Improving the Schedulability and Energy Efficiency for Weakly Hard Real-Time Embedded Systems

Abstract—For real-time embedded systems, schedulability and energy efficiency are two highly co-related important design issues. In this paper, we explore how to improve the schedulability and energy efficiency for real-time systems with weakly hard temporal constraints. The weakly hard temporal constraints are modeled by the \((m,k)\)-constraints, which require that at least \(m\) out of any \(k\) consecutive jobs of a task meet their deadlines. Two off-line approaches were proposed in managing the appropriate mandatory/optional job partition for given task sets with \((m,k)\)-constraints. Through extensive experiments, the results demonstrate that our proposed techniques significantly outperformed the previous research in both schedulability and energy efficiency for weakly hard real-time embedded systems.

I. INTRODUCTION

In traditional hard real-time embedded systems, all task instances are required to meet their deadlines and any deadline miss will crash the entire application or system. However, few real-time applications are truly hard real-time. For example, many real-time applications, such as multimedia processing and real-time communication systems, can often tolerate occasional deadline misses. Some other applications may have soft deadlines where tasks that do not finish by their deadlines can still be completed with a reduced value [1] or they can simply be dropped provided that user’s perceived quality of service (QoS) are satisfied. To quantify the QoS requirements, some statistic information such as the average deadline miss rate is commonly used. Although the statistical deadline miss rate can ensure the quality of service in a probabilistic manner, this metric can be problematic for some real-time applications. For example, for certain real-time systems, when the deadline misses happened to some tasks, the information carried by those tasks can be estimated in a reasonable accuracy using techniques such as interpolation. However, even a very low overall miss rate tolerance cannot prevent a large number of deadline misses from occurring in such a short period of time that the data cannot be successfully reconstructed. To avoid possible severe consequences, one can always treat the system as a hard real-time system. The problem, however, is that when the system is overloaded \(^1\), the approaches for hard real-time systems are no longer valid as deadline missing will become inevitable in such kind of scenarios. As a result, critical information that cannot be reconstructed may be lost, thus the service quality can be severely degraded from the user’s perspective. The weakly hard real-time systems are more suitable to model such kind of applications. In the weakly hard real-time model, the systems should not only support the overall guarantee of the QoS statistically, but also be able to provide the lower bounded, predictable level of QoS at each specified time interval. The \((m,k)\)-model, proposed by Hamdaoui et al. [2], can serve well for this purpose. According to this model, a repetitive task of the system is associated with an \((m,k)(0 < m \leq k)\) constraint requiring that \(m\) out of any \(k\) consecutive job instances of the task meet their deadlines. A dynamic failure occurs, which implies that the temporal QoS constraint is violated and the scheduler is thus considered failed, if within any \(k\) consecutive jobs less than \(m\) job instances meet their deadlines. Based on this \((m,k)\)-model, Ramanathan et al. [3] proposed to partition the jobs into mandatory and optional jobs. So long as all the mandatory jobs can meet their deadlines, the \((m,k)\)-constraints can be ensured. West et al. [4] introduced another model, called the window-constrained model, which requires that at least \(m\) jobs over any non-overlapped and consecutive windows containing \(k\) jobs meet their deadlines. It can be readily concluded that window constraints are weaker than the \((m,k)\)-constraints, as if a schedule is feasible under the \((m,k)\)-constraints, it is also feasible under the window constraints.

For real-time embedded systems, energy minimization has also come to be recognized as one of the primary design goals. And dynamic voltage/frequency scaling (DVFS), e.g. [5], which dynamically varies the supply voltage and frequency of the processing unit, has proven to be an effective way to reduce energy consumption. Moreover, since energy consumption is a convex function of the processor speed, which is proportional to the supply voltage [6], the energy efficiency of embedded systems is closely related to the schedulability of the system. In other words, if the schedulability of the task sets can be improved, the running speeds of the processor to execute the jobs can be lowered correspondingly and thus the energy consumption can be reduced.

In this paper, we study the feasibility issues for weakly hard real-time systems. Specifically, we will explore how to improve the schedulability and energy efficiency for such kind of systems through more adaptive mandatory/optional job partition while ensuring the \((m,k)\)-guarantee. Since compared to the fixed-priority scheme, the earliest deadline first (EDF) scheme allows the task set to be executed under higher CPU utilizations and thus has better voltage scaling potentials, we assume that the real-time tasks are scheduled according to the EDF scheduling policy.

Two approaches are proposed in this paper in determining the appropriate mandatory/optional job partition for a given task set. In the first approach, the mandatory job sets of each task for the given systems are evenly distributed, which is based on enhancing the previous work proposed in [3]. Due to the inherent properties of some real world applications (for

\(^1\)this is a common case when additional constraints such as energy constraints are imposed on the systems, as conserving energy often needs to reduce the processor voltage/frequencies which will increase the execution times of the tasks and thus easily cause the systems to be “overloaded"
example, the task periods of some practical systems are often close to harmonic), strictly evenly distributed pattern might not always be desired for the purpose of improving schedulability and energy efficiency. To incorporate these scenarios, we also proposed the second approach in which the mandatory jobs are determined more adaptively based on the simulated annealing (SA) algorithm. Through our extensive experiments, the results show that our proposed approaches can significantly improve the schedulability and energy efficiency for weakly hard real-time systems simultaneously.

The rest of the paper is organized as follows. Section II introduces the related work. Section III presents the system models and motivations. Section IV introduces our first approach in enhancing the previous mandatory/optimal job partition based on evenly distributed pattern. Section V introduces our second approach in determining the more general mandatory/optimal job partition based on simulated annealing algorithm. The effectiveness of our approach are demonstrated using experimental results in section VI. In section VII, we offer the conclusion and future work.

II. RELATED WORK

Due to its intuitiveness and capability of capturing not only statistical but also deterministic QoS requirements, the \((m,k)\)-model has been widely studied, e.g., [7], [2], [3]. In [7], Quan et al. formally proved that the problem of scheduling with \((m,k)\)—guarantee for arbitrary value of \(m\) and \(k\) is NP-hard in the strong sense. To guarantee the \((m,k)\)-constraints, Ramanathan et al. [3] proposed a strategy to partition the jobs into mandatory and optional jobs. The mandatory jobs are the jobs that must meet their deadlines in order to satisfy the \((m,k)\)-constraints, while the optional jobs can be executed to further improve the quality of the service or simply be dropped to save the computing resources.

In [7], Quan et al. improved the above partitioning strategy by reducing the maximal interference from high priority mandatory jobs to lower priority mandatory jobs. The limitation of this approach mainly lies in that each time the maximal “interference” can only be computed among mandatory jobs between two tasks. Consequently, the improvement achievable on the schedulability of the whole task set may be limited. On the other hand, since the “interference” is computed for task sets based on fixed-priority assignment, it is not applicable when the system is scheduled under EDF scheme. Also in [7], the same authors introduced a genetic algorithm (GA) to find the more general partitioning strategy for task sets scheduled under fixed-priority scheme. Their approach is based on an objective function which assumes the worst case interference from higher priority tasks will happen to the lower priority task(s) simultaneously, which can be too conservative in practice. And for the same reason the objective function cannot be applied to task sets scheduled under EDF scheme, either.

In [8], Bernat et al. proposed to use a Bi-Modal scheduler to schedule the systems with \((m,k)\)—constraints. The tasks are first scheduled according to the generic scheduling policy in the normal mode, and switched to panic mode if the dynamic failure is likely to occur. This work also targets systems with fixed-priority assignment. And none of the works above has taken energy/power consumption into consideration.

Some other researches [9], [10], [11] have also been conducted in conserving the energy consumption for real-time systems with QoS constraints. However, the application of all these approaches are limited to the systems that are already schedulable under certain known mandatory/optimal job partitioning strategies such as the deeply-red pattern [12] or the evenly distributed pattern [3]. Otherwise the \((m,k)\)—constraints cannot be guaranteed in the worst case. As will be shown in Section III-B and Section VI-A in this paper, deeply-red pattern [12] and evenly distributed pattern [3] can only account for a limited categories of feasible partitioning strategies and there still exists quite a number of mandatory/optimal job partitions which are schedulable but cannot be identified by these known partitioning strategies. Few works have been reported in reducing the energy by searching for better pattern and enhancing the mandatory/optimal job partition of the whole system directly. To the best of our knowledge, this is the first work to combine these two important issues and improve the schedulability and energy efficiency for weakly hard real-time systems simultaneously.

III. PRELIMINARIES

In this section, we first introduce the system models and problem definitions. Then we present the motivations to our research.

A. System models

The real-time system considered in this paper contains \(n\) independent periodic tasks, \(T = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}\), scheduled according to the earliest deadline first (EDF) policy [13]. Each task contains an infinite sequence of periodically arriving instances called jobs. Task \(\tau_i\) is characterized using five parameters, i.e., \((P_i, D_i, C, m_i, k_i)\). \(P_i\), \(D_i\), \(D_i \leq P_i\), and \(C_i\) represent the period, the deadline and the worst case execution time for \(\tau_i\), respectively. A pair of integers, i.e., \((m_i, k_i)\) \((0 < m_i \leq k_i)\), represent the QoS requirement for \(\tau_i\), requiring that, among any consecutive \(k_i\) jobs of \(\tau_i\), at least \(m_i\) jobs must meet their deadlines. The \(P\)-th job of task \(\tau_i\) is represented with \(J_{d_i}\) and its arrival time and absolute deadline are represented as \(r_{d_i}\) and \(d_{d_i}\).

With the above system models, the problems to be explored in this paper can be stated as follows:

\textbf{Problem 1}: Given system \(T = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}\), \(\tau_i = (P_i, D_i, C_i, m_i, k_i), i = 0, \cdots, (n-1)\), schedule \(T\) with EDF such that all \((m,k)\)-constraints are satisfied.

\textbf{Problem 2}: Given system \(T = \{\tau_0, \tau_1, \cdots, \tau_{n-1}\}\), \(\tau_i = (P_i, D_i, C_i, m_i, k_i), i = 0, \cdots, (n-1)\), schedule \(T\) with EDF on a variable voltage processor with limited supply voltage levels such that all \((m,k)\)-constraints are guaranteed and the energy consumption is minimized.

Obviously Problem 1 and Problem 2 are co-dependent and the solution for Problem 1 will be very useful in solving Problem 2.

In what follows, we will introduce about some concepts as well as the motivations related to these problems.
B. Motivations

To solve the above problems, a key problem is to judiciously partition the jobs into mandatory jobs and optional jobs [7] and schedule them most efficiently. For ease of explanation, we adopt the concept of \((m, k)\)–pattern as introduced in [7].

Definition 1: [7] The \((m, k)\)–pattern of task \(\tau_i\), denoted by \(\Pi_i\), is a binary string \(\Pi_i = \{\pi_0, \pi_1, ..., \pi_{(k-1)}\}\) which satisfies the following: (i) \(\pi_j\) is a mandatory job if \(\pi_j = 1\) and optional if \(\pi_j = 0\), and (ii) \(\sum_{j=0}^{k-1} \pi_{ij} = m_i\).

By repeating the regular \((m, k)\)–pattern \(\Pi_i\), we can get a mandatory job pattern for task \(\tau_i\). It is not difficult to see that the \((m, k)\)–constraint for \(\tau_i\) can be satisfied if the mandatory jobs of \(\tau_i\) are selected accordingly.

Two well-known regular \((m, k)\)–patterns proposed in literature are the deeply-red pattern (or R-pattern) [12] and the evenly distributed pattern (or E-pattern) [3]. According to R-pattern, the pattern \(\pi_{ij}\) is defined by

\[
\pi_{ij} = \begin{cases} 
1 & \text{if } 0 \leq j < m_i \\
0 & \text{otherwise}
\end{cases}
\]

And according to E-pattern, the pattern \(\pi_{ij}\) is defined by

\[
\pi_{ij} = \begin{cases} 
1 & \text{if } j = \left\lfloor \frac{m_i}{k_i} \right\rfloor \times \frac{k_i}{m_i} \\
0 & \text{otherwise}
\end{cases}
\]

The mandatory optional job partitions according to equation (2) has the interesting property that it helps to spread out the mandatory jobs evenly in each task along the time. In [9], the E-pattern is reversed horizontally to get an reversal version of it called the \(E^R\)–pattern.

It is not hard to see that the mandatory job set assigned according to E-pattern or \(E^R\)–pattern is easier to be schedulable than that by R-pattern due to its even distribution of mandatory jobs within each task [9]. However, even though the mandatory jobs for each task are evenly distributed, the overall mandatory workload is not necessarily evenly distributed. As a result the schedulability of the task set with \((m, k)\)–constraints may be seriously compromised. For example, consider a task set of three tasks \((\tau_1 = (4, 4, 3, 3, 6); \tau_2 = (6, 6, 4, 1, 4); \tau_3 = (8, 8, 2, 1, 3))\). As shown in Figure 1(a) and (b), neither the job partition based on E-pattern nor that based on \(E^R\)–pattern is schedulable, while other patterns such as the one in Figure 1(c) can be well schedulable.

Even though it is shown [9] that E-pattern (or \(E^R\)–pattern) has very good schedulability compared with other \((m, k)\)–patterns when all \((k_i \times P_i)\)'s are (or close to) co-prime with each other, in practical applications, the task periods often contain some large common divisors (e.g., the webphone [14], CNC (Computerized Numerical Control) machine controller [15], Avionics and INS (Inertial Navigation System) [16], etc.). E-pattern (or \(E^R\)–pattern) is not necessarily the best candidate in this scenario. Moreover, it is proven in [7] that to find the optimal \((m, k)\)–patterns for the general case is NP-hard in the strong sense. In this paper, we will explore how to determine the appropriate \((m, k)\)–patterns for a given task set. Our approaches are based on the following definitions and observations:

\[
\text{Definition 2: (Intensity)} \text{ For a real-time task set whose \(M_j(t_i, t_f)\) are the release time and deadline of some job instance, correspondingly.}
\]

\[
I(t_i, t_f) = \frac{\sum_j M_i(t_i, t_f) \times C_j}{t_f - t_i}
\]

\[
\text{Definition 3: (Most intensive interval)} \text{ An interval } [t_i, t_f] \text{ is called the most intensive interval (represented as } [t_i^M, t_f^M]) \text{ if it is the interval with the maximum intensity compared with any other intervals within the hyperperiod.}
\]

\[
\text{Definition 4: (Least intensive interval)} \text{ An interval } [t_i, t_f] \text{ is called the least intensive interval (represented as } [t_i^L, t_f^L]) \text{ if it is the interval with the minimum intensity compared with any other intervals within the hyperperiod.}
\]

\[
\text{Lemma 1: Given system } \mathcal{T} = \{\tau_0, \tau_1, ..., \tau_{n-1}\} \text{ with } \tau_i = (P_i, D_i, C_i, m_i, k_i) \text{ to be scheduled with EDF under given } (m, k) \text{–pattern, } \mathcal{T} \text{ is schedulable if the intensity of the most intensive interval, i.e., } I(t_i^M, t_f^M), \text{ is less than or equal to } 1.
\]

\[
\text{Proof: If we assume the maximal normalized processor speed to be } 1, \text{ the result follows the conclusions in [5].}
\]
Note that given a task set with arbitrary \((m,k)\)-pattern, to check its feasibility is NP-hard for the general case [7] and the usual way is to resort to simulation to check whether each mandatory job can be finished before its deadline. However, with Lemma 1, it is sufficient to just inspect whether the intensity \(I(t^M, t^M_f)\) of the most intensive interval is greater than 1 or not. Since \(I(t^M, t^M_f)\) can be computed in \(O(N \log N)\) time, it also provides us a more efficient way to check the schedulability for task sets under arbitrary \((m,k)\)-pattern.

As seen in the motivation example in Figure 1, by distributing the mandatory workload of the whole task set more evenly, the schedulability of the task set could be improved. Moreover, with energy consumption as a convex function of the job speed [6] we could expect to reduce the energy consumption of the whole system effectively if the mandatory workload of the whole task set is distributed more evenly. In order to do so, we should try to determine the \((m,k)\)-pattern appropriately such that the difference between the intensities of the most/least intensive intervals is reduced as much as possible. Obviously to achieve this goal, one effective way is to determine the \((m,k)\)-pattern in such a way that the maximum intensity \(I(t^M, t^M_f)\) of the system is minimized. In what follows, we will formulate this issue as a programming problem. Specifically, given a system \(T\), our goal is to find the appropriate regular \((m,k)\)-pattern for each task \(\tau_i\) such that the corresponding \(I(t^M, t^M_f)\) of the task set is minimized. Based on Lemma 1, the optimization problem can be stated as:

\[
\begin{align*}
\text{Minimize:} & \quad I(t^M, t^M_f) \\
\text{subject to:} & \quad \sum_{j=0}^{k-1} \pi_{ij} = m_i; \\
& \quad \pi_{ij} \in \{0, 1\}, \forall \tau_i \in T.
\end{align*}
\]

Unfortunately, the problem above is also NP-hard as it can be reduced in linear time, based on Lemma 1, to the feasibility problem of finding the optimal \((m,k)\)-pattern for a given task set. The latter one has already proven to be NP-hard in the strong sense [7]. However, it does give us some hints in searching for the appropriate \((m,k)\)-pattern based on some existing results. From Lemma 1, it is not hard to see that the feasibility of the task set depends on the maximum intensity of the system, i.e., \(I(t^M, t^M_f)\). Therefore if the value of \(I(t^M, t^M_f)\) can be reduced, the schedulability of the task set can be improved correspondingly.

For the existing \((m,k)\)-patterns, i.e., E-pattern, R-pattern, and \(E^R\)-pattern, we know that the schedulabilities of E-pattern and \(E^R\)-pattern are much better than R-pattern [9]. However, from the motivation example in Figure 1, we saw that the E-pattern or \(E^R\)-pattern is not always desirable. Moreover, from Figure 1(c) we can see that if the E-pattern could be adjusted properly, the schedulability of the task set can be improved effectively.

In next section, we first introduce how to enhance the E-pattern to improve the schedulability of the task sets.
Algorithm 1 Enhancing the evenly distributed pattern. (Algorithm Alg-EE)

1: Input: $\mathcal{T}$ with each task initialized with E-pattern;
2: Output: Rotated pattern for each task in $\mathcal{T}$;
3: $\mathcal{T}' = \{t_i\}$;
4: for $t_i \in \mathcal{T}, i = 2, 3, ..., N$ do
5: Compute $[t_i^M, t_i^P]$ and $[\tilde{t}_i^M, \tilde{t}_i^P]$ of the tasks in $\mathcal{T}$ using Yao’s algorithm [5];
6: $g = \gcd(LCM_{t_j \in \mathcal{T}}(k_jP_j), k_jP_j)$;
7: if $g = 1$ then
8: $\mathcal{T}' = \mathcal{T} + t_i$;
9: Continue; Il move on to the next iteration
10: else
11: $\min \text{dist}_1 = \infty$; $\min \text{dist}_2 = 0$;
12: for $s_i = 0$ to $(k_i - 1)$ do
13: $O_i = s_iP_i$;
14: $\min_1 = \min \{(O_i - \tilde{t}_i^P) \mod g, (\tilde{t}_i^M - O_i) \mod g\}$;
15: $\min_2 = \min \{(O_i - t_i^M) \mod g, (\tilde{t}_i^M - O_i) \mod g\}$;
16: if $\min_1 \leq \min_\text{dist}_1$ and $\min_2 \geq \min_\text{dist}_2$ then
17: $\gamma_i = s_i$;
18: $\min_\text{dist}_1 = \min_1$;
19: $\min_\text{dist}_2 = \min_2$;
20: end if
21: end for
22: Rotate pattern of task $t_i$ right by $\gamma_i$;
23: $\mathcal{T}' = \mathcal{T} + t_i$;
24: end if
25: end for

is assigned according to the E-pattern. And we put the tasks whose patterns have already been decided into $\mathcal{T}'$. Then during each iteration, we determine the patterns of the following tasks based on those of the tasks in $\mathcal{T}'$. Specifically, each time we first compute the most and least intensive intervals\(^2\), i.e., $[\tilde{t}_i^M, \tilde{t}_i^P]$ and $[t_i^M, t_i^P]$ of the previous $(i - 1)$ tasks in $\mathcal{T}$ using Yao’s algorithm [5]. Then we try to rotate the patterns of task $t_i$ logically such that the minimum distance between CMT and $\tilde{t}_i^P$ is minimized while the minimum distance between CMT and $t_i^P$ is maximized. In other words, we try to make CMT of task $t_i$ as close to $\tilde{t}_i^P$ as possible and at the same time, make CMT and $t_i^P$ as far apart as possible. With this heuristic, the maximal intensity $I(t_i^M, t_i^P)$ of the resulting task set could be expected to be reduced and the schedulability of the task set should be improved correspondingly.

Note that in Algorithm 1 the most critical point is how to determine the value of $\gamma_i$ (we call it rotating factor) for rotating the E-pattern of task $t_i$. Obviously, by rotating the original E-pattern of task $t_i$ right by a factor of $\gamma_i$, we essentially move the first CMT of task $t_i$ from 0 to $\gamma_iP_i$. Therefore after rotating the CMT, it is still a periodic event with period $k_iP_i$ but with initial time changed to $\gamma_iP_i$. It is not hard to see that the intervals $[t_i^M, t_i^P]$ and $[\tilde{t}_i^M, \tilde{t}_i^P]$ of the tasks in $\mathcal{T}$ are also periodic events with period $LCM_{t_j \in \mathcal{T}}(k_jP_j)$ and initial times $t_i^P$ and $\tilde{t}_i^P$, correspondingly. With Lemma 2, by adjusting the value of $\gamma_i$ properly, we can make CMT of task $t_i$ as close to $\tilde{t}_i^P$ of the previous $(i - 1)$ tasks in $\mathcal{T}$ as possible and at the same time, set CMT and $t_i^P$ as far as possible. Thus the maximal intensity $I(t_i^M, t_i^P)$ of the resulting task set could be

\(^2\)In case there are multiple most (or least) intensive intervals, we choose the longest one to be considered.
In the general SA algorithm, first an initial solution is randomly generated and used as the starting point. Then the cost/energy at this point is evaluated. The objective of the algorithm is to minimize the final cost/energy. To continue, an iterative loop is performed. During each iteration a neighbor solution is chosen randomly and its cost/energy is evaluated. If the neighbor solution is better than the initial solution, then it is accepted as the new initial solution. Otherwise, the neighbor solution is accepted only with a probability which is exponentially decreasing with the cost/energy difference and is slowly lowered with time. According to [18], the probability can be defined as \( \text{min}(1, e^{-\Delta \beta}) \), where \( \Delta \) is the difference of the cost/energy between the initial solution and the new solution and \( T \) is the control parameter corresponding to the temperature of the physical analogy. By repeating the above iterations (until some stopping criterion is met) on slow reduction of temperature, the algorithm eventually converges to the global optimum.

In SA algorithm, it is important to select the initial configuration as it will affect both the quality of the solution and also the computation time. In our approach, instead of choosing the initial solution randomly, we sought to make use of the \((m,k)\)-pattern that is output by Algorithm 1 as the starting point. Then during each loop, by iteratively improving the initial \((m,k)\)-pattern based on its cost/energy evaluation with the objective function in Equation (4), we can expect to eventually find the most appropriate \((m,k)\)-pattern that can minimize the objective function (4). The salient part of our SA approach is presented in Algorithm 2.

As shown in Algorithm 2, our approach firstly takes the task set with \((m,k)\)-pattern generated by Algorithm 1 as the input. Then based on it the most/least intensive intervals, \([\hat{t}_s^M, \hat{t}_f^M]\) and \([\hat{t}_s^L, \hat{t}_f^L]\) of the task set are computed with Yao’s algorithm. The neighbor solution is obtained by randomly picking out one task \(\tau_i\) and changing one of the mandatory pattern out of its \((m,k)\)-pattern to optional. Meanwhile, to satisfy the \((m,k)\)-constraint, we should also promote one optional pattern out of its \((m,k)\)-pattern to mandatory. After that, we update the repeated regular \((m,k)\)-pattern for \(\tau_i\) and recompute the most/least intensive intervals for the whole task set, \([\hat{t}_s,M, \hat{t}_f,M, \hat{t}_s,L, \hat{t}_f,L]\) of the new solution. The modification is accepted only if the maximal intensity \(I(\hat{t}_s,M, \hat{t}_f,M)\) of the new solution is less than that from the initial solution (line 9-10) or with a probability which decreases exponentially with their intensity difference and is lowered in each iteration (line 13-18). The algorithm stops when some criterion is met (line 21), for example, when the maximal intensity \(I(\hat{t}_s,M, \hat{t}_f,M)\) of the new solution has been frozen and stopped decreasing in the last few temperatures or the temperature has been reduced to zero.

The complexity of the SA approach in Algorithm 2 mainly comes from the loop in which Yao’s algorithm is used to compute the the most/least intensive intervals. With proper data structure, Yao’s algorithm can be implemented in \(O(N\log N)\) time [5], where \(N\) is the number of jobs within the hyperperiod. So the total time complexity of Algorithm 2 is \(O(kN\log N)\). Here \(k\) is the number of loops to be performed, which is determined by the initial temperature and the cooling rate. Note that, SA is inherently sequential and hence very slow for problems with large search spaces. In [19], [20], advanced exploration algorithms based on Markov Decision Process were proposed to minimize the number of necessary evaluations. Although similar approaches can be applied to our work to speed up the search process of SA, here we adopt to narrow the search space by exploiting the domain knowledge provided by the most/least intensive intervals, \([\hat{t}_s,M, \hat{t}_f,M, \hat{t}_s,L, \hat{t}_f,L]\), which is much easier to implement. Specifically, in selecting the task to switch its mandatory/optional patterns within its \((m,k)\)-pattern, we try to select such kind of task as has mandatory/optional instances as close to \([\hat{t}_s,M, \hat{t}_f,M, \hat{t}_s,L, \hat{t}_f,L]\) as possible (line 6). The priority is given to the task with both mandatory and optional instances completely falling within \([\hat{t}_s,M, \hat{t}_f,M]\) and \([\hat{t}_s,L, \hat{t}_f,L]\) simultaneously. By switching the mandatory/optional patterns of such kind of task within its \((m,k)\)-pattern, we could expect to make the mandatory workload within the hyper period distribute more evenly from the perspective of the whole task set.

In SA algorithm, the efficiency of the algorithm is also depending on the initial temperature and the cooling rate. In our algorithm, we determined the initial temperature based on the maximal intensity difference \(\Delta \beta\) between the most and least intensive intervals of the initial solution and the acceptance probability under the initial temperature, i.e., set \(\text{Init Temperature} = \frac{\Delta \beta}{\ln \alpha}\), where \(\alpha\) is the value of the cooling.
factor, which is a value between (0,1) and should be determined empirically to give good results. In particular, in our approach, the cooling factor is set to be 0.95 to generate results with good quality and within acceptable time cost.

One example As an example of showing the process of SA algorithm, consider the same task set in Figure 2(a). The output by Algorithm 1 is shown in Figure 2(c) and taken as input. Based on it, our SA algorithm works as follows:

First, the most and least intense intervals in the configuration of Figure 2(c) are computed with Yao’s algorithm, which are [0,12] and [16,20] with corresponding intensities as $\frac{7}{8}$ and 0, respectively. The initial temperature was set to be $\text{Init}_\text{Temperature} = \frac{\ln 0.95}{\ln 0.90} = 22.75$. Then during the first iteration, we try to select the tasks with mandatory and optional instances completely falling within [0,12] and [16,20] simultaneously to switch its mandatory/optional job pattern correspondingly. For this particular configuration only task $T_1$ is qualified (in case there are multiple such kind of tasks, we just randomly pick out one of them). After the mandatory and optional job patterns of $T_1$ within [0,12] and [16,20] are switched correspondingly as shown in Figure 3(a), we recompute the most and least intensive intervals of the current configuration with Yao’s algorithm, which are [0,12] and [12,24] with corresponding intensities as $\frac{7}{8} = 0.92$ and $\frac{1}{2} = 0.75$, respectively. After that, we evaluate the new configuration based on $\Delta I$, which is computed as $\frac{12}{12} - \frac{7}{2} = -0.25$. Since $\Delta I < 0$, the change is confirmed and the temperature $T$ is reduced with the cooling rate, i.e., $T = 22.75 \times 0.95 = 21.61$.

The above procedures are repeated until the maximal intensity $I^M = I_{R}^{M} + I_{F}^{M}$ of the new solution has been frozen and stopped decreasing in the last few temperatures or the temperature was reduced to zero. Samples of the configurations at different temperatures (with their corresponding probabilities) and the intensities of the different intervals (represented by the height of jobs belonging to them) are shown in Figure 3(b), (c), (d), (e), and (f). Note that, since all of them have higher values of maximal intensities than the configuration in Figure 3(a), they can either be canceled or kept with rapidly deceasing probabilities with the reduction of temperature $T$. After some iterations, the state will eventually be frozen at the configurations similar to that in Figure 3(a) or Figure 3(f). It is not hard to see that all of them have the same value of maximal intensity and each one is schedulable. Note that for this particular example, the configuration converges quickly well before the temperature is reduced to zero.

Note that, when evaluating the acceptance of the configurations with regard to minimize the overall energy consumption, although we can always use the total energy consumption as the metric for evaluation, we still prefer to use the intensity instead because it greatly reduces the time consumption. Moreover, sometimes it is better to move back to a solution that was significantly better rather than always moving from the current state. This is called “restarting”. Restarting is implemented by saving the best solution so far in $S_{\text{Best}}$ and restart the cooling schedule from there. The schedule could be restarted after a fixed number of steps, or when the current intensity is too high compared to the best solution so far.

Compared with the genetic algorithm (GA) in [7], one of the advantages of our SA algorithm is that in SA, the pattern modification is directed towards lower maximal intensity while the GA algorithm in [7] do not have such kind of features. For example, given two task sets $T_1$ and $T_2$, let the objective function in [7] be $f_1$ and the objective function in Equation (4) for our approach be $f_2$. In the GA algorithm in [7], if $f_1(T_1) < f_1(T_2)$ and $T_1$ is schedulable, $T_2$ is NOT guaranteed to be schedulable, while in our SA approach, if $f_2(T_1) > f_2(T_2)$ and $T_1$ is schedulable, $T_2$ is guaranteed to be schedulable. So generally our SA approach is more efficient in improving the schedulability of task sets with given $(m,k)$-constraints.

VI. PERFORMANCE EVALUATION

In this section, we use experiments to demonstrate the effectiveness of our approach in scheduling the real-time systems with $(m,k)$-constraints. Two groups of experiments were conducted to evaluate the corresponding schedulability guaranteeing capability and energy efficiency of different approaches for given task sets with $(m,k)$-constraints.

A. Comparison on schedulability guaranteeing capability

In the first group of experiments, we compare the performance of the different approaches in terms of providing prior off-line guarantee on the schedulability for randomly generated task sets. Specifically, we compared the performance of five different approaches. For the first approach $Alg - E$, the task sets are partitioned with E-pattern from [3]. For the second approach $Alg - E_R$, the task sets are partitioned with $E_R$-pattern from [9]. For the third approach $Alg - R$, the task sets are statically partitioned with R-pattern from [12], which is also the pattern on which the approach in [21] is based in order to provide the prior off-line guarantee for its online algorithm. The fourth approach $Alg - EE$ is our approach in Section IV which seeks to rotate the E-pattern adaptively to improve its schedulability. The fifth approach $Alg - SA$ is our approach in Section V which adopts simulated annealing to determine the general appropriate regular $(m,k)$-pattern for any given task set. (We did not compare with the approaches...
in [7] because the approaches in [7] are based on computing the “interference” from higher priority tasks on lower priority tasks under fixed-priority scheme, which are not applicable to systems scheduled under EDF scheme.)

The periodic tasks sets were randomly generated with at least five tasks each. The periods were randomly chosen in the range of [10ms, 100ms] and the relative deadlines were randomly set to be less than or equal to the periods. The worst case execution time (WCET) was set to be uniformly distributed from 1 to its deadline. The $m_i$ and $k_i$ for the $(m,k)$-constraints were also randomly generated such that $k_i$ is uniformly distributed between 2 to 10, and $m_i < k_i$. Since it is proved in [9] that any task sets schedulable with R-pattern can be schedulable with any other $(m,k)$-pattern, we discarded those task sets schedulable with R-pattern as they do not provide any insight on the schedulability comparison between the different approaches. Consequently we did not include the results for Alg − R in the evaluation for this group of comparison, either.

To reduce the statistical error, the $(m/k)$-utilization, i.e., $\sum_{i \in T} \frac{m_i C_i}{k_i}$, is divided into intervals of length 0.1 and within each interval, the task sets were randomly generated such that at least 100 task sets were schedulable by at least one of the approaches (except Alg − R) or at least 5000 task sets have been generated for the interval. Note that although we can always resort to simulation to check the schedulability of the task sets, some faster feasibility conditions can help greatly reduce the time consumption. In our experiment, for task sets to be scheduled with Alg − E, we check their schedulability with the feasibility condition in [9]. And for task sets with approaches Alg − ER, Alg − EE and Alg − SA, we check their schedulability based on Lemma 1. The numbers of task sets schedulable by each of the approaches are shown in Figure 4.

As seen from Figure 4, when the $(m/k)$-utilization is very low (e.g., less than 0.2), the schedulability of the different approaches are close to each other. However, as the $(m/k)$-utilization increases, schedulability of Alg − E begins to degrade quickly. The schedulability of Alg − ER has marginal improvement over Alg − E but not very significant for most of the utilization intervals. With pattern adjustment based on rotation, Alg − EE can improve Alg − E significantly when the $(m/k)$-utilization is not too high or too low. The maximal improvement can be around 20%. However, as the $(m/k)$-utilization further increases, the improvement of Alg − EE over Alg − E also becomes limited. With more advanced pattern adjustment based on simulated annealing, Alg − SA can further improve Alg − EE effectively and it always has the best performance among the different approaches. Moreover, the improvement becomes more significant as the $(m/k)$-utilization increases. For example, when the utilization is between [0.0,0.8], the average improvement of Alg − SA over Alg − E can be more than 36%. From the experiment results, it also conforms to our expectation that all task sets schedulable with Alg − E or Alg − ER are schedulable with Alg − EE and all task sets schedulable with the previous approaches are schedulable with Alg − SA.

B. Energy performance evaluation

In the second group of experiments, we test the performance of the different approaches in terms of energy consumption of the system. Note that for this group of experiments, we replaced the algorithm Alg − ER above with the energy conservation algorithm $HYB_{ER}$ in [9] which adopts dynamic pattern based on $R$-pattern. We also implemented Alg − R with the energy constrained algorithm from [22] based on R-pattern. At the same time, we also compared with the online algorithm $DYN_{DP}$ in [21] which adopts dynamic pattern based on R-pattern to provide prior off-line guarantee.

We simulate the execution of the real time task sets on a DVFS processor model that can operate at a finite set of discrete supply voltage levels $V = \{V_1, ..., V_{max}\}$, each with an associated speed $s_i$. We compute the energy consumption of the task sets based on the following power model.

1) Power model: The power consumption of the DVFS processor can be divided into two parts: the speed-dependent part $P_{dep}(s)$ and the speed-independent part $P_{ind}$. The speed-dependent power consumption $P_{dep}(s)$ mainly comes from the dynamic power consumption and short-circuit power consumption, while the speed-independent power consumption mainly comes from the leakage. The dynamic power mainly consists of the switching power for charging and discharging the load capacitance and can be represented [6] as $P_{dy} = C_l V^2 f$, where $C_l$ is the load capacitance, $V$ is the supply voltage, and $f$ is the system clock frequency. The leakage power is mainly due to the subthreshold leakage current and the reverse bias junction current in the CMOS circuit. If the leakage power consumption is related to the speeds/voltages, it can be divided into two parts that contribute to $P_{dep}(s)$ and $P_{ind}$ accordingly. So the total power consumption when the processor is in its active state, i.e., $P_{act}(s)$, is thus $P_{act}(s) = P_{dep}(s) + P_{ind}$. The speed dependant power $P_{dep}(s)$ can be modeled as a strictly convex and increasing function of $s$. And it can usually be expressed as $P_{dep}(s) = \alpha s^m$, where $m$ is a constant between 2 and 3 [23].

The processor can be in one of the three states: active, idle and sleeping states. When the processor is idle, the major portion of the power consumption comes from the leakage which increases rapidly with the dramatic increasing of the leakage power consumption. Shutting-down strategy, i.e., put the processor into its sleeping state, can greatly reduce the leakage energy. However, it has to pay extra energy and
timing overhead to shut down and later wake up the processor. Assume that the power consumptions of a processor in its idle state and sleeping state are $P_{idle}$ and $P_{sleep}$, respectively, and the energy overhead and the timing overhead of shutdown/wakeup is $E_o$ and $t_o$. Then the processor can be shut down with positive energy gains only when the length of the idle interval is larger than $T_h = \max(\frac{P_{idle}-P_{sleep}}{E_o}, t_o)$. We call $T_h$ as the shut down threshold interval.

When processor is active, without the consideration of the leakage, it would be most energy efficient to run the tasks with voltage as low as possible. However, reducing the supply voltage usually requires the reduction of the threshold voltage to maintain the circuit performance, which will dramatically increase the leakage current and, hence, the leakage power consumption. Considering a job with workload $w$ and total power function as $P_{act}(s)$, the total energy ($E_{act}(s)$) consumed to finish this job with speed $s$ can be represented as $E_{act}(s) = P_{act}(s) \times \frac{s}{2}$. Hence, to minimize $E_{act}(s)$, let $\frac{dE_{act}(s)}{ds} = 0$. When $P_{dep}(s) = ax^3$, the derived speed is $s = (\frac{E_{max}}{E_o})^{\frac{1}{2}}$, which is the optimal speed to minimize the active energy for executing a job. We therefore call this speed the critical speed (denoted as $s_{crit}$) and use it as the lower bound for our speed/voltage scaling.

2) Results on energy consumption: The processor model used in our experiments was the Intel XScale processor model [24] which we assumed to be able to run with available discrete speeds in the range of $[0, 1 \text{ GHz}]$ with step of 50MHz. According to [25], the power consumption function for Intel XScale [24] can be modeled approximately as $P_{act}(s) = 0.08 + 1.52s^2$ Watt by treating 1GHz as the reference speed 1. And the normalized critical speed in such a model is about 0.3 (at 297 MHz) with power consumption 0.12W. We assume the shut down overhead to be $E_o = 800\mu$J [25]. If the minimal processor speed is 0, the idle power consumption of the processor is 0.08 Watt and the corresponding shut down threshold will be $T_h = 10\text{ms}$.

In this group of experiments, for the approaches based on static pattern (i.e., Alg-R, Alg-E, Alg-EE, and Alg-SA), we adopt the job level speed scaling because it can be more close to the lower bound of energy consumption by the ideally optimal voltage schedule. In doing that, the speed schedules of each task set within its hyperperiod are derived with Yao’s algorithm [5] first. Then all job speeds less than the critical speed $s_{crit}$ are rounded up to $s_{crit}$. For the approaches based on dynamic pattern (i.e., $HYB_{ER}$ and $DYN_{DP}$), since Yao’s algorithm is not applicable any more, we just scale the speed of each task to as low as the critical speed $s_{crit}$ at the task level with the feasibility conditions in [9] and [21] correspondingly, and then during the run-time, the job speeds will be updated dynamically based on the approaches in [9] and [21] respectively. For all approaches above, whenever a job speed is less than the critical speed $s_{crit}$, it will be rounded up to $s_{crit}$. Also for all approaches compared, the total energy consumption for each task set within twice of its hyperperiod will be computed with simulations.

We first studied the energy consumption of these approaches based on the synthesized task sets. The periodic task sets (including the periods, deadlines, WCETs, and $m_i$ and $k_i$ values) were randomly generated in the same way as in Section VI-A. And we also divided the $(m,k)$—utilization, i.e., $\sum_i m_i k_i$ into intervals of length 0.1 and for each interval we randomly generated 50 feasible task sets. The energy consumption for each approach was collected and normalized to that by Alg-R, and the results are shown in Figure 5 (a).

From Figure 5 (a), the energy consumption of Alg—R is always the highest among the approaches based on static $(m,k)$—patterns due to the poor schedulability of R-pattern. Also when the utilization is very low, the energy consumption by the other approaches based on static $(m,k)$—patterns are close to each other because their schedulabilities are similar, as shown in the first group of experiments. However, as the $(m,k)$—utilization increases, the energy consumption of Alg—EE becomes much better than Alg—E. The maximal energy reduction by Alg—EE over Alg—E can be around 25%. This is because, by rotating the E-pattern adaptively, Alg—EE is able to shift the mandatory work demand among different tasks away and redistribute the mandatory workload of the whole task set more evenly within the hyperperiod. As a result, the intensities of the intervals can be effectively reduced. So the required speeds to execute the mandatory jobs within those intervals can also be lowered correspondingly. Furthermore, with our simulated annealing approach, the energy can be reduced more significantly. For example, when the utilization is not high, the energy reduction achievable by Alg—SA over Alg—E can be nearly 34%. This is because, by adjusting the patterns artistically with techniques based on simulated annealing, the patterns can be determined more flexibly and the intensities of the intervals can be further reduced effectively. As a result the required job speeds can be reduced more.

![Fig. 5. The energy comparison of different approaches for (a) synthesized task sets; (b) Webphone; (c) INS.](image-url)
aggressively. This is especially true when the periods of the tasks are not coprime with each other. It is also interesting to note that although the approaches based on dynamic patterns, i.e., HYB\(_k\) and DYNN\(_P\) can reduce the dynamic energy efficiently as shown in [9] and [21], their performances degrade severely when both dynamic and leakage energy are considered simultaneously. For example, when the utilization is less than 0.2, the total energy consumption by HYB\(_k\) and DYNN\(_P\) can be even higher than that by Alg \( R \). This is mainly because HYB\(_k\) and DYNN\(_P\) optionally executed quite a few redundant jobs whose total energy cost (including dynamic and leakage part) cannot be compensated by the savings in dynamic energy alone. Another reason is generally the job level speed schedule adopted by the approaches based on static patterns are more close to the lower bound of energy consumption by the ideally optimal case and thus can be more energy efficient than the task level speed slowdown adopted by the approaches based on dynamic patterns.

Next, we compare the energy performance of the different approaches in a more practical environment. The test cases contained two real world applications: webphone [14], and INS (Inertial Navigation System) [16]. The timing parameters such as the deadlines, periods, and execution times were adopted from these practical applications. The \((m,k)\) -constraints were generated as we did for the synthesized task sets. The normalized total energy consumptions are shown in Figure 5 (b) and (c).

As seen in Figure 5 (b) and (c), the experimental results based on the practical applications further demonstrate the effectiveness of our approaches in saving energy. From Figure 5 (b) and (c), for both the webphone and INS applications, the energy consumption of Alg \(- E\) still outperformed Alg \(- R\), but not as much as for the randomly generated task sets. This is because for randomly generated task sets, the task periods are less likely to contain very large common divisors. By rotating the \((m,k)\) -pattern adaptively, Alg \(- EE\) can still achieve a maximal improvement of 15% for Webphone and 9% for INS when compared with Alg \(- E\). Alg \(- SA\) has much better performance in this group of experiments. As seen, the maximal improvement of Alg \(- SA\) over Alg \(- E\) can be around 38% for webphone and 43% for INS. This is mainly because, for these real-world applications, their periods contain large greatest common divisors. In this scenario approaches based on evenly distributed pattern cannot reduce the job speed very significantly. Contrary to that, Alg \(- SA\) can determine the pattern more adaptively and thus reduce the intensities of the intervals and the required job speeds more aggressively. And similar to the results with synthesized task sets, the energy consumption of the approaches based on dynamic pattern, i.e., HYB\(_k\) and DYNN\(_P\) are much higher than that by Alg \(- EE\) and Alg \(- SA\) for the same reasons as stated above.

**VII. Conclusions**

In this paper, we proposed two off-line approaches to improve the schedulability and energy efficiency for weakly hard real-time embedded systems. The first one intends to improve the previous evenly distributed \((m,k)\) -pattern through adaptive pattern adjustment. And the second one determines the more generalized regular \((m,k)\) -pattern with techniques based on simulated annealing. Through extensive experiments, the results demonstrate that our proposed techniques outperformed the previous work significantly in both schedulability and energy efficiency for weakly hard real-time embedded systems.

For the future work, we plan to explore the parallel simulated annealing techniques to improve the schedulability and energy efficiency for weakly hard real-time systems on multicore platforms.

**References**


