Chapter 1

Reliability-Aware Power Management for Real-Time Embedded Systems

Dakai Zhu

The University of Texas at San Antonio

Hakan Aydin

George Mason University

1.1 Introduction

The performance of modern computing systems has steadily increased in the past decade thanks to the ever-increasing processing frequencies and integration levels. However, such performance improvements have resulted in drastic increases in the power consumption of computing systems and promoted energy to be a first-class system resource. Hence, *power-aware computing* has become an important research area and several hardware and software power management techniques have been proposed. The common strategy to reduce power consumption in a computing system is to operate system components at low-performance (thus, low-power) states, whenever possible.

As one of the popular and widely exploited power management techniques,

dynamic voltage and frequency scaling (DVFS) reduces the processor energy consumption by scaling down the supply voltage and processing frequency simultaneously [46, 48]. However, executing an application at lower processing frequencies (that is, at lower speeds) normally increases the computation time. Consequently, for real-time embedded systems with stringent timing constraints, special provisions are needed to avoid deadline misses when DVFS is employed to save energy. For various real-time task models, a number of power management schemes have been proposed to minimize the energy consumption while meeting the deadlines [4, 10, 12, 31, 36, 37].

Reliability has been a traditional requirement for computer systems. During the operation of a computing system, both *permanent* and *transient faults* may occur due to, for instance, the effects of hardware defects, electromagnetic interferences or cosmic ray radiations, and result in system *errors*. In general, fault tolerance techniques exploit *space* and *time redundancy* [33] to detect and possibly recover from system errors caused by various faults. It has been shown that transient faults occur much more frequently than permanent faults [9, 23], especially with the continued scaling of CMOS technologies and reduced design margins for higher performance [19]. For transient faults, which will be the focus of this chapter, the *backward error recovery* is an effective fault tolerance technique. This technique restores the system state to a previous *safe state* and repeats the computation when an error has been detected [33].

Until recently, energy management through DVFS and fault tolerance through redundancy have been studied independently in the context of real-time systems. However, there is an interesting trade-off between system energy efficiency and reliability as both DVFS and backward recovery techniques are based on (and compete for) the active use of the available CPU time (also known as *slack*). Moreover, it has been recently shown that DVFS has a direct and adverse effect on the transient fault rates, especially for those induced by cosmic ray radiations [15, 19, 61], which further complicates the problem. Therefore, for safety-critical real-time embedded systems (such as satellite and surveillance systems) where both reliability and energy efficiency are important, *reliability-aware power management (RAPM)* becomes a necessity. A number of reliability-aware power management schemes have been recently developed by the research community to address the negative effects of DVFS on system reliability. These schemes, which are the main focus of this chapter, typically guarantee system reliability requirements by scheduling proper recovery tasks while still saving energy with the remaining system slack.

In this chapter, we first present system models and state our assumptions (Section 1.2). Then, the fundamental idea of reliability-aware power management is introduced in the context of a single real-time task (Section 1.3.1). For systems with multiple real-time tasks that share a common deadline, the reliability-aware power management framework with individual recovery tasks is presented (Section 1.3.2), followed by the scheme that adopts a shared recovery task (Section 1.3.3). For general periodic real-time tasks

that have different deadlines, the task-level recovery technique is first introduced (Sections 1.4.1); then the utilization-based RAPM schemes for the earliest deadline-first (EDF) algorithm and priority-monotonic RAPM schemes for the rate-monotonic scheduling (RMS) algorithm are discussed (in Sections 1.4.2 and 1.4.3, respectively). The dynamic RAPM schemes that exploit dynamic slack generated at runtime for better energy savings are also covered (Section 1.5). We further discuss the RAPM schemes for energy-constrained systems aiming at maximizing system reliability (Section 1.6). We provide an overview of other related studies and identify some open research problems (Section 1.7). At the end, a brief summary concludes this chapter (Section 1.8).

1.2 Background and System Models

In this chapter, we consider a single processor system with DVFS capability. We consider the problems of how to minimize energy consumption without sacrificing system reliability, and to maximize system reliability with a given energy budget, while guaranteeing the timing constraints of real-time tasks. To better characterize the settings and define the scope of the discussion, in what follows, we first present task, system power and fault models and state our assumptions.

1.2.1 Real-Time Tasks

In real-time systems, applications are generally modeled by a set of *tasks*, which arrive periodically and need to finish their executions before a certain *deadline* after their arrivals [29]. More specifically, we consider an application that consists of a set of n independent real-time tasks $\Gamma = \{T_1, \ldots, T_n\}$. The task T_i is characterized by a pair (c_i, p_i), where c_i denotes its worst-case execution time (WCET) and p_i represents its period (which coincides with its relative deadline). That is, task T_i generates an infinite number of task instances (also called *jobs*) and the inter-arrival time of two consecutive jobs of T_i is p_i. Once a job of task T_i arrives at time t, it needs to execute (in the worst case) for c_i time units before its deadline $t + p_i$ (which is also the arrival time of the task T_i's next job).

Given that the system under consideration adopts a variable-frequency processor, we assume that the WCET c_i of task T_i is obtained under the maximum processing frequency f_{max} of the processor. Note that, a number of studies have indicated that the execution time of tasks may not always scale linearly due to memory accesses or I/O operations [6, 40]. However, to simplify our discussions, in this chapter, the execution time of a task is assumed to scale *linearly* with the processing frequency. That is, at the scaled processing

frequency f ($\leq f_{max}$), task T_i will take (in the worst-case) $\frac{c_i \cdot f_{max}}{f}$ time units to complete its execution.

1.2.2 Dynamic Voltage and Frequency Scaling

In many execution scenarios, real-time tasks may complete their executions well before their deadlines and leave the processor idle. Such idle time can be exploited through the *dynamic power management (DPM)* technique by putting the system to low-power sleep states to save energy. However, a more preferable strategy is to execute tasks at low processing frequencies and complete them just in time before the deadlines. This is due to the fact that the dynamic power P_d normally dominates in processors. P_d is linearly related to processing frequency f and quadratically related to the supply voltage V_{dd} (that is, $P_d \approx f \cdot V_{dd}^2$) [7]. Moreover, the operating frequency for CMOS circuits is almost linearly related to the supply voltage. Hence, the dynamic power becomes essentially a convex function of the processor frequency when applying DVFS [46] where the supply voltage and frequency are adjusted simultaneously. In modern processors, such frequency changes can be performed quite efficiently (in a few cycles or microseconds [2, 13]). Therefore, we assume that the overhead for DVFS is negligible (or such overhead can be incorporated into the WCETs of tasks). Note that we will use the term *frequency change* to stand for scaling both processing frequency and supply voltage in the rest of this chapter.

In addition to dynamic power, another component in system power consumption is leakage power, which becomes increasingly important due to scaled technology size and increased levels of integration [25]. Moreover, it is necessary to consider all power-consuming system components (such as memory [27]) in an effective power management framework. Several such system-level power models have been recently proposed [12, 22, 25]). In this chapter, we adopt a simple *system-level power model*, where the system power consumption $P(f)$ of a computing system at processing frequency f is given by [61]:

$$P(f) = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{ef} f^m) \tag{1.1}$$

Here, P_s denotes the *static power*, which includes the power to maintain basic circuits and to keep the clock running. It can be removed only by powering off the whole system. When there is computation in progress, the system is *active* and $\hbar = 1$. When the system is turned off or in power-saving sleep modes, $\hbar = 0$.

P_{ind} is the *frequency-independent active power*, which corresponds to the power that is independent of processing frequency. For simplicity, unless specified otherwise, P_{ind} is assumed to be constant and same for all the tasks; it can be efficiently removed (typically, with acceptable overhead) by putting the system components (e.g. main memory) into sleep state(s) [27]. P_d is the *frequency-dependent active power*, which includes the processor's dynamic

power and *any* power that depends on the system processing frequency f (and the corresponding supply voltage) [7, 27]. The effective switching capacitance C_{ef} and the dynamic power exponent m (in general, $2 \leq m \leq 3$) are system-dependent constants [7]. Despite its simplicity, this power model captures the essential components of an effective system-wide energy management framework.

The energy-efficient frequency: Since *energy* is the integral of power over time, the energy consumed by a task executing at constant frequency f ($\leq f_{max}$) is $E(f) = P(f) \cdot t(f) = P(f) \cdot \frac{c \cdot f_{max}}{f}$. Here. $t(f) = \frac{c \cdot f_{max}}{f}$ denotes the execution time of the task at frequency f. From the power model in Equation (1.1), intuitively, executing the task at lower processing frequencies can result in less energy consumption due to the frequency-dependent active power P_d. However, at lower frequencies, the task needs more time to complete its execution and thus consumes more energy due to the static and frequency-independent active power. Therefore, there exists a minimal *energy-efficient frequency* f_{ee} below which a task starts to consume more system energy [22, 25]. Considering the prohibitive overhead of turning on and off a system (e.g., tens of seconds), we assume that the system is on for the operation interval considered and P_s is always consumed. By differentiating $E(f)$ with respect to f and setting it to 0, we can find that [61]:

$$f_{ee} = \sqrt[m]{\frac{P_{ind}}{C_{ef} \cdot (m-1)}} \cdot f_{max} \tag{1.2}$$

Consequently, for energy efficiency, the frequency f to execute any task should be limited to the range $f_{ee} \leq f \leq f_{max}$. Moreover, to simplify the discussion, we assume that *normalized* frequencies are used and that the maximum frequency is $f_{max} = 1$.

1.2.3 Transient Faults and Backward Recovery

Unlike *crash failures* that result from permanent faults, a *soft error* that follows a transient fault typically do not last for long and disappears when the computation is repeated. With the aim of tolerating transient faults, we exploit temporal redundancy (that is, system slack) and employ *backward recovery* techniques to tolerate soft errors. More specifically, when soft errors due to transient faults are detected by, for example, *sanity* (or *consistency*) checks at the completion time of a task, a recovery task is dispatched in the form of re-execution [33]. The overhead for error detection is assumed to be incorporated into tasks' worst-case execution times.

Traditionally, transient faults have been modeled through a Poisson process with an average arrival rate of λ [49, 51]. With the continued scaling of technology sizes and reduced design margins, it has been shown that DVFS has a direct and negative effect on the arrival rate λ due to the increased number of transient faults (especially the ones induced by cosmic ray radiations)

at lower supply voltages [15, 19]. For systems with a DVFS-enabled processor, the average rate of soft errors caused by transient faults at scaled processing frequency f (and the corresponding supply voltage) can be expressed as [61]:

$$\lambda(f) = \lambda_0 \cdot g(f) \tag{1.3}$$

where λ_0 is the average error rate corresponding to the maximum processing frequency f_{max}. That is, $g(f_{max}) = 1$. With reduced processing frequencies and supply voltages, the average error rate generally increases and $g(f) > 1$ for $f < f_{max}$. In other words, $g(f)$ is a non-increasing function of the processor frequency f.

Exponential fault rate model: The radiation-induced transient faults in semiconductor circuits have been known and well studied for decades [67]. When high-energy particles strike a sensitive region in semiconductor devices, a dense track of electron-hole pairs are deposited, which can accumulate and exceed the minimum charge (i.e., the *critical charge*) required to flip the value stored in a memory cell [21], or be collected by pn-junctions via drift and diffusion mechanisms to form a current pulse and cause a logic error [26]. However, it has been a great challenge to model such soft errors caused by transient faults considering the various factors, such as cosmic ray flux (i.e., number of particles per area), technology feature size, chip capacity, supply voltage and operating frequency [39, 41, 68].

In general, when the supply voltage of a semiconductor device decreases, the critical charge becomes smaller, which can lead to exponentially increased transient fault rates [21, 41]. Such effects have been observed on both processors [38] and memory subsystems [68]. Moreover, in addition to high-energy particles such as cosmic rays, which are more likely to cause transient faults in circuits with smaller critical charge, lower-energy particles do also exist, and in much larger quantities [67]. Therefore, considering the number of particles in the cosmic rays and the relationship between transient fault rate, critical charge and supply voltage, an *exponential* rate model for soft errors caused by transient faults has been derived as [61]:

$$\lambda(f) = \lambda_0 \cdot g(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{ee}}} \tag{1.4}$$

In the expression above, the exponent d (> 0) is a constant which indicates the sensitivity of the rate of soft errors to DVFS. The maximum error rate at the minimum energy efficient processing frequency f_{ee} (and corresponding supply voltage) is assumed to be $\lambda_{max} = \lambda_0 10^d$. For example, when $d = 3$, the average rate of soft errors at the minimum frequency can be 1000 times higher than that at the maximum frequency.

1.3 Reliability-Aware Power Management

The preceding discussion shows that when the processing frequency (and supply voltage) of a task is scaled down through DVFS to save energy, the probability of incurring soft errors due to transient faults during the scaled execution will drastically increase, which in turn leads to significantly-reduced reliability. To compensate such reliability loss at lower supply voltage, special provisions are needed when DVFS is employed. Suppose that the *original reliability* of a system denotes the probability of successful operation without incurring any errors caused by transient faults for a given interval when there is no power management (i.e. when all tasks are executed at the maximum processing frequency f_{max}). With the goal of preserving system's original reliability in the presence of DVFS, we now present the details of the *reliability-aware power management (RAPM)* framework.

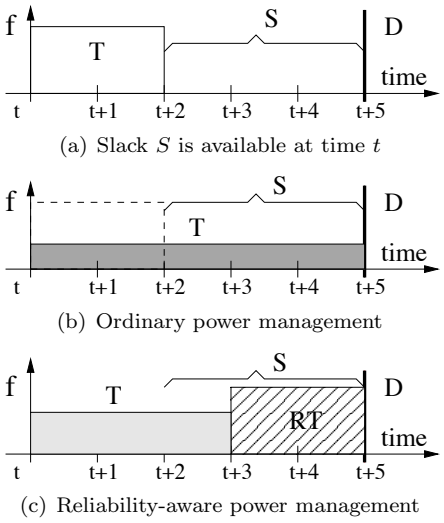
Specifically, we first introduce the fundamental idea of reliability-aware power management in the context of a single real-time task. The approach involves scheduling an additional recovery task to recuperate reliability loss induced by power management (DVFS). Next, for systems with multiple real-time tasks that share a common deadline/period, the RAPM scheme with individual recovery tasks is discussed, followed by the scheme with a single shared recovery task. The RAPM schemes for general periodic tasks with different deadlines will be presented in Section 1.4.

1.3.1 The Case with a Single Real-Time Task

For systems with a single real-time task T, let R^0 denote the *original* reliability of an instance of task T when it runs at the maximum frequency f_{max}. R^0 is the probability of successfully completing the execution of the task instance without incurring soft errors caused by transient faults. Since system reliability depends on the correct execution of every instance of the task, to achieve the objective of maintaining the system's original reliability, we can preserve the original reliability R^0 for each instance of task T.

The central idea behind RAPM is to schedule a recovery task using the available slack before exploiting the slack for DVFS to save energy [55, 56]. As a concrete example, we consider a single task T that has the worst-case execution time c and period p as 2 and 5, respectively. Suppose that an instance of task T arrives at time t, which needs to complete its execution by time $t + 5$. As shown in Figure 1.1(a), we can see that there are $S = 3$ units of slack available.

Without paying special attention to system reliability, the *ordinary* power management would utilize all the available slack to scale down the processing frequency of the task instance through DVFS for maximum energy savings [4, 46]. Therefore, the scaled frequency for the task instance under the ordinary



(a) Slack S is available at time t

(b) Ordinary power management

(c) Reliability-aware power management

FIGURE 1.1: Ordinary and reliability-aware power management schemes.

power management will be $f = \frac{2}{2+3} \cdot f_{max} = 0.4$ (recall that $f_{max} = 1$), as shown in Figure 1.1(b). In the figures, the X-axis represents time, the Y-axis represents processing frequency (e.g., cycles executed per time unit), and the area of the task box defines the workload (e.g., number of cycles) of the task instance. By focusing on active power and assuming that $P_{ind} = 0.1, C_{ef} = 1$ and $m = 3$ [7], from Equation (1.1), we can easily find that the ordinary power management can save 63% of the *active energy* when compared to that of *no power management (NPM)* where all tasks run at the maximum frequency.

However, as discussed earlier, with reduced processing frequency and supply voltage, the execution of task T is more susceptible to errors caused by transient faults [19, 61]. Suppose that the exponent in the exponential fault rate model is $d = 2$ (see Equation (1.4)). At the reduced frequency $f = 0.4$, the probability of incurring errors due to transient fault(s) during T's execution can be found as:

$$\begin{aligned} & \rho_1 = 1 - R_1 = 1 - e^{-\lambda_0 10^{\frac{d(1-f)}{1-f_{ee}}} \frac{c}{f}} \\ = \;& 1 - e^{-\lambda_0 10^{\frac{d(1-f)}{1-f_{ee}}} \frac{c}{f}} = 1 - e^{-\lambda_0 c 10^{\frac{2(1-0.4)}{1-0.37}} \frac{1}{0.4}} \\ \approx \;& 1 - (R^0)^{200} = 1 - (1-\rho^0)^{200} \approx 200\rho^0 \end{aligned} \tag{1.5}$$

Here, the minimum energy-efficient frequency can be found as $f_{ee} = 0.37$. R^0 is the original reliability of task T and ρ^0 $(= 1 - R^0)$ is the corresponding *probability of failure* of task T's scaled execution. Since ρ^0 is in general a very small number (usually $< 10^{-4}$), we can see that, executing the task instance of T at scaled frequency $f = 0.4$ can lead to approximately 200 times higher probability of failure figures. Such increase in the probability of

failure during the execution of individual tasks will degrade the overall system reliability, which is unbearable, especially for safety-critical systems where the requirement for high levels of reliability is strict.

Instead of using all the available slack for DVFS to save energy, we can also reserve part of the available slack for temporal redundancy to re-execute the tasks incurring soft errors and thus increase system reliability [33]. Hence, the central idea of *reliability-aware power management* is to reserve part of the available slack and schedule a recovery task, which can recuperate the reliability loss due to energy management. Then, the remaining slack can be utilized by DVFS to scale down the execution of tasks for energy savings. For simplicity, we assume that the recovery task takes the form of *re-execution* and has the same size of the task to be recovered.

For the above example, 2 units of the available slack can be reserved for scheduling a recovery task RT, as shown in Figure 1.1(c). The remaining 1 unit of slack can be utilized by DVFS to scale down the processing frequency of task T to $f = \frac{2}{2+1} = 0.67$. Note that, the recovery task RT will be invoked only if the scaled execution of task T is subject to a soft error caused by transient faults. Without considering the energy consumption of the recovery task, executing task T at the scaled frequency $f = 0.67$ could yield 26% energy savings when compared to that of no power management. Moreover, since the recovery task takes the form of re-execution of its primary task, it will be executed at the maximum frequency $f_{max} = 1$ to ensure that there is no deadline miss.

With the additional recovery task RT, the reliability R of task T will be the summation of the probability of the primary task T being executed correctly and the probability of incurring errors due to transient faults during task T's execution while the recovery task RT is executed correctly. With the assumption that the recovery task RT is essentially the re-execution of task T at the maximum frequency f_{max}, its probability of successful execution will be $R^0 = e^{-\lambda_0 c}$. Thus, we have:

$$R = e^{-\lambda(f)S} + \left(1 - e^{-\lambda(f)S}\right) \cdot R^0 > R^0 \tag{1.6}$$

where $\lambda(f)$ is the average rate of soft errors due to transient faults at the reduced frequency f. From the above equation, we can see that, the resulting reliability for task T under RAPM is always better than its original reliability R^0.

Therefore, when the amount of available slack is larger than the worst-case execution time of a task, the RAPM scheme can reserve part of the slack to schedule a recovery task while using the remaining slack for DVFS to save energy. With the help of the recovery task, which is assumed to take the form of re-execution at the maximum frequency in case the scaled execution of the primary task fails, the RAPM scheme guarantees to preserve a real-time task's original reliability while still obtaining energy savings using the remaining slack, regardless of different error rate increases (i.e., different values of d) and the scaled processing frequency [55, 56].

1.3.2 Multiple Tasks with a Common Deadline

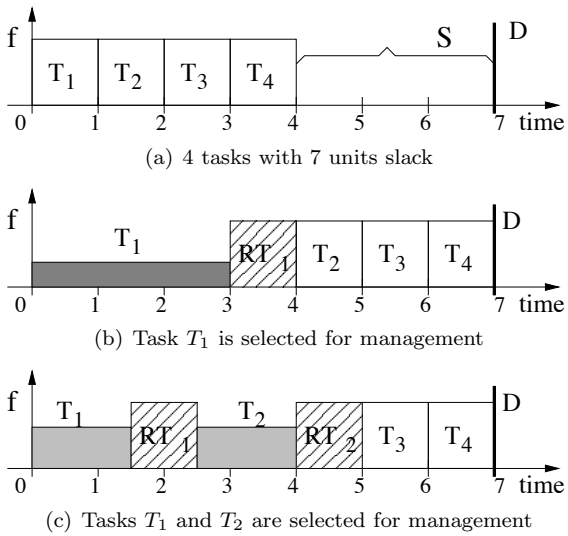
When there are multiple real-time tasks, the reliability-aware power management problem gains a new dimension. Here, we need to allocate the available slack to multiple tasks, *possibly in different amounts*, to maximize the energy savings while preserving system reliability. We start our analysis with systems where all tasks have the same period, which have been denoted as *frame-based task systems* [35]. In such systems, the common period is considered as the *frame* of the application. Due to their periodicity, we consider only the tasks within one frame for such systems, where all tasks arrive at the beginning of a frame and need to complete their executions by the end of the frame (that is, the common deadline).

Recall that the reliability of a system generally depends on the correct execution of all its tasks. Although it is possible to preserve the original reliability of a given system while sacrificing the reliability of some individual tasks, for simplicity, we focus on preserving the original reliability of each task to guarantee the system original reliability. From the above discussions, we know that, for any task whose execution is scaled down, the reliability-aware power management can schedule a recovery task to recuperate the reliability loss due to power management at the reduced supply voltage levels.

For ordinary power management that does not consider system reliability, the optimal scheme for obtaining the maximum energy savings is to allocate the available slack to all tasks proportionally and scale down their executions uniformly [3, 46]. The optimality comes from the convex relationship between the energy consumption of tasks and their processing frequencies. However, when system reliability is considered, the proportional slack allocation scheme may not be the most energy-efficient approach, especially for cases where the amount of available slack is not enough to accommodate a separate recovery task for each and every task.

An Example: We explain the key idea of the reliability-aware power management for systems with multiple real-tasks that share a common deadline, again, through a concrete example. Here, the system consists of four tasks, which have the same period of 7. If the worst-case execution time of each task is 1 time unit, 3 units of slack time exist within each frame as shown in Figure 1.2(a). It is not hard to see that the available slack is not sufficient to accommodate a separate recovery task for each of the four tasks and not all tasks can be scaled down under RAPM for energy savings.

If we adopt a greedy approach and allocate all the available slack to the first task T_1, a recovery task RT_1 (which takes 1 unit of slack) can be scheduled and the remaining 2 units of slack can be utilized to scale down the processing frequency of task T_1 to $f = \frac{1}{3}$, as shown in Figure 1.2(b). Assuming the same parameters as in in the power model in Section 1.3.1, simple calculation shows that about 22% energy savings can be obtained compared to that of no power management case. As explained earlier, the original reliability of task T_1 is



(a) 4 tasks with 7 units slack

(b) Task T_1 is selected for management

(c) Tasks T_1 and T_2 are selected for management

FIGURE 1.2: RAPM for multiple tasks with a common deadline.

preserved with the help of the recovery task RT_1. Moreover, as tasks T_2, T_3 and T_4 are executed at the maximum frequency f_{max}, their original reliabilities are preserved as well. Therefore, the overall system reliability is guaranteed to be no worse than the system original reliability.

However, the greedy strategy of allocating all available slack to one task may not be the most energy-efficient one and we can select more tasks for more energy savings. In fact, the available slack is not enough to slow down all four tasks. Moreover, if we select three tasks, all 3 units of available slack will be utilized for scheduling the recovery tasks, which leaves no slack for DVFS and no energy savings can be obtained. Instead, suppose that two tasks T_1 and T_2 are selected. After scheduling their recovery tasks RT_1 and RT_2 and using the remaining 1 unit of slack to scale down the processing frequency of tasks T_1 and T_2, we obtain the schedule shown in Figure 1.2(c). Here, the scaled frequency for tasks T_1 and T_2 is $f_{12} = \frac{2}{3}$ and the energy savings can be calculated as 28% – a significant improvement over the greedy approach. Observe that in this solution, the original system reliability is still preserved.

Optimal Task Selection: From this example, we can see that, one of the key problems in the reliability-aware power management for frame-based real-time tasks is to select an *appropriate* subset of tasks to apply DVFS. The execution of the selected tasks will be scaled down after reserving part of the slack to schedule a separate recovery task for each of them. The remaining tasks are left intact and they run at the maximum processing frequency in order to preserve their original reliability.

Intuitively, for a system with a given amount of available slack S, when

more tasks are selected, more slack needs to be reserved for the recovery tasks, which reduces the amount of slack for DVFS to scale down the selected tasks and thus reduce the energy savings. Similarly, if fewer tasks are selected, using the large amount of slack to scale down a few tasks may not be energy efficient either. Therefore, there should exist an optimal subset of selected tasks for which the energy savings are maximized. A natural question to ask is whether there exists a fast (that is, polynomial-time) solution to the problem of reliability-aware energy management for multiple tasks. Unfortunately, the answer is negative, as we argue below.

Let us denote the total workload (computation requirement) of all tasks by $L = \sum_{i=1}^{n} c_i$. The available slack is $S = D - L$, where D is the common deadline (also the frame of the task set). Without loss of generality, assume that a subset of tasks are selected, where the total computation time of the selected tasks is X. We have $X \leq L$ and $X \leq S$. After reserving X units of slack for recovery tasks, the remaining $S - X$ units of slack could be used to scale down the processing frequency for the selected tasks. Considering the convex relationship between energy consumption and processing frequency, the optimal solution to get the maximum energy savings can be obtained by uniformly scaling down the execution of the selected tasks. Therefore, the amount of *fault-free* energy consumption (without considering the execution of recoveries that are needed only with a small probability) will be:

$$\begin{aligned} E_{total} &= P_s \cdot D + S\left(P_{ind} + c_{ef} \cdot \left(\frac{X}{S}\right)^m\right) + \\ &\quad (L - X)(P_{ind} + c_{ef} \cdot f_{max}^m) \end{aligned} \tag{1.7}$$

where the first part represents the energy consumption due to the *static power*, which is always consumed during the operation. The second part is the energy consumption for the *selected tasks* and the third part is the energy consumption of the *unselected tasks*. Simple algebra shows that, to minimize E_{total}, the total size of the selected tasks should be equal to:

$$X_{opt} = S \cdot \left(\frac{P_{ind} + C_{ef}}{m \cdot C_{ef}}\right)^{\frac{1}{m-1}} \tag{1.8}$$

Hence, the value of X_{opt} can serve as a guideline for task selection to obtain the best energy savings. If $X_{opt} \geq L$, all tasks should be selected and scaled down uniformly for maximum energy savings. Otherwise (the case where $X_{opt} < L$), if there exist a subset of tasks such that the summation of their computation is *exactly* X_{opt}, selecting that subset would definitely be optimal.

However, finding a subset of tasks with exactly X_{opt} units of total computation time turns out to be NP-hard [57]. Note that, the reliability of any single selected task under RAPM is better than its original reliability with the help of its recovery task (Equation 1.6). To select as many tasks as possible, we can adopt an efficient task selection heuristic, namely the *smallest-task-first (STF)* scheme. That is, after obtaining X_{opt} for a given task set, we can find

the largest value of k, such that the total computation of the first k smallest tasks is no more than X_{opt}. The drawback of this heuristic is that, the difference between X_{opt} and the total computation of the selected tasks (denoted as *selection error*) can be large, which may result in less energy savings. As the second scheme, we can select tasks following the *largest-task-first (LTF)* heuristic. That is, the tasks can be selected by processing them in decreasing order of their size, as long as the selection error is larger than the current task. Hence, LTF can ensure that the selection error is bounded by the size of the smallest task. Interested readers are referred to [57] for further details.

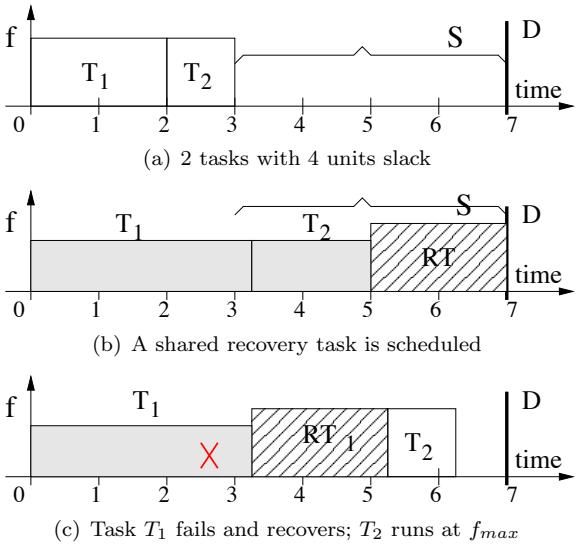
The results of performance evaluation through extensive simulations, given in [55, 56, 57], show that the RAPM technique is able to preserve (even *improve*) the overall system reliability, while the ordinary (but reliability-ignorant) power management technique can result in reliability degradations of several orders of magnitude. Moreover, RAPM schemes can still obtain up to 40% energy savings where the gains tend to be higher at lower workloads. However, since some amount of CPU time needs to be reserved for recoveries, the energy savings achieved by RAPM schemes are generally 20% to 30% lower compared to those of ordinary power management schemes.

1.3.3 Shared Recovery Technique

The probability of incurring an error due to transient faults during the execution of a task is rather low, even for scaled execution at low supply voltages. Hence, most of the recovery tasks will not be invoked at run-time. Moreover, the required slack for multiple separate recovery tasks reduces the prospects for energy savings with less available slack for DVFS. That is, the RAPM scheme with individual recovery tasks is, in some sense, *conservative*. In fact, when the execution of a scaled task completes successfully without incurring any error, its recovery task will not be activated and the corresponding slack time can immediately be made available for the *next* scaled task. Furthermore, to preserve the original reliability of a task, it is sufficient to ensure that, at the dispatch time of a task whose execution is scaled down, there is sufficient slack time for a recovery (to re-execute the task) at the maximum frequency.

The key idea of the *shared-recovery* based RAPM scheme is to reserve slack time for only one recovery block, which can be shared by all selected tasks at run-time [53]. Therefore, more slack can be left for DVFS to scale down the execution of the selected tasks and save more energy. Here, to ensure that the recovery block is large enough to re-execute any scaled task at the maximum frequency, its size needs to be the same as the largest selected task. As long as there is no error caused by transient faults during the execution of scaled tasks at run-time, the next scaled task can run at its pre-calculated low processing frequency. When an error is detected during the execution of a scaled task, the recovery will be triggered and the faulty task will be re-executed at the maximum frequency. After that, a *contingency* schedule is adopted in the sense

that all remaining selected tasks are executed at the maximum frequency until the end of the current frame to preserve the system reliability.



(a) 2 tasks with 4 units slack

(b) A shared recovery task is scheduled

(c) Task T_1 fails and recovers; T_2 runs at f_{max}

FIGURE 1.3: 2 tasks managed by the shared recovery technique.

The idea can be further illustrated with the following example. As shown in Figure 1.3(a), there are two tasks T_1 and T_2 that share a common period/deadline of 7. The worst-case execution times of T_1 and T_2 are 2 and 1, respectively. The system potentially has access to $7-3=4$ units of slack when both tasks execute at the maximum frequency. If we consider the individual-recovery based RAPM scheme, it can be found that selecting only task T_1 to utilize all slack (while task T_2 runs at the maximum frequency) will yield maximum energy savings. With the same parameters for the system power model as in Section 1.3.1, the energy savings can be calculated as 33% with respect to no power management scheme.

For the shared-recovery based RAPM scheme, only one recovery of size 2 units (the maximum size of tasks T_1 and T_2) is scheduled, which can be shared by both tasks. The remaining 2 units of slack can be used to scale down the execution of both tasks T_1 and T_2 at the frequency of $f = \frac{3}{5}$, as shown in Figure 1.3(b). Compared to the no power management scheme, it can be found that 40% energy can be saved, which is better than that of the individual-recovery based RAPM scheme.

Figure 1.3(c) further shows the scenario where, after the detection of an error due to transient faults, the shared recovery block is utilized to re-execute task T_1 at the maximum frequency. From previous discussions, we know that the reliability of task T_1 is no worse than its original reliability. Moreover, by switching to the contingency schedule after recovering the faulty task T_1

and enforcing task T_2 run at the maximum frequency, the original reliability of task T_2 is also preserved. To conclude, the shared-recovery based RAPM scheme can still preserve the original reliability of a task system [53].

Our evaluation results show that the energy savings performance of the shared-recovery based RAPM scheme is excellent – in fact, very close (within 5% difference for most cases) to that of the ordinary power management. Perhaps unexpectedly, the results also indicate that the RAPM scheme with a shared recovery task can offer non-trivial gains on the reliability side over individual-recovery based RAPM schemes as well [53]. This comes from the fact that, the shared-recovery based RAPM scheme provides a better protection for *all* single-fault scenarios that are typically much more likely than some multiple-fault scenarios that the individual-recovery based RAPM schemes provision for. Since optimizing the most common case is a well-known system design technique, overall reliability achieved by the shared-recovery technique tends to be better in most cases.

1.4 RAPM for Periodic Real-Time Tasks

For frame-based real-time tasks that share a common period/deadline, the available slack within a frame is accessible to all tasks, which allows the use of the shared-recovery based RAPM scheme. However, for general periodic real-time tasks with different periods, the active task instances (or jobs) of different tasks can have different deadlines, preventing them from sharing a common recovery task. In addition, the available system slack may also have different expiration times [4], which makes it difficult (if not impossible) to develop the shared-recovery based RAPM scheme for general periodic real-time tasks. Hence, in this section, we focus on the individual-recovery based RAPM schemes, where each task instance whose execution is scaled down will have a separate recovery task.

For periodic real-time tasks, the workload in a system is normally represented by *system utilization*. The utilization of a periodic real-time task T_i is defined as $u_i = \frac{c_i}{p_i}$, where c_i is its worst-case execution time and p_i is its period. The system utilization of a task set Γ with n tasks is correspondingly defined as $U = \sum_{i=1}^{n} u_i$. For a task set to be schedulable on a uniprocessor system, it is necessary to have $U \leq 1$ [29]. For task sets with $U < 1$, the spare processor capacity is denoted as $1-U$, which indicates the amount of available *static slack* in the system. In this section, we address the problem of exploiting spare processor capacity (that is, static slack) to maximize energy savings while guaranteeing the original reliability of the system under consideration.

1.4.1 Task-Level RAPM Techniques

Theoretically, it is possible to select only a subset of jobs of each task to obtain the maximum energy savings. That is, the jobs of a given task can be handled differently, where the execution of only a subset of selected jobs are scaled down (with a separate recovery job scheduled before the deadline for each), while the remaining jobs can run at the maximum frequency. However, such a job-oriented approach requires the consideration of all jobs within the *hyper-period* (defined as the least common multiple, LCM, of all tasks' periods) of the task set, which may be arbitrarily long. As a result, such an approach cannot be generally considered as a computationally efficient technique.

This section focuses on a particular efficient *task-level* RAPM technique, where all jobs of the same task will get the same treatment [58, 59, 65]. In particular, the jobs of all the unselected tasks will run at the maximum frequency for reliability preservation. Moreover, all the jobs of a given selected task T_k will be scaled down to the same low processing frequency f_k and each such job will need to have a recovery scheduled before its deadline to preserve its original reliability. To provide the recovery time needed by all the jobs of T_k, we can construct a *periodic recovery task (PRT)*, with the same timing parameters (WCET and period) as those of task T_k. By incorporating such a periodic recovery task into the task set, we can ensure that there is a recovery job scheduled before the deadline of each job of T_k.

Periodic RAPM Problem: Let us denote by Φ ($\subseteq \Gamma$) the subset of the selected tasks. A periodic recovery task will be constructed for each task T_k ($\in \Phi$) and the scaled frequency for task T_k is f_k ($< f_{max}$). To preserve reliability, the remaining (unselected) tasks run at the maximum frequency f_{max}. Assume that the *augmented* task set, which incorporates the newly created recovery tasks and the scaled execution of the selected tasks, is schedulable under a given scheduling policy. Without considering the energy consumed by recovery tasks (which normally have a small probability of being executed), the *fault-free* energy consumption within the hyper-period (LCM) of the task set is:

$$E(\Phi) = \sum_{T_i \in (\Gamma - \Phi)} \frac{LCM}{p_i} P(f_{max}) c_i + \sum_{T_k \in \Phi} \frac{LCM}{p_k} P(f_k) \frac{c_k f_{max}}{f_k} \tag{1.9}$$

where the first part represents the energy consumed by the *unselected* tasks and the second part corresponds to the energy consumed by the *selected* tasks. Hence, the RAPM problem for periodic real-time tasks can be stated as follows: for a given scheduling policy, find the subset Φ of tasks and their corresponding scaled frequencies to minimize $E(\Phi)$ while preserving the system original reliability and meeting all deadlines.

Not surprisingly, finding the optimal subset Φ and the scaled frequencies to minimize E_Φ is NP-hard [59]. In fact, if all tasks have the same period, the special case of the periodic RAPM problem becomes essentially the RAPM

problem for multiple tasks sharing a common deadline, which is NP-hard as discussed in Section 1.3.2. However, for different uniprocessor scheduling policies, such as preemptive *earliest-deadline first (EDF)* and *rate-monotonic-scheduling (RMS)*, there are different schedulability conditions that result in different task selection and frequency assignment strategies.

1.4.2 Utilization-Based RAPM for EDF Scheduling

Under preemptive EDF scheduling, a set of n periodic real-time tasks are schedulable on a uniprocessor system as long as the system utilization $U \leq 1$ [29]. The spare processor capacity is denoted as $sc = 1 - U$, which can be exploited for energy and reliability management. Suppose that the total utilization of the selected tasks in Φ is $X = \sum_{T_i \in \Phi} u_i < sc$. After incorporating the newly constructed periodic recovery tasks, the remaining spare processor capacity is $(sc - X)$, which can be used to scale down the execution of all jobs of the selected tasks to save energy. Again, due to the convex relation between power and processing frequency (see Equation 1.1), the optimal solution that minimizes system energy consumption will consist in uniformly scaling down all jobs of the selected tasks. The scaled processing frequency can be found as $f = \frac{X}{X+(sc-X)} = \frac{X}{sc}$. Note that, with the newly constructed recovery tasks and the scaled execution of the selected tasks, the system utilization of the augmented task set can be calculated as:

$$U' = (U - X) + \frac{X}{f} + X = U + \frac{X}{X/sc} = 1 \tag{1.10}$$

That is, the augmented task set is still schedulable under EDF scheduling.

Based on system utilization, Equation (1.9) can be re-written as:

$$\begin{aligned} E(\Phi) &= LCM \cdot P_s + LCM(U - X)(P_{ind} + c_{ef} \cdot f_{max}^m) \\ &\quad + LCM \cdot sc \left(P_{ind} + c_{ef} \cdot \left(\frac{X}{sc} \right)^m \right) \end{aligned} \tag{1.11}$$

where the first part denotes the energy consumption due to static power, the second part captures the active energy consumption of unselected tasks, and finally, the third part represents the active energy consumption of the selected tasks.

The formulation is similar to that of the case with multiple tasks sharing a common deadline (Section 1.3.2). We can find that when the total utilization of the selected tasks is $X_{opt} = sc \cdot \left(\frac{P_{ind} + C_{ef}}{m \cdot C_{ef}} \right)^{\frac{1}{m-1}}$, $E(\Phi)$ is minimized. Again, finding the optimal subset of tasks with total utilization equal to exactly X_{opt} turns out to be NP-hard. Following a similar reasoning, the efficient heuristics *largest-utilization-first (LUF)* and *smallest-utilization-first (SUF)* can be adopted when selecting tasks [58, 59].

1.4.3 Priority-Monotonic RAPM for Rate Monotonic Scheduling

In rate monotonic scheduling (RMS), tasks are assigned static priorities and the ones with smaller periods have higher priorities. Several feasibility tests have been proposed for task sets under RMS with different levels of accuracy and complexity. A simple well-known feasibility test is based on the system utilization of a task set. A task set is feasible as long as its utilization $U \leq LLB(n) = n(2^{\frac{1}{n}} - 1)$, where n is the number of tasks in the task set and $LLB(n)$ denotes the *Liu-Layland bound* [29]. Following the same approach as that for EDF scheduling, similar utilization-based RAPM schemes can be developed using the Liu-Layland bound for the feasibility of task sets scheduled by RMS. Interested readers can refer to [65] for detailed discussions.

Here, we focus on the feasibility test with the exact *time demand analysis (TDA)* technique. The time demand function $wq_i(t)$ of task T_i is defined as [28]:

$$wq_i(t) = c_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil c_k, \text{for } 0 < t \leq p_i \tag{1.12}$$

The task set is considered feasible if, for every task T_i in the task set under consideration, it is possible to find a time instant t such that $wq_i(t) \leq t \leq p_i$.

First, by incorporating the scaled tasks in Φ and their corresponding recovery tasks, the *modified* time demand function $mwq_i(t)$ for task T_i ($\in \Gamma$) can be defined as:

$$\begin{aligned} mwq_i(t) &= \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil c_k + \sum_{T_k \in \Phi, 1 \leq k \leq i-1} \left\lceil \frac{t}{p_k} \right\rceil \frac{c_k \cdot f_{max}}{f_k} \\ &+ \begin{cases} c_i & \text{if } T_i \notin \Phi; \\ c_i \cdot (1 + \frac{f_{max}}{f_i}) & \text{if } T_i \in \Phi. \end{cases} \end{aligned} \tag{1.13}$$

The above function incorporates the time demand from all (scaled and unscaled) tasks as well as the required recovery tasks. It is not difficult to see that the augmented task set is schedulable if, for every task $T_i \in \Gamma$, there is a time instant t such that $mwq_i(t) \leq t \leq p_i$.

Therefore, the periodic RAPM problem for RMS with TDA-based analysis can be formally expressed as:
find the subset Φ and the scaled frequencies for selected tasks so as to

$$minimize(E(\Phi))$$

subject to

$$\forall T_i \in \Gamma, \exists t, mwq_i(t) \leq t \leq p_i, \text{where } 0 < t \leq p_i$$

Following the idea of *priority-monotonic* frequency assignment [36], we assume that the first x highest priority tasks are selected and $\Phi = \{T_1, \cdots, T_x\}$.

That is, it is assumed that tasks are ordered by their priorities where T_1 has the highest priority and T_n has the lowest priority. Depending on how the scaled frequencies for the selected tasks are determined, we present two priority-monotonic RAPM schemes, which assume the same and different frequency settings for the selected tasks, respectively.

Single Frequency for Selected Tasks: Starting with the single frequency assignment for the selected tasks, the *RAPM-TDA* scheme resorts to the exact TDA test that considers all the time instants within a task's period to get a lower scaled frequency and thus better energy savings. The scaled frequency $f_{RAPM-TDA}(x)$ for the selected x highest priority tasks is given by:

$$f_{RAPM-TDA}(x) = \max\left\{f_{ee}, \max_{T_i \in \Gamma}\{f_i(x)\}\right\} \tag{1.14}$$

$$f_i(x) = \min_{0<t\leq p_i}\{f_i(x,t)\} \tag{1.15}$$

$$f_i(x,t) = \begin{cases} \frac{\sum_{k=1}^{i}\left\lceil \frac{t}{p_k}\right\rceil c_k}{\sum_{k=1}^{i}\left\lceil \frac{t}{p_k}\right\rceil c_k + (t - mwq_i(t))} f_{max} & \text{if } i \leq x; \\ \frac{\sum_{k=1}^{x}\left\lceil \frac{t}{p_k}\right\rceil c_k}{\sum_{k=1}^{x}\left\lceil \frac{t}{p_k}\right\rceil c_k + (t - mwq_i(t))} f_{max} & \text{if } i > x. \end{cases} \tag{1.16}$$

where $f_i(x)$ is the scaled frequency determined by task T_i and $mwq_i(t)$ is the modified time demand function as defined in Equation (1.13) with $f_k = f_{max}$.

If there does not exist a feasible time instant for any task T_i (that is, $mwq_i(t) > t$ for $\forall t\ 0 < t \leq p_i$), it is not valid to select exactly x highest priority tasks. Otherwise, by applying Equation (1.9), we can get the energy consumption when the x highest priority tasks are scaled down to the frequency of $f_{RAPM-TDA}(x)$. Searching through all the feasible task selections, RAPM-TDA can find out the optimal number of highest priority tasks x_{opt} and the corresponding scaled frequency $f_{RAPM-TDA}(x_{opt})$ that give the minimal energy consumption in pseudo-polynomial time. The complexity of RAPM-TDA can be easily found to be $\mathbf{O}(n^3 r)$, where $r = \frac{p_n}{p_1}$ is the ratio of the largest period to the smallest period.

Multiple Frequencies for Selected Tasks: In RAPM-TDA, it is possible that the resulting single scaled frequency is constrained by a high priority task T_k in the subset Φ. That is, T_k has more stringent timing constraints and requires a higher scaled frequency. For such cases, by exploiting the slack time from the high frequency assignment for high priority tasks, the *RAPM-TDAM* scheme iteratively re-calculates and assigns a lower frequency for low priority tasks in the subset Φ, and thus saves more energy.

More specifically, for a given subset Φ with x highest priority tasks, if the single scaled frequency obtained by RAPM-TDA is $f_{RAPM-TDA}(x) = f_k(x)$ and $k < x$, we can assign the frequency $f_{RAPM-TDA}(x)$ to the first k highest priority tasks. Then, we can re-calculate the scaled frequency for the remaining

tasks in the subset Φ. When the frequency assignment for the first k highest priority tasks is fixed, the modified work demand function and scaling factor for task T_i $(k < i \leq n)$ can be re-calculated as:

$$mwq_i(t) = \sum_{j=1}^{k} \left\lceil \frac{t}{p_j} \right\rceil \left(1 + \frac{f_{max}}{f_j}\right) c_j + \begin{cases} \sum_{j=k+1}^{i} \lceil t/p_j \rceil \, 2 \cdot c_j & \text{if } i \leq x; \\ \sum_{j=k+1}^{x} \lceil t/p_j \rceil \, 2 \cdot c_j + \sum_{j=x+1}^{i} \left\lceil \frac{t}{p_j} \right\rceil c_j & \text{if } i > x. \end{cases} \tag{1.17}$$

$$f_i(x,t) = \begin{cases} \frac{\sum_{j=k+1}^{i} \left\lceil \frac{t}{p_j} \right\rceil c_j}{\sum_{j=1}^{i} \left\lceil \frac{t}{p_j} \right\rceil c_j + (t - mwq_i(t))} f_{max} & \text{if } i \leq x; \\ \frac{\sum_{j=k+1}^{x} \left\lceil \frac{t}{p_j} \right\rceil c_j}{\sum_{j=1}^{x} \left\lceil \frac{t}{p_j} \right\rceil c_j + (t - mwq_i(t))} f_{max} & \text{if } i > x. \end{cases} \tag{1.18}$$

After re-calculating the scaled frequencies for tasks T_{k+1} to T_n, we can obtain a new maximum frequency $f^{new}(x)$. Suppose that $f^{new}(x) = f_q^{new}(x)$, where $k+1 \leq q \leq n$. If $q \geq x$, the scaled frequency for the remaining tasks in the subset Φ will be $f^{new}(x)$. Otherwise, we can assign $f^{new}(x)$ as the scaled frequency for tasks T_{k+1} to T_q, and then repeat the above process until we complete the frequency assignment for all tasks in the subset Φ. Then, the energy consumption for the case of selecting x highest priority tasks can be calculated. Checking through all possible values of x, finally we could obtain the optimal value of x_{opt} and corresponding frequency settings that result in the minimum energy consumption. With an additional round to assign the possible different scaled frequencies for tasks in the subset, one can derive the complexity of the RAPM-TDAM scheme as $\mathbf{O}(n^4 r)$, where, again, $r = \frac{p_n}{p_1}$ is the ratio of the largest period to the smallest period.

The performance of the task-level RAPM schemes for periodic real-time tasks follows a trend similar to those of the individual-recovery based RAPM schemes for frame-based tasks. First, system reliability can be preserved by all the RAPM schemes. Second, energy savings of the RAPM schemes generally increase at smaller system utilization values since additional static slack provides better slow-down opportunities [58, 59, 65].

1.5 Dynamic RAPM with Online Slack Reclamation

It is well-known that real-time tasks typically take a small fraction of their worst-case execution times at run-time [20]. Moreover, for the RAPM framework with backward recovery techniques, the recovery tasks are invoked and

executed only if the executions of their corresponding scaled tasks fail. Otherwise, the processor time reserved for those recovery task can be freed and becomes dynamic slack as well. Therefore, significant amount of dynamic slack can be expected at run-time, which should be exploited to further scale down the processing frequency of selected tasks for additional energy savings or to select additional tasks and enhance system reliability.

Dynamic RAPM for Frame-Based Tasks: For frame-based task systems where the tasks share a common period (and deadline), any dynamic slack generated at run-time is accessible to all remaining tasks within the current frame. Therefore, whenever additional dynamic slack is generated at run time, one approach would be to re-distribute all system slack by solving a smaller RAPM problem for the remaining tasks. However, the required computational overhead can be high, especially for systems with many tasks.

Another straightforward approach is to allocate all available dynamic slack to the next task to be dispatched. If the next task already has a statically assigned recovery task, such dynamic slack can be utilized to further scale down the processing frequency for the task as long as the resulting frequency is not lower than the energy-efficient frequency f_{ee}. Otherwise (i.e. if there is no statically assigned recovery task), in case that the dynamic slack is not enough to schedule a recovery for the next task, it cannot be reclaimed by the next task and will be saved for future tasks. For cases where the amount of dynamic slack is large enough, after scheduling a recovery for the next task, the remaining dynamic slack can be utilized to scale down the processing frequency of the task. It has been shown that such greedy slack reclamation is quite effective for additional energy savings [55, 56].

Dynamic RAPM for Periodic Tasks: In periodic execution settings, the dynamic slack may be generated at different priorities and may not always be *reclaimable* by the next ready job. A piece of slack is reclaimable for a job only if the slack has higher priority than that of the job [4]. Moreover, possible preemptions that a job could experience *after* it has reclaimed some slack further complicate the problem. This is because, in the RAPM framework, once the execution of a job is scaled through DVFS, additional slack must be reserved for the potential recovery operation to preserve system reliability. Therefore, conserving the reclaimed slack by a job until it completes its execution (at which point the slack may be used for recovery operation if errors occur, or freed otherwise) is essential in reliability-aware power management settings.

Slack management for periodic real-time tasks has been studied extensively (as in the CASH-queue [8] and α-queue [4] techniques) for different purposes. By borrowing and also extending some fundamental ideas from these studies, we discuss the *wrapper-task* mechanism to track and manage dynamic slack, which can guarantee the *conservation* of the reclaimed slack by a job for dynamic RAPM schemes. Essentially, each wrapper-task represents a piece

of dynamic slack generated at run-time. At the highest level, there are three rules for managing dynamic slack with wrapper-tasks:

- **Rule 1 (slack generation):** When new slack is generated due to early completion of jobs or unneeded recovery jobs, a new wrapper-task is created with the following two timing parameters: a *size* that equals the amount of dynamic slack generated and a *priority* that is equal to that of the job whose early completion gave rise to this slack. Then, the newly created wrapper-task is added into a wrapper-task queue (*WT-Queue*), which is used to track available dynamic slack. The wrapper-tasks with the same priority can be merged into a larger wrapper-task with size equal to the summation of these wrapper-tasks' sizes.

- **Rule 2 (slack reclamation):** The slack is reclaimed when: **(a)** the next dispatched highest priority job is a non-scaled job and its reclaimable slack is larger than the job's WCET (which ensures that a recovery, in the form of re-execution, can be scheduled to preserve reliability); or, **(b)** the next dispatched job has already been scaled (that is, CPU time has been already reserved for its recovery) but its scaled frequency is still higher than the energy efficient frequency f_{ee} and reclaimable slack exists. After reclamation, the corresponding wrapper-tasks are removed from the *WT-Queue* and destroyed, which guarantees the conservation of slack for the job under consideration.

- **Rule 3 (slack exchange):** After slack reclamation, the remaining wrapper-tasks in the *WT-Queue* compete for the processor along with ready jobs. When a wrapper-task has the highest priority and is "scheduled": **(a)** if there are available ready jobs, the wrapper-task will "fetch" the highest priority job and *"wrap"* the execution of that job during the interval when the wrapper-task is "executed". In this case, the corresponding slack is actually borrowed to the ready job. When the job returns such slack to the system, the new slack will have a lower priority than that of the job; **(b)** otherwise, if there is no ready job, the processor becomes idle. The wrapper-task is said to "execute no-ops" and the corresponding dynamic slack is consumed/wasted during this time interval.

It has been shown that the wrapper-task based mechanism can effectively manage dynamic slack at run-time for dynamic RAPM schemes [58, 59]. In general, higher energy savings can be obtained when more dynamic slack is generated at run time (with higher workload variability). Also, by potentially managing more tasks, higher system reliability can be achieved by the dynamic RAPM schemes. Interested readers are referred to [58, 59] for detailed discussions.

1.6 Reliability Maximization with Energy Constraints

In the RAPM schemes discussed so far, the goal was to minimize the system energy consumption while preserving system original reliability by appropriately scheduling the required recovery tasks. In this section, considering the energy-constrained operation settings where the system energy consumption for any given interval must not exceed a hard bound [1, 11, 35, 47], we discuss the schemes that aim at *maximizing system reliability*. Here, the negative effects of DVFS on system reliability are incorporated but no recovery task is considered/scheduled. That is, we consider the problem of determining task level frequency assignments to maximize overall system reliability (the probability of completing all tasks successfully) within the given energy budget and without missing the deadlines of tasks. For simplicity, we focus on frame-based task systems where tasks share a common deadline, the approach can be extended to periodic real-time tasks as well [54].

Let E_{budget} be the energy budget that can be utilized by all tasks within a frame that has the period (deadline) of D. Suppose that the processing frequency for task T_i is f_i ($\leq f_{max}$), which can vary from task to task. The system energy consumption within a frame can be given as follows:

$$\begin{aligned} E(f_1, \ldots, f_n) &= P_s \cdot D + \sum_{i=1}^{n} E_i(f_i) \\ &= P_s \cdot D + \sum_{i=1}^{n} (P_{ind_i} \cdot \frac{c_i}{f_i} + C_{ef} \cdot c_i \cdot f_i^2) \end{aligned} \tag{1.19}$$

Here, $E_i(f_i)$ stands for the active energy consumed by task T_i, which comes from the frequency-independent and frequency dependent power (see Section 1.2.2). We consider a general case where the frequency-independent power P_{ind_i} can vary from task to task. Note that $E_i(f_i)$ is a strictly convex function and is minimized when $f_i = f_{ee_i}$, the energy-efficient frequency for task T_i that can be derived through Equation (1.2).

Given that the rate of soft errors caused by transient faults follows a Poisson distribution as discussed in Section 1.2, the reliability (that is, the probability of completing a task without having errors caused by transient faults) of task T_i in one frame at the frequency f_i is $R_i(f_i) = e^{-\lambda(f_i) * \frac{c_i}{f_i}}$, where $\lambda(f_i)$ is given by Equation (1.3). Let $\varphi_i(f_i) = \lambda(f_i) \cdot \frac{c_i}{f_i}$. Considering that the correct operation of a system depends on all its tasks, the system reliability within a frame can be represented as:

$$R(f_1, \ldots, f_n) = \prod_{i=1}^{n} R_i(f_i) = e^{-\sum_{i=1}^{n} \varphi_i(f_i)} \tag{1.20}$$

Considering the well-known features of the exponential functions, to maximize

system reliability $R(f_1, \ldots, f_n)$, we need to minimize $\sum_{i=1}^{n} \varphi_i(f_i)$. Therefore, the *energy-constrained reliability management (ECRM)* problem can be stated as: find the processing frequency f_i $(1 \leq i \leq n)$ so as to:

$$\text{minimize} \quad \sum_{i=1}^{n} \varphi_i(f_i) \tag{1.21}$$

Subject to:

$$E(f_1, \ldots, f_n) \leq E_{budget} \tag{1.22}$$

$$\sum_{i=1}^{n} \frac{c_i}{f_i} \leq D \tag{1.23}$$

$$f_{ee_i} \leq f_i \leq f_{max} \qquad (1 \leq i \leq n) \tag{1.24}$$

Here, the first inequality corresponds to the hard energy constraint; the second one encodes the deadline constraint. The last constraint set gives the range of feasible frequency assignments for the tasks.

Let E_{min} be the minimum energy that must be allocated to the given task system to allow their completion before or at the deadline D. Given the task parameters, E_{min} can be computed by the polynomial-time algorithm developed in [3]. As a by-product, the same algorithm yields also the optimal task-level frequency assignments $(fl_1, fl_2, ..., fl_n)$ for the tasks when the total system energy consumption is *exactly* E_{min}. Obviously, if the energy budget E_{budget} is less than E_{min}, there is no solution to the problem as the system would lack the minimum energy needed for timely completion.

Moreover, let $E_{max} = E(f_{max}, \ldots, f_{max})$ be the maximum energy consumption of the task set when all tasks run at f_{max}. As another boundary condition, when the given energy budget $E_{budget} \geq E_{max}$, executing all tasks at the maximum frequency is the optimal solution. Note that $R_i(f_i)$ is a strictly concave and increasing function of f_i. Therefore, in what follows, we will focus exclusively on settings where $E_{min} \leq E_{budget} < E_{max}$.

We first explain that, in the optimal solution to the ECRM problem, the resulting total energy consumption $E(f_1^{opt}, \ldots, f_n^{opt})$ must be equal to E_{budget}. Otherwise, with the assumption that $E_{budget} < E_{max}$, there must be a frequency for task T_i such that $f_i^{opt} < f_{max}$. In this case, it should be possible to increase f_i^{opt} by ε (> 0) such that $f_i' = (f_i^{opt} + \varepsilon) \leq f_{max}$ and $E(f_1^{opt}, \ldots, f_i', \ldots, f_n^{opt}) \leq E_{budget}$. It is clear that the deadline and energy constraints are still satisfied after this modification. Further, as $R_i(f_i)$ increases monotonically with the increasing frequency f_i, the overall system reliability can be improved due to the execution of T_i at frequency f_i', which is higher than f_i^{opt}. Hence, in the optimal solution $(f_1^{opt}, \ldots, f_n^{opt})$, the resulting total energy consumption $E(f_1^{opt}, \ldots, f_n^{opt})$ must be equal to E_{budget}, the given energy budget within a frame.

Therefore, the energy constraint for the original ECRM problem (Equa-

tion (1.22)) can be re-written as:

$$E(f_1, \ldots, f_n) = E_{budget} \tag{1.25}$$

which leads to a new non-linear (convex) optimization ECRM problem.

The new ECRM problem can be solved, for instance, by *Quasi-Newton* techniques developed for constrained non-linear optimization [5]. The technique exploits the well-known Kuhn-Tucker optimality conditions for non-linear programs in an iterative fashion by transforming the original problem to a quadratic programming problem and solving it optimally. While optimal, a theoretical complication with this approach is that it is very difficult to express the maximum number of iterations as a function of the number of unknowns which, in this case, corresponds to the number of tasks n.

The new ECRM problem can be also tackled with an efficient heuristic scheme (denoted as ECRM-LU) that provably runs in polynomial-time and satisfies the energy deadline, and frequency constraints. ECRM-LU proceeds as follows. We temporarily ignore the deadline constraint (Equation (1.23)) and solve the problem only by considering the new energy constraint (Equation (1.25)) and frequency range constraints (Equation (1.24)). Notice that, by excluding the deadline constraint, the problem is transformed to a separable convex optimization problem with n unknowns, $2n$ inequality constraints and a single equality constraint. This problem, in turn, can be solved in time $O(n^3)$ by iteratively manipulating the Kuhn-Tucker optimality conditions in a way similar to the technique illustrated in algorithm given in [3]. Now, if the resulting solution satisfies also the deadline constraint, obviously it is the solution to the ECRM problem.

Otherwise, we can re-write the frequency constraint set as:

$$fl_i \leq f_i \leq f_{max} \;\; (1 \leq i \leq n) \tag{1.26}$$

where fl_i is the frequency assignment to task T_i in the solution where the task set completes *at exactly* $t = D$ and with energy consumption E_{min}. Again, the corresponding $\{fl_i\}$ values can be computed in time $O(n^3)$ [3]. By enforcing the new frequency constraint set given above, we make sure that the final frequency assignments satisfy also the deadline constraint. Once again, this version of the problem where the deadline constraint is handled implicitly by enforcing the lower bounds on frequency assignments can be solved in time $O(n^3)$. Hence, the overall time complexity of ECRM-LU is also $O(n^3)$.

The evaluation through extensive simulations show that ECRM-LU yields reliability figures that are extremely close (within 1%) to those achieved by the optimal solution [52, 54]. In general, the achieved system reliability increases with increasing energy budget as more energy enables the system to operate at higher processing frequencies, which results in fewer errors due to reduced number of transient faults. Moreover, the system workload also has an interesting effect on system reliability. Increasing the workload leads to increased execution time, which in turn means increased probability of incurring errors

caused by transient faults. However, higher system workload also forces the system to adopt higher frequencies in order to meet the timing constraints of tasks, which instead has a positive impact on the system reliability due to reduced error rates. Our results show that, for low workload (for instance, where the amount of slack is more than the amount of computation of tasks), the first effect dominates. After the workload reaches a certain point, the second effect becomes the dominant factor and leads to better system reliability.

1.7 Other Techniques and Future Directions

In addition to the RAPM framework discussed above, there have been other studies on the co-management of system reliability and energy consumption, with and without the consideration of the negative effect of DVFS on system reliability, respectively. After a brief overview of other reliability and energy co-management techniques, in this section, we point out some open problems and possible future directions for this line of research.

Co-Management of Energy and Reliability: One of the earliest energy-aware fault tolerance schemes has been proposed by Unsal *et al.* for a set of independent periodic real-time tasks [43]. Based on the primary and backup fault tolerance model, the scheme postpones as much as possible the execution of backup tasks to minimize the overlap between primary and backup executions and thus to reduce system energy consumption. With the goal of tolerating a fixed number of transient faults, Melhem *et al.* explored the optimal number of checkpoints, *uniformly* or *non-uniformly* distributed, to achieve the minimum energy consumption for a duplex system (where two hardware processing units are used to run the same software concurrently for fault detection) [30]. To reduce the energy consumption in a traditional TMR system (Triple Modular Redundancy, in which three hardware processing units are used to run the same software simultaneously to detect and mask faults), Elnozahy *et al.* studied an interesting *Optimistic-TMR* (OTMR) scheme, which allows one processing unit to run at a scaled processing frequency provided that it can catch up and finish the computation before the deadline if there is a fault [18]. In [63], Zhu *et al.* further explored the optimal frequency settings for an OTMR system and presented detailed comparisons among Duplex, TMR and OTMR for reliability and energy consumption. For parallel real-time tasks running on multiprocessor systems, Zhu *et al.* studied the optimal redundant configuration of the processors to tolerate a given number of transient faults through backward recovery techniques [62, 64].

With the assumption that the arrivals of transient faults follow a Poisson distribution with a constant arrival rate, Zhang *et al.* studied an adaptive checkpointing scheme to tolerate a fixed number of transient faults during the

execution of a single real-time task [49]. The scheme dynamically adjusts the checkpointing interval during the execution of a task based not only on the slack time but also on the occurrences of transient faults during task's execution, so as to reduce system energy consumption. The adaptive checkpointing scheme was extended to a set of periodic real-time tasks on a single processor system with the EDF (earliest deadline first) scheduling algorithm [51]. In [50], the authors further considered the cases where faults may occur within checkpoints. Following a similar idea and considering a fixed priority rate-monotonic scheduling (RMS) algorithm, Wei *et al.* studied an efficient online scheme to minimize energy consumption by applying DVFS with the consideration of the run-time behaviors of tasks and fault occurrences while still satisfying the timing constraints [44]. In [45], the authors extended the study to multiprocessor real-time systems. Izosimov *et al.* studied an optimization problem for mapping a set of tasks with reliability constraints, timing constraints and precedence relations to processors and for determining appropriate fault tolerance policies (re-execution and replication) for the tasks [24]. However, these studies did not address the negative effects of DVFS on system reliability due to the higher rate of soft errors caused by transient faults at lower supply voltages.

Taking such negative effects into consideration, in addition to the RAPM schemes discussed in this chapter, for real-time periodic tasks that have different reliability requirements, a research effort that proposes the schemes that selectively recover a subset of jobs for each task is given in [66]. For real-time tasks with known statistical execution times, an optimistic RAPM has also been proposed. The scheme deploys smaller size recovery tasks while still preserving the system's original reliability [60].

Ejlali *et al.* studied schemes that combine the information (about hardware resources) and temporal redundancy to save energy and to preserve system reliability [17]. By employing a feedback controller to track the overall miss ratio of tasks in soft real-time systems, Sridharan *et al.* [42] proposed a reliability-aware energy management algorithm to minimize the system energy consumption while still preserving the overall system reliability. Pop *et al.* studied the problem of energy and reliability trade-offs for distributed heterogeneous embedded systems [32]. The main idea is to transform the user-defined reliability goals to the objective of tolerating a fixed number of transient faults by switching to pre-determined contingency schedules and re-executing individual tasks. A constrained logic programming-based algorithm is proposed to determine the voltage levels, process start time and message transmission time to tolerate transient faults and minimize energy consumption while meeting the timing constraints of the application. Dabiri *et al.* studied the problem of assigning frequency and supply voltage to tasks for energy minimization subject to reliability as well as timing constraints [14].

More recently, following the similar idea in OTMR [18], Ejlali *et al.* studied a standby-sparing energy efficient hardware redundancy technique for fault-tolerance, where a standby processor is operated at a low power state when-

ever possible provided that it can catch up and finish the tasks in time [16]. This scheme was shown to have better energy performance when compared to that of the backward recovery based RAPM approach. For frame-based task systems with independent tasks, Qi *et al.* investigated global scheduling based RAPM problem and studied several individual recovery based schemes depending on when and how the tasks are selected [34].

Future Research Directions. Among the open problems, the extension of the shared recovery technique to the preemptive periodic execution settings needs to be mentioned. Moreover, with the emergence of multicore processors, extending that technique to multiprocessor real-time systems can also be an interesting direction. With the inherent hardware redundancy, multicore systems provide excellent opportunities to tolerate permanent faults. However, how to effectively integrate the hardware and temporal redundancy to tolerate both permanent and transient faults while reducing energy consumption, especially considering the intriguing interplay between power/energy, temperature and rate of system failure, is another major open problem.

1.8 Summary

In this chapter, we discussed several reliability-aware power management (RAPM) schemes for real-time tasks running on a single processor. The motivation comes from the negative effect of the widely-deployed power management technique, namely *dynamic voltage and frequency scaling (DVFS)*, on system reliability due to the increased transient fault rates associated with operation at low supply voltages. The key idea of these RAPM schemes is to recuperate the reliability loss due to DVFS by scheduling proper recovery tasks, while still exploiting the remaining system slack to save energy.

Starting with the case of a single real-time task, we showed that, by scheduling a recovery task before scaling down the execution of the task, the RAPM scheme can preserve the system reliability while still obtaining energy savings. Then, for frame-based task systems where multiple real-time tasks share a common deadline, the RAPM problem is shown to be NP-hard and two efficient heuristics with individual recovery tasks are discussed. Aiming to address the pessimism of the individual-recovery based schemes, the RAPM scheme with a shared recovery task is presented. For general periodic real-time tasks that have different deadlines, we presented a task-level RAPM technique where all jobs of the same periodic task receive the same treatment. Then, for the *earliest-deadline-first (EDF)* scheduling policy, the utilization-based RAPM scheme are discussed, followed by the priority-monotonic RAPM schemes for *rate-monotonic scheduling (RMS)*. Dynamic RAPM schemes that exploit dynamic slack generated at run time are further

considered. For energy-constrained systems, we discussed the RAPM scheme that aims at maximizing system reliability. At the end, an overview of other related studies is provided and a few open research problems are identified.

Bibliography

[1] T.A. AlEnawy and H. Aydin. On energy-constrained real-time scheduling. In *Proc. of the 16th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 165–174, Washington, DC, USA, 2004. IEEE Computer Society.

[2] AMD. Amd opteron quad-core processors; http://www.amd.com/us/products/embedded/processors/, 2009.

[3] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proc. of The* 27^{th} *IEEE Real-Time Systems Symposium (RTSS)*, Piscataway, NJ, USA, 2006. IEEE CS Press.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. on Computers*, 53(5):584–600, 2004.

[5] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, 2005.

[6] E. Bini, G.C. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *Proc. of the* 17^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 3–10, Washington, DC, USA, 2005. IEEE Computer Society.

[7] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of the 28th Hawaii International Conference on System Sciences (HICSS)*, pages 288–297, Washington, DC, USA, 1995. IEEE Computer Society.

[8] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proc. of the 21st IEEE Real-Time Systems Symposium (RTSS)*, pages 295–304, Washington, DC, USA, 2000. IEEE Computer Society.

[9] 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.

[10] J.-J. Chen and T.-W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *Proc. of the 2005*

International Conference on Parallel Processing (ICPP), pages 13–20, Washington, DC, USA, 2005. IEEE Computer Society.

[11] J.-J. Chen and T.-W. Kuo. Voltage-scaling scheduling for periodic real-time tasks in reward maximization. In *Proc. of the 26th IEEE Real-Time Systems Symposium (RTSS)*, pages 345–355, Washington, DC, USA, 2005. IEEE Computer Society.

[12] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Proc. of the 2007 IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD)*, pages 289–294, Piscataway, NJ, USA, 2007. IEEE Press.

[13] Intel Corp. Intel embedded quad-core xeon; http://www.intel.com/products/embedded/processors.htm, 2009.

[14] F. Dabiri, N. Amini, M. Rofouei, and M. Sarrafzadeh. Reliability-aware optimization for dvs-enabled real-time embedded systems. In *Proc. of the 9th int'l symposium on Quality Electronic Design (ISQED)*, pages 780–783, Washington, DC, USA, 2008. IEEE Computer Society.

[15] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. Soft errors issues in low-power caches. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 13(10):1157–1166, Oct. 2005.

[16] A. Ejlali, B. M. Al-Hashimi, and P. Eles. A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems. In *Proc. of the 7th IEEE/ACM Int'l conference on Hardware/software codesign and system synthesis (CODES)*, pages 193–202, New York, NY, USA, 2009. ACM.

[17] 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)*, pages 281–286, New York, NY, USA, 2005. ACM.

[18] E. (Mootaz) 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 (RTSS)*, pages 256–265, Washington, DC, USA, 2002. IEEE Computer Society.

[19] 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.

[20] 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 (ICCAD)*, pages 598–604, New York, NY, USA, 1997. ACM.

[21] P. Hazucha and C. Svensson. Impact of cmos technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. on Nuclear Science*, 47(6):2586–2594, 2000.

[22] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proc. of the 14th Annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 37–46, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

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

[24] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In *Proc. of the conference on Design, Automation and Test in Europe (DATE)*, pages 864–869, Washington, DC, USA, 2005. IEEE Computer Society.

[25] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proc. of the* 41^{st} *Design automation conference (DAC)*, pages 275–280, New York, NY, USA, 2004. ACM.

[26] T. Juhnke and H. Klar. Calculation of the soft error rate of submicron cmos logic circuits. *IEEE Journal of Solid-State Circuits*, 30(7):830–834, 1995.

[27] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Proc. of the 9th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 105–116, New York, NY, USA, 2000. ACM.

[28] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization andaverage case behavior. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 166–171, Washington, DC, USA, 1989. IEEE Computer Society.

[29] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

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

[31] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 89–102, New York, NY, USA, 2001. ACM.

[32] P. Pop, K.H. 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, New York, NY, USA, 2007. ACM.

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

[34] X. Qi, D. Zhu, and H. Aydin. Global reliability-aware power management for multiprocessor real-time systems. In *Proc. of the IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 183–192, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[35] C. Rusu, R. Melhem, and D. Mossé. Maximizing rewards for real-time applications with energy constraints. *ACM Transactions on Embedded Computing Systems*, 2(4):537–559, Nov. 2003.

[36] 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 (RTAS)*, pages 106–115, Washington, DC, USA, 2003. IEEE Computer Society.

[37] C. Scordino and G. Lipari. A resrouce reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks. *IEEE Trans. on Computers*, 55(12):1509–1522, 2006.

[38] N. Seifert, D. Moyer, N. Leland, and R. Hokinson. Historical trend in alpha-particle induced soft error rates of the alpha^{TM} microprocessor. In *Proc. of the* 39^{th} *IEEE Annual International Reliability Physics Symposium*, pages 259–265, Washington, DC, USA, 2001. IEEE Computer Society.

[39] Tezzaron Semiconductor. Soft errors in electronic memory: A white paper. available at http://www.tachyonsemi.com/about/papers/, 2004.

[40] 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 (RTSS)*, pages 200–224, New York, NY, USA, 2003. ACM.

[41] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, pages 389–398, Washington, DC, USA, 2002. IEEE Computer Society.

[42] R. Sridharan, N. Gupta, and R. Mahapatra. Feedback-controlled reliability-aware power management for real-time embedded systems. In *Proc. of the 45th annual Design Automation Conference (DAC)*, pages 185–190, New York, NY, USA, 2008. ACM.

[43] O. S. Unsal, I. Koren, and C. M. Krishna. Towards energy-aware software-based fault tolerance in real-time systems. In *Proc. of The International Symposium on Low Power Electronics Design (ISLPED)*, pages 124–129, New York, NY, USA, 2002. ACM.

[44] T. Wei, P. Mishra, K. Wu, and H. Liang. Online task-scheduling for fault-tolerant low-energy real-time systems. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 522–527, New York, NY, USA, 2006. ACM.

[45] T. Wei, P. Mishra, K. Wu, and H. Liang. Fixed-priority allocation and scheduling for energy-efficient fault tolerance in hard real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19:1511–1526, 2008.

[46] 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 (OSDI)*, Berkeley, CA, USA, 1994. USENIX Association.

[47] H. Wu, B. Ravindran, and E.D. Jensen. Utility accrual real-time scheduling under the unimodal arbitrary arrival model with energy bounds. *IEEE Transactions on Computers*, 56(10):1358–1371, Oct. 2007.

[48] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. of The* 36^{th} *Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, Washington, DC, USA, 1995. IEEE Computer Society.

[49] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proc. of the conference on Design, Automation and Test in Europe (DATE)*, pages 918 – 923, Washington, DC, USA, 2003. IEEE Computer Society.

[50] Y. Zhang and K. Chakrabarty. Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems. In *Proc. of IEEE/ACM Design, Automation and Test in Europe Conference(DATE)*, pages 1170 – 1175, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[51] Y. Zhang, K. Chakrabarty, and V. Swaminathan. Energy-aware fault tolerance in fixed-priority real-time embedded systems. In *Proc. of the 2003 IEEE/ACM int'l Conference on Computer-Aided Design (ICCAD)*, pages 209–214, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[52] B. Zhao, H. Aydin, and D. Zhu. Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems. In *Proc. of the IEEE International Conference on Computer Design (ICCD)*, pages 633–639, Piscataway, NJ, USA, 2008. IEEE CS Press.

[53] B. Zhao, H. Aydin, and D. Zhu. Enhanced reliability-aware power management through shared recovery technique. In *Proc. of the ACM/IEEE Int'l Conference on Computer Aided Design (ICCAD)*, pages 63–70, New York, NY, USA, 2009. ACM.

[54] B. Zhao, H. Aydin, and D. Zhu. On maximizing reliability of real-time embedded applications under hard energy constraint. *IEEE Trans. on Industrial Informatics*, 6(3):316–328, 2010.

[55] 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 (RTAS)*, pages 397 – 407, Piscataway, NJ, USA, 2006. IEEE CS Press.

[56] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. *ACM Trans. on Embedded Computing Systems (TECS)*, 10(2):26.1–26.27, Dec. 2010.

[57] 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 (ICCAD)*, pages 528–534, New York, NY, USA, 2006. ACM Press.

[58] 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 (RTAS)*, pages 225–235, Piscataway, NJ, USA, 2007. IEEE CS Press.

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

[60] D. Zhu, H. Aydin, and J.-J. Chen. Optimistic reliability aware energy management for real-time tasks with probabilistic execution times. In *to appear in the Proc. of the 29th IEEE Real-Time Systems Symposium (RTSS)*, pages 313–322, Piscataway, NJ, USA, 2008. IEEE CS Press.

[61] 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 (ICCAD)*, pages 35–40, New York, NY, USA, 2004. ACM Press.

[62] D. Zhu, R. Melhem, and D. Mossé. Energy efficient configuration for qos in reliable parallel servers. In *Proc. of the Fifth European Dependable Computing Conference (EDCC)*, Lecture Notes in Computer Science, pages 122–139, New York, NY, USA, Apr. 2005. Springer.

[63] D. Zhu, R. Melhem, D. Mossé, and E.(Mootaz) Elnozahy. Analysis of an energy efficient optimistic tmr scheme. In *Proc. of the* 10^{th} *Int'l Conference on Parallel and Distributed Systems*, pages 559–568, Piscataway, NJ, USA, 2004. IEEE CS Press.

[64] D. Zhu, D. Mossé, and R. Melhem. Energy efficient redundant configurations for real-time parallel reliable servers. *Journal of Real-Time Systems*, 41(3):195–221, Apr. 2009.

[65] 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)*, pages 629–635, Piscataway, NJ, USA, 2007. IEEE CS Press.

[66] D. Zhu, X. Qi, and H. Aydin. Energy management for periodic real-time tasks with variable assurance requirements. In *Proc. of the IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 259–268, Piscataway, NJ, USA, 2008. IEEE CS Press.

[67] J. F. Ziegler. Terrestrial cosmic ray intensities. *IBM Journal of Research and Development*, 42(1):117–139, 1998.

[68] J. F. Ziegler. Trends in electronic reliability: Effects of terrestrial cosmic rays. available at http://www.srim.org/SER/SERTrends.htm, 2004.