Reliability-Aware Scheduling for (m,k)-firm Real-Time Embedded Systems under Hard Energy Budget Constraint

Linwei Niu[a,*], Jonathan Musselwhite[1]

[a]Department of Electrical Engineering and Computer Science, Howard University, Washington, DC, 20059, U.S.A.

Abstract

For real-time embedded systems, feasibility, Quality of Service (QoS), reliability, and energy constraint are among the primary design concerns. In this research, we proposed a reliability-aware scheduling scheme for real-time embedded systems with (m,k)-firm deadlines under hard energy budget constraint. The (m,k)-firm systems require that at least m out of any k consecutive jobs of a real-time task meet their deadlines. To achieve the dual goals of maximizing the feasibility and QoS for such kind of systems while satisfying the reliability requirement under given energy budget constraint, we propose to reserve recovery space for real-time jobs in an adaptive way based on the mandatory/optional job partitioning strategy. The evaluation results demonstrate that the proposed techniques significantly outperform the previous research in maximizing the feasibility and QoS for (m,k)-firm real-time embedded systems while preserving the system reliability under hard energy budget constraint. Moreover, the proposed work has also addressed some insufficiency in [26] in terms of preserving the system reliability.

Keywords: Feasibility, QoS, Reliability, Energy Constraint, Scheduling

1. Introduction

With the advance of IC technology, energy constraint has been an increasingly important factor for the design of real-time embedded systems. In some real-time

*Corresponding author. Tel: +1 2028064822.

Email addresses: linwei.niu@howard.edu (Linwei Niu), jonathan.musselwhite@bison.howard.edu (Jonathan Musselwhite) applications, the systems are driven by power supplies with limited energy budget constraint, which has to remain operational during a well-defined mission cycle. Examples include Heart Pacemakers [40] or other portable embedded devices whose power supply can only be charged to full capacity right before the beginning of certain mission/operation cycle/period(s). For such kind of applications, efforts must be made by all means to avoid exhausting the energy budget before the end of the mission cycle.

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, in many practical real-time applications such as multimedia processing and real-time communication systems, occasional deadline misses can often be tolerated. Some other applications may have soft deadlines where tasks that do not finish by their deadlines can still be completed with a reduced value [23] or they can simply be dropped provided that the user's perceived quality of service (QoS) is satisfied.

QoS requirements dictate under what conditions the system will provide its service to real-time tasks executed in the embedded processor. To quantify the QoS requirements, some statistic information such as the average deadline miss rate can be used. However, even a low overall miss rate (e.g., 2%) cannot prevent the scenarios where a very large number of deadline misses (e.g., 20 deadline misses out of 1000 jobs) occur consecutively in a short period of time, which could generate undesirable results.

The deterministic QoS model is more appropriate for such kind of systems. Under the deterministic QoS model, tasks have both firm deadlines (i.e., task(s) with deadline(s) missed generate(s) no useful values) and a throughput requirement (i.e., *sufficient* task instances must finish before their deadlines to provide acceptable QoS levels) [30]. Two well known deterministic QoS models are the (m,k)-model [12] and the *window-constrained* model [43]. The (m,k)-model requires that m jobs out of any *sliding* window of k consecutive jobs of the task meet their deadlines, whereas the *window-constrained* model requires that m jobs out of each *fixed* and *nonoverlapped* window of k consecutive jobs meet their deadlines. It is not hard to see that the *window-constrained* model is weaker than the (m,k)-model as the latter one is more restrictive.

To ensure the (m,k)-constraints, Ramanathan *et al.* [35] proposed a partitioning strategy which divides the the jobs into *mandatory* ones and *optional* ones. The mandatory ones are jobs that must meet their deadlines in order to satisfy the (m,k)-constraints. In [43], West *et al.* tried to set up a corresponding relationship from the *window-constrained* model to the (m,k) model. They also found a method to map the *window-constraints* to the (m,k)-constraints.

In the meantime, with the aggressive scaling of transistor size in CMOS circuits, more and more transistors are integrated into a single die and the power consumption of IC chips has been increasing dramatically. For the past two decades, extensive researches on power management techniques (e.g. [13]) have been reported on energy minimization for embedded real-time systems. Among them dynamic voltage scaling (DVFS) and Dynamic Power Management (DPM) are two widely used techniques to reduce energy dissipation. DVFS dynamically adjusts the supply voltage and working frequency to reduce power consumption at the cost of extended circuit delay, whereas DPM techniques try to put the processing unit in a low-power inactive state when it is not in use.

In recent years, reliability has become increasingly important in the design of fault tolerant computer systems as system fault could occur anytime during the execution cycle of real-time jobs [54]. Moreover, it has been shown that DVFS could affect system reliability negatively because the probability of transient faults could be much higher at lower supply voltages [54]. In order to satisfy the reliability requirement of a given job, a widely adopted strategy is to reserve a recovery job for it before scaling its speed using DVFS [54]. If the job has failed due to transient faults [17], the recovery job should be invoked for execution to compensate the failed job.

In the context of reliability assurance, the energy-constrained issue is especially critical as the recovery job(s) (for preserving the system reliability) will also occupy part of the system utilization, which could leave much less space for adjusting the speeds of the mandatory jobs to keep the total energy consumption under control. When the QoS is taken into consideration, the problem becomes even more challenging as we need to ensure that the baseline (m,k)-constraints be satisfied all the time without exceeding the given hard energy budget constraint.

In this paper, we study the problem of maximizing the feasibility, QoS, and energy performance for (m,k)-firm real-time embedded systems while satisfying the reliability requirement under given hard energy budget constraint.

The rest of the paper is organized as follows. Section 2 talks about the related work. Section 3 presents the preliminaries. Section 4 presents the motivations. Section 5 presents our general scheduling algorithm. In section 6, we present our evaluation results. In section 7 we offer the conclusions.

2. Related Work

In last decades, plenty of work has been done in integrating QoS constraint into scheduling for real-time systems. For systems with transient overloaded conditions, Chetto *et al.* [5] explored scheduling algorithms for firm real-time systems. For real-time applications organised as pipelines of tasks using resources of different type, Cucinotta *et al.* [6] explored how to effectively share the resources such that robust invariance is preserved. Their approach adopted QoS requirements tightly related to the end-to-end temporal behaviour of the application, which is different from our (m,k)-firm model dealing with QoS locally. For mixed-criticality systems, Gettings *et al.* [11] and Bruggen *et al.* [41] proposed new approaches that can provide QoS-guarantee for low-criticality tasks. Moreover, for general fixed-priority weakly-hard real-time systems, schedulability analysis based on the Mixed Integer Linear Programming (MILP) formulation are provided in [39]. For mixed systems consisting of both periodic and aperiodic tasks, Buttazzo *et al.* [3] studied minimizing aperiodic response times in a firm real-time environment without considering energy consumption. With given energy budget constraint in mind, Alenawy *et al.* [2] proposed an approach to reduce the number of (m,k)-violations for weakly hard real-time systems. Also to minimize the number of dynamic failures, Kooti *et al.* [18] proposed a QoS-aware approach for (m,k)-firm real-time systems with long-term variations of the harvested energy.

With energy consumption in mind, some previous researches have been conducted under the deterministic QoS models. In [29, 25], energy-efficient scheduling schemes were proposed for real-time systems with (m,k) and *window-constraints*, respectively. When system-wide energy consumption is considered, in [30], Niu *et al.* proposed approaches to reduce the energy for both the processor and peripheral devices. Recently, with reliability becoming an important concern for ubiquitous computing systems, a lot of works have also been reported in combining reliability and energy management for real-time embedded systems. In [54], Zhu *et al.* formulated the reliability as the probability of executing the real-time tasks/jobs successfully. They found that DVFS can affect the system reliability adversely. In [48], a recovery-sharing scheme was proposed to reduce energy for "frame-based" real-time task sets. In [50], the above scheme was extended to systems with precedence constraint. In [20], Li *et al.* introduced an adaptive checkpointing scheme to minimize energy consumption under reliability requirement. Their work targeted the "frame-based" real-time systems only. For systems with more general real-time constraints, Zhao *et al.* [49] proposed an approach to reduce energy for periodical real-time tasks with reliability requirement quantified for each task individually. When considering shared resource synchronization, Zhang *et al.* [42] proposed a scheme to reduce energy consumption under reliability requirement. In [24], Neukirchner *et al.* presented a methodology for contract based dynamic task management for mixed-criticality real-time systems. They provides a close coupling between an admission control scheme and task management facilities which ensures that the effects of ill-specified components are properly contained and thus allows safe reconfiguration of mixed criticality systems. Their approach mainly focused on hard real-time systems and did not take energy consumption into consideration.

For multicore/multiprocessor systems, some works have also been done to reduce energy consumption under reliability requirement. In [7], Das *et al.* proposed an offline approach for mapping tasks onto processor cores to minimize energy consumption for all processor fault-scenarios. In [15], Haque *et al.* proposed a standby-sparing technique which makes use of the hardware redundancy in dual-processor systems to tolerate both transient and permanent faults. In [36], Roy *et al.* proposed a scheduling scheme to reduce energy consumption for heterogeneous multicore systems while guaranteeing the system reliability. In [45], a passive primary/backup technique is proposed to reduce power/energy consumption in heterogeneous multicore systems by considering real-time and peak power constraints while preserving the reliability requirement of the system at a satisfactory level. The proposed method tries to map the primary and backup tasks in a mixed manner to remove the overlap between the execution of the primary and backup tasks. In [8], Dobias *et al.* explored online mapping and scheduling of hard real-time systems adopting primary/backup technique in order to improve the system reliability subject to various constraints regarding such as time, space, and energy.

Note that most of the above energy-aware approaches are focused on reducing the energy as much as possible. However, for systems that are driven by power supplies with limited energy budget, the above best-effort approaches might not be able to ensure that the system could remain operational during a well-defined mission cycle. For systems with given fixed energy budget for its operation, Zhao *et al.* [47] proposed an approach to maximize the overall reliability for hard real-time systems. In [31], energy-constrained scheduling schemes for standby-sparing systems consisting of two Non-DVFS processors are studied. To the best of our knowledge, energy-constrained DVFS scheduling for (m,k)-firm systems under the reliability requirement has not been studied yet. In this work, we assume the system is operating in an energy-constrained system equipped with DVFS capability in which the energy consumption of the system must not exceed a given fixed budget. Based on it, we explore maximizing the feasibility and QoS for the system while satisfying the (m,k)-constraint under the reliability requirement.

3. Preliminaries

3.1. System models

The real-time task set considered in this paper contains n independent periodic tasks, $\mathcal{T} = \{\tau_1, \tau_1, \cdots, \tau_n\}$, scheduled according to the earliest deadline first (EDF) policy [22]. Each task contains an infinite sequence of periodically arriving instances called *jobs*. Task τ_i is characterized using five parameters, *i.e.*, $(P_i,\ D_i,\ C_i,\ m_i,\ k_i)$. P_i, D_i $(D_i \leq P_i)$, and C_i denote the period, the deadline and the worst case execution time for τ_i, respectively. A pair of integers, *i.e.*, (m_i, k_i) $(0 < m_i \leq k_i)$, are used to represent the (m,k)-constraint for task τ_i which requires that, among any k_i consecutive jobs, at least m_i jobs are executed successfully. The j^{th} job of task τ_i is denoted as J_{ij} and its arrival time and absolute deadline are denoted as r_{ij} and d_{ij}, respectively. In addition, if job J_{ij} is used as a recovery job, it is also represented as $J_{ij}(R)$. The hyper-period of the task set, represented by H, is defined as $H = LCM\{k_i T_i\}$.

We assume the task set is to be executed in a uni-processor system with a limited energy supply of $\bar{E}$ units during its mission cycle. Moreover, we assume this energy budget is a hard constraint in a sense that it cannot be exceeded at any time during its mission cycle. Without loss of generality, we let the mission cycle be the hyper period of the task set and assume that the energy supply can only be charged to full capacity right before the beginning of each hyper period.

3.2. Fault and Recovery Models

Similar to [32, 14], we focus on transient faults in this paper since transient faults occur much more frequently than permanent faults in modern semi-conductor devices [10]. We assume that faults can be detected using sanity (or consistency) checks [33] when a job finishes its execution and the overhead for detection can be integrated into the job's worst case execution time. Moreover, when transient fault occurs and is detected at the end of a job's execution, the affected job can be addressed by re-executing a recovery job with the same worst-case execution time as the original one [33]. Once released, the recovery job will be executed like a normal job. Note that, in order to preserve the reliability, the recovery jobs should always be executed with the maximal speed s_{max}.

Following the fault model in [54], we assume that the transient faults will present Poisson distribution [46] and the average transient fault rate for systems running at speed s (and the corresponding supply voltage) is [54]:

$$\lambda(s) = \lambda_0 \times 10^{(d \times \frac{1-s}{1-s_{min}})} \tag{1}$$

where λ_0 is the average fault rate corresponding to the maximum speed s_{max} and $d(>0)$ is a constant representing the sensitivity of transient faults on voltage scaling. Larger value of d means the transient faults are more sensitive to voltage scaling.

3.3. Power Model

The power/energy consumption on a DVFS processor can be divided into two parts: the speed-dependent part $P_d(s)$ and the speed-independent part P_{nd}. The speed-dependent part $P_d(s)$ mainly comes from the dynamic power consumption and short-circuit power consumption and can be represented as $P_d(s) = \alpha s^m$, where α and m are constants [54], while the speed-independent part P_{nd} mainly comes from the leakage which is due to the subthreshold leakage current and the reverse bias junction current in the CMOS circuit. The total power consumption when the processor is active, *i.e*, $P_a(s)$, is thus $P_a(s) = P_d(s) + P_{nd}$.

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 is increasing rapidly with the continued scaling of IC technology size. 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 shutting-down/waking-up the processor are E_o and t_o, respectively. Then the processor can be shut down with positive energy gains only when the length of the idle interval is larger than $T_{th} = \max(\frac{E_o}{P_{idle}-P_{sleep}}, t_o)$ [32].

When the processor is active executing a job with workload w, the total energy $(E_a(s))$ consumed to finish this job with speed s can be represented as $E_a(s) = P_a(s) \times \frac{w}{s}$. Hence, to minimize the energy consumption, we let $\frac{\partial E_a(s)}{\partial s} = 0$. When $P_d(s) = \alpha s^m$, the derived speed is $s = (\sqrt[m]{\frac{P_{nd}}{(m-1)\alpha}})$, which is the optimal speed to minimize the active energy for executing a job. We therefore call this speed the *energy-efficient speed*, and denote it as s_{ee} [54]. Note that since the s_{ee} derived here has not taken timing constraint into consideration, it might not always be feasible and can only be used as a lower bound of speed scaling for real-time jobs.

3.4. Performance Metrics

3.4.1. Feasibility

Given system $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ to be scheduled on a variable voltage processor with energy budget $\bar{E}$, $\mathcal{T}$ is said to be *feasible* if the (m,k)-constraints of all tasks in $\mathcal{T}$ are satisfied under a speed schedule $\mathcal{S}$ with total energy consumption not exceeding the given energy budget $\bar{E}$.

3.4.2. Reliability

For any job J_{ij} of task τ_i, the reliability of it under speed s_{ij}, represented by $\gamma(J_{ij}, s_{ij})$, is defined as the probability that J_{ij} could be completed successfully. According to [54], $\gamma(J_{ij}, s_{ij})$ is given as:

$$\gamma(J_{ij}, s_{ij}) = e^{-\lambda(s_{ij}) \times \frac{C_i}{s_{ij}}} \tag{2}$$

Under the QoS-constraint (either in the (m,k)-constraint of (m_i,k_i), or the window-constraint of m_i/k_i), the reliability of the l^{th} window W_{il} of task τ_i, represented by $\gamma(W_{il})$, is defined as the probability that at least m_i jobs in W_{il} could be completed successfully.

Based on it, the reliability of task τ_i is defined as,

$$\gamma(\tau_i) = \prod_{l=1}^{z_i} \gamma(W_{il}) \tag{3}$$

where z_i is the number of windows in τ_i that need to be inspected.

And the reliability of the whole system $\mathcal{T}$ is defined as,

$$\gamma(\mathcal{T}) = \prod_{i=1}^{n} \gamma(\tau_i) \tag{4}$$

3.4.3. Quality of Service

Based on the above system models, we define the metrics to measure the quality of service of a task τ_i, represented by $QoS(\tau_i)$ to be the ratio of the number of jobs completed successfully and the total number of jobs within the hyperperiod H. Based on it, we define the actual quality of service of a task τ_i, represented by QoS_i, as followed:

$$QoS_i = \frac{x_i}{y_i} \tag{5}$$

where x_i is the number of valid jobs and y_i is the total number of jobs, both within the hyperperiod H.

And the system quality of service is defined as

$$QoS_{sys} = \sum_{i=1}^{n} QoS(\tau_i)\omega_i \tag{6}$$

where ω_i is the weight of task τ_i (could be user-defined).

Meanwhile, with the window reliability in mind, we also define the expected quality of service of a task τ_i, represented by $\widetilde{QoS}_i$, as followed:

$$\widetilde{QoS}_i = \frac{\Sigma_{l=1}^{z_i}(m_i \times \gamma(W_{il}))}{z_i \times k_i} = \frac{m_i}{k_i} \times \frac{\Sigma_{l=1}^{z_i}\gamma(W_{il})}{z_i} \tag{7}$$

Correspondingly, the expected system quality of service $\widetilde{QoS}_{sys}$ is defined as

$$\widetilde{QoS}_{sys} = \sum_{i=1}^{n} \widetilde{QoS}_i \omega_i \tag{8}$$

where ω_i is the weight of task τ_i.

4. Motivations

To satisfy the reliability requirement, one essential part is to reserve recovery jobs for the tasks when applying DVFS to reduce energy. Prior to that, a key problem for ensuring the (m,k)-constraints is to judiciously partition the jobs into *mandatory* jobs and *optional* jobs [34]. A well-known partitioning method is called the *the evenly distributed pattern* (or E-pattern) [35]. According to E-pattern, the pattern π_{ij} for job J_{ij}, *i.e.*, the j^{th} job of a task τ_i, is used to indicate whether job J_{ij} is partitioned as mandatory or optional job. Specifically, π_{ij} is defined as followed:

$$\pi_{ij} = \begin{cases} \text{“1”} & \text{if } j = \lfloor \lceil \frac{(j-1)\times m_i}{k_i} \rceil \times \frac{k_i}{m_i} \rfloor + 1 \\ \text{“0”} & \text{otherwise} \end{cases} \qquad j = 1, 2, \cdots \tag{9}$$

where $\pi_{ij} = 1$ represents the mandatory job and $\pi_{ij} = 0$ represents the optional job. In [21], a variation of E-pattern called E^R-pattern was achieved by reversing the pattern horizontally to let the optional jobs happen first, which can preserve the schedulability of E-pattern [21].

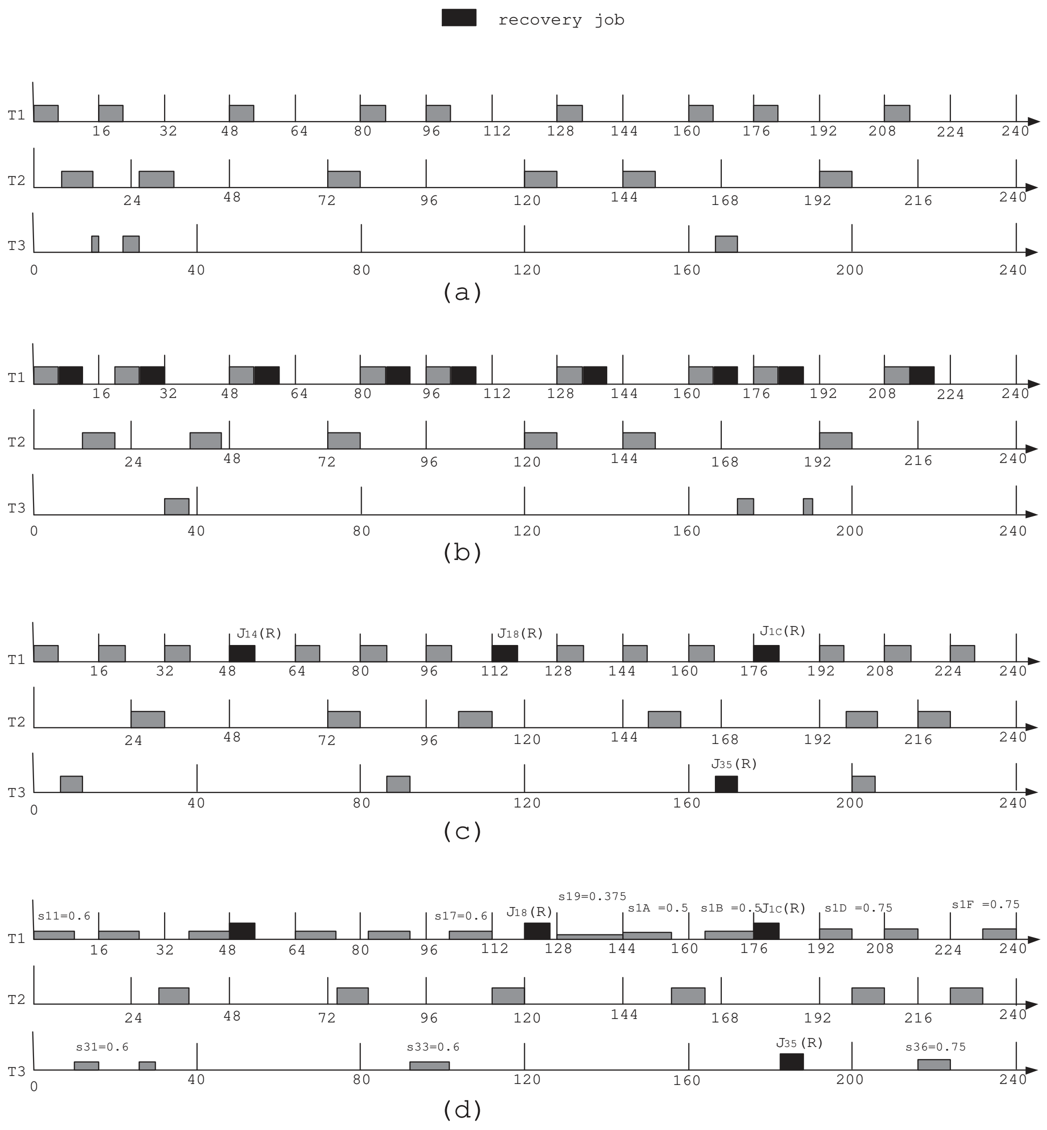


Figure 1: (a) The original schedule for task set $\{\tau_1 = (16,16,6,3,5);\ \tau_2 = (24,24,8,3,5);\ \tau_3 = (40,40,6,2,8)\}$ based on E-pattern without reliability/energy management; (b) The schedule based on the original (m,k)-constraint managed with recovery jobs (only recovery jobs for task τ_1 can be accommodated);(c) The schedule under the maximal speed with recovery jobs reserved for τ_1 and τ_3 based on *window-constraints* that could be transferred to their original (m,k)-constraints; (d) The schedule for the tasks in (c) after the speeds of the mandatory jobs are scaled.

Note that based on the E-pattern given in Equation (9), without energy management, *i.e.*, all mandatory jobs to be executed under the maximal speed without recovery, the reliability of any window W_{il} of k_i jobs could be calculated as:

$$\gamma(W_{il}) = \prod_{p=(l-1)\times k_i+1}^{(l-1)\times k_i+k_i} \{\gamma(J_{ip}, s_{max}) \mid \pi_{ip} = 1\} \tag{10}$$

It is also noted that the job patterns defined with E-pattern have the property that they define a *minimal set* of mandatory jobs that "just" satisfies the given (m,k)-constraint in each sliding window. Due to this property, before the speed for any task is scaled to save energy, a popular approach is to reserve a recovery job for each mandatory job of the task to satisfy the reliability of the same task. Conversely, under window-constraint, this requirement could be greatly relaxed as we only need to reserve *one* recovery job within each separate window so long as we can guarantee that reliability of the window under consideration is preserved[1], *i.e.*, not lower than that calculated with Equation (10). However, the problem is, since window-constraint is weaker than (m,k)-constraint, a task τ_i satisfying the window-constraint of m_i/k_i does not necessarily satisfy the (m,k)-constraint of (m_i,k_i). Fortunately, in [43] it is shown that the *window-constraint* of m_i/k_i can be transferred to the (m,k)-constraint of $(m_i, 2k_i - m_i)$. That means if we can reserve recovery jobs for the tasks such that it could satisfy the window constraint of m_i/k_i, it will satisfy the (m,k)-constraint of $(m_i, 2k_i - m_i)$ automatically. Since in this case we only need to reserve one recovery job for each separate window of k_i jobs, it could leave us more space to keep the energy under control than reserving recovery jobs for all mandatory jobs under E-pattern for tasks with (m,k)-constraints of $(m_i, 2k_i - m_i)$, which could be illustrated using the following example.

Consider a task set of three tasks, *i.e.*, $\tau_1 = (16,16,6,3,5)$, $\tau_2 = (24,24,8,3,5)$, and $\tau_3 = (40,40,6,2,8)$, to be executed in a processor with energy budget before time 240 to be 150 mJoule. Figure 1(a) shows the schedule for the mandatory jobs based on E-pattern. If we assume the probability of transient fault for the system running at full speed is 10^{-6}, without reliability management, based on Equa-

[1]Note that, in this paper, similar to the work in [53, 51] we assume the reliability of the original system executing at the maximal speed is satisfactory and our goal is to schedule the system while guaranteeing that the resulting reliability of the system is never lower than that under the maximal speed, *i.e.*, with the reliability preserved. But we can also set the reliability requirement higher than that if necessary.

tion (10), the reliability of each window in τ_1, τ_2, and τ_3 will be 0.9999820002, 0.9999760003, and 0.9999880001, respectively. Based on Equation (4), the corresponding system reliability will be 0.9999460015. Meanwhile, if we assume all tasks in Equation (8) have equal weights, *i.e.*, $\omega_i = \frac{1}{n}$ for all tasks in $\mathcal{T}$, the QoS of the whole system based on Equation (8) will be 0.48332393344.

If we apply the state-of-the-art reliability management strategy, *i.e.*, to reserve a recovery job for each mandatory job whenever possible to do so, Figure 1(b) shows the schedule for the mandatory jobs based on E-pattern for the original given (m,k)-constraints, with a recovery job reserved for each mandatory job of task τ_1. Note that in this case after reserving recovery jobs for task τ_1, there is not enough room for reserving recovery jobs for task τ_2 or task τ_3 any more. Therefore, only the reliability of each window in τ_1 could be improved to 0.99999999996 while the reliability of each window in tasks τ_2 and τ_3 will remain the same. As a result, the reliability of the whole system based on Equation (4) is now 0.99996400056. And in this case the QoS of the whole system based on Equation (8) is 0.48332753339. Compared with the above schedule in Figure 1(a) without reliability management, there is some improvement in the system reliability and very slight improvement in system QoS. However, the problem is that it might not be able to meet the given hard energy budget constraint. For ease of presentation here we assume the task set to be executed in an Intel XScale processor [4]. According to [4], the power consumption function for Intel XScale [16] can be modeled approximately as $P_a(s) = 1.52s^3 + 0.08$ *Watt* by treating 1GHz as the reference speed 1. Other parameters such as idle power consumption and shut down overhead can be found in [4]. Note that, after recovery jobs are reserved for task τ_1, there is no room for reducing the speed for task τ_1 any more. Consequently no task in the whole task set got a chance to have its speed reduced. Also note that in this example even though the reserved space for the recovery jobs in task τ_1 could be freed and become slack time reclaimable by other mandatory jobs of τ_2 (or τ_3) after their "primary jobs" are completed successfully, according to [53], the slack times need to be used to reserve recovery jobs for the job(s) under consideration (for example, J_{21}) first. In this particular example, after reserving recovery jobs for the mandatory jobs of τ_2 (or τ_3) using the slack from freed recovery jobs in τ_1, there will be no slack time left that could be used to help scale the speeds of the mandatory job(s) anymore. Consequently still no mandatory job could get chance to have its speed reduced. As a result the total worst case energy consumption before the hyper period, *i.e.*, 240, is 182.4 mJoule, which has exceeded the given energy budget constraint. Therefore the task set is infeasible.

Different from the above schedule, if we reserve the recovery jobs to satisfy the window-constraints first, we can still meet the original (m,k)-constraints with opportunities to reduce the speeds of the mandatory jobs, thereby meeting the given energy budget constraint while preserving the reliability. This is illustrated in Figure 1(c). As can be seen, in this case the mandatory jobs of τ_1 and τ_3 are determined to satisfy the window constraints of $3/4$ and $2/5$ first. Then according to the aforementioned mapping relationship they can meet their original (m,k)-constraints of $(3,5)$ and $(2,8)$, respectively. Note that in this particular example although there is enough space for reserving recovery jobs for task τ_2 as well, we still prefer to determine its mandatory jobs using E^R-pattern based on its original (m,k)-constraint due to the need of leaving more space to facilitate speed scaling for the mandatory jobs of tasks τ_1 and τ_3. Based on the above configuration, with the aforementioned flexibility of window-constraints in preserving reliability in each separate window, we only need to reserve one recovery job for each window under consideration. For convenience here we assume the last optional job within each separate window is reserved as the shared recovery job for all mandatory jobs in the same window, as shown in Figure 1(c). It is not hard to see that in this case after reserving recovery jobs for tasks τ_1 and τ_3, there is still some space that could be used to scale the speeds of the mandatory jobs. Since both τ_1 and τ_3 have already had recovery jobs reserved for them, we can scale the speeds of all mandatory jobs in them arbitrarily, which will be very helpful in keeping the overall energy consumption under control, *i.e.*, within the given energy budget constraint.

After the mandatory/recovery job patterns are determined in Figure 1(c), we can apply DVFS to scale the speeds of all mandatory jobs within the windows with recovery jobs reserved for them to be as close to the energy-efficient speed as possible, which is shown in Figure 1(d). It is not hard to see that in this case all mandatory jobs of τ_1 and τ_3 got chance to reduce their speeds. As a result the total worst case energy consumption before time 240 is 140.9 mJoule, which is within the energy budget before time 240, therefore feasible. Moreover, under the new speed schedule, the reliability of each window W_{il} of task τ_i should be calculated as:

$$\begin{aligned}\gamma(W_{il}) &= \prod_{p=(l-1)k_i+1}^{(l-1)k_i+k_i} \{\gamma(J_{ip},s_{ip}) \mid \pi_{ip}=1\} \\ &+ \sum_{p=(l-1)k_i+1}^{(l-1)k_i+k_i} \{ \prod_{q=(l-1)k_i+1}^{(l-1)k_i+k_i} \{\gamma(J_{iq},s_{iq}) \mid \pi_{iq}=1,\ q\neq p\} \\ &\times\ (1-\gamma(J_{ip},s_{ip}))\times\gamma(J_{ip},s_{max})\} \qquad (11)\end{aligned}$$

Based on Equation (11), the reliability of each window in task τ_1 will be calculated as $\gamma(W_{11}) = \gamma(W_{12}) = 0.9999999218$, $\gamma(W_{13}) = 0.9999995614$, $\gamma(W_{14}) = 0.9999999931$. Since none of them is lower than that calculated using Equation (10), the reliability of task τ_1, *i.e.*, $\gamma(\tau_1)$, under the new speed schedule will not be lower than its original reliability with all mandatory jobs executed under the maximal speed. Similarly, we can verify that the reliabilities of tasks τ_2 and τ_3, *i.e.*, $\gamma(\tau_2)$ and $\gamma(\tau_3)$, under the new speed schedule are not lower than their original reliabilities under the maximal speed, either. Thus the reliability of the whole system is preserved. In addition, the system QoS based on Equation (8) in this case is improved to 0.5833224976, which is 20.7% higher than that in Figure 1(b) (and that without energy/reliability management in Figure 1(a) as well).

Note that in the above example in Figure 1(d), tasks τ_1 and τ_3 have their own recovery jobs reserved separately and no recovery jobs are shared between them. In practice, some recovery jobs from different tasks could overlap with each other in time and it is possible to share some of them across different tasks to reduce the energy further. For example, two recovery jobs in Figure 1(d), *i.e.*, $J_{1C}(R)$ and $J_{35}(R)$, from τ_1 and τ_3, respectively, can actually be merged into one "shared" recovery job across tasks τ_1 and τ_3, leaving more space for other mandatory jobs to have their speeds reduced further to save more energy. More details for the implementation of it could be found in [26].

From the above example, it is easy to see that it is very promising to determine the job/recovery patterns in such a way that the task set satisfies the window-constraint first which could be transferred to the corresponding original (m,k)-constraint automatically. In [26], more advanced techniques are also introduced to determine the recovery jobs that could be shared most efficiently. However, even with such kind of techniques the system might not be able to accommodate recovery jobs for all tasks yet. Therefore, how to select the subset of tasks that can be managed with reliability/energy will be an important issue. Regarding that, in next section we will propose a method based on "branch-and-bound" to choose the proper subset of tasks that could be managed with reliability/energy.

5. The General Algorithm

In this section, we will introduce our general scheduling algorithm. Our algorithm consists of two stages: an off-line stage and an on-line stage.

The goal for the off-line stage is to determine the (shared) recovery jobs for the tasks and set up the static processor speed schedule for the mandatory jobs such that the task set is feasible under the given energy budget constraint and the

Algorithm 1 Reserving recovery jobs for the task set

```
1: Input: task set $\mathcal{T}$ with mandatory jobs determined by $E^R$-pattern (with no recovery job(s) reserved for any task yet);
2: Output: task set $\Gamma$=$\Omega\cup\Theta$, where $\Omega$ is the subset of tasks in $\mathcal{T}$ with recovery jobs reserved for each task and $\Theta$ is the subset of tasks in $\mathcal{T}$ with no recovery jobs;
3: $\Omega = \emptyset$; $\Theta = \mathcal{T}$; $\Gamma$=$\Omega\cup\Theta$;
4: Sort the tasks in $\Theta$ according to non-increasing order of $\frac{m_i C_i}{k_i P_i}, i = 1,..,n$;
5: $\bar{E}$ = The given hard energy budget constraint of the system ;
6: $\widetilde{QoS}_b$ = The $\widetilde{QoS}_{sys}$ of task set $\mathcal{T}$ based on Equation (8) under original $(m,k)$-constraints;
7: Recovery-Reservation ($\Omega$, $\Theta$, $\widetilde{QoS}_b$, $\Gamma$);
8: Recompute the total energy consumption $E$ of task set $\Gamma$ based on line 27;
9: if $\Gamma$ is non-schedulable with $LPEDF^R$ or $E > \bar{E}$ then
10:     Output: Task Set Infeasible!
11: else
12:     Output ($\Gamma$);
13: end if
14:
15: FUNCTION Recovery-Reservation($\Omega$, $\Theta$, $\widetilde{QoS}_b$, $\Gamma$)
16: for each task $\tau_i \in \Theta$ do
17:     Re-determine the mandatory jobs of $\tau_i$ using E-pattern based on the window-constraint that can be transferred to its original $(m,k_)$-constraint;
18:     Remove $\tau_i$ from $\Theta$;
19:     Add $\tau_i$ to $\Omega$;
20:     Determine the recovery jobs of the tasks in $\Omega$ by reserving the last optional job within each separate window as the share recovery job;
21:     Apply the $LPEDF^R$ algorithm from [32] to the task set $\Omega\cup\Theta$;
22:     if $\Omega\cup\Theta$ is schedulable with $LPEDF^R$ then
23:         Get the simulation trace based on the speed schedule generated by $LPEDF^R$;
24:         Merge the recovery jobs for the tasks in $\Omega$ according to Algorithm 2 in [26];
25:         Re-apply the $LPEDF^R$ algorithm from [32] to the task set $\Omega\cup\Theta$ with recovery jobs merged;
26:         Compute the new $\widetilde{QoS}_{sys}$ of the updated task set $\Omega\cup\Theta$ based on Equation (8);
27:         Recompute the total energy consumption $E$ of task set $\mathcal{T}$ as $E = \sum_{J_i\in\Omega\cup\Theta}\{(\alpha\times(s_i)^m+P_{nd})\times\frac{C_i}{s_i}\}+\sum_{J_{<x>}(R)\in\tilde{\mathcal{R}}}\{(\alpha\times(s_{max})^m+P_{nd})\times C_{<x>}\times(1-\prod\{\gamma(J_{ip},s_{ip})\mid J_{ip}\in W_y$ managed by $J_{<x>}(R)$ and $\pi_{ip}=1\})$ $+P_{idle}\times(H-\sum_{J_i\in\Omega\cup\Theta}\{\frac{C_i}{s_i}\})$, where $s_i$ is the scaled speed of any mandatory job $J_i$ under $LPEDF^R$.
28:         if $E \leq \bar{E}$ and $\widetilde{QoS}_{sys} > \widetilde{QoS}_b$ then
29:             $\widetilde{QoS}_b = \widetilde{QoS}_{sys}$;
30:             $\Gamma$=$\Omega\cup\Theta$;
31:         end if
32:         Recovery-Reservation ($\Omega$, $\Theta$, $\Gamma$, $\widetilde{QoS}_b$);
33:     else
34:         Restore the mandatory jobs of $\tau_i$ to the original ones based on E-pattern (with no recovery job(s)) and put it back to $\Theta$;
35:     end if
36: end for
```

expected QoS of the system is maximized while the system reliability is preserved. One essential part of it is to determine the mandatory jobs for the tasks and reserve recovery jobs for them based on the window-constraints (that can be transferred to the corresponding original (m,k)-constraints) if possible. However, since the system might not be able to accommodate recovery jobs for all tasks, how to select the subset of tasks that can be managed with reliability/energy is not a trivial problem. Meanwhile, since recovery jobs also consume part of the processor utilization, reserving recovery jobs for too many tasks might leave much less space to scale the speeds of the mandatory jobs, therefore causing the overall energy consumption to go beyond the hard energy budget constraint. On the other hand, in order to preserve the system reliability, tasks with no recovery jobs reserved for them should not have their speeds scaled [53]. Then the problem becomes how to select the subset of tasks to be managed with recovery jobs to achieve the best energy efficiency. In [53], it is shown that even without consideration of QoS and energy constraint this problem is NP-hard. Although some heuristics [53] are proposed for hard real-time systems regarding the selection of tasks, they are not applicable any more for soft real-time systems with (m,k)-constraints. In order to solve the problem, in this section, we propose a "branch-and-bound" method to divide the task set $\mathcal{T}$ into two parts: the subset Ω in which the tasks will be managed with recovery jobs and the subset Θ in which the tasks will not be managed with recovery jobs. The details are presented in Algorithm 1.

As can be seen in Algorithm 1, by applying the branch-and-bound strategy, our approach determines task by task if the mandatory jobs of each task should be based on the original (m,k)-constraint (with no recovery jobs reserved) or based on the window-constraint (with recovery jobs reserved according to Algorithm 1 from [26]). When Algorithm 1 is finished, it is possible to reach certain hybrid configuration in which the tasks in Ω are partitioned based on window-constraints with recovery jobs reserved in each separate window, while the tasks in Θ are still partitioned based on the original (m,k)-constraints with no recovery jobs reserved for them. Moreover, to make the mandatory/recovery workload of the whole task set distribute more evenly (to facilitate the speeds scaling), we let the tasks in Ω be partitioned under E-pattern while the tasks in Θ be partitioned under E^R-pattern [21]. And the resulting configuration should be the one with the maximal expected QoS and with energy consumption not exceeding the energy budget constraint $\bar{E}$.

The branch-and-bound strategy applied in Algorithm 1 is commonly used to solve the optimization problem. Although it is essentially an exhaustive search strategy with worst case time complexity of $O(2^n)$, since it can be done offline, we

Algorithm 2 Window Reliability Management

```
1:  Input: Merged recovery job set $\tilde{\mathcal{R}}$ output by Algoirthm 2 in [26];
2:  for each recovery job $J_{<x>}(R) \in \tilde{\mathcal{R}}$ do
3:      $\mathcal{W} = \emptyset$;
4:      for each Window $W_y$ (jointly) managed by $J_{<x>}(R)$ do
5:          $\mathcal{W} = \mathcal{W} \cup W_y$;
6:      end for
7:      Let $\hat{\gamma}(\mathcal{W})$ be the original reliability of $\mathcal{W}$, calculated based on Equation (10), with
        all mandatory jobs in it executed under $s_{max}$;
8:      Let $\gamma(\mathcal{W})$ be the reliability of $\mathcal{W}$, calculated based on Equation (11), with all
        mandatory jobs in it executed under their current speed schedule;
9:      if $\gamma(\mathcal{W}) < \hat{\gamma}(\mathcal{W})$ then
10:         repeat
11:             Increase the speed(s) of the job(s) with the lowest speed in $\mathcal{W}$ to be the next
                higher level speed in the whole speed schedule;
12:             Re-calculate $\gamma(\mathcal{W})$ under the new speed schedule;
13:         until $\gamma(\mathcal{W}) \geq \hat{\gamma}(\mathcal{W})$
14:     end if
15: end for
```

can afford to have it with task sets with not very large size n of task sets. Moreover, to improve its efficiency, the task set is initially sorted according to the values of $\frac{m_iC_i}{k_iP_i}, i = 1,..,n$ (line 4) because the higher the value, the higher the possibility that the task set becomes unschedulable when the task has recovery jobs reserved based on window-constraint and its job speeds are reduced accordingly.

Note that in lines 17-23 of Algorithm 1, a variation of Yao's LPEDF Algorithm [44], *i.e.*, $LPEDF^R$ [32], is applied to determine the speeds of the mandatory jobs and to compute their expected energy consumption within each iteration. The details of it could be found in [32].

By running Algorithm 1, after the subset Ω of a given task set $\mathcal{T}$ is determined, the static speed schedule for the mandatory jobs from all tasks in it is also available (generated by the $LPEDF^R$ algorithm in line 22). At the same time, the speeds of all mandatory jobs from tasks in Θ should be set as the maximum speed s_{max}.

It does not escape our attention that, similar to the work in [26], in some occasional cases, for example, when the length of a window managed by a recovery job is very long and the scaled speed(s) of the job(s) belonging to the window is (are) extremely low, it is possible that the reliability of the window under consideration could drop below its original reliability (under the maximal speed, calculated by

Algorithm 3 Online algorithm

```
 1:  Upon job arrival:
 2: Run jobs in the ready queue according to EDF scheme with their predetermined
    speed;
 3:
 4:  Upon job completion (assuming $J_i$ is just completed at $f_i$):
 5: if the execution of $J_i$ is found to have failed then
 6:     Let $J_i$'s original non-shared recovery job in $\mathcal{R}$ be $J_w(R)$
 7:     Invoke the shared recovery job in $\tilde{\mathcal{R}}$ which contains $w$ in its index vector and set
        its actual execution time to be $C_i$ [26];
 8:     Increase the speeds of all the other mandatory jobs after the faulty job within all
        windows (jointly) managed by shared recovery job to $s_{max}$;
 9: else
10:     if $J_i \in \Omega$ and all windows sharing the same shared recovery job as $J_i$'s have got
        enough number of jobs belonging to them completed successfully then
11:         Drop the shared recovery job of $J_i$;
12:     end if
13:     if the ready queue is empty then
14:         Compute the latest starting time, i.e., $T_{LS}(\mathcal{J}_n)$, for the upcoming manda-
            tory/recovery job set $\mathcal{J}_n$;
15:         if $(T_{LS}(\mathcal{J}_n) - f_i) > T_{th}$ then
16:             Shut down the processor and set up the wake-up timer to be $(T_{LS}(\mathcal{J}_n) - f_i)$;
17:         end if
18:     end if
19: end if
```

Equation (10)). For such kind of windows, to preserve the reliability, we need to revise the job speed(s) in them to increase their reliability correspondingly. The details are provided in Algorithm 2.

As shown in Algorithm 2, for each shared recovery job $J_{<x>}(R)$ in $\tilde{\mathcal{R}}$ [26], we scan all windows (jointly) managed by $J_{<x>}(R)$ and put them in a temporary pool $\mathcal{W}$. If the reliability of all mandatory jobs in $\mathcal{W}$ under the current speed schedule is less than their original reliability under the maximal speed s_{max} (line 9), we need to increase the lowest job speeds in $\mathcal{W}$ to certain level such that their original reliability could be preserved [2] (line 10-13). The time complexity of Algorithm 2 depends on the number of shared recovery jobs in $\tilde{\mathcal{R}}$, the maximal possible number of tasks jointly managed by any shared recovery job, and the number of mandatory jobs in each window, which are $\max\{\frac{H}{\frac{m_i+k_i}{2}P_i}\}$, n and $\max\{m_i\}$, respectively. From Algorithm 2, the time complexity of it should be $O(\frac{H}{\frac{m_i+k_i}{2}P_i}nm_i)$, which is pseudo-polynomial in the number of tasks n and still suitable for an offline algorithm due to its low overhead in practice.

Moreover, during the online stage, if any mandatory job encountered transient fault, since its shared recovery job $J_{<x>}(R)$ will be consumed by the faulty job, all the other mandatory jobs after the faulty job within all windows (jointly) managed by $J_{<x>}(R)$ needs to have their speeds raised to the maximal speed s_{max} in order to preserve the original reliability, which is reflected in the online Algorithm 3 (line 8) as well.

Based on the adjusted speed schedule output by Algorithm 2, the mandatory jobs can be scheduled according to the EDF scheme during the online stage.

During the online stage, as shown in Algorithm 3, a job ready queue will be maintained. Upon arrival, a mandatory job determined during the off-line stage is inserted into the ready queue. Note that a recovery job needs to be inserted into the ready queue to be executed only if some mandatory job within the same window has failed. Otherwise the recovery job will simply be dropped. All jobs in the ready queue will be executed following the EDF scheme. If the current job J_i is found to have failed at its completion time, its shared recovery job in $\tilde{\mathcal{R}}$ [26] will be invoked with actual execution time set to be the worst execution time of J_i and inserted into the ready queue at its arrival time (line 7).

If the execution of J_i is successful, Algorithm 3 determines whether its shared

[2]Note that we can also set the target reliability for the mandatory jobs in $\mathcal{W}$ higher if necessary, which is beyond the scope of this paper.

recovery job, if any, should be dropped depending on the number of successful jobs so far in each window sharing it (lines 10-12). At the same time, if the job ready queue is empty, Algorithm 3 determines whether it is necessary to shut down the processor or to keep it idle depending on if the predicted idle interval length, with future mandatory/recovey jobs delayed to $T_{LS}(\mathcal{J}_n)$, is larger than the shut down threshold T_{th} (lines 13-18).

One important part to the success of Algorithm 3 is to compute the latest starting time $T_{LS}(\mathcal{J}_n)$ for the upcoming mandatory/recovery job set when the ready queue is empty (line 14), which is also the main complexity of Algorithm 3. The latest starting time could be computed using an advanced version of Theorem 1 from [28], whose online time complexity is $O(N'M')$, where M' is the number of mandatory/recovery jobs with arrivals before the earliest deadline of the upcoming mandatory jobs and N' is the total number of jobs arriving within the interval between the arrival time and the deadline of any job aforementioned. Since N' and M' are usually very small for periodic task sets [27], the complexity above is suitable for online use as well. More details could be found in [26].

6. Evaluation

In this section, we evaluate the performance of our approach by comparing with the existing approaches in literature. Specifically, the performance of four different approaches were studied:

- *NPM* The task sets are partitioned with E-pattern, and all mandatory jobs are always executed with the highest speed.
- *RAEM* The task sets are partitioned with deeply-red pattern to satisfy the given (m,k)-constraints. Then the mandatory jobs are scheduled with the approach from [49].
- *ECRA* This is our newly proposed approach in Section 5.
- *$ECRA_{SS}$* This is our newly proposed approach in Section 5 enhanced with the shared slack time reclaiming technique in [26].

The periods of the real-time task sets were randomly chosen in the range of $[30ms, 100ms]$. The worst case execution time (WCET) was set to be uniformly distributed and the m_i and k_i for the user defined (m,k)-constraints were also randomly generated such that k_i is uniformly distributed between 2 to 10, and $m_i < k_i$.

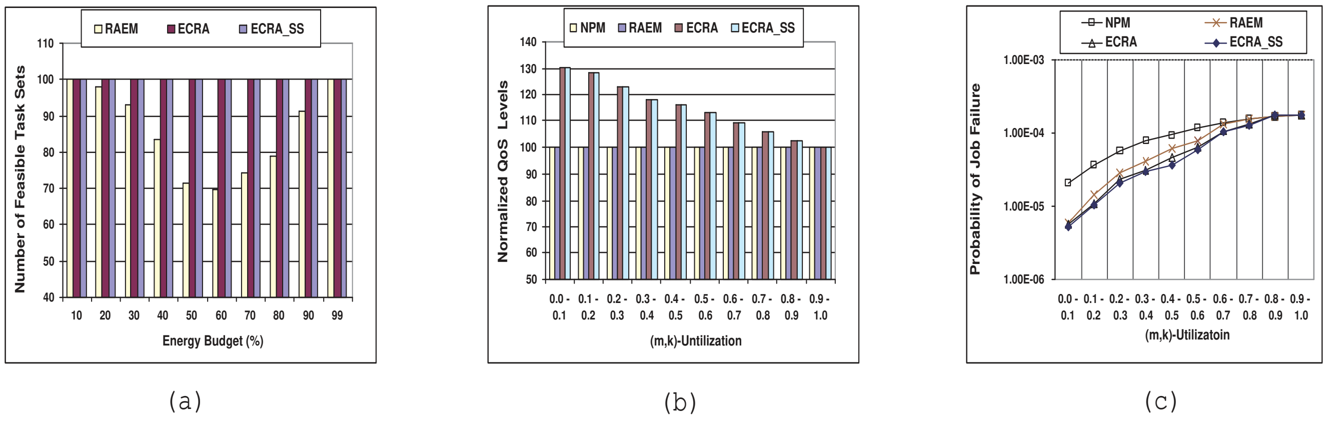


Figure 2: (a) The feasibility comparison for the different approaches under given hard energy budget constraint; (b) The QoS comparison for the different approaches; (c) The comparison on the probability of job failure (PoJF) for the different approaches.

For the processor model we adopted a widely used embedded processor model, *i.e.*, the Freescale MPC8536E [38]. The default value of P_{nd} was set to be 10% of the maximum speed-dependent power. The shutdown threshold T_{th} is set to be 1 ms.

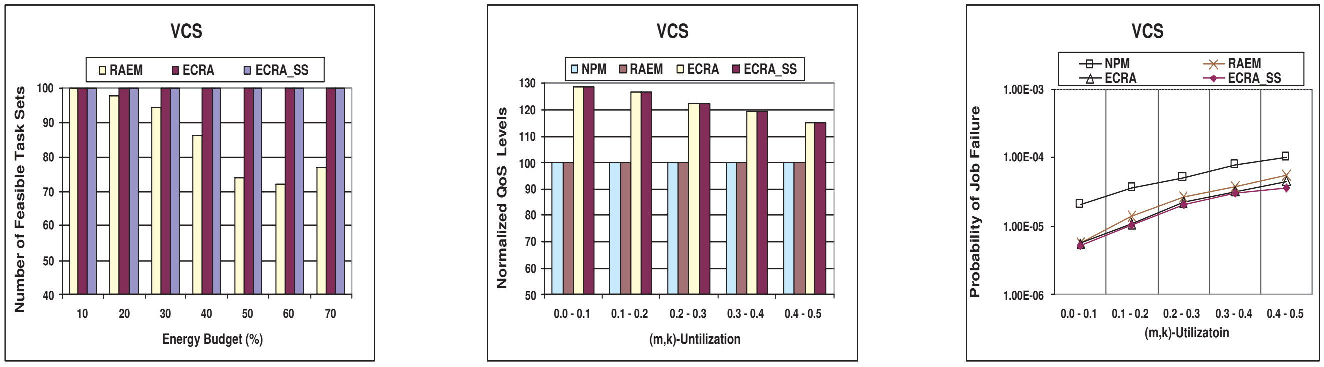
For the fault and recovery model we adopted the same model as used in [54], *i.e.*, the transient faults are assumed to follow the Poisson distribution with an average fault rate of $\lambda_0 = 10^{-6}$ at the maximum speed s_{max} (and corresponding supply voltage). We assume the sensitivity value $d = 3$. That is, the average fault rate is 1000 times higher at the lowest speed s_{min} [54].

The simulations were performed using a discrete event simulator in a desk top computer with 4.7Ghz processor and Windows 10 OS. During the simulation, whenever an activated job has reached its deadline with pending/uncompleted workload, it is regarded as having missed its deadline and will be discarded. In particular, when simulating with the (m,k) model, any mandatory job activation not completing by its deadline will be aborted at the deadline and any optional job will be dropped to save energy.

Firstly, we inspect the feasibility of the different approaches under different energy budget constraints. We assume the maximal energy budget constraint to be 99% of the energy consumption by *NPM* and then scale the energy budget constraint with a step reduction of about 10%. Based on it we checked the feasibility of the task sets by the different approaches. Note that since under this scenario

NPM is not feasible in nearly all cases, we just show the feasibility of *RAEM* and *ECRA* as well as $ECRA_{SS}$. The results (normalized to that by *ECRA*) are shown in Figure 2(a). As can be seen, when the energy budget is sufficiently large, *i.e.*, larger than 90%, usually the task set could be feasible by all of them because their total energy consumptions estimated based on the statically scaled speed schedules would seldom exceed the energy budget constraint in this case. So the total number of task sets feasible by them are close to each other. However, when the energy budget constraint became tighter, for example, between 40% and 80%, the feasibilities of *ECRA* and $ECRA_{SS}$ are much better than that of *RAEM*. This is because, by reserving recovery jobs at each separate window frame (to satisfy the (m,k)-constraints through the corresponding window-constraints) and sharing the recovery jobs whenever possible to do so, *ECRA* and $ECRA_{SS}$ can greatly reduce the space required f or reserving recovery jobs. As a result, there was more space in them to help scale the speeds of the mandatory jobs. Meanwhile, the schedulability of the deeply-red pattern adopted by *RAEM* is not as good as the E-pattern adopted in our approach [21], which also limited the feasibility of *RAEM* to some extent. Moreover, as a side effect, generally *ECRA* also allowed more tasks from the task set to have recovery jobs reserved for them. So it could provide more freedom for the mandatory jobs of the whole task set to scale their speeds more aggressively. As a result, the total energy consumption required by *ECRA* and $ECRA_{SS}$ could be much lower than that by *RAEM*, therefore allowing better feasibility when the energy budget constraint is relatively tighter. Also it is not surprising to see that when the energy budget is extremely low, most task sets are not feasible by any of the approaches. So under this scenario their feasibilities are also close to each other.

Next, we inspect the maximal QoS levels each approach can provide. This time we set the energy budget constraint to be 100% of the energy consumption by *NPM*. For simplicity, we assumed all tasks in Equation (8) were assigned the equal weights, *i.e.*, $\omega_i = \frac{1}{n}$ for any task $\tau_i \in \mathcal{T}$. To investigate the performance of different approaches under different workload, we divided the total (m,k)-utilization, *i.e.*, $\sum_i \frac{m_i C_i}{k_i P_i}$, into intervals of length 0.1. To reduce the statistical errors, we required that each interval contain at least 20 task sets schedulable with E-pattern, or until at least 5000 task sets within each interval had been generated. The results (normalized to that by *NPM*) are shown in Figure 2(b). From Figure 2(b), we can see that the newly proposed approaches, *i.e.*, *ECRA* and $ECRA_{SS}$ can achieve much better QoS levels than the previous approaches. Compared with *RAEM* and *NPM*, the maximal QoS improvement could be nearly 30%. This is because, different from *RAEM* and *NPM* which could only provide a minimum set



(a) (b) (c)

Figure 3: (a) The feasibility comparison for the different approaches under given hard energy budget constraint; (b) The QoS comparison for the different approaches; (c) The comparison on the probability of job failure (PoJF) for the different approaches.

of jobs that "just" satisfied the (m,k)-constraints, *ECRA* and $ECRA_{SS}$, by adopting more adaptive recovery job reservation and sharing techniques, could utilize the time space more efficiently. Therefore it could generally accommodate more valid jobs in its schedule, generating better QoS levels.

Finally, with system reliability in mind, we also inspected the *probability of job failure* (denoted as PoJF) of the different approaches, which is defined as the ratio of the number of jobs failed over the total number of jobs executed. The results in different (m,k)-utilization intervals are shown in Figure 2(c). As seen, in most utilization intervals, the PoJF of *RAEM* is lower than that by *NPM*. The effect is especially obvious when the utilization is not very high. That conforms to the conclusion in [52] that the reservation of recovery jobs could generally help reduce the probability of job failure. The average PoJFs of *ECRA* and $ECRA_{SS}$ are even lower than that by *RAEM*. This is mainly because, by reserving shared recovery jobs based on window-constraints (that can be transferred to the corresponding original (m,k)-constraints) first, the mandatory jobs of more tasks could have (shared) recovery jobs reserved for them when compared with *RAEM*. Moreover, it is noted that the average PoJs of $ECRA_{SS}$ is stil little lower than that by *ECRA* mainly because its capability of sharing the slack time among different mandatory jobs such that more jobs could use the slack time to reserve recovery jobs for them when necessary.

6.1. Evaluations with Real World Benchmark

Next, we tested our conclusions in a more practical environment. The test is based on a real world benchmark: VCS (Vehicle Control System) [19]. The timing parameters such as the deadlines, periods, and execution times were adopted from the practical application directly [19]. For the processor model used in this group of experiments, we adopt the Samsung Exynos 4210 processor [9] to test the energy saving performance of our approaches. The Samsung Exynos 4210 processor can operate with five operating frequencies: (0.3, 0.6, 0.8, 1.008, 1.2) GHz with power consumption of (0.1018, 0.2738, 0.4607, 0.7548, 1.0675) Watt, respectively [1, 37].

Similar to the experiments on synthesized task sets, we also performed three sets of experiments to check the feasibility, QoS. and probability of job failure of the different approaches. The results are shown in Figure 3.

From Figure 3, the above analysis on synthesized task sets also conforms to the experimental results on the real world benchmark VCS as well. For example, *ECRA* and $ECRA_{ss}$ still have much better feasibility and QoS levles than the other approaches. As shown in Figure 3(a), compared with *RAEM*, when the energy budget constraint is between 40% and 80%, the feasibilities of *ECRA* and $ECRA_{ss}$ are much better than *RAEM*. Meanwhile, in terms of QoS, *ECRA* and $ECRA_{ss}$ can also achieve much better QoS levels than the previous approaches with maximal QoS improvement of nearly 30%. Also, for probability of job failure (PoJF), the PoJF of *RAEM* is lower than that by *NPM* and the average PoJFs of *ECRA* and $ECRA_{ss}$ are even lower than that by *RAEM* for the same reasons as stated above.

Overall, the experimental results for synthesized systems as well as real world application have clearly demonstrated the effectiveness of our approaches in maximizing the feasibility and QoS while satisfying the (m,k)-constraints and preserving the system reliability under hard energy budget constraint.

7. Conclusions

In this paper, we explored maximizing the feasibility and QoS for (m,k)-firm real-time embedded systems while satisfying the reliability requirement under given hard energy budget constraint. Regarding that, we proposed a reliability-aware energy-constrained scheduling scheme which implemented recovery space sharing for real-time jobs in an adaptive way based on the mandatory/optional job partitioning strategy. Through extensive simulations, our evaluation results demonstrated that the proposed techniques significantly outperformed the previous research in feasibility and QoS while preserving the system reliability under given hard energy budget constraint. Moreover, the proposed work has also addressed some insufficiency in [26] in terms of preserving the system reliability.

Acknowledge*

This work is supported in part by NSF under projects HRD-2135345 and ECCS-2302651.

References

[1] Samsung exynos 4210 processor user's manual, Samsung Corporation, October 2012.

[2] T. A. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. *RTSS*, 2005.

[3] G. C. Buttazzo and M. Caccamo. Minimizing aperiodic response times in a firm real-time environment. *IEEE Transactions on Software Engineering*, 25(1):22–32, 1999.

[4] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, 2007.

[5] M. Chetto. Graceful overload management in firm real-time systems. *Journal of Information Technology and Software Engineering*, 5(3):1–3, 2015.

[6] T. Cucinotta and L. Palopoli. Qos control for pipelines of tasks using multiple resources. *IEEE Transactions on Computers*, 59(3):416–430, 2010.

[7] A. Das, A. Kumar, and B. Veeravalli. Energy-aware task mapping and scheduling for reliable embedded computing systems. *ACM Trans. Embed. Comput. Syst.*, 13(2s):72:1–72:27, Jan. 2014.

[8] P. Dobias. *Online Fault Tolerant Task Scheduling for Real-Time Multiprocessor Embedded Systems*. PhD thesis, 10 2020.

[9] L.-T. Duan, B. Guo, Y. Shen, Y. Wang, and W.-L. Zhang. Energy analysis and prediction for applications on smartphones. *Journal of Systems Architecture*, 59(10, Part D):1375 – 1382, 2013.

[10] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *Micro, IEEE*, 24(6):10–20, 2004.

[11] O. Gettings, S. Quinton, and R. I. Davis. Mixed criticality systems with weakly-hard constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 237–246, 2015.

[12] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computes*, 44:1443–1451, Dec 1995.

[13] J.-J. Han, M. Lin, D. Zhu, and L. Yang. Contention-aware energy management scheme for noc-based multicore real-time systems. *Parallel and Distributed Systems, IEEE Transactions on*, 26(3):691–701, March 2015.

[14] Q. Han, L. Niu, G. Quan, S. Ren, and S. Ren. Energy efficient fault-tolerant earliest deadline first scheduling for hard real-time systems. *Real-Time Syst.*, 50(5-6):592–619, Nov. 2014.

[15] M. A. Haque, H. Aydin, and D. Zhu. Energy-aware standby-sparing for fixed-priority real-time task sets. *Sustainable Computing: Informatics and Systems*, 6:81 – 93, 2015.

[16] INTEL-XSCALE. http://developer.intel.com/design/xscale/. 2003.

[17] B. P. R. J. J. Srinivasan, A. S.V. and C.-K. Hu. Ramp: A model for reliability aware microprocessor design. *IBM Research Report, RC23048*, 2003.

[18] H. Kooti, N. Dang, D. Mishra, and E. Bozorgzadeh. Energy budget management for energy harvesting embedded systems. In *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 320–329, Aug 2012.

[19] J. Li, Y. Song, and F. Simonot-Lion. Providing real-time applications with graceful degradation of qos and fault tolerance according to (m,k)-firm model. *Industrial Informatics, IEEE Transactions on*, 2(2):112–119, May 2006.

[20] Z. Li, S. Ren, and G. Quan. Energy minimization for reliability-guaranteed real-time applications using dvfs and checkpointing techniques. *Journal of Systems Architecture*, 61(2):71–81, Feb. 2015.

[21] Linwei Niu and Gang Quan. Energy minimization for real-time systems with (m,k)-guarantee. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(7):717–729, July 2006.

[22] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 17(2):46–61, 1973.

[23] J. Liu. *Real-Time Systems*. Prentice Hall, NJ, 2000.

[24] M. Neukirchner, S. Stein, H. Schrom, J. Schlatow, and R. Ernst. Contract-based dynamic task management for mixed-criticality systems. In *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, pages 18–27, 2011.

[25] L. Niu. Energy efficient scheduling for real-time systems with qos guarantee. *Journal of Real-Time Systems*, 47(2):75–108, 2011.

[26] L. Niu. Reliability-aware energy-efficient scheduling for (m, k)-constrained real-time systems through shared time slots. *Microprocessors and Microsystems*, 77:103110, 2020.

[27] L. Niu and W. Li. Energy-efficient scheduling for embedded real-time systems using threshold work-demand analysis. *Journal of Circuits, Systems and Computers*, 26:1750091, 02 2017.

[28] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. *CASES'04*, Sep 2004.

[29] L. Niu and G. Quan. Leakage-aware scheduling for embedded real-time systems with (m, k)-constraints. *International Journal of Embedded Systems*, 5(4):189–207, 2013.

[30] L. Niu and G. Quan. Peripheral-conscious energy-efficient scheduling for weakly hard real¨ctime systems. *International Journal of Embedded Systems*, 7(1):11–25, 2015.

[31] L. Niu and D. B. Rawat. Energy-constrained standby-sparing for weakly hard real-time systems. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 257–269, 2020.

[32] L. Niu and J. Xu. Improving schedulability and energy efficiency for window-constrained real-time systems with reliability requirement. *Journal of Systems Architecture*, 61(5):210–226, May 2015.

[33] D. K. Pradhan, editor. *Fault-tolerant Computing: Theory and Techniques; Vol. 2*. Prentice-Hall, Inc., USA, 1986.

[34] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *RTSS*, pages 79–88, 2000.

[35] P. Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantee. *IEEE Trans. on Paral. and Dist. Sys.*, 10(6):549–559, Jun 1999.

[36] A. Roy, H. Aydin, and D. Zhu. Energy-aware standby-sparing on heterogeneous multicore systems. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017.

[37] Samsung-Exynos-4210-Processor. file/sys/devices/system/cpu/cup0/cpufreq.

[38] F. Semiconductor. Mpc8536e powerquicc iii integrated processor hardware specifications, document number: Mpc8536eec rev. 7, 07/2015. *https://www.nxp.com/docs/en/data-sheet/MPC8536EEC.pdf*, 2015.

[39] Y. Sun and M. D. Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Trans. Embed. Comput. Syst.*, 16(5s):171:1–171:19, Sept. 2017.

[40] A. Taherin, M. Salehi, and A. Ejlali. Reliability-aware energy management in mixed-criticality systems. *IEEE Transactions on Sustainable Computing*, 3(3):195–208, July 2018.

[41] G. v. d. Bruggen, K. Chen, W. Huang, and J. Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 303–314, Nov 2016.

[42] Y. wen Zhang, H. zhen Zhang, and C. Wang. Reliability-aware low energy scheduling in real time systems with shared resources. *Microprocessors and Microsystems*, 52:312 – 324, 2017.

[43] R. West, Y. Zhang, K. Schwan, and C. Poellabauer. Dynamic window-constrained scheduling of real-time streams in media servers. *IEEE Trans. on Computers*, 53(6):744–759, June 2004.

[44] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *AFCS*, pages 374–382, 1995.

[45] S. Yari-Karin, R. Siyadatzadeh, M. Ansari, and A. Ejlali. Passive primary/backup-based scheduling for simultaneous power and reliability management on heterogeneous embedded systems. *IEEE Transactions on Sustainable Computing*, 8(1):82–93, 2023.

[46] Y. Zhang, K. Chakrabarty, and V. Swaminathan. Energy-aware fault tolerance in fixed-priority real-time embedded systems. In *ICCAD*, 2003.

[47] B. Zhao, H. Aydin, and D. Zhu. On maximizing reliability of real-time embedded applications under hard energy constraint. *IEEE Trans. Industrial Informatics*, pages 316–328, 2010.

[48] B. Zhao, H. Aydin, and D. Zhu. Generalized reliability-oriented energy management for real-time embedded applications. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 381–386, 2011.

[49] B. Zhao, H. Aydin, and D. Zhu. Energy management under general task-level reliability constraints. In *Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, RTAS '12, pages 285–294, Washington, DC, USA, 2012.

[50] B. Zhao, H. Aydin, and D. Zhu. Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. *ACM Trans. Des. Autom. Electron. Syst.*, 18(2):23:1–23:21, Apr. 2013.

[51] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. *ACM Trans. Embed. Comput. Syst.*, 10:26:1–26:27, January 2011.

[52] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *ICCAD*, 2006.

[53] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. *Computers, IEEE Transactions on*, 58(10):1382–1397, 2009.

[54] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *ICCAD*, 2004.