Department of Computer Science, UTSA
Technical Report: CS-TR-2008-007

Energy Management for Periodic Real-Time Tasks with Variable Assurance Requirements*

Dakai Zhu, Xuan Qi
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX, 78249
{dzhu,xqi}@cs.utsa.edu

Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, VA 22030
aydin@cs.gmu.edu

Abstract

Reliability-aware power management (RAPM) *schemes, which consider the negative effects of voltage scaling on system reliability, were recently studied to save energy while preserving system reliability. However, in previous RAPM schemes, real-time tasks are occasionally treated* unfairly *and the selected jobs are determined in greedy fashion. In this paper, we study static* flexible *RAPM schemes for real-time periodic tasks, which consider the* assurance requirements *of tasks and manages a subset of jobs for* every *task accordingly. The problem is shown to be NP-hard in the strong sense and the upper bounds on energy savings are discussed. For a special case of tasks' recovery patterns, a pseudo-polynomial static scheme is proposed. Dynamic schemes that explore dynamic slack for better energy savings and reliability enhancement are also discussed. The schemes are evaluated extensively through simulations. The results show that, compared to the previous RAPM schemes, the new flexible RAPM schemes can guarantee the assurance requirements and provide* fairness *for all tasks, but at the cost of decreased energy savings. However, when combining with dynamic schemes, such cost can be effectively recovered.*

*A preliminary version of this paper appeared in IEEE RTCSA 2007. This work was supported in part by NSF awards CNS-0720651, CNS-0720647 and NSF CAREER Award CNS-0546244.

1 Introduction

Energy has been recognized as a first-class resource in computing systems, especially for battery-operated embedded devices that have limited energy budget. To manage power consumption in such systems, many energy efficient hardware and software techniques have been proposed and power aware computing has recently become an important research area. As a common strategy for saving energy, system components are operated at low-performance (thus low-power) states, whenever possible. For instance, through *dynamic voltage and frequency scaling (DVFS)* [26], the supply voltage and operating frequency of modern processors can be scaled down to save energy.

However, with lower processor operating frequencies (i.e., slower processing speeds), applications will generally take more time and run longer. For real-time systems, where applications may have stringent timing constraints and the consequences of missing a deadline can be catastrophic, special provisions are needed when exploring DVFS for energy savings. In the recent past, exploiting the available static and/or dynamic *slack* in the system, several research studies explored the problem of minimizing energy consumption while meeting all the deadlines for various real-time systems [3, 19, 23].

More recently, the negative effect of DVFS on system reliability due to increased transient fault rates has been studied [32]. With the continued scaling of CMOS technologies and reduced design margins for higher performance, it is expected that, in addition to the systems that operate in electronics-hostile environments (such as those in outer space), practically all digital computing systems will be much more vulnerable to transient faults [9]. Hence, for safety-critical real-time systems (such as satellite and surveillance systems) where reliability is as important as energy efficiency, *reliability-cognizant* energy management becomes a necessity.

There has been some recent work that addressed energy efficiency and system reliability simultaneously [8, 18, 25, 28, 33]. However, most of the previous research either focused on tolerating a fixed number of faults [8, 18, 25, 33] or assumed constant transient fault rate [28].

Reliability-Aware Power Management As an initial study, focusing on tolerating transient faults, we have proposed the concept of *reliability-aware power management (RAPM)* [29]. Taking the negative effects of DVFS on system reliability (due to increased transient fault rates at lower supply voltages [32]) into consideration, for reliability preservation, **the central idea of RAPM schemes is to reserve a portion of available slack to schedule one *recovery* for any task to be scaled down before utilizing the remaining slack for energy management**. This is different from the

ordinary power management schemes that exploit *all* the available (dynamic or static) slack for energy savings through DVFS techniques. The recovery jobs are invoked for execution only if their corresponding scaled tasks fail, at the *maximum* processing speed. It has been proved that, with the help of recovery jobs, the RAPM scheme can guarantee to preserve the system reliability while still obtaining energy savings using the remaining slack, regardless of different fault rate increases and scaled processing speeds [29].

The RAPM scheme has been extended to consider multiple real-time tasks sharing a common deadline [30] as well as periodic real-time tasks scheduled by *earliest-deadline-first (EDF)* [31] and *rate-monotonic (RMS)* scheduling algorithms [34]. Although the static *task-level* RAPM schemes for periodic real-time tasks can achieve significant energy savings while preserving system reliability, a few problems remain open.

It was not clear that, instead of *managing* a subset of tasks (i.e., scheduling corresponding recovery tasks and scaling down the execution of *all* their jobs), whether managing a subset of jobs from *every* task could further improve energy savings. Moreover, by selecting tasks for management in greedy fashion, the task-level schemes treat tasks *unequally/unfairly*. Considering that the reliability of any scaled job is actually *enhanced* with the help of the scheduled recovery job [29], the jobs of selected tasks will have enhanced reliability while the reliability for the jobs of other tasks remains unchanged. For applications where it is important to improve the *quality of assurance* for *all* tasks *simultaneously* (e.g., satellite signal processing in GPS applications), excluding any task from RAPM schemes may not be desired.

In this paper, considering different quality of assurance requirements (e.g., reliability enhancements) of individual tasks, we study preemptive EDF-based *flexible* RAPM schemes for periodic real-time tasks. Compared to the previous task-level RAPM schemes, the new schemes manage a subset jobs for *every* task according to individual tasks' *assurance requirements*. Simulation results show the effectiveness of the proposed schemes on guaranteeing the assurance requirements of all tasks while achieving considerable amount of energy savings.

The contributions of this paper are summaried as follows:

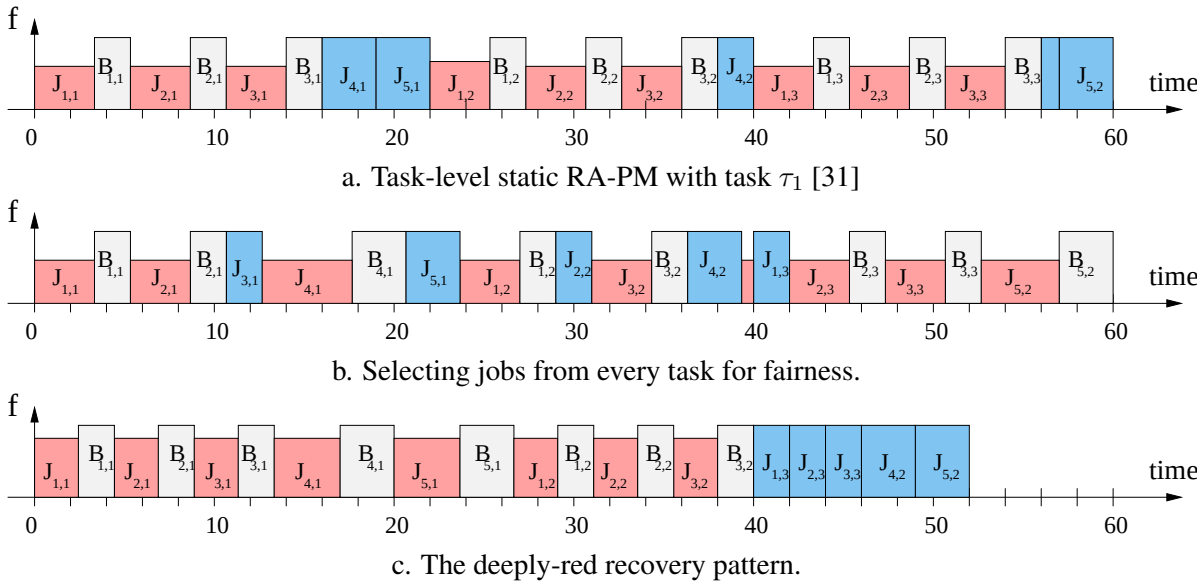
- Establishment of an upper bound of energy savings for static RAPM schemes;
- Based on the idea of skip task model [17], the necessary as well as sufficient conditions for the feasibility test of periodic real-time tasks with variable assurance requirements are presented;
- For tasks with variable assurance requirements, the static RAPM problem is proved to be NP-hard;

- Both static (for a special case) and dynamic RAPM schemes are proposed and evaluated;

We first elaborate on an upper bound of energy savings for static RAPM schemes. For the feasibility of the task set considered, we discuss the necessary condition as well as sufficient condition using tasks' utilization information. Then, based on the idea of the skip task model [17], we formally formulate the RAPM problem and show that the problem is NP-hard in its general setting. For a special pattern of required recovery jobs, we propose one pseudo-polynomial static scheme. Exploring dynamic slack, dynamic RAPM schemes are also discussed to further improve system performance on both energy efficiency and reliability.

In what follows, Section 2 presents a motivational example and Section 3 presents system models. In Section 4, we address the static flexible RAPM problem, which is shown to be NP-hard in the strong sense, and propose a pseudo-polynomial scheme for a special case after presenting the upper bounds on energy savings. Dynamic RAPM schemes are further discussed in Section 5. Simulation results are presented and discussed in Section 6. Section 7 concludes the paper.

2 Motivational Example



a. Task-level static RA-PM with task τ_1 [31]

b. Selecting jobs from every task for fairness.

c. The deeply-red recovery pattern.

Figure 1. Motivational Example

Before formally presenting the problem that will be addressed in this work, we first consider an example. For a task set with five periodic tasks $\{\tau_1(2, 20), \tau_2(2, 20), \tau_3(2, 20), \tau_4(3, 30), \tau_5(3, 30)\}$, where the first number associated with each task is its worst case execution time (WCET) and the

second number is the task's period, the system utilization is 0.5 and the spare CPU capacity (i.e., *static slack*) is 0.5 (i.e. 50%). The slack can be used for both energy and reliability management. In the task-level static RAPM scheme [31], for any task that is selected for management, a *recovery task* will be created with the same timing parameters as the managed task. When the managed task is scaled down using the remaining slack, the recovery task will provide necessary recovery jobs to preserve system reliability.

In the example, although the spare CPU capacity is enough to create a recovery task for every task, doing so leaves no slack for energy management and no energy savings can be obtained. Suppose that, the static task-level RAPM scheme selects three tasks (τ_1, τ_2 and τ_3) for management, after creating the required recovery tasks and scaling down the jobs of the managed tasks [31]. Figure 1a shows the schedule for the interval of $[0, 60]$. In the figures, the X-axis represents time, the Y-axis represents CPU processing speed (e.g., cycles per time units) and the area of the task box defines the amount of work (e.g., number of CPU cycles) needed to execute the task. Here, 30% CPU capacity is used to accommodate the newly created recovery tasks and the remaining spare CPU capacity (which is 20%) is exploited to scale all jobs of the three managed tasks to the frequency of $0.6f_{max}$ (f_{max} is assumed to be the maximum frequency).

Note that, with the recovery tasks, all scaled jobs of the three managed tasks have a recovery job each within their deadlines and system reliability will be preserved [31]. However, such a *greedy* selection of tasks in the previous RAMP scheme does not consider different requirements of individual tasks and the task-level selection may result in *unfairness*. For the case shown in Figure 1a, although the reliability of the three managed tasks is enhanced due to the scheduled corresponding recovery tasks, the reliability for the other two tasks (τ_4 and τ_5) remains *unchanged*. When the overall performance of a real-time system is limited by the task with the lowest quality, such as signal processing for the multiple channels in satellite/GPS/ATR applications [22], improving the quality-of-assurance for *every* task *simultaneously* is necessary.

For the above example, instead of managing only jobs of tasks τ_1, τ_2 and τ_3, we can manage two out of three jobs for these three tasks and one out of two jobs for other two tasks. Through *wise* selection of jobs for each task (e.g., the first two jobs of task τ_1, the first and the third jobs of task τ_2, the second and the third jobs of task τ_3, the first job of task τ_4 and the second job of task τ_5), Figure 1b shows the schedule within the interval considered. Here, after scheduling the recovery jobs, all the selected jobs are also scaled to the frequency of $0.6f_{max}$ and the same energy savings is obtained as in Figure 1a. Moreover, tasks are *fairly* treated and the reliability of all tasks is *simultaneously* enhanced.

However, as discussed in Section 4, finding the *optimal* subset of jobs for each task to *maximize* energy savings while ensuring *fairness* and/or meeting different assurance requirements for individual tasks is not trivial.

3 System Models

3.1 Task Model

We consider a set of independent periodic real-time tasks $\{\tau_1, \ldots, \tau_n\}$, where task τ_i $(i = 1, \ldots, n)$ is represented by its WCET c_i and period p_i. Considering the variable speed processor, it is assumed that c_i is given under the maximum processing frequency f_{max}. Moreover, for simplicity, we assume that the execution time of a task scales *linearly* with the processing speed[1]. That is, at the scaled frequency f, the execution time of task τ_i is assumed to be $c_i \cdot \frac{f_{max}}{f}$. The utilization of task τ_i is defined as $u_i = \frac{c_i}{p_i}$ and $U = \sum_{i=1}^{n} u_i$ is the system utilization. We further assume that tasks are synchronous and the first job of every task arrives at time 0. The j'th job $J_{i,j}$ of task τ_i arrives at time $(j-1) \cdot p_i$ and has the deadline of $j \cdot p_i$ $(j \geq 1)$.

In this work, following the idea in the skip task model [7, 17], we use a single skip parameter k_i to present the assurance requirement for task τ_i. It means that, for the purpose of reliability enhancements, $(k_i - 1)$ out of any consecutive k_i jobs of task τ_i need to have recovery jobs and *only* such jobs can be scaled down to save energy. Here, k_i can range from 1 to ∞. When $k_i = 1$, no job of τ_i needs a recovery job and cannot be scaled down. With higher values of k_i, more jobs need recoveries and better reliability enhancement can be obtained for task τ_i. For the case of $k_i = \infty$, all jobs of τ_i must have recovery jobs. For the example in Figure 1b, the assurance parameters are $k_1 = k_2 = k_3 = 3$ and $k_4 = k_5 = 2$ for the five tasks, respectively.

Although it is possible to adopt two numbers to represent the assurance requirement for each task, as in the general firm (i.e., (m, k)) task model [12, 21], we underline that, by enforcing the minimal interval between two consecutive jobs without recoveries, the skip assurance model describes the changes in quality of assurance more smoothly. Moreover, as shown in Section 4, even with the skip assurance model, the static flexible RAPM problem is found to be intractable.

Note that the assurance parameters for tasks can be determined following various rules (such as design requirements, importance/criticality of tasks and/or fairness). However, the discussion on

[1] A number of studies have indicated that the execution time of tasks does not scale linearly with reduced processing speed due to accesses to memory [24] and/or I/O devices [5]. However, exploring the full implications of this observation is beyond the scope of this paper and is left as our future work.

how to choose the best assurance parameters for tasks is beyond the scope of this paper and will be addressed in our future work. In this paper, for a set of tasks with *given* assurance requirements, we focus on the flexible RAPM schemes that maximize energy savings while ensuring such requirements.

3.2 Energy Model

We adopt the *system-level power model* proposed in [32], where the power consumption of a computing system at frequency f ($\leq f_{max}$; and corresponding supply voltage) is given by:

$$P(f) = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{ef} f^m) \tag{1}$$

Here, P_s is the *static power*, P_{ind} is the *frequency-independent active power*, and P_d is the *frequency-dependent active power*. The effective switching capacitance C_{ef} and the dynamic power exponent m (in general, $2 \leq m \leq 3$) are system-dependent constants [6] and f is the frequency. $\hbar = 1$ when the system is *active* (i.e., computation is in progress); otherwise, $\hbar = 0$.

Despite its simplicity, the above power model captures the essential power components in a system. Moreover, from the above equation, one can easily find the minimal *energy-efficient frequency* $f_{ee} = \sqrt[m]{\frac{P_{ind}}{C_{ef} \cdot (m-1)}}$ [32]. Consequently, we assume that the frequency is never reduced below the threshold f_{ee} for energy efficiency. Moreover, normalized frequencies are used (i.e. $f_{max} = 1.0$) and we assume that the frequency can vary continuously[2] from f_{ee} to f_{max}.

3.3 Fault Model

Considering that *transient* faults occur much more frequently than *permanent* faults [14], especially with the continued scaling of CMOS technologies and reduced design margins [9], we focus on transient faults in this paper and explore backward recovery techniques to recovery them. It is assumed that the faults are detected using *sanity* (or *consistency*) checks at the completion of a job's execution, and if needed, the recovery task is dispatched, in the form of re-execution [20].

Assuming that transient faults follow Poisson distribution [27], the average transient fault rate for systems running at frequency f (and corresponding supply voltage) can be modeled as [32]:

$$\lambda(f) = \lambda_0 \cdot g(f) \tag{2}$$

[2]For discrete frequency levels, we can use two adjacent levels to emulate the execution at any frequency [13].

where λ_0 is the average fault rate corresponding to the maximum frequency f_{max}. That is, $g(f_{max}) = 1$. Considering the negative effect of DVFS on the transient fault rate, in general, there is $g(f) > 1$ for $f < f_{max}$ [32].

3.4 Problem Description

We consider a set of real-time tasks to be executed on a uni-processor system according to the preemptive *earliest deadline first (EDF)* scheduling policy. For a task set with system utilization U, the spare CPU capacity (i.e., *static slack*) will be $sc = 1 - U$. To guarantee the assurance requirements and achieve the desired reliability, the static slack will be first used to schedule the *required* recovery jobs for every task; then, the remaining static slack (if any) can be used to scale down *only* the jobs that have recoveries, for energy savings.

Simple Manageability Conditions: Considering the assurance requirements of tasks, the *manageability* of a task set is defined as there exists a schedule in which all the required recovery jobs can be accommodated within the timing constraints. When the system utilization of a task set is $U \leq 0.5$, the spare capacity will be large enough to schedule a recovery task for every task [31]. That is, a recovery job can be scheduled for *each* and *every* job of *all* tasks within the timing constraints, regardless of different assurance requirements for tasks. Therefore, $U \leq 0.5$ is the **sufficient condition** for the manageability of a set of tasks with assurance requirements.

However, without taking the specific assurance requirements of tasks into consideration, scheduling a recovery task for every task may not be the most energy efficient approach as shown in the above example. When more slack is used to schedule the *unnecessary* recovery jobs, less slack is left for energy savings. Define the *augmented system utilization* of the task set with assurance requirements as:

$$AU = U + \sum_{i=1}^{n} \frac{(k_i - 1) * c_i}{k_i * p_i} \tag{3}$$

where the second summation term denotes the workload from the required recovery jobs. It is easy to find out that, if $AU > 1$, the spare capacity will not be enough to schedule the required recovery jobs for all tasks and the task set is not manageable. Therefore, the **necessary condition** for a task set to be manageable is $AU \leq 1$.

Problem Statement In this work, for a set of tasks with assurance requirements where $AU \leq 1$, the problems to be addressed are: **a.) how to effectively exploit spare CPU capacity (i.e., static slack) to maximize the energy savings while guaranteeing the assurance requirement for each task; and b.) how to efficiently use the dynamic slack that can be generated at run-time, to further improve energy savings and/or system reliability.**

4 Static Flexible RAPM Schemes

Note that, there are two steps involved in the *static* flexible RAPM problem. First, considering the assurance requirements of tasks, the subset of jobs to which a recovery job for each will be allocated needs to be determined. If all the required recovery jobs can be accommodated within the timing constrains, we say that the task set is *schedulable* with such job selection. Second, for the schedulable job selection, the scaled frequencies need to be determined for the jobs with recoveries to save energy. Here, we can see that the schedulability (as well as the potential energy savings) of a task set directly depends on, for each task, the selection of jobs to which recovery jobs will be allocated.

4.1 Definitions

Recovery Patterns: Given a real-time task τ_i ($i = 1, \ldots, n$) with the assurance requirement k_i, the *recovery pattern* is defined as a binary string of length k_i: $RP_i(k_i)$ ="$r_0 r_1 \cdots r_{k_i-1}$". Here, the value of r_j ($j = 0, \ldots, k_i - 1$) is either 0 or 1, and $\sum r_j = k_i - 1$. Consider the first k_i jobs of task τ_i. If $r_{j-1} = 1$ ($j = 1, \ldots, k_i$), then the j'th job $J_{i,j}$ of task τ_i needs a recovery; otherwise, if $r_{j-1} = 0$, no recovery is needed for $J_{i,j}$. For simplicity, we assume that the recovery pattern will be *repeated* for the remaining jobs of task τ_i. That is, the $(j + q \cdot k_i)$'th job of task τ_i has the same recovery requirement as job $J_{i,j}$, where q is a positive integer. By repeating the recovery pattern, the assurance requirement of a task will be satisfied.

For the example in Figure 1b, the recovery patterns for the five tasks are: $RP_1(3)$ ="110", $RP_2(3)$ ="101", $RP_3(3)$ ="011", $RP_4(2)$ ="10" and $RP_5(2)$ ="01". Note that, in that example, these recovery patterns provide the best energy management opportunity and lead to the maximum energy savings. However, as shown in Section 4.2, finding such recovery patterns and the corresponding scaled frequencies is not trivial.

Augmented Processor Demand: For a set of given recovery patterns for tasks with assurance requirements, as the first step, we need to find out whether the task set is manageable (i.e., the required recovery jobs can be scheduled within timing constraints) or not. For such purpose, we first re-iterate the concept of *processor demand* and the fundamental result in the feasibility analysis of task systems scheduled by preemptive EDF [4, 16]. Then, the concept and the feasibility analysis are extended accordingly.

Definition 1 *The* processor demand *of a real-time job set Φ in an interval $[t_1, t_2]$, denoted as $h_\Phi(t_1, t_2)$, is the sum of computation times of all jobs in Φ with arrival times greater than or equal to t_1 and deadlines less than or equal to t_2.*

Theorem 1 ([4, 16]) *A set of independent real-time jobs Φ can be scheduled (by EDF) if and only if $h_\Phi(t_1, t_2) \leq t_2 - t_1$ for all intervals $[t_1, t_2]$.*

■

For a set of tasks with assurance requirements and given recovery patterns $RP_i(k_i)$ $(i = 1, \ldots, n)$, by incorporating the workload from the required recovery jobs, the *augmented processor demand* in the interval $[t_1, t_2]$ can be formally defined as:

$$APD(t_1, t_2) = \sum_{i=1}^{n} \sum_{j=a}^{b} (1 + r_{x(i,j)}) c_i \tag{4}$$

where

$$a = \left\lfloor \frac{t_1}{p_i} \right\rfloor + 1 \tag{5}$$

$$b = \left\lfloor \frac{t_2}{p_i} \right\rfloor \tag{6}$$

$$x(i, j) = (j - 1) \bmod k_i \tag{7}$$

That is, the augmented processor demand $APD(t_1, t_2)$ includes the workload of all jobs of the tasks, as well as the required recovery jobs, with arrival times greater than or equal to t_1 and deadlines less than or equal to t_2.

Note that, when a job requires a recovery job according to the task's assurance requirement and recovery pattern, it seems like the worst-case workload of that job is doubled. Following the similar reasoning as to the one in [2], we can obtain the following result.

Theorem 2 *For a set of real-time tasks with assurance requirements and given recovery patterns $RP_i(k_i)$ $(i = 1, \ldots, n)$, all jobs and the required recovery jobs of the tasks can be scheduled by preemptive EDF if and only if $APD(t_1, t_2) \leq t_2 - t_1$ for all the intervals $[t_1, t_2]$.*

■

Define the *super-period* of the task set considered as $SP = LCM(k_1 p_1, \ldots, k_n p_n)$, where the function $LCM()$ denotes the *least common multiple (LCM)* of all the numbers. It is easy to see that the recovery patterns of tasks may cross $LCM(p_1, \ldots, p_n)$ and all recovery patterns will repeat after the super-period SP. Therefore, to check the schedulability of a set of real-time tasks with assurance requirements and given recovery patterns, according to Theorem 2, we need to check $APD(t_1, t_2) \leq t_2 - t_1$ for all intervals $[t_1, t_2]$ where $0 \leq t_1, t_2 \leq SP$.

Note that, if $APD(t_1, t_2) < t_2 - t_1$ for all the intervals $[t_1, t_2]$, it means that more slack exists in the schedule and can be used to scale down the jobs with recoveries (for reliability purpose) to save energy. Considering the scaled execution of some jobs, the *augmented processor demand* can be further extended as:

$$EAPD(t_1, t_2) = \sum_{i=1}^{n} \sum_{j=a}^{b} \left(\frac{1}{f_{i,j}} + r_{x(i,j)} \right) c_i \tag{8}$$

where $f_{i,j}$ is the processing frequency for job $J_{i,j}$. Here, the energy consumption of job $J_{i,j}$ will be $E(i,j) = P(f_{i,j}) \frac{c_i}{f_{i,j}}$, in which $P(f)$ is defined as in Equation 1.

With these definitions, the static flexible RAPM problem considered in this work can be formally stated as: for a set of real-time tasks with assurance requirements, find the recovery patterns and the scaled frequencies so as to:

$$\text{Minimize} \quad \sum_{i \in [1,n], j \in [1, SP/p_i]} E(i,j) \tag{9}$$

subject to

$$\sum_{j=0}^{k_i - 1} r_j \geq k_i - 1, \; i = 1, \ldots, n \tag{10}$$

$$f_{i,j} = f_{max}, \text{ if } r_{x(i,j)} = 0 \tag{11}$$

$$f_{i,j} \leq f_{max}, \text{ if } r_{x(i,j)} = 1 \tag{12}$$

$$EAPD(t_1, t_2) \leq t_2 - t_1, \; \forall t_1, t_2 \in [0, SP] \tag{13}$$

where the first condition confines the skip model recovery patterns; the second and third condition states that only jobs with recoveries can be scaled down; and the last condition ensures that, with the recovery patterns and scaled frequencies, the task set is schedulable.

Note that, for a set of tasks with assurance requirements, even with $AU \leq 1$, it is still possible that no schedule exists to be able to accommodate the required recovery jobs within timing constraints. That is, the task set is not manageable with the given assurance requirements. In this case, it is possible to adjust the assurance requirements of tasks (e.g., by appropriately decreasing k_i for task τ_i) to make them feasible. However, exploring such possibility is well beyond the scope of this paper.

4.2 Intractability of the Static Problem

Note that, for a real-time task τ_i with assurance requirement k_i, there are k_i different recovery patterns. Therefore, for the task set considered, the number of different combinations of tasks' recovery patterns is $\prod_{i=1}^{n} k_i$. To find the optimal solution that maximizes energy savings, all these combinations of recovery patterns for tasks need to be examined, and scaled frequencies need to be determined. In fact, finding the optimal solution for the static flexible RAPM problem turns out to be intractable:

Theorem 3 *For a periodic real-time task set where tasks have individual assurance requirements, the static flexible RAPM problem is NP-hard, in the strong sense.*

■

The proof of this problem is provided in the Appendix. We underline that, due to this result, finding the optimal solution even in pseudo-polynomial time seems to be unlikely (unless $NP = P$).

4.3 Upper Bounds on Energy Savings

Considering the intractability of the static RAPM problem, instead of focusing on designing heuristic/approximation algorithms, we first discuss the upper bounds on energy savings. Then, for a special case of the recovery patterns for the tasks, a pseudo-polynomial scheme to find a single scaled frequency for the jobs with recoveries is presented in Section 4.4.

For a task set with system utilization U and spare capacity $sc = 1 - U$, suppose that the utilization for the managed workload is X ($\leq \min\{U, sc\}$). After accommodating the required recovery jobs, the remaining spare capacity (i.e., $sc - X$) could be used to scale down the managed workload to save energy. Considering the convex relation between energy and processing frequency [6], to

minimize the energy consumption, the workload should be scaled down *uniformly* (if possible) and the scaled frequency will be $f(X) = \max\{f_{ee}, \frac{X}{X+(sc-X)}\} = \max\{f_{ee}, \frac{X}{sc}\}$. Without considering the energy consumed by recovery jobs (which are only executed when the corresponding scaled jobs fail with a very small probability), the amount of total *fault-free* energy consumption of the task set within LCM can be calculated as:

$$\begin{aligned} E(X) &= LCM \cdot P_s + LCM(U - X)(P_{ind} + c_{ef} \cdot f_{max}^m) \\ &\quad + LCM \cdot \frac{X}{f(X)} (P_{ind} + c_{ef} \cdot f(X)^m) \end{aligned} \tag{14}$$

where the first part is the energy consumption due to static power, the second part captures the energy consumption of the unscaled workload, and the third part represents the energy consumption of the managed workload.

An Absolute Upper Bound As shown in [31], by differentiating Equation (14), $E(X)$ is minimized when

$$X_{opt} = \min \left\{ U, sc \cdot \left(\frac{P_{ind} + C_{ef}}{m \cdot C_{ef}} \right)^{\frac{1}{m-1}} \right\} \tag{15}$$

Therefore, without considering the assurance requirements for individual tasks, the *absolute* upper bound on the energy savings will be:

$$ES_{abs-upper} = E(0) - E(X_{opt}). \tag{16}$$

where $E(0)$ denotes the energy consumption when no task is managed (i.e., all tasks are executed at f_{max}). This bound actually provides an upper limit on energy savings for *all* possible RAPM schemes (including the previous task-level scheme [31] and the flexible scheme proposed in this paper).

K-Upper Bound with Assurance Parameters Note that, the previous upper bound relies only on system utilization. Taking the assurance parameters of tasks into consideration, we can get a *tighter* upper bound on the energy savings for the flexible scheme. Note that, for a task set where each task has its assurance requirement and a subset of its jobs need recoveries, the workload from the such

jobs (which can be scaled down for saving energy) is:

$$U_{assurance} = \sum_{i=1}^{n} \frac{(k_i - 1) * c_i}{k_i * p_i} \tag{17}$$

Assuming that, after accommodating the required recovery jobs, all such jobs are scaled down uniformly using the remaining slack, a tighter upper bound on the energy savings within LCM can be given as:

$$ES_{k-upper} = E(0) - E(U_{assurance}) \tag{18}$$

4.4 Deeply-Red Recovery Pattern

In the weakly-hard real-time scheduling literature, the *"deeply-red"* execution pattern has been adopted widely [7, 17]. There are two main reasons: first, if a task set is schedulable under the deeply-red execution pattern, it will be schedulable for any other execution patterns; second, the simplified feasibility test can be used. Instead of checking the processor demand between any interval from time 0 to the super-period SP, only the intervals that start at time 0 and end at a time instance not larger than $LCM(p_i, \ldots, p_n)$ need to be considered.

Following the same idea, in this work, the *"deeply-red"* recovery pattern will be used and is defined as the one with leading 1's followed by a 0. For instance, for the task τ_i with assurance requirement k_i, the first $(k_i - 1)$ characters of its recovery pattern $RP_i(k_i)$ are $'1'$'s and the last character is a $'0'$. Following the same line of reasoning as in [7, 17], and using the augmented processor demand function $APD()$ defined in Equation 4, we have:

Theorem 4 *For a real-time task set, if all tasks with assurance requirements adopt the deeply-red recovery pattern, the task set can be scheduled by preemptive EDF if and only if* $APD(0, L) \leq L$ *for* $\forall L, 0 \leq L \leq LCM(p_1, \ldots, p_n)$. ■

Define the *manageable workload* for a set of tasks with assurance requirements in the interval $[t_1, t_2]$ as:

$$MW(t_1, t_2) = \sum_{i=1}^{n} \sum_{j=a}^{b} r_{x(i,j)} c_i \tag{19}$$

where a, b and $x(i, j)$ are the same as defined in Equations (5), (6) and (7), respectively. If $APD(0, L) < L$, more slack exists and can be used to scale down the execution of the jobs with

recoveries to save energy. Assuming that all manageable jobs are scaled down uniformly [1], the scaled frequency f_{dr} can be calculated as:

$$f_{dr} = \max\left\{\frac{MW(0,L)}{MW(0,L) + (L - APD(0,L))}\right\} \tag{20}$$

where $0 < L \leq LCM(p_1, \ldots, p_n)$. Note that, when evaluating f_{dr}, it is sufficient to consider L values that correspond to period boundaries of tasks, which will take pseudo-polynomial time.

For the example shown in Figure 1c, where the deeply-red recovery pattern is used for every task, the scaled frequency can be calculated as $\frac{9}{11}$. Here, we can see that, although the deeply-red recovery pattern simplifies the feasibility test, the required recovery jobs can "clash" (i.e. need to be scheduled during the same time interval). Therefore, the single scaled frequency calculated can be pessimistic. The performance of this simplified scheme is evaluated and compared to the upper bounds on energy savings in Section 6.

5 Dynamic Online RAPM Schemes

Note that, the statically scheduled recovery jobs are executed only if their corresponding scaled jobs fail. Otherwise, the CPU time reserved for those recovery jobs is freed and becomes *dynamic slack* at run-time. Moreover, it is well-known that real-time tasks typically take a small fraction of their WCETs [10]. Therefore, significant amount of dynamic slack can be expected at run time, which should be exploited to further save energy and/or enhance system reliability.

In [31], an effective dynamic slack management mechanism, called *wrapper-task* approach, has been studied. In this approach, wrapper tasks are used to represent dynamic slack generated at run-time. When dynamic slack is reclaimed or consumed, the corresponding wrapper task(s) will be destroyed, which prevents the slack from being reused by high priority task instances later. Therefore, the slack reserved for recovery blocks will be conserved across preemption points, which is essential for reliability preservation in the RA-PM schemes.

However, in the previous dynamic RAPM algorithm using wrapper-tasks, task sets are assumed to have system utilization of $U = 1$ and no recovery job is scheduled statically [31]. Therefore, all jobs of tasks arrive at the maximum frequency f_{max}, and are scaled down only if the available dynamic slcak is enough to schedule the necessary recovery jobs. Combining with the static flexible RAPM scheme, where the required recovery jobs are statically scheduled for a subset jobs of every task and the scaled frequency is determined accordingly, we can extend the dynamic algorithm as shown in Algorithm 1. Note that, when applying the dynamic algorithm to a statically managed task set, the

Algorithm 1 Flexible Dynamic RAPM Algorithm

t_{past} is the elapsed time since last scheduling point. J and WT represent the current job and wrapper-task, respectively (if there is no such a job or wrapper-task, they have a $NULL$ value). $J.rem$ and $WT.rem$ denote the remaining time requirements; $J.d$ and $WT.d$ are the deadlines.
Step 1:
  if (J!=NULL and $J.rem - t_{past} > 0$) {
    $J.rem -= t_{past}$;//job was preempted or completes
    if (J completes)
      CreateWT($J.rem$, $J.d$);//slack of early completion
    else *Enqueue(J, Ready-Q)*;}
  if (WT!=NULL and $WT.rem - t_{past} > 0$) {
    $WT.rem -= t_{past}$; *Enqueue(WT, WT-Queue)*;}
  if (WT!=NULL and J!=NULL)
    CreateWT(t_{past}, $J.d$);//push forward slack;
  if (J is scaled and succeeds){
    RemoveRecoveryJob(J,*Ready-Q*);
    CreateWT($J.c$, $J.d$);//slack from recovery job;}
Step 2:
  for (all newly arrived job $NJ_{i,j}$){
    $x(i,j) = (j-1) \bmod k_i$;
    if ($r_{x(i,j)} == 1$) {
      CreateRecoveryJob($NJ_{i,j}$);$NJ_{i,j}.f = f_{i,j}$;
      $NJ_{i,j}.rem = \frac{c_i}{f_{i,j}}$;$NJ_{i,j}.scaled = true$;
    }**else**{$NJ_{i,j}.scaled = false$;
      $NJ_{i,j}.f = f_{max}$;$NJ_{i,j}.rem = c_i$;}
    Enqueue($NJ_{i,j}$, Ready-Q);}
Step 3://in the following, J and WT will represent the next job and wrapper-task to be processed, respectively;
  J=Dequeue(Ready-Q);
  if (J!=NULL) ReclaimSlack(J, *WT-Queue*);
  WT=Header(WT-Queue);
  if (J!=NULL){
    if ($WT! = NULL$ and $WT.d < J.d$)
      //WT wraps J's execution (a timer is needed)
      WT = *Dequeue(WT-Queue)*;
    else $WT = NULL$;//normal execution of J
    Execute(J);}
  else if (WT!=NULL)
    WT = *Dequeue(WT-Queue)*;//WT executes no-ops

task-level RAPM scheme can be viewed as a special case of the flexible scheme, where the selected tasks have assurance requirements of $k = \infty$ with recovery patterns of all 1's and other tasks have $k = 0$ with recovery patterns of all 0's.

Here, the task set is assumed to be schedulable under preemptive EDF with the given recovery patterns and scaled frequencies that are determined by the static RAPM schemes. The major differ-

ence between Algorithm 1 and the dynamic scheme in [31] comes from Step 2 (lines 15 to 23). In the new algorithm, according to the recovery pattern of a task (lines 17 and 18), if needed, the recovery job is created at the time of a new job's arrival and the scaled frequency is set correspondingly (line 19).

For the detailed discussion on other part of the algorithm, the interested readers are referred to [31]. However, we would like to emphasize that the dynamic slack reclamation through the management of wrapper tasks will not violate any timing constraint that is statically guaranteed.

6 Simulation Results and Discussions

To evaluate the performance of the proposed schemes, we developed a discrete event simulator using C++. In the simulations, we implemented the simplified flexible static RAPM scheme (**Flexible**) that assumes that all tasks take the deeply-red recovery pattern. For simplicity, if a task set is not manageable with the deeply-red recovery pattern, we assume that no recovery jobs will be scheduled and no power management will be applied (i.e., all tasks will be executed at f_{max}). The **dynamic** RAPM scheme is also implemented. In addition, we consider two different schemes for comparison. First, the scheme of no power management (**NPM**), which does not schedule any recovery job and executes all tasks/jobs at f_{max} while putting system to sleep states when idle, is used as the baseline. Second, for the task-level static RAPM scheme, we consider the smaller utilization task first heuristic (**SUF**) [31].

The parameters employed in the simulations are similar to the ones used in [31]. Focusing on active power and assuming $P_s = 0, P_{ind} = 0.05$, $C_{ef} = 1$ and $m = 3$, the energy efficient frequency can be calculated as $f_{ee} = 0.29$ (see Section 3). Moreover, the transient faults are assumed to follow the Poisson distribution with an average fault rate of $\lambda_0 = 10^{-6}$ at the maximum frequency f_{max} (and corresponding supply voltage). For the fault rates at lower frequencies/voltages, we adopt the exponential fault rate model $g(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{ee}}}$ and assume that $d = 2$ [32]. That is, the average fault rate is 100 times higher at the lowest frequency f_{ee} (and corresponding supply voltage).

We consider synthetic real-time task sets where each task set contains 10 periodic tasks. The periods of tasks (p) are uniformly distributed within the range of $[10, 20]$. The WCET (c) of a task is uniformly distributed in the range of 1 and its period. Finally, the WCETs of tasks are scaled by a constant such that the desired system utilization is reached [19]. For the assurance requirements of tasks, we consider two different settings. In the first setting, all tasks have the same assurance requirement (e.g., $k = 2$). For the second setting, the assurance parameters of tasks are randomly

generated within the range of $[2, 10]$. For each run of the simulation, approximately 20 million jobs are executed. Moreover, each result point in the graphs corresponds to the average of 100 runs.

6.1 Performance of the Static Schemes

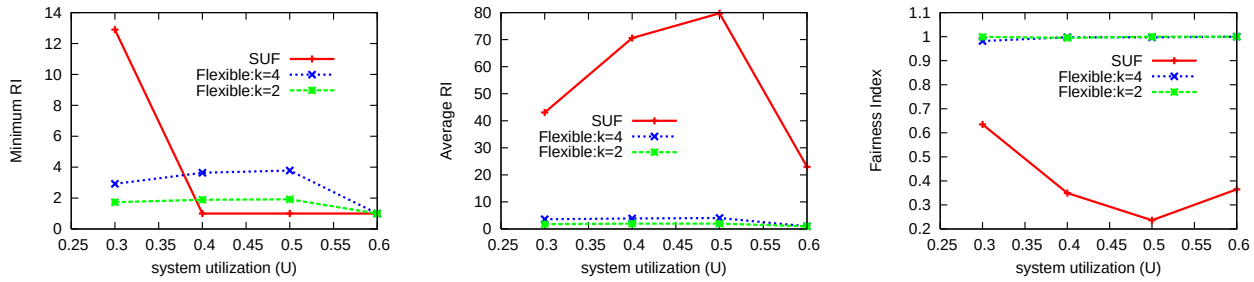
Reliability: Note that, under RAPM schemes, recovery tasks/jobs are scheduled and the reliability of the corresponding tasks will be improved. Define the *probability of failure* (i.e., $1-$*reliability*) $PoF_i(S)$ of a task τ_i under any scheme S as the ratio of the number of failed jobs over the total number of jobs executed. Consider the NPM scheme as the baseline, the *reliability improvement* of a task τ_i under a scheme S can be defined as:

$$RI_i(S) = \frac{PoF_i(NPM)}{PoF_i(S)} = \frac{\text{\# of failed jobs under NPM}}{\text{\# of failed jobs under S}}$$

That is, larger $RI_i(S)$ values indicate better reliability improvement. Moreover, to quantify the fairness on reliability improvement to tasks, following the idea in [15], the *fairness index* of a scheme S is defined as:

$$FI(S) = \frac{(\sum_i RI_i(S))^2}{n \sum_i RI_i(S)^2} \tag{21}$$

From this equation, we can see that, the value of fairness index has the range of $(0, 1]$, and the higher values mean that tasks are treated more fairly.



a. minimum RI b. average RI c. fairness index

Figure 2. Reliability improvement and fairness index for the static schemes.

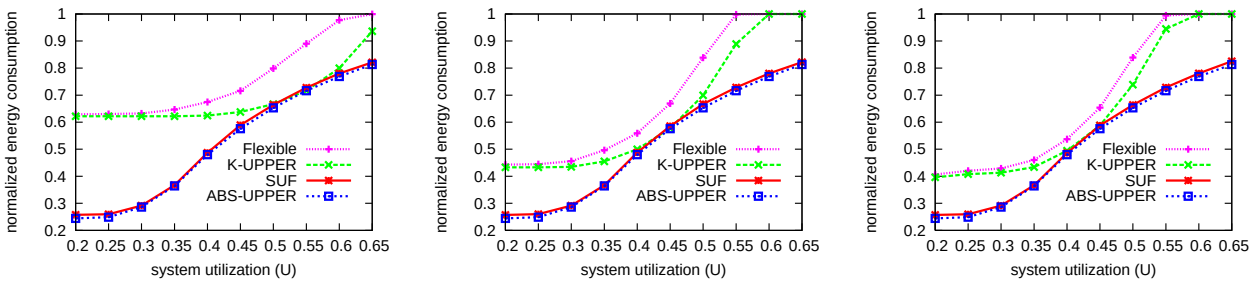
In the first set of experiments, we consider task sets with 10 tasks that have the same assurance parameter k. Figure 2 shows the reliability improvements and the fairness index for the static schemes. In the figures, "Flexible:k=i" means that all tasks have the same assurance parameter $k = i$ in the static flexible RAPM problem. The X-axis represents the system utilization.

For applications where the system reliability is determined by the task with *lowest* quality, Figure 2a shows the *minimum* reliability improvement of the tasks. Here, as mentioned before, larger numbers mean better improvement. From the figure, we can see that, when the system utilization is low (e.g., $U \leq 0.4$), the task-level static scheme SUF will manage all the tasks and performs better than the flexible scheme. However, when the system utilization is large (e.g., $U \geq 0.4$), at least one task will not be managed and its reliability will not have any improvement. For the flexible scheme, the minimum reliability improvement of the tasks is rather stable and directly related to the assurance parameter k. For example, when $k = 4$, only 1 out of 4 jobs will not have a recovery job for each task and, compared to NPM, the reliability improvement is roughly around 4 times. The same result is obtained for the case of $k = 2$. However, for large system utilization (e.g., $U \geq 0.6$), the flexible RAPM scheme cannot guarantee the assurance requirements for all the tasks.

If the overall system reliability depends on the total successfully executed jobs from all tasks, Figure 2b shows that the average reliability improvement of the tasks under SUF is much better than the flexible RAPM scheme. The reason is as follows: while SUF always tries to manage as many jobs as possible up to the workload X_{opt}, the manageable jobs under flexible scheme are limited by the assurance parameters of tasks. Figure 2c further shows the fairness index of the tasks under different system utilizations. Here, we can see that, with the same assurance parameter, the flexible scheme provides excellent fairness to tasks. From the results, we can conclude that the task-level SUF scheme should be used if the overall system reliability depends on the average behaviors of tasks. However, if the system reliability is limited by the lowest quality task or fairness is required for tasks, the flexible scheme should be employed.

Energy Savings: For different settings of the assurance requirements for tasks, Figure 3 shows the normalized energy consumption for the static flexible RAPM scheme. For comparison, the energy consumption for SUF and the upper bounds is also shown. Here, "K-UPPER" denotes the upper bound that considers the assurance requirements of tasks and "ABS-UPPER" is for the absolute upper bound. Note that, higher energy consumption means less energy savings.

From the results, we can see that the energy consumption of SUF is very close to the absolute bound (ABS-UPPER), which coincides with our previous results [31]. For the flexible RAPM scheme, its energy performance is almost the same as that of K-UPPER at low system utilization (e.g., $U \leq 0.3$) since all manageable jobs are scaled down to the same frequency (e.g., f_{ee}). However, at high system utilization, due to conflicts of the required recovery jobs under deeply-red recovery patterns, the scaled frequency of the flexible scheme is higher than that of K-UPPER (which as-



a. $k = 2$ b. $k = 4$ c. $k \in [2, 10]$

Figure 3. Normalized energy consumption for the static schemes.

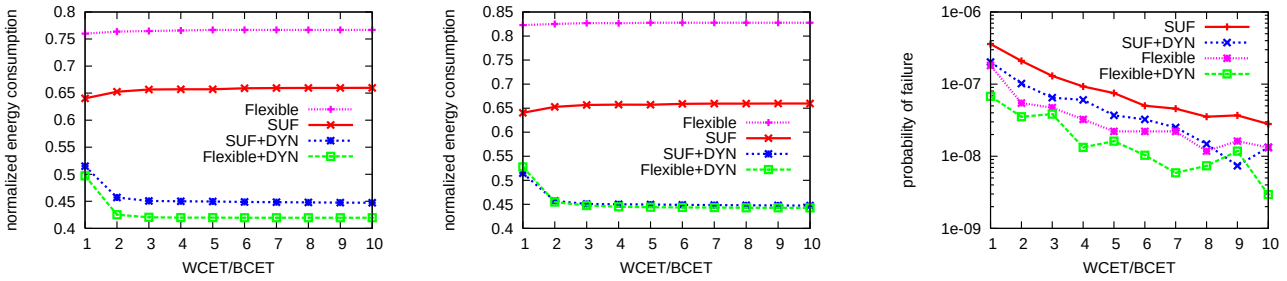
sumes all remaining static slack can be used by DVFS) and thus consumes more energy. Moreover, when compared to SUF, as shown in Figure 3a, the flexible RAPM scheme performs worse with $k = 2$ due to limited number of manageable jobs. For larger values of k (Figures 3b and 3c), the energy performance difference becomes smaller, especially for the moderate system utilization (e.g., $0.35 \leq U \leq 0.45$).

Therefore, we can conclude that the flexible static RAPM scheme, which considers different assurance requirements of individual tasks, can guarantee such assurance requirements and/or provide fairness to tasks, but at the cost of increased energy consumption. Moreover, we can see that, when choosing the assurance requirements for tasks, in addition to satisfying tasks' reliability requirements, to maximize the energy savings, the overall manageable workload should consider X_{opt} and use it as a reference.

6.2 Dynamic Schemes

In this section, we evaluate the dynamic schemes for their energy savings and reliability enhancements over static schemes. Here, Algorithm 1 is applied on top of the static flexible scheme (referred as "Flexible+DYN") as well as the static task-level SUF scheme (referred as "SUF+DYN"). Note that, in the dynamic algorithm, all jobs from all tasks are treated equally. That is, any job can reclaim available dynamic slack at run time, regardless of whether it is scaled by the static scheme or not.

To emulate the run-time behaviors of real-time tasks/jobs, the variability of a task's workload is controlled by the ratio of $\frac{WCET}{BCET}$ (that is, the *worst-case* to *best-case* execution time ratio), where larger values of the ratio imply more dynamic slack can be expected from the early completion of tasks/jobs. At run time, the actual execution time of a real-time job follows a normal distribution with mean and standard deviation being $\frac{WCET+BCET}{2}$ and $\frac{WCET-BCET}{6}$, respectively [3].



a. normalized energy for $k = 4$ b. normalized energy for $k \in [2, 10]$ c. probability of failure for $k \in [2, 10]$

Figure 4. Energy and reliability improvement with dynamic schemes at $U = 0.5$.

Considering the system utilization of task sets to be $U = 0.5$, Figure 4 shows the performance improvement of the dynamic scheme over static schemes on both energy and reliability. Note that, even if the ratio $\frac{WCET}{BCET} = 1$ (i.e., there is no variation in the execution time of tasks), dynamic slack is still available at run time due to the free of statically scheduled recovery jobs when there is no error during the execution of their corresponding scaled jobs. From the Figures 4a and 4b (which correspond to tasks having the same assurance requirement $k = 4$ and tasks with different assurance requirements randomly generated between $[2, 10]$, respectively), we can see that the dynamic scheme can significantly improve the energy performance over static schemes (upto 33% for the flexible scheme and 20% for SUF). However, the performance improvement is rather stable after $\frac{WCET}{BCET} \geq 3$. The reason comes from the fact that, with larger values of the ratio, excessive dynamic slack is available from jobs' the early completion and almost all jobs can reclaim the slack and run at the frequency f_{ee}.

Moreover, we can see that the difference (from 10% to 15% for the cases considered) between the energy performance of the static schemes is effectively diminished (only around 2%) after combining with the dynamic scheme. Therefore, although the static flexible scheme itself may consume more energy, the combination of the flexible scheme with dynamic algorithm can recuperate its energy inefficiency while guaranteeing the individual assurance requirements of tasks statically.

For the case of randomly generated assurance requirements for tasks, Figure 4c shows the *overall* probability of failure (i.e., $1-$*reliability*) of the system under different schemes considered. Here, smaller values indicate better system reliability. From the results, we can see that, by allowing the statically unscaled jobs (which have no recovery job initially) to reclaim dynamic slack, additional recovery jobs can be scheduled online and the dynamic algorithm can further improve system reliability. For larger values of $\frac{WCET}{BCET}$, the actual execution time of jobs becomes shorter and the

reliability for all schemes increases slightly.

7 Conclusion and Future Work

Energy has been recognized as a first-class resource in computing systems and power aware computing has recently become an important research area. Consider the negative effects of voltage scaling on system reliability due to increased transient fault rates, *reliability-aware power management (RAPM)* schemes have been studied to save energy while preserving system reliability. The central idea of RAPM schemes is to reserve a portion of the available slack to schedule one *recovery* to preserve reliability. However, in previous RAPM schemes, real-time tasks are treated *unequally* and are selected for management in greedy fashion without considering individual tasks' reliability requirements.

In this paper, for a set of periodic real-time tasks with different assurance requirements, we study *flexible* static RAPM schemes, which targets at maximizing energy savings while ensuring tasks' assurance requirements by manages a subset of jobs for *every* task. The problem is formally presented and is shown to be NP-hard, in the strong sense. The upper bounds on energy savings are discussed. For a special case, where the requirements for recovery jobs follow a deeply-red pattern, a pseudo-polynomial static scheme is proposed. Dynamic schemes that explore dynamic slack for better energy savings and reliability enhancement are also discussed. The schemes are evaluated extensively through simulations. From the results, we conclude that, compared to the previous task-level RAPM schemes, the new flexible RAPM schemes can guarantee the assurance requirements and provide *fairness* for all tasks, but at the cost of decreased energy savings. However, when combining with dynamic schemes, such cost for the flexible scheme can be effectively recovered.

For our future work, we will study schemes that can automatically choose the best assurance parameters for tasks to maximize energy savings and/or reliability enhancements.

References

[1] T. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. In *Proc. of The* 26^{rd} *IEEE Real-Time Systems Symposium*, Dec. 2005.

[2] H. Aydin. Exact fault-sensitive feasibility analysis of real-time tasks. *IEEE Trans. on Computers*, 56(10):1372–1386, 2007.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proc. of IEEE Real-Time Systems Symposium*, 2001.

[4] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2, 1990.

[5] E. Bini, G. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *Proc. of the* 17^{th} *Euromicro Conference on Real-Time Systems*, 2005.

[6] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of The HICSS Conference*, Jan. 1995.

[7] M. Caccamo and G. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *Proc. of the 18th IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1997.

[8] E. M. Elnozahy, R. Melhem, and D. Mossé. Energy-efficient duplex and tmr real-time systems. In *Proc. of The* 23^{rd} *IEEE Real-Time Systems Symposium*, Dec. 2002.

[9] 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. *IEEE Micro*, 24(6):10–20, 2004.

[10] R. Ernst and W. Ye. Embedded program timing analysis based on path clustering and architecture classification. In *Proc. of The Int'l Conference on Computer-Aided Design*, pages 598–604, 1997.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Mathematical Sciences Series. Freeman, 1979.

[12] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Trans. on Computers*, 44(5):1443–1451, 1995.

[13] T. Ishihara and H. Yauura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of The Int'l Symposium on Low Power Electronics and Design*, 1998.

[14] R. Iyer, D. J. Rossetti, and M. Hsueh. Measurement and modeling of computer reliability as affected by system activity. *ACM Trans. on Computer Systems*, 4(3):214–237, Aug. 1986.

[15] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report TR-301, DEC Research, Sep. 1984.

[16] K. Jeffay and D. L. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Proc. of the IEEE Real-Time Systems Symposium*, Dec. 1993.

[17] G. Koren and D. Shasha. Skip-over: algorithms and complexity for overloaded systems that allow skips. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 110–117, Dec. 1995.

[18] R. Melhem, D. Mossé, and E. M. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Trans. on Computers*, 53(2):217–231, 2004.

[19] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of* 18^{th} *ACM Symposium on Operating Systems Principles*, Oct. 2001.

[20] D. K. Pradhan. *Fault Tolerance Computing: Theory and Techniques*. Prentice Hall, 1986.

[21] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *Proc. of the IEEE Real-Time Systems Symposium*, Nov. 2000.

[22] J. A. Ratches, C. P. Walters, R. G. Buser, and B. D. Guenther. Aided and automatic target recognition based upon sensory inputs from image forming systems. *IEEE Tran. on Pattern Analysis and Machine Intelligence*, 19(9):1004–1019, 1997.

[23] S. Saewong and R. Rajkumar. Practical voltage scaling for fixed-priority rt-systems. In *Proc. of the* 9^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[24] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. In *Proc. of the* 24^{th} *IEEE Real-Time System Symposium*, 2003.

[25] O. S. Unsal, I. Koren, and C. M. Krishna. Towards energy-aware software-based fault tolerance in real-time systems. In *Proc. of The Int'l Symposium on Low Power Electronics Design*, 2002.

[26] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proc. of The First USENIX Symposium on Operating Systems Design and Implementation*, Nov. 1994.

[27] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proc. of IEEE/ACM Design, Automation and Test in Europe Conference(DATE)*, 2003.

[28] Y. Zhang, K. Chakrabarty, and V. Swaminathan. Energy-aware fault tolerance in fixed-priority real-time embedded systems. In *Proc. of Int'l Conference on Computer Aided Design*, Nov. 2003.

[29] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[30] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proc. of the Int'l Conf. on Computer Aidded Design*, Nov. 2006.

[31] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, 2007.

[32] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the Int'l Conf. on Computer Aidded Design*, 2004.

[33] D. Zhu, R. Melhem, D. Mossé, and E. Elnozahy. Analysis of an energy efficient optimistic tmr scheme. In *Proc. of the* 10^{th} *Int'l Conference on Parallel and Distributed Systems*, 2004.

[34] D. Zhu, X. Qi, and H. Aydin. Priority-monotonic energy management for real-time systems with reliability requirements. In *Proc. of the IEEE International Conference on Computer Design (ICCD)*, 2007.

APPENDIX: Proof of Theorem 3

Our strategy in proving the theorem has two phases. In the first phase, we will focus on the the well-known real-time scheduling problem with skip parameters [17] that we will call, in short, RT-SKIP.

RT-SKIP: Consider a set of independent periodic real-time tasks $\{\tau_1, \ldots, \tau_n\}$, where task τ_i is represented by its WCET c_i, period p_i and the skip parameter k_i. Is there a schedule where each task τ_i meets $k_i - 1$ deadlines in each consecutive k_i invocations?

We will first show that a *special case* of RT-SKIP, called RT-SKIP*, is NP-Hard in the strong sense. Then, we will establish that that RT-SKIP* itself is equivalent to a specific instance of the flexible RA-PM problem, implying that the latter is also NP-Hard in the strong sense.

Specifically, RT-SKIP* is defined as the special case of RT-SKIP where all n tasks have the same period p, the same skip parameter k and the total utilization is equal to $U_{tot} = \sum \frac{c_i}{p} = \frac{k}{k-1}$.

Theorem 5 *RT-SKIP* is NP-Hard in the strong sense.*

Proof: Consider the following 3-PARTITION problem, which is known to be NP-Hard in the strong sense [11].

3-PARTITION: Consider integers m, k and a set of items $A = \{a_1 \dots a_{3m}\}$, where an integer size s_i is associated with each a_i. It is known that $\sum_{i=1}^{3m} s_i = m.B$ and $\frac{B}{4} < s_i < \frac{B}{2},\ \forall_i$. Is there a partitioning of A into m bins $A_1 \dots A_m$, such that the sizes of the items placed in each bin sum up to exactly $\sum_{a_i \in A_j} s_i = B, \forall_j$? (Note that the constraints given in the problem imply that, if there exists such a partitioning, *exactly* three items will appear in each bin A_j.)

Given an instance of the 3-PARTITION problem, specified by m, B and $\{s_i\}$ values, we will construct an equivalent instance of RT-SKIP*, and show that the 3-PARTITION instance admits a YES answer if and only if the corresponding RT-SKIP* instance admits a YES answer.

First, observe that, in any YES instance of 3-PARTITION, the $3m-3$ items that are *not* placed in a given bin A_k can be seen as forming a "mirror" (or, complement) bin $\overline{A_k}$ with total size $m \cdot B - B = (m-1).B$

Since each item a_i is supposed to appear in exactly one bin in the YES instance of the 3-PARTITION problem, we can conclude that a_i will appear in exactly $m-1$ mirror bins. Given an instance of the 3-PARTITION, construct the corresponding RT-SKIP* instance as follows: we have $3m$ periodic tasks $\tau_1, \dots, \tau_{3m}$, each with period $p_i = p = (m-1)B$, execution time $c_i = s_i$ and the skip parameter $k_i = k = m$. Observe that, complying with the specification of the RT-SKIP* problem definition, the total utilization of the task set is:

$$U_{tot} = \sum_{i=1}^{3m} \frac{c_i}{p} = \sum_{i=1}^{3m} \frac{s_i}{p} = \frac{m\ B}{(m-1)\ B} = \frac{k}{(k-1)}$$

Since $k_i = k = m\ \forall i$, it is sufficient and necessary to provide the schedule in the interval $[0, m(m-1)B]$ where all tasks meet their deadlines according to the skip parameters (and repeat that schedule thereafter).

In these settings, the *minimum* CPU time needed to meet all the "skip" requirements of the tasks in the interval $[0, m(m-1)B]$ is $\sum_{i=1}^{3m} c_i \cdot m \cdot \frac{m-1}{m} = m \cdot (m-1) \cdot B$, which indicates that the timeline must be fully utilized and exactly $m-1$ instances of each task (out of m total instances) must be scheduled in the same interval.

Given such a schedule, consider partitioning the items (in the 3-PARTITION instance) to bins in such a way that a_i is placed in A_j if and only if τ_i is *not* scheduled in the j^{th} period. Recalling the relationship between the bins and the "mirror" bins, we conclude that the 3-PARTITION instance admits also a YES answer. Similarly, when the 3-PARTITION instance has a YES answer, one can obtain a YES instance for the RT-SKIP* instance by scheduling τ_i in the j^{th} period whenever a_i is *not* placed in A_j; concluding the proof. ■

Having proven that RT-SKIP* is NP-Hard in the strong sense, we can also state the following:

Corollary 1 *RT-SKIP is NP-Hard in the strong sense.*

We believe that the above result is also worthy of attention on its own. Because, in [17], Koren and Shasha only showed that RT-SKIP is NP-Hard in the *weak* sense. Later, Quan and Hu proved that the *general* problem of scheduling with (m, k)-firm deadlines is NP-Hard in the strong sense [21]. The RT-SKIP problem is only a special case of the general problem of scheduling with (m, k)-firm deadlines, where $m = k - 1$; hence, the fact that RT-SKIP is NP-Hard in the strong sense is *not* implied by the existing results[3].

Now, we are ready to show that the flexible RA-PM problem is also NP-Hard in the strong sense. We will show that the flexible RA-PM problem can be reduced to the RT-SKIP* problem. Since the former is just shown to be NP-Hard in the strong sense above, the conclusion will follow.

Given an instance of the RT-SKIP* problem with a task set $\tau = \{\tau_1, \ldots, \tau_n\}$ where each task τ_i has an execution time c_i, the same skip parameter k and the same period p, and the total utilization is $U_{tot} = \sum \frac{c_i}{p} = \frac{k}{k-1}$, we construct the corresponding flexible RA-PM instance as follows: We have a periodic task set $\tau' = \{\tau'_1, \ldots, \tau'_n\}$, where each task τ'_i has the execution time $c'_i = \frac{c_i}{2}$, the same skip parameter $k' = k$ and the same period $p' = p + \sum_{i=1}^{n} c'_i$. In this specific instance, the frequency-independent power component is assumed to be negligible. Hence, the energy-efficient frequency of each task is equal to to the minimum frequency available on the CPU, which is assumed not to exceed 0.5 (when normalized with respect to the maximum CPU frequency).

[3]To be accurate, the proof of Quan and Hu made use of a different special case where $m = 1$ for all tasks.

In the flexible RA-PM instance, we will need to provide a schedule in which all the *managed* tasks are selected according to the assurance requirement $k' = k$, in interval $[0, kp']$. In each period of length p' as defined above, the total workload that needs to be executed at f_{max} consists of the tasks that are *not* managed and the recovery tasks for those that are managed. It is easy to see that the total CPU time that *must* be reserved for this component is equal to $\sum_{i=1}^{n} c'_i = \sum_{i=1}^{n} \frac{c_i}{2}$ in every period p'.

Also observe that, in every period p', $p' - \sum_{i=1}^{n} c'_i = p$ (where p is the common period of tasks in the original RT-SKIP* instance), is the CPU time available for the execution of the managed tasks at the scaled frequency(ies).

Hence, the total CPU time available for the scaled tasks in the interval $[0, kP']$ is

$$W_1 = k \cdot p' - k \sum c'_i = k \cdot p$$

Next, note that, the *minimum* managed/selected task workload[4] that *must* be completed in the same interval $[0, kp']$ is:

$$W_2 = (k-1) \sum_{i=1}^{n} c'_i = (k-1) \sum \frac{c_i}{2}$$

Recalling that $U_{tot} = \sum \frac{c_i}{p} = \frac{k}{k-1}$, we have:

$$\frac{W_2}{W_1} = \frac{(k-1) \sum \frac{c_i}{2}}{k \cdot p} = 0.5$$

Since $\frac{W_2}{W_1} = 0.5$, the maximum possible energy savings can be obtained by using the *uniform* frequency 0.5, thereby fully utilizing the timeline, **subject to the condition that a feasible schedule exists with that frequency for the $k-1$ instances of each task in the k consecutive common period p'**. Observe that, if such a schedule exists:

- the execution of each scaled instance of τ_i will be $\frac{c'_i}{0.5} = c_i$, and,
- we will have exactly $k' = k$ time intervals, each of length $p' - \sum c'_i = p$, during which exactly $(k-1)$ scaled instances of each τ'_i must be scheduled.

But the above conditions characterize precisely the necessary and sufficient conditions under which the original RT-SKIP* instance would admit a YES answer. Hence, a flexible RA-PM optimization routine that would run in polynomial-time would be able to check if the minimum energy

[4]We are expressing the workload as normalized to the maximum CPU speed.

flexible RA-PM schedule yields a total energy figure which is equal to the one obtained by scaling exactly $(k-1)$ instances of each task during the super-period, using the frequency 0.5.

As shown above, this would be possible if and only if there exists a positive answer to the RT-SKIP* instance. Since the RT-SKIP* problem was shown to be NP-Hard in the strong sense, it follows that the general flexible RA-PM problem is NP-Hard in the strong sense, as well.