO}) \cdot (f^{\mathrm{LO}} - y)^{\alpha-1} \tag{22}$$

where $(f^{\mathrm{LO}})^{\alpha-1} = \frac{1}{f^{\mathrm{LO}}} \cdot (f^{\mathrm{LO}})^{\alpha}$. The first part of Eq. 22 always consumes less energy as each job is assigned the same processor frequency and the finishing time of each job is less than t^{Δ}. From Eq. 22, we can verify that $\sum_{j_q \in J \wedge d_q^{\Delta} \leq t^{\Delta}} C_q(\mathrm{LO}) \cdot (f^{\mathrm{LO}})^{\alpha-1}$ will always consume minimum energy as the processor frequency for each job is minimum (f^{LO}), because further reducing the frequency will create empty spaces in the schedule. We need to show that any processor frequency other than $f^{\mathrm{LO}} + x$ and $f^{\mathrm{LO}} - y$ results in increasing the total energy consumption. It is easy to see that decreasing the value of δ decreases the processor frequency $f^{\mathrm{LO}} + x$ which will lead the job j_{ik} to miss its finishing time d_{ik}^{Δ}. Furthermore, we need to show that the energy consumption will increase with the increase of δ. This is because, increasing δ results in increasing $f^{\mathrm{LO}} + x$ and decreasing $f^{\mathrm{LO}} - y$. So we need to show that the sum of the last two terms of Eq. 22 is an increasing function with respect to δ. Now we express the sum of the last two terms of Eq. 22 with respect to δ as follows:

$$\sum_{j_r \in J \wedge a_r \geq t^{\Delta} \wedge d_r^{\Delta} \leq d_{ik}^{\Delta}} C_r(\mathrm{LO}) \cdot \left(f^{\mathrm{LO}} + \frac{K}{\Psi - \delta} - f^{\mathrm{LO}} \right)^{\alpha-1} + \sum_{j_s \in J \wedge d_s^{\Delta} > d_{ik}^{\Delta}} C_s(\mathrm{LO}) \cdot \left(f^{\mathrm{LO}} - f^{\mathrm{LO}} + \frac{L}{\Omega + \delta} \right)^{\alpha-1} \tag{23}$$

We simplify Eq. 23 to get the following equation:

$$\frac{K^{\alpha}}{(\Psi - \delta)^{\alpha-1}} + \frac{L^{\alpha}}{(\Omega + \delta)^{\alpha-1}} \tag{24}$$

To show the function in Eq. 24 is an increasing function of δ, we need to differentiate the function with respect to δ. The derivative of the function in Eq. 24 with respect to δ is:

$$\frac{(\alpha - 1) \cdot K^{\alpha}}{(\Psi - \delta)^{\alpha}} - \frac{(\alpha - 1) \cdot L^{\alpha}}{(\Omega + \delta)^{\alpha}} \tag{25}$$

Now

$$\begin{aligned}
&\frac{(\alpha-1)\cdot K^{\alpha}}{(\Psi-\delta)^{\alpha}} - \frac{(\alpha-1)\cdot L^{\alpha}}{(\Omega+\delta)^{\alpha}} \geq 0 \\
&\Leftrightarrow \frac{K^{\alpha}}{(\Psi-\delta)^{\alpha}} \geq \frac{L^{\alpha}}{(\Omega+\delta)^{\alpha}} \\
&\Leftrightarrow \left(\frac{K}{L}\right)^{\alpha} \geq \left(\frac{\Psi-\delta}{\Omega+\delta}\right)^{\alpha} \\
&\Leftrightarrow \frac{K}{L} \geq \frac{\Psi-\delta}{\Omega+\delta} \qquad (26)
\end{aligned}$$

If the above condition is true, then the function in Eq. 24 is an increasing function with respect to δ. It is easy to see that decreasing the value of δ leads to a deadline miss. Hence we cannot decrease the value of δ. If we increase δ, then the right-hand side of the inequality 26 decreases and is always less than the left-hand side of the inequality 26. This is because, increasing δ results in increasing the value of denominator of the right-hand side and decreasing the value of the numerator. So the right-hand side will decrease with increasing δ and the condition is true with increasing δ. Hence, the function in Eq. 24 is an increasing function with respect to δ. □

As a corollary to Lemma 3 and Lemma 5, we have our main result.

Theorem 2 *If Algorithm 3 does not declare failure, then the energy consumption in a schedule produced by it is optimal for TT-Merge.*

Proof Algorithm 3 achieves the minimum value of δ without missing any deadline. Hence the resulting energy consumption is minimum among all possible feasible schedules according to TT-Merge. □

Theorem 3 *The schedule according to our algorithm consumes no more energy than the one produced by the energy-efficient EDF-VD algorithm.*

Proof From Theorem 1, we know that EDF-VD schedules fewer task sets than the TT-Merge algorithm.

Here we consider the case where both energy-efficient EDF-VD and energy-efficient TT-Merge can schedule a task set. We know that the schedules constructed by the energy-efficient TT-Merge algorithm are without any idle time. Without loss of generality we assume that schedules produced by energy-efficient EDF-VD also have no idle time, as in that case we can always expand a job and decrease the total energy consumption. Since the energy-efficient EDF-VD algorithm finds an optimal solution in two cases, we need to consider both.

Case 1 Energy efficient EDF-VD algorithm assigns the minimum frequency $f_{\min}$ to each job.

According to Lemma 3, our algorithm assigns the same processor frequency $\frac{f_b}{r_{\mathrm{LO}}}$ to each job, if all job meet their finishing time d^{Δ}. If $\frac{f_b}{r_{\mathrm{LO}}} \leq f_{\min}$, then we set $\frac{f_b}{r_{\mathrm{LO}}}$ to $f_{\min}$. From Lemma 3, we know that the maximum processor frequency between $f_{\min}$ and $\frac{f_b}{r_{\mathrm{LO}}}$ will always result in the minimum energy solution. Since both the algorithms use the EDF algorithm to check for schedulability, if energy-efficient EDF-VD assigns $f_{\min}$ to each job then so does our algorithm.

Case 2 Energy efficient EDF-VD fails to find the lowest possible energy solution (i.e., a job does not meet its deadline using the processor frequency $f_{\min}$):

Then energy-efficient EDF-VD computes two separate processor frequencies $f_{\text{LO}}^{\text{LO}}$ and $f_{\text{HI}}^{\text{LO}}$ for LO-criticality and HI-criticality jobs, respectively. We assume that energy-efficient EDF-VD successfully schedules all the jobs using processor frequencies $f_{\text{LO}}^{\text{LO}}$ and $f_{\text{HI}}^{\text{LO}}$. Let the processor frequencies for each job assigned by energy-efficient EDF-VD be less than those assigned by our algorithm. We know that our algorithm assigns two different processor frequencies to the job of task set $\mathcal{T}$ if and only if there is a finishing time miss by a job j_{ik} with processor frequency $\frac{f_b}{r_{\text{LO}}}$. Then the processor frequencies of those jobs whose finishing times before d_{ik}^{Δ} are increased using the function CheckEmptySpace() and the rest of the jobs whose finishing times lie after d_{ik}^{Δ} are decreased. In Lemma 5, we have already proved that increasing or decreasing the processor frequency of any job from those that are assigned by our algorithm will lead to deadline misses or increase in the total energy consumption. If $f_{\text{LO}}^{\text{LO}}$ and $f_{\text{HI}}^{\text{LO}}$ are less than the frequencies assigned by our algorithm, then the jobs will miss their deadline. This is a contradiction. On the other hand, it is easy to see that if $f_{\text{LO}}^{\text{LO}}$ and $f_{\text{HI}}^{\text{LO}}$ are greater than the frequencies assigned by our algorithm, then there will be empty spaces in the schedule. This is a contradiction.

Case 3 One of the processor frequencies $f_{\text{LO}}^{\text{LO}}$ and $f_{\text{HI}}^{\text{LO}}$ computed by energy-efficient EDF-VD is less or the other one is greater than the processor frequency assigned by our algorithm to some job. Here we assume that our algorithm faces only one finishing time miss which results in two different processor frequency assignment for each job. In this case, we get either a finishing time miss when the processor frequency assigned by EDF-VD is less than that assigned by our algorithm or an empty space when the processor frequency assigned by EDF-VD is greater than that assigned by our algorithm. Suppose job j_{ik} misses its finishing time d_{ik}^{Δ}. Then the function SetFrequency() reduced the running time (by increasing the processor frequency) of all the jobs whose deadline is less than d_{ik}^{Δ} and increasing the running time (by decreasing the processor frequency) of all the jobs whose deadline is greater than d_{ik}^{Δ}. We know that both the algorithms use EDF algorithm to verify schedulability of the jobs. Clearly, EDF-VD has to schedule all those jobs before d_{ik}^{Δ} whose finishing time is less than or equal to d_{ik}^{Δ} as they will miss their deadlines if scheduled later. This is because the finishing time computed by our algorithm for each job is as close to its deadline as possible. From Lemma 3, we know that the same processor frequency assigned to each job results in minimum energy consumption. Our algorithm assigns two different processor frequency to each job scheduled before and after d_{ik}^{Δ}, respectively. Suppose EDF-VD meets the finishing time d_{ik}^{Δ} and jobs present before and after d_{ik}^{Δ} contains both LO- and HI-criticality. Using Lemma 3, we know that the same processor frequency assigned to each job results in minimum energy consumption. Hence the schedule generated by the energy-efficient TT-Merge algorithm results in minimum energy consumption. If jobs present before and after d_{ik}^{Δ} contains LO- and HI-criticality, respectively, then the schedule computed by both the algorithm consumes same energy. On the other hand, suppose EDF-VD schedules the jobs present on the left hand side of d_{ik}^{Δ} quickly and the jobs present on the right hand side of d_{ik}^{Δ} slowly as compared to our algorithm. This means EDF-VD increases the value of δ. From Lemma 5, we know that increasing δ leads to increase in energy consumption. Using Lemmas 3 and 4, we can easily conclude that the energy-efficient schedule constructed by EDF-VD gets either a deadline miss or has empty spaces in the schedule. This is a contradiction. This case can be easily extended for more than two processor frequencies assigned by our algorithm to a job.

So the processor frequencies assigned by the energy-efficient EDF-VD algorithm are equal or greater than those assigned by our algorithm. Hence the energy consumption for the energy-efficient EDF-VD algorithm is equal or greater than the energy consumed for our algorithm. □

5.2 Extension for discrete frequency levels

In previous sections, we discussed the minimization of energy consumption in a mixed-criticality system using continuous processor frequency levels. In practice, continuous processor frequency levels will not be available. We need to extend our algorithm to find a schedule which consumes less energy as compared to the energy-efficient EDF-VD algorithm while considering only discrete processor frequency levels.

Suppose the set of processor frequencies available in the model is $F = \{f_1, f_2, \ldots, f_{|F|}\}$, where $f_1 = f_{\min}$ and $f_{|F|} = f_{\max}$. We need to find processor frequencies for each job such that the energy consumption can be minimized in the LO-criticality scenario using DVFS method without affecting the schedulability of the system in both the scenarios.

We can modify our algorithm discussed in Sect. 5 to assign processor frequencies from F. For example, if a job j_{ik} is assigned a processor frequency $f_m < f_{ik}^{\mathrm{LO}} < f_n$, then processor frequency f_n can be assigned to job j_{ik}, where f_m and f_n are two consecutive processor frequency in F. Note that the resulting schedule may not be optimal with respect to energy consumption.

6 Extension of the proposed algorithm for dependent task sets

In previous sections, we have considered task sets with independent tasks. To the best of our knowledge, minimizing the energy consumption of mixed-criticality systems has not been explored for dependent tasks. Here we consider the case of dual-criticality instances with dependent tasks. There exists a time-triggered algorithm [13] for mixed-criticality systems with dependent tasks which finds the two scheduling tables, i.e., $\mathcal{S}_{\mathrm{LO}}$ and $\mathcal{S}_{\mathrm{HI}}$, hereafter, abbreviated as the TT-Merge-Dep algorithm. Here we focus on integrating DVFS with the TT-Merge-Dep algorithm to minimize the energy consumption in mixed-criticality systems with dependency constraints. As in the case of independent tasks, we propose an algorithm which finds a processor frequency f_{ik}^{LO} and energy-efficient LO-criticality WCET $\tilde{C}_{ik}(\mathrm{LO})$ for each job j_{ik} of a task set which minimizes the energy consumption. These $\tilde{C}_{ik}(\mathrm{LO})$ units of execution time are used by the TT-Merge-Dep algorithm to find the scheduling tables which gives minimum energy consumption.

6.1 Model

The task model is similar to the one discussed in Sect. 2. The dependency between the tasks is defined by a *directed acyclic graph* (DAG). A task set is represented in the form of $\mathcal{T}(V, E)$, where V represents the set of tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ and E represents the dependency between the tasks. If there is an edge $\tau_i \rightarrow \tau_j$, then the execution of τ_i must precede the execution of τ_j, denoted by $\tau_i \prec \tau_j$. We assume that no HI-criticality job can depend on a LO-criticality job. This means, there will be no task set where an outward edge from a LO-criticality task becomes an inward edge to a HI-criticality task.

Definition 4 A dual-criticality MC task set with task dependencies is said to be **time-triggered schedulable** if it is possible to construct the two scheduling tables $\mathcal{S}_{\mathrm{LO}}$ and $\mathcal{S}_{\mathrm{HI}}$ for task set $\mathcal{T}$ without violating the dependencies, such that the run-time scheduler schedules $\mathcal{T}$ in a correct manner.

6.2 Problem formulation

Here we formally present our energy optimization problem for mixed-criticality systems with dependent tasks. Our goal is to minimize the LO-criticality scenario energy consumption by expanding the task executions in the schedule while ensuring that they do not miss their deadlines without violating the dependency constraints. Without loss of generality, we calculate the energy consumption minimization up to the hyperperiod P of the task set. As in the earlier case, we find a LO-criticality WCET $\tilde{C}_{ik}(\text{LO})$ for each job of a task set. This is accomplished by expanding the execution time of a task as much as possible so that the minimum expansion of each job is maximized and the processor works on a slower frequency to minimize the energy consumption and each task also meets its deadline without violating the dependency constraints. Then using these LO-criticality WCETs $\tilde{C}_{ik}(\text{LO})$, the TT-Merge-Dep algorithm finds a minimized energy consumption schedule. When the system switches to HI-criticality scenario, the processor frequency is set to f_b. Here the problem is similar to Problem 1 in Sect. 4 with dependency constraints. So our energy objective to be minimized in the LO-criticality scenario by varying the processor frequency is:

$$\frac{1}{P} \cdot \sum_{\tau_i \in \mathcal{T} \wedge 1 \le k \le N_i} C_i(\text{LO}) f_b \cdot \frac{1}{f_{ik}^{\text{LO}}} \cdot \beta \cdot (f_{ik}^{\text{LO}})^{\alpha} \tag{27}$$

The energy minimization is constrained by:

– Bound for the frequency for each job:

$$f_i \in [f_{min}, f_{max}]. \tag{28}$$

where f_i is the frequency of job j_{ik}.

– Construction of table $\tilde{\mathcal{T}}_{\text{LO}}$ and $\tilde{\mathcal{T}}_{\text{HI}}$:

> It should be possible to construct $\tilde{\mathcal{T}}_{\text{LO}}$ and $\tilde{\mathcal{T}}_{\text{HI}}$ using $\tilde{C}_i(\text{LO})$ and $\tilde{C}_i(\text{HI})$ units of execution time of job j_{ik}, respectivley without missing any deadline [13] (29)

where $\tilde{C}_i(\text{LO})$ and $\tilde{C}_i(\text{HI})$ are the time taken to execute $C_i(\text{LO})$ and $C_i(\text{HI})$ units of execution time, respectively.

– Construction of table $\tilde{\mathcal{S}}_{\text{LO}}$:

> It should be possible to construct $\tilde{\mathcal{S}}_{\text{LO}}$ while ensuring the failure situation (4) in [13] does not occur, at any time t (i.e., $\tilde{\mathcal{T}}_{\text{LO}}[t]$ is non-empty and $\tilde{\mathcal{T}}_{\text{HI}}[t]$ is non-empty, at time t. (30)

where $\tilde{\mathcal{T}}_{\text{LO}}[t]$ contains a LO-criticality job and $\tilde{\mathcal{T}}_{\text{HI}}[t]$ contains a HI-criticality job at time t, respectively.

– Dependency constraints:

$$\text{For all i, k, if } \forall j_{ik} \prec j_{il}, \quad \text{then} \quad e_{ik} < s_{il} \tag{31}$$

where e_{ik} and s_{il} are the completion time and starting time of task j_{ik} and j_{il}, respectively in the scheduling table.

Finally, our energy minimization problem is

$$\begin{aligned} &\textbf{minimize} \quad (27) \\ &\textbf{s.t.} \quad (28), (29), (30), (31) \end{aligned} \tag{32}$$

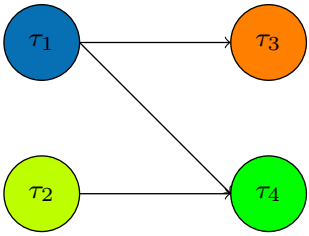


Fig. 6 Dependency between the tasks given in Table 1

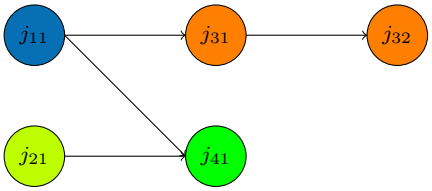


Fig. 7 Dependency between the tasks given in Table 1 after unroll

6.3 The algorithm

In this section, we present an algorithm to find the processor frequency f_{ik}^{LO} for each job which can be used by the TT-Merge-Dep algorithm to schedule a dependent task set successfully. First, we find the tables E_{LO} and E_{HI}. Then the table E_{FINAL} is constructed using these two tables. The length of all these tables are P, i.e., the hyperperiod of all the tasks in the task set. These steps are similar to the TT-Merge-Dep algorithm from [13]. But inputs for the TT-Merge-Dep algorithm are non-recurrent jobs. So we need to find all the jobs and the dependencies between them over a hyperperiod P which can be done by the method given in [6]. The steps to unroll a DAG are presented below.

- Let P denote the least common multiple of the periods of the blocks: $P := lcm\{p_1, p_2, \ldots, p_n\}$.
- For each task τ_i, there will be $\frac{P}{p_i}$ jobs labeled $j_{i1}, j_{i2}, \ldots, j_{i\frac{P}{p_i}}$.
- The job j_{ik} is assigned a released time $(k-1) \cdot p_i$ and a deadline $k \cdot p_i$.
- For each job j_{ik}.
  - For each task τ_m such that there is a dependency (τ_m, τ_i)
    1. define $l := \max\{x \in N | x \cdot p_m \leq k \cdot p_i\}$
    2. Add edge (j_{ml}, j_{ik})
    3. If the node $j_{m(l+1)}$ exists then add the edge $(j_{ik}, j_{m(l+1)})$.

Then we can use the same function SetFrequency() of Sect. 5. But we need to check the dependency among the jobs. The first step of function SetFrequency(E_{LO},E_{HI},E_{FINAL}) for the dependent jobs moves each segment of a job j_{ik} as close to its finishing time in E_χ as possible without disturbing the dependency constraints, where χ is the criticality of j_{ik}. Then all the steps are similar to the function SetFrequency(E_{LO},E_{HI},E_{FINAL}) of Sect. 5.

We explain the algorithm with the help of an example.

Example 3 Consider the task set given in Table 1. The dependencies between the tasks are shown in Fig. 6. Here the $f_{\min}$ and $f_{\max}$ are set to 0.5 and 1, respectively. We assume $\alpha = 3$.

First, we unroll the graph given in Fig. 6 to get all the jobs and their dependencies over a hyperperiod using the method given in [6]. The resulting graph is shown in Fig. 7.

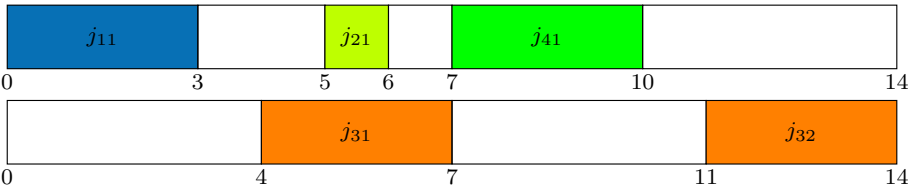


Fig. 8 Tables E_{LO} and E_{HI}

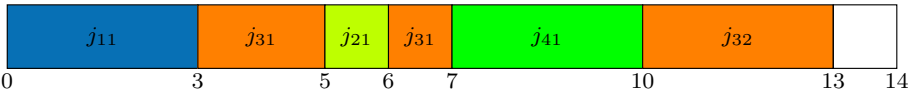


Fig. 9 Table E_{FINAL}

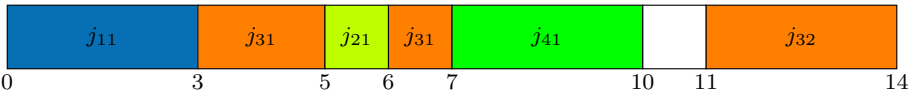


Fig. 10 Table E_{FINAL} after each job is moved to its finishing time in E_χ

Now we find the tables E_{LO} and E_{HI} as shown in Fig. 8 using the TT-Merge-Dep algorithm. Then we construct table E_{FINAL} by merging the tables E_{LO} and E_{HI} using the TT-Merge-Dep algorithm as shown in Fig. 9.

- Then starting from the right, each job j_{ik} is moved as close to its finishing time in E_χ as possible without disturbing the dependency constraints, where χ is the criticality of job j_{ik}. So job j_{32} is moved to its right from [10, 13] to [11, 14] as its finishing time in table E_{LO} is 14. The final table E_{FINAL} is given in Fig. 10.
- Here $EMPTY_1^S$ and P are 10 and 14, respectively. The total execution time between $EMPTY_1^S$ and P in table E_{FINAL} is 3.
- $r_{\text{LO}} = \frac{P - EMPTY_1^S}{\sum_{\forall j_{ik} \in J} C_{ik}(\text{LO})} = \frac{14-10}{3} = \frac{4}{3}$ where $f_{\min} < \frac{1}{r_{\text{LO}}}$. This means the lowest possible processor frequency for each job in J is 0.75.
- We find the $\tilde{C}_{ik}(\text{LO})$ for each job $\langle j_{11}, j_{21}, j_{31}, j_{41}, j_{32} \rangle$ to be $\langle 3, 1, 3, 3, 4 \rangle$.
- Now we use formula 13 to check the schedulability of each job with the $\tilde{C}_{ik}(\text{LO})$ units of execution time. All the jobs pass this test.
- Then we find the processor frequencies $f_{ik}(\text{LO})$ for each job $\langle j_{11}, j_{21}, j_{31}, j_{41}, j_{32} \rangle$ to be $\langle 1, 1, 1, 1, 0.75 \rangle$.
- The total normalized energy consumption by the TT-Merge energy-efficient algorithm using the above processor frequencies is 0.835.

7 Experimental results and evaluation

In this section we present the experiments conducted to evaluate our algorithm. The experiments show the impact of the utilization of task sets on the energy-efficient TT-Merge algorithm versus the energy-efficient EDF-VD algorithm. The comparison is done over numerous task sets with randomly generated parameters. The task generation policy may have a significant effect on the experiments. We follow a similar task generation policy as in [13]. The details of the task generation policy are as follows.

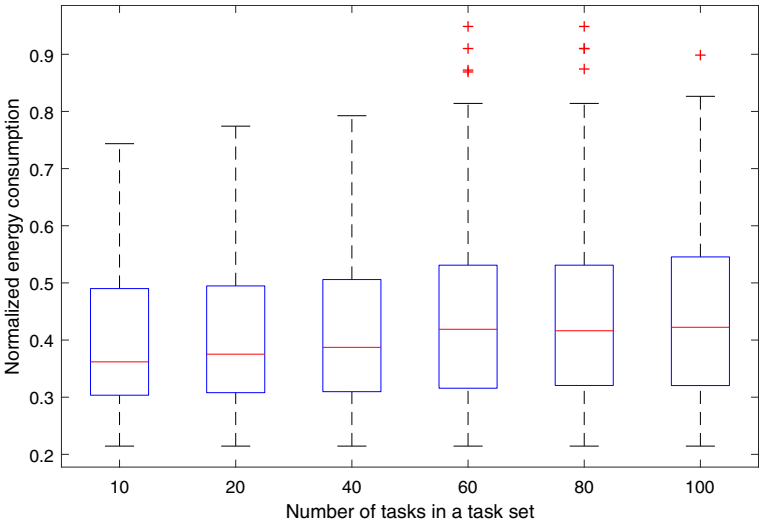
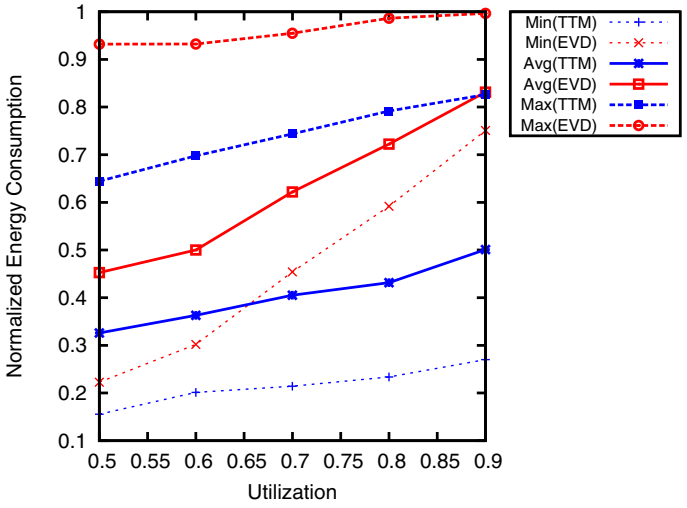


Fig. 11 Performance of our algorithm where LO-criticality utilization is 0.7 and more than 50% of the tasks in a task set are of HI-criticality

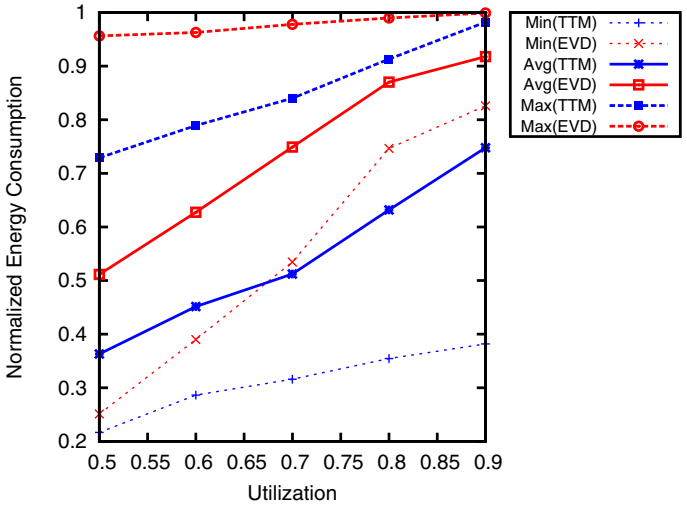
- The utilization (u_i) of tasks in a task set are generated according to the UUniFast algorithm [14].
- The periods (p_i) of the tasks are generated by using exponential distribution proposed by Davis et al [17].
- The C_i(LO) units of execution time of the tasks are calculated by $u_i \times p_i$.
- The C_i(HI) units of execution time of the tasks are calculated as $C_i(\text{HI}) = \text{CF} \times C_i(\text{HI})$, where CF is the criticality factor which varies between 2 and 6 for each HI-criticality tasks
- Each task set contains at least one HI-criticality job and one LO-criticality job.

In these experiments, we consider only those results where both the energy-efficient TT-Merge algorithm and energy-efficient EDF-VD algorithm successfully find a schedule. For each point on the X-axis, we plot the average results of 200 runs. The energy consumption plotted in the graphs are normalized energy consumption. We vary the utilization at the LO-criticality scenario of the task sets between 0.5 and 0.9, and we set periods of the tasks between 1 and 2000. The processor frequency f_{min} is set to 0.2, β is set to 1 and α to 2.5. The names TTM and EVD used in the graphs are shorthand for the energy-efficient TT-Merge and EDF-VD algorithms, respectively.

First, we show the performance of our algorithm with a different number of tasks in a task set using the Box-whisker plots in Fig. 11, where the LO-criticality utilization is 0.7, and more than 50% of the tasks in a task set is HI-criticality. In Fig. 11, we can see the average energy consumption for all the task sets is very similar. This is because the LO-criticality utilization of each task set is fixed to 0.7. We can verify that as the number of tasks increases in a task set, the energy consumptions increase slightly, whereas in a few cases, it is a bit higher. The main reason behind the fact is the increase in the number of HI-criticality tasks in the task set. The HI-criticality tasks reserve slots to execute their C_i(HI) units of execution time, which leads to a lack of slack time in the time-line in a LO-criticality scenario.



(a) At least one LO-criticality and one HI-criticality task

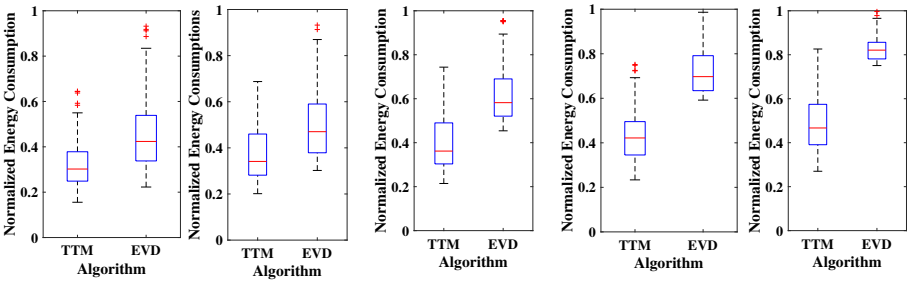


(b) More than 50% of HI-criticality tasks

Fig. 12 Comparison of normalized energy consumption between energy-efficient EDF-VD and TT-Merge

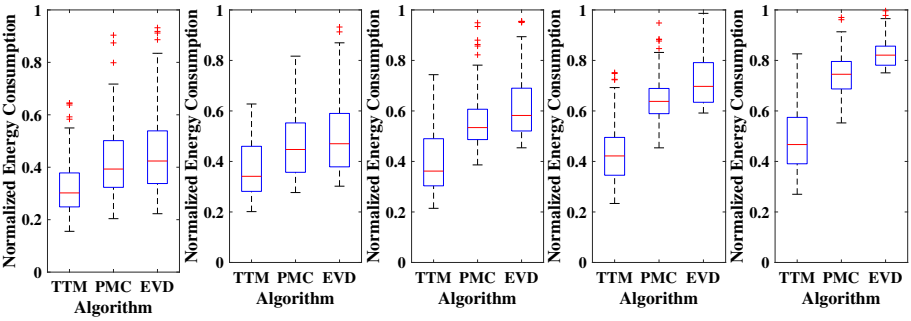
In the second experiment, we plot the minimum, maximum and average energy consumption of both the algorithms for each utilization. The number of tasks in the task set is set to 10. The graph in Fig. 12a shows that the energy consumption increases for both the algorithms with the increase in the utilization. It is evident from the graph that the schedule constructed according to the energy-efficient TT-Merge algorithm consumes less energy than the one constructed by the energy-efficient EDF-VD algorithm for all utilizations. For detailed analysis, we use Box-whisker plots.

From the graphs in Fig. 13, we can see that the maximum energy consumption of task sets by our algorithm lies between 0.2 and 0.6 for all utilizations, where it is between 0.3 and 0.8 for energy-efficient EDF-VD. For example, when the LO-criticality scenario utilization



(a) U = 0.5 **(b)** U = 0.6 **(c)** U = 0.7 **(d)** U = 0.8 **(e)** U = 0.9

Fig. 13 Comparison between TTM and EVD where LO-criticality scenario utilization ranges from 0.5 to 0.9



(a) U = 0.5 **(b)** U = 0.6 **(c)** U = 0.7 **(d)** U = 0.8 **(e)** U = 0.9

Fig. 14 Comparison between TTM, PCM and EVD where LO-criticality scenario utilization ranges from 0.5 to 0.9

is 0.7, the energy consumption by the schedule constructed by our algorithm is between 0.3 and 0.5 for maximum number of task sets and the same for the energy-efficient EDF-VD is between 0.5 and 0.7, respectively. Clearly, our algorithm outperforms the energy-efficient EDF-VD algorithm.

The next experiment finds the impact of the number of HI-criticality tasks in a task set. Here each task set contains more than 50% of HI-criticality tasks. The other parameters are the same as the previous experiment. From the graph in Fig. 12b, we can observe that the LO-criticality scenario energy consumption increases with the number of HI-criticality tasks increases. This is because of the lack of slack times due to the time reservation for HI-criticality jobs.

Now we compare our algorithm with energy-efficient EDF-VD and the work in [1], which we abbreviated as PMC. Here we assume that the available processor frequencies are {0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}. Now we plot the Box-whisker plots to show the minimum, maximum and average energy consumption for all the three algorithms for each utilization. From Fig. 14, it is clear that the schedule constructed according to our algorithm consumes less energy than both the existing algorithms. We also can see that the schedule constructed according to our algorithm consumes 20% and 25% less energy than the one constructed by the energy-efficient EDF-VD and PMC algorithms, respectively at higher LO-criticality utilization.

8 Conclusion

In this article, we propose a new algorithm to find energy efficient time-triggered schedules using the TT-Merge algorithm. The proposed algorithm finds a processor frequency and the corresponding LO-criticality WCET for all the jobs of a task set which are then used by the TT-Merge algorithm. We prove that the schedule constructed according to the energy-efficient TT-Merge algorithm consumes less energy than the one constructed by the energy-efficient EDF-VD algorithm. The experiments also verify the fact. We also discuss the impact of discrete processor frequency levels on our algorithm. Then we extend our algorithm for dependent task sets. As part of future work, we plan to extend this algorithm to the multiprocessor case. The algorithm proposed in this paper can be used in a multiprocessor system using the partitioned scheduling technique. We also plan to extend our technique for a global multiprocessor scheduling algorithm.

Acknowledgements We thank Sanjoy Baruah and Arnab Sarkar for helpful comments and feedback on earlier versions of this draft.

References

1. Ali I, Seo J, Kim KH (2015) A dynamic power-aware scheduling of mixed-criticality real-time systems. In: 14th IEEE international conference on ubiquitous computing and communications, IUCC 2015, pp 438–445
2. Asyaban S, Kargahi M, Thiele L, Mohaqeqi M (2016) Analysis and scheduling of a battery-less mixed-criticality system with energy uncertainty. ACM Trans Embed Comput Syst 16(1):23:1–23:26
3. Audsley NC (2001) On priority assignment in fixed priority scheduling. Inf Process Lett 79(1):39–44
4. Awan MA, Masson D, Tovar E (2015) Energy-aware task allocation onto unrelated heterogeneous multicore platform for mixed criticality systems. In: IEEE real-time systems symposium, RTSS 2015, San Antonio, Texas, USA, 1–4 December 2015, p 377
5. Awan MA, Masson D, Tovar E (2016) Energy efficient mapping of mixed criticality applications on unrelated heterogeneous multicore platforms. In: 11th IEEE symposium on industrial embedded systems, SIES 2016, Krakow, Poland, 23–25 May 2016, pp 63–72
6. Baruah S (2014) Implementing mixed-criticality synchronous reactive programs upon uniprocessor platforms. Real Time Syst 50(3):317–341
7. Baruah S (2018) Mixed-criticality scheduling theory: scope, promise, and limitations. IEEE Des Test 35(2):31–37
8. Baruah S, Bonifaci V, D'Angelo G, Li H, Marchetti-Spaccamela A, Megow N, Stougie L (2012) Scheduling real-time mixed-criticality jobs. IEEE Trans Comput 61(8):1140–1152
9. Baruah S, Bonifaci V, DAngelo G, Li H, Marchetti-Spaccamela A, Ster SVD, Stougie L (2012) The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In: 2012 24th Euromicro conference on real-time systems (ECRTS), pp 145–154. IEEE
10. Baruah SK, Bonifaci V, D'Angelo G, Marchetti-Spaccamela A, van der Ster S, Stougie L (2011) Mixed-criticality scheduling of sporadic task systems. In: Demetrescu C, Halldórsson MM (eds) Algorithms – ESA 2011. ESA 2011. Lecture Notes in Computer Science, vol 6942. Springer, pp 555–566
11. Baruah S, Fohler G (2011) Certification-cognizant time-triggered scheduling of mixed-criticality systems. In: 32nd IEEE real-time systems symposium (RTSS). IEEE, pp 3–12
12. Baruah S, Vestal S (2008) Schedulability analysis of sporadic tasks with multiple criticality specifications. In: Euromicro conference on real-time systems, 2008. ECRTS'08. IEEE, pp 147–155
13. Behera L, Bhaduri P (2017) Time-triggered scheduling of mixed-criticality systems. ACM Trans Des Autom Electron Syst (TODAES) 22(4):74
14. Bini E, Buttazzo G (2005) Measuring the performance of schedulability tests. Real Time Syst 30(1–2):129–154
15. Burns A, Davis R (2013) Mixed criticality systems: a review. Technical report, Department of Computer Science, University of York

16. Chen JJ, Kuo CF (2007) Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In: 13th IEEE international conference on embedded and real-time computing systems and applications, 2007. RTCSA 2007. IEEE, pp 28–38
17. Davis RI, Zabos A, Burns A (2008) Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans Comput 57(9):1261–1276
18. Ernst R, Di Natale M (2016) Mixed criticality systems: a history of misconceptions? IEEE Des Test 33(5):65–74
19. Esper A, Nelissen G, Nélis V, Tovar E (2015) How realistic is the mixed-criticality real-time system model? In: Proceedings of the 23rd international conference on real time and networks systems, RTNS '15. ACM, New York, pp 139–148
20. Henzinger TA, Sifakis J (2006) The embedded systems design challenge. In: FM 2006: formal methods. Springer, pp 1–15
21. Huang P, Kumar P, Giannopoulou G, Thiele L (2014) Energy efficient DVFs scheduling for mixed-criticality systems. In: 2014 International conference on embedded software (EMSOFT). IEEE, pp 1–10
22. Legout V, Jan M, Pautet L (2013) Mixed-criticality multiprocessor real-time systems: energy consumption vs deadline misses. In: First workshop on real-time mixed criticality systems (ReTiMiCS), pp 1–6, Taipei, Taiwan
23. Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61
24. Narayana S, Huang P, Giannopoulou G, Thiele L, Prasad RV (2016) Exploring energy saving for mixed-criticality systems on multi-cores. In: 2016 IEEE real-time and embedded technology and applications symposium (RTAS). IEEE, pp 1–12
25. Pagani S, Chen JJ (2014) Energy efficiency analysis for the single frequency approximation (SFA) scheme. ACM Trans Embed Comput Syst 13(5s):158
26. Valavanis KP (2008) Advances in unmanned aerial vehicles: state of the art and the road to autonomy, vol 33. Springer, Berlin
27. Vestal S (2007) Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: 28th IEEE international real-time systems symposium, 2007. RTSS 2007, pp 239–243
28. von der Brüggen G, Chen K, Huang W, Chen J (2016) Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In: 2016 IEEE real-time systems symposium (RTSS), pp 303–314
29. Zhu D, Melhem R, Mossé D (2004) The effects of energy management on reliability in real-time embedded systems. In: IEEE/ACM international conference on computer aided design, 2004. ICCAD-2004. IEEE, pp 35–40