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. The existing RAPM schemes for periodic tasks may be, however, inherently unfair in that they can manage only some tasks at the expense of the other remaining tasks. In this work, we propose the flexible *reliability-aware power management framework, which allows the management of all the tasks in the system, according to their* assurance requirements. *Optimally solving this problem is shown to be NP-hard in the strong sense and upper bounds on energy savings are derived. Then, by extending the processor demand analysis, a pseudo-polynomial-time static scheme is proposed for the "deeply red" recovery patterns. On-line schemes that manage 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 for all the tasks, but at the cost of slightly decreased energy savings. However, when combined with dynamic reclaiming, the new schemes become as competitive as the previous ones on the energy dimension, while improving overall reliability.*

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. 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)* [25], the supply voltage and operating frequency of modern processors can be scaled down to save energy. However, at low processor operating frequencies, applications will generally take more time to complete. In the recent past, several research studies explored the problem of minimizing energy consumption while meeting all the timing constraints for various system models [3, 18, 23], by exploiting the available static and/or dynamic *slack* in the system.

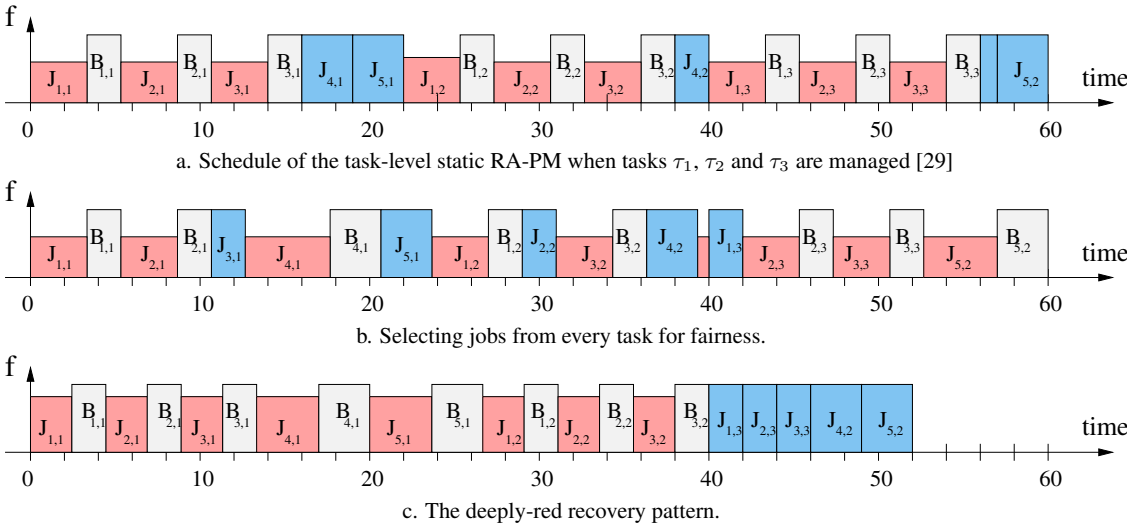
More recently, the adverse effect of DVFS on system reliability due to increased transient fault rates has been studied [30]. With the continued scaling of CMOS technologies and reduced design margins for higher performance, it is expected that, in addition to the systems that operate in electronics-hostile environments (such as those in outer space), practically all digital computing systems will be much more vulnerable to transient faults [10]. 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.

Some recent studies addressed energy efficiency and system reliability simultaneously [8, 9, 17, 19, 24, 26, 31]. However, most of the previous research either focused on tolerating a fixed number of faults [9, 17, 24, 31] or assumed constant transient fault rate [26]. By taking the negative impact of DVFS on system reliability (due to increased transient fault rates at lower supply voltages [30]) into consideration, we proposed and analyzed *reliability-aware power management (RAPM)* schemes for different real-time task models [27, 28, 29, 32]. Unlike the *ordinary* power management schemes that exploit *all* the available slack for energy savings [3, 18, 23], **the central idea of RAPM is to reserve a portion of available slack to schedule a *recovery job* for any job whose execution is scaled down through DVFS** [27]. The remaining slack is still used to save energy by reducing the execution frequency of the job. It should be noted

*This work was supported in part by NSF awards CNS-0720651, CNS-0720647 and NSF CAREER Award CNS-0546244.

1533-2306/08 $25.00 © 2008 IEEE
DOI 10.1109/RTCSA.2008.48

IEEE computer society



a. Schedule of the task-level static RA-PM when tasks τ_1, τ_2 and τ_3 are managed [29]

b. Selecting jobs from every task for fairness.

c. The deeply-red recovery pattern.

Figure 1. Motivational Example

that the recovery jobs are invoked for execution only if their corresponding scaled tasks fail, and they are executed at the *maximum* processing speed if invoked. It has been proved that, with the help of recovery jobs, the RAPM scheme can guarantee to preserve (even, enhance) the system reliability, regardless of the extent of the fault rate increases and processing frequency reductions [27].

Although the previously proposed static *task-level* RAPM schemes for periodic real-time tasks can achieve significant energy savings while preserving system reliability [29, 32], a few problems remain open. For instance, the previous schemes are based on *managing exclusively* a subset of tasks (i.e., scheduling corresponding recovery tasks and scaling down the execution of *all* their jobs), while leaving out the remaining tasks (and all their jobs). An interesting question is whether managing a subset of jobs from *every* task could further increase energy savings. Moreover, considering that the reliability of any scaled job is actually *enhanced* with the help of the scheduled recovery job [27], for real-time applications (e.g., ATR with multiple-channel satellite signal processing [22]) where the overall performance is limited by the task with the lowest quality, the investigation of the techniques to **improve the quality-of-assurance for all the tasks simultaneously** is warranted.

In this paper, considering different *quality of assurance requirements* (e.g., reliability enhancement requirements) of individual tasks, we study preemptive EDF-based *flexible* RAPM schemes for periodic real-time tasks. We develop schemes to manage a subset jobs for *every* task according to individual tasks' assurance requirements such that the quality of assurance for *all* tasks is improved *simultaneously*. 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 remainder of this paper is organized as follows. Section 2 presents a motivational example and Section 3 presents system models. Section 4 and Section 5 elaborate on the static and dynamic flexible RAPM problems, respectively. Simulation results are presented and discussed in Section 6 and Section 7 concludes the paper.

2 Motivational Example

To illustrate various trade-off dimensions in the RAPM problem, we consider a motivational example. Consider 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 found to be 0.5 (i.e. 50%). The slack can be used for both energy and reliability management. In the task-level static RAPM scheme [29], for any task that is selected for management, a *recovery task* will be created with the same timing parameters as the managed task. That is, a separate recovery job will be created for all the jobs of the *managed* tasks.

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 [29]. Figure 1a shows the schedule in 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). The recovery job associated with $J_{q,r}$ is assumed to have the same WCET and is denoted by $B_{q,r}$.

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 [29]. However, such a *greedy* task selection 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*.

Instead of managing exclusively 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 τ_4 and τ_5. Figure 1b shows the schedule within the interval considered, after a *judicious* selection of jobs to be managed for *each* task. 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 figures are *simultaneously* enhanced for all the tasks.

3 System Models

3.1 Application and Task Models

We consider applications with 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. We assume preemptive Earliest-Deadline-First (EDF) policy for scheduling the periodic tasks. It is assumed that c_i is given under the maximum processing frequency f_{max}, and 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. The j'th job $J_{i,j}$ of task τ_i arrives at time $(j-1) \cdot p_i$ and has the deadline of $j \cdot p_i$ ($j \geq 1$).

3.2 Energy Model

We adopt the system-level power model where the power consumption of the computing system considered is given by [30, 31]:

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

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 [5] 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.

By setting the derivative of Equation 1 to zero, a minimal *energy-efficient frequency* f_{ee} below which DVFS ceases to be energy-efficient, can be obtained [30]. 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[1] from f_{ee} to f_{max}.

3.3 Fault and Recovery Models

Considering that *transient* faults occur much more frequently than *permanent* faults [7, 13], especially with the continued scaling of CMOS technologies and reduced design margins [10], we focus on transient faults in this paper. At the end of jobs' execution, the transient fault is detected using *sanity* (or *consistency*) checks [20]. For jobs with recovery job being scheduled, should a transient fault be detected, the system's state is restored to a previous safe state and the recovery job is executed. Note that this approach exploits the temporal redundancy, falls along the lines of backward error recovery techniques [20], and was adopted in previous works as well [2, 19, 26]. The recovery job may take the form of *re-execution* of the job or a functionally comparable, alternative recovery block [2]. The results of this paper would remain valid, as long as the worst-case execution time of the recovery job does not exceed that of the (main) job.

Assuming that transient faults follow Poisson distribution [26], the average transient fault rate for systems running at frequency f (and corresponding supply voltage) can be modeled as [30]:

$$\lambda(f) = \lambda_0 \cdot g(f) \quad (2)$$

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, we have $g(f) > 1$ for $f < f_{max}$ [30].

3.4 Problem Description

In this work, we develop a flexible RAPM framework that attempts to save energy while preserving and enhancing reliability of *every* task, as specified by the quality of *assurance requirements* (defined as the number of jobs should be managed). Following the idea in the skip task model [6, 16], we use a single skip parameter k_i to present the assurance requirement for task τ_i. Specifically, for the purpose of enhancing system reliability, $(k_i - 1)$ out of any consecutive k_i jobs of task τ_i

[1]For discrete frequency levels, we can use two adjacent levels to emulate the execution at any frequency [12].

need to have recovery jobs. Here, k_i can range from 1 to ∞. 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 given as $k_1 = k_2 = k_3 = 3$ and $k_4 = k_5 = 2$.

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 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.**

Considering the assurance requirements of tasks, the *manageability* of a task set can be defined as the existence of a schedule in which all the required recovery jobs can be accommodated within the timing constraints. For task sets with system utilization $U \leq 0.5$, the spare capacity ($sc = 1 - U$) will be large enough to schedule a recovery task for every task [29], regardless of different assurance requirements for tasks.

However, without taking the assurance requirements of tasks into consideration, scheduling a recovery task for every task may not be the most energy efficient approach. 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.

Problem Statement In this work, for a set of periodic real-time tasks with assurance requirements where $AU \leq 1$, the problems to be addressed are: **a.) how to effectively exploit the 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, for each task, considering the assurance requirements, the subset of jobs to which recovery jobs 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 a given 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 optimal execution 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 periodic task systems scheduled by preemptive EDF [4, 15]. Then, the analysis is extended to the flexible RAPM framework.

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, 15]) *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:

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

where

$$a = \left\lceil \frac{t_1}{p_i} \right\rceil + 1 \quad (5)$$

$$b = \left\lfloor \frac{t_2}{p_i} \right\rfloor \quad (6)$$

$$x(i,j) = (j-1) \bmod k_i \quad (7)$$

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.

The recovery jobs introduce additional computational demand that must be taken into consideration when assessing the feasibility. Following similar reasoning as 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 SP as $LCM(k_1 p_1, \ldots, k_n p_n)$, where the $LCM()$ function denotes the *least common multiple (LCM)* of its arguments. 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 $0 \leq t_1, t_2 \leq SP$. It is necessary and sufficient to evaluate this function only at time points that are period boundaries of tasks [2, 4].

If the workload is feasible with the given recovery patterns, additional slack may still exist in the schedule and this can be exploited to scale down the jobs with recoveries to save energy. In addition, jobs without *statically* scheduled recoveries will have the default speed of f_{max}, but these too can reclaim dynamic slack at run time for reliability preservation and energy savings (Section 5).

Considering the scaled execution of managed jobs, the *augmented processor demand* can be written as:

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

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:

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

subject to

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

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

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

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

where the first condition corresponds to the quality of assurance requirements expressed through recovery patterns, the second and third condition state 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 should be schedulable.

4.2 Intractability of the Static Problem

For a real-time task τ_i with assurance requirement k_i, there are k_i different recovery patterns. Therefore, the number of different combinations of tasks' recovery patterns is $\prod_{i=1}^{n} k_i$, for a given task set with n tasks. 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.* ■

Due to the space limitations, the proof of the theorem is omitted and can be found in [33]. Moreover, the report contains more deliberation and detailed comparison of this result to other related intractability results [16, 21]. 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

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 [5], to minimize the energy consumption, the managed 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:

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

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

An Absolute Upper Bound As shown in [29], 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.

K-Upper Bound with Assurance Parameters 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, the workload for the jobs that need recoveries 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 real-time scheduling literature addressing the skip model, the *"deeply-red"* execution pattern has been frequently adopted [6, 16]. In fact, if a task set is schedulable under the deeply-red execution pattern, it will be schedulable for any other execution patterns. Also, with the deeply-red pattern, only the intervals that start at time 0 and end at a time instance no larger than $LCM(p_i, \ldots, p_n)$ need to be considered for processor demand evaluation (as opposed to the super-period SP).

In a similar vein, in this work, we will adopt the *"deeply-red"* recovery patterns. Specifically, a deeply-red recovery pattern is defined as the one with leading 1's followed by a single 0. Following the same line of reasoning as in [6, 16], and using the augmented processor demand function $APD()$ defined in Equation (4), we can obtain:

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 $APD(0, L) < L$, additional slack exists and it 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 result in pseudo-polynomial time complexity.

In 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 may "clash" in time (i.e. may need to be scheduled during the same time interval). 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 [11]. 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 [29], an effective dynamic slack management mechanism, called *wrapper-task* approach, has been studied for periodic tasks. In that scheme, wrapper tasks are used to represent dynamic slack generated at run-time. A primary feature of the scheme is that the slack reserved for recovery blocks is preserved across preemption points during the execution of the slack reclaiming algorithm: this is essential for reliability preservation in every RAPM scheme.

We have extended the wrapper task approach to the flexible RAPM framework. The detailed discussion of the algorithm is omitted due to space limitations and the interested readers are referred to [29, 33] for more details. However, we would like to emphasize that the dynamic slack reclamation through the management of wrapper tasks will not cause any timing constraint violation.

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 flexible static RAPM scheme (**Flexible**) where all tasks have 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, as an example to the task-level static RAPM scheme, we consider the one with smaller-utilization-task-first (**SUF**) heuristic, which is shown to have very good performance [29].

The parameters employed in the simulations are similar to the ones used in [29]. 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 $f_{ee} = 0.29$ [30]. 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 $d = 2$ [30]. 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 [18]. For the assurance requirements of tasks, we consider two different settings. In the first setting, all tasks have the same assurance requirement (e.g., $k = 2$). In 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, the reliability of any task that assumes recovery jobs will be improved [27]. 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. By considering the NPM scheme as the baseline, the *reliability improvement* of a task τ_i under a scheme S can be defined as:

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

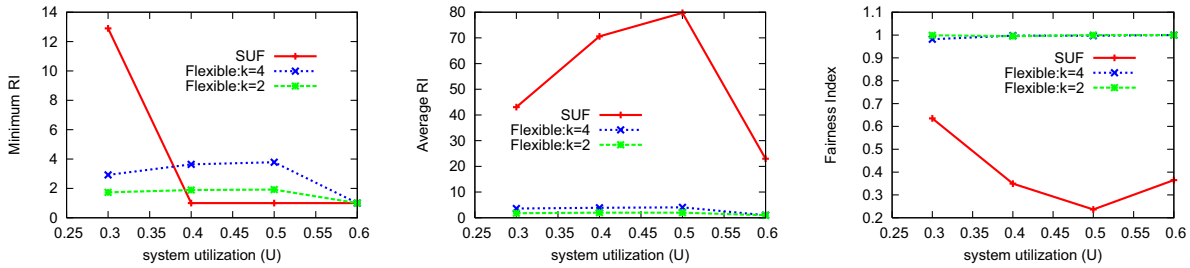
That is, larger $RI_i(S)$ values indicate better reliability improvement. Moreover, to quantify the fairness on reliability improvement to tasks, following the idea in [14], the *fairness index* of a scheme S is defined as:

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

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.

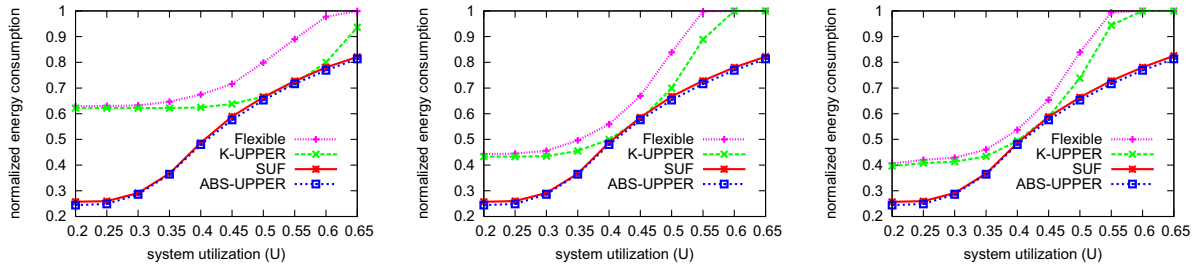
In the first set of experiments, we consider task sets with 10 tasks that have the same assurance parameter k. Figure 2 shows the reliability improvements and the fairness index for the static schemes. In the figures, "Flexible:k=i" means that all tasks have the same assurance parameter $k = i$ in the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

For applications where the system reliability is determined by the task with *lowest* reliability, Figure 2a shows the *minimum* reliability improvement among all 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 manages 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



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

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



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

Figure 3. Normalized energy consumption for the static schemes.

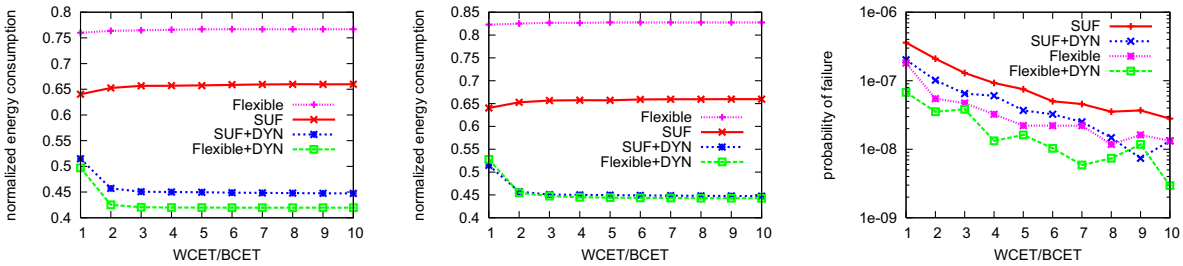
improvement. For the flexible scheme, the minimum reliability improvement of the tasks is rather stable and directly related to the assurance parameter k. For example, when $k = 4$, only 1 out of 4 jobs will not have a recovery job for each task and, compared to NPM, the reliability improves approximately by a factor of 4. 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 always guarantee the assurance requirements for all the tasks (e.g., when $AU > 1$).

If the overall system reliability depends on the total number of successfully executed jobs in the entire task set, Figure 2b shows that the average reliability improvement of the tasks under SUF is better than the flexible RAPM scheme. Indeed, 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 conclude that the task-level SUF scheme should be used if the overall system reliability depends on the average behavior of tasks. However, if the system reliability is limited by the task with lowest reliability improvement, or fairness is targeted 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 [29]. 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 the scheduling 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 assumes 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 between the flexible RAPM scheme and SUF becomes smaller.

Therefore, we can conclude that the flexible static RAPM scheme can guarantee such quality of assurance requirements and/or provide *fairness* to tasks, but at the cost of slightly increased energy consumption. Moreover,



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 $U = 0.5$.

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, the augmented dynamic algorithm [29] 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").

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].

Figure 4 shows the performance improvement of the dynamic scheme over static schemes on both energy and reliability, when $U = 0.5$. Similar results are obtained for other utilization values and are omitted due to space limitation. 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 online removal of statically scheduled recovery jobs when there is no error during the execution of their corresponding scaled jobs. From 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 (up to 33% for the flexible scheme and 20% for SUF). However, the performance improvement is rather stable after $\frac{WCET}{BCET} \geq 3$. This is because, 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 of the energy performance between the static schemes (from 10% to 15% for the cases considered) has effectively disappeared with the dynamic extension (only around 2%). The reason is that, although the managed jobs and their scaled frequency are limited under the flexible RAPM scheme, the slack generated from the removal of statically scheduled recovery jobs under the dynamic algorithm can be re-used to manage more jobs and/or to further scale down the execution of managed jobs for more energy savings. Therefore, although the static flexible RAPM scheme itself may perform worse than task-level SUF scheme in terms of energy savings, the dynamic version can recuperate its energy inefficiency while still 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. 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

In this paper, we presented a *flexible* reliability-aware power management (RAPM) framework for periodic tasks with variable assurance requirements. Extending the existing RAPM frameworks (that manage all the jobs of the selected tasks at the expense of some other unselected tasks), the main tenet of the work is to provide quality of assurance guarantees to all the tasks by considering their individual assurance requirements. We showed that the problem, in general, is NP-Hard in the strong sense. Then, we proposed static and dynamic schemes that are experimentally shown to perform successfully to achieve energy savings and improve reliability.

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

[6] 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.

[7] X. Castillo, S. McConnel, and D. Siewiorek. Derivation and caliberation of a transient error reliability model. *IEEE Trans. on computers*, 31(7):658–671, 1982.

[8] A. Ejlali, M. T. Schmitz, B. M. Al-Hashimi, S. G. Miremadi, and P. Rosinger. Energy efficient seu-tolerance in dvs-enabled real-time systems through information redundancy. In *Proc. of the Int'l Symposium on Low Power and Electronics and Design (ISLPED)*, 2005.

[9] 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.

[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. *IEEE Micro*, 24(6):10–20, 2004.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] P. Pop, K. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proc. of the 5th IEEE/ACM Int'l Conference on Hardware/software codesign and System Synthesis (CODES+ISSS)*, pages 233–238, 2007.

[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] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] D. Zhu, X. Qi, and H. Aydin. Energy management for periodic real-time tasks with variable assurance requirements. Technical Report CS-TR-2008-007, Dept. of Computer Science, UTSA, 2008. available at http://www.cs.utsa.edu/~dzhu/papers/tr-08-007.pdf.