The Interplay of Power Management and Fault Recovery in Real-Time Systems

Rami Melhem, *Fellow*, *IEEE*, Daniel Mossé, *Member*, *IEEE*, and
Elmootazbellah (Mootaz) Elnozahy, *Member*, *IEEE*

Abstract—This paper describes how to exploit the scheduling slack in a real-time system to reduce energy consumption and achieve fault tolerance at the same time. During failure-free operation, a task takes checkpoints to enable recovery from failure. Additionally, the system exploits the slack to conserve energy by reducing the processor speed. If a task fails, it will restart from a saved checkpoint and execute at maximum speed to guarantee that the deadlines are met. The paper shows that the number of checkpoints and their placements interact in subtle ways with the power management policy. We study two checkpoint placement policies for aperiodic tasks and analytically derive the optimal number of checkpoints to conserve energy under each. This optimal number allows the CPU speed to be slowed down to the level that yields minimum energy consumption, while still guaranteeing recoverability of tasks under each checkpointing policy. The results show that traditional periodic checkpointing is not the best policy for the combined purpose of conserving energy and guaranteeing recovery. Instead, better energy savings are possible through a nonuniform distribution of checkpoints that takes into account the energy consumption and reliability factors. Depending on the amount of slack and the checkpointing overhead, energy can be reduced by up to 68 percent under nonuniform checkpointing. We also demonstrate the applicability of these checkpoint placement policies to periodic tasks.

Index Terms—Checkpointing, fault tolerance, frequency scaling, power management, real-time systems, reliability, voltage scaling.

1 Introduction

Slack that exists in a schedule has been used for fault tolerance purposes, for example, to restart a task, or a part of a task, after a fault occurs [9], [18], [20], [21], [30], [33]. Checkpoints may need to be inserted in situations where the slack in scheduling may not allow an entire task to be restarted [8]. This paper presents a study showing how this slack can also be exploited to simultaneously tolerate failures and reduce the energy consumption of the system. The idea behind this work is simple and intuitive. During failure-free operation, we exploit the slack to conserve energy by slowing down the processor such that the tasks can meet their deadlines and also recover from potential failures. If a failure occurs, we set the processor to operate at maximum speed (and, consequently, maximum energy consumption) to reexecute the lost computation.

Power management has recently attracted a large body of research [7], [11], [12], [23], [24], [27], [28], [32], [35], [36], [37]. This increasing attention has been motivated initially by the limitations on battery life in portable devices. There are several aspects to the problem, including controlling the power of the processor, display, disk subsystem, and memory [6]. The interactions of these techniques with failure recovery are not well understood and we are not aware of any previous study that simultaneously addresses both fields of research. Our work is relevant in real-time systems where reliability and low power consumption are required. Examples include autonomous airborne and seaborne systems working on limited battery supply, space systems working on a limited combination of solar and battery power supply, or time-sensitive systems deployed in remote locations where a steady power supply is not available.

In this paper, we present two checkpoint placement policies. The first is the standard, straightforward periodic checkpointing. Under this policy, we compute the optimal number of checkpoints that must be inserted to minimize the energy consumption, subject to the constraints of recovering from a single failure and completing before the task's deadline. The second checkpoint placement policy takes a more aggressive approach at reducing processor speed, resulting in more energy savings than can be obtained by the uniform placement alone. It places checkpoints in a manner where the frequency of checkpointing starts slowly at the beginning and increases as we approach the task deadline. An analysis shows that, depending on the workload, this method can achieve up to 68 percent energy savings while tolerating one potential failure and meeting the task deadline.

The scope of this paper is limited to the analysis of energy conservation in the context of a fault-tolerant real-time system. We also discuss some issues for possible implementation of the algorithms presented here. However, the focus is on the fundamental issues and the theoretical derivation to analyze the potential for energy conservation in the context of the checkpointing policies under study.

This paper is organized as follows: In Section 2, we present the real-time, power consumption, fault, and

- *R. Melhem and D. Mossé are with the Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260.
E-mail: {melhem, mosse}@cs.pitt.edu.*
- *E. Elnozahy is with the System Software Department, IBM Austin Research Laboratory, Austin, TX 78758.
E-mail: mootaz@us.ibm.com.*

*Manuscript received 17 May 2001; revised 19 July 2002; accepted 3 Apr. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114167.*

0018-9340/04/$20.00 © 2004 IEEE Published by the IEEE Computer Society

recovery models. In Section 3, we derive the necessary conditions for determining the optimal number of checkpoints that minimize energy consumption, when checkpoints are placed uniformly in a task. Section 4 presents and analyzes results when the checkpoints are placed nonuniformly in the tasks and Section 5 describes how to apply our scheme to periodic tasks and discusses an illustrative example. Implementation issues are discussed in Section 6, while Section 7 presents related work. The paper then concludes with Section 8.

2 Models

2.1 Task and Real-Time Models

A typical real-time system model assumes that a task, τ, has a worst-case execution time and a deadline, D, which is derived from hard real-time constraints. Without loss of generality, we assume that a task is ready at time 0 and, therefore, D can be seen as the time interval within which τ is allowed to execute. However, given that variable voltage CPUs are available, the time to execute task τ depends on the processor speed. We therefore characterize a task τ by a fixed quantity, namely, its *worst-case* number of CPU cycles, C, needed to execute the task.

To simplify analysis and to allow for the derivation of analytical formulas, we would like to assume that C is independent of the CPU speed for a given processor architecture. This assumption, however, does not hold if the speed of the memory system is independent of the speed of the CPU since memory references will consume a larger number of cycles when the processor speed is high, thus increasing the total number of cycles needed to execute the program. For this reason, we assume that C is the worst-case number of CPU cycles needed to execute a program at the maximum processor speed.

We have conducted a number of simulation experiments using the Simple-Scalar simulator to determine the degree of pessimism in the definition of C. These experiments show that, with on-chip caches and low cache miss rates, C does not change substantially with the processor speed. For the Li, Perl, Go, and Compress programs from the SPEC benchmarks [4], changing the processor's speed from 700 MHz to 300 MHz changed the number of CPU cycles needed to execute the benchmarks by 0.01 percent, 1.2 percent, 1.9 percent, and 0.6 percent, respectively. The main reason for the small change in the number of cycles is the small cache miss rate. In all the experiments, the default simple-scalar configurations of 16K data and instruction L1 caches and 256K L2 cache are used. No disk I/O is performed during execution since, without this assumption, it is extremely hard to estimate the execution times of tasks. For the rest of this paper, we normalize the units of C such that the maximum processor speed is 1. That is, if the maximum processor speed is S cycles per second, then we express the number of cycles in units of S cycles and thus normalize the maximum processor speed to $S_{max} = 1$.

The difference between the deadline, D, and the worst-case execution time, C, is defined in this paper as the $static$ $slack$, or simply, the $slack$. This is different from the $dynamic$ $slack$ which results at runtime when a task consumes less than its worst-case execution time. In this paper, we will mainly be concerned with the static slack and briefly discuss the dynamic slack in Section 6.3.

Last, when describing the periodic model in Section 5, each task τ_i has associated with it a period, T_i, which represents the minimum interarrival of consecutive instances of the task. Let $U = \sum_{i=1}^{N} \frac{C_i}{T_i}$ be the total utilization of the task set under the maximum processor speed. In order for U to be dimensionless, T_i is expressed in terms of the number of CPU cycles at the maximum processor speed of $S_{max} = 1$. It is a well-known result that, if $U \leq 1$ and Earliest Deadline First (EDF) scheduling is used, then each instance of every task will terminate execution before the end of its period, thus meeting its deadline [22].

2.2 Power Consumption Model

Variable-voltage CPUs can reduce power consumption *quadratically* or *cubically* at the expense of *linearly* increased delay (reduced speed) [14]. Thus, any effective Variable Voltage Scaling (VVS) scheme should be able to vary the voltage fed to the system component and the frequency of the system clock. The power consumption of the processor under the speed S is given by $g(S)$, which is assumed to be a strictly increasing convex function, represented by a polynomial of at least the second degree [14]. If task τ_i occupies the processor during the time interval $[t_1, t_2]$, then the *energy* consumed during this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g(S(t))dt$. For the rest of this paper, we use the notation E without parameters when no confusion arises. We assume that the CPU speed can be changed between a minimum speed S_{min} (minimum supply voltage necessary to keep the system functional) and a maximum speed S_{max} and that $0 \leq S_{min} \leq S_{max} = 1$.

2.3 Fault and Recovery Models

We assume a real-time system that is subject to transient and intermittent faults, which are faults that have instantaneous duration [17]. Furthermore, we start with the assumption that no more than one fault can occur before the task's deadline. For the periodic task model, we consider two schemes, one assumes that at most one fault occurs in each task's instance and the other assumes that faults are separated in time by intervals T_{max}, where T_{max} is the longest period.

The possible causes of transient faults include limitations in the accuracy of electromechanical devices, electromagnetic radiation received by interconnections (such as long buses acting like receiving antennas), power fluctuations not properly filtered by the power supply, and the effects of ionizing radiation on semiconductor devices [3]. Our focus is on the *transient fault tolerance* scheme because it has been shown that transient faults are significantly more frequent than permanent faults [3], [16].

We assume a real-time system with tasks that take checkpoints in main memory and use them to restart if a fault occurs. We assume that the system will detect a fault before the next checkpoint is taken, for instance, by running internal integrity tests, using hardware assistance, or employing techniques from test theory. We assume that the overhead of running the self-tests and the error detection is included in the checkpointing overhead. This is a reasonable assumption, given that the previous

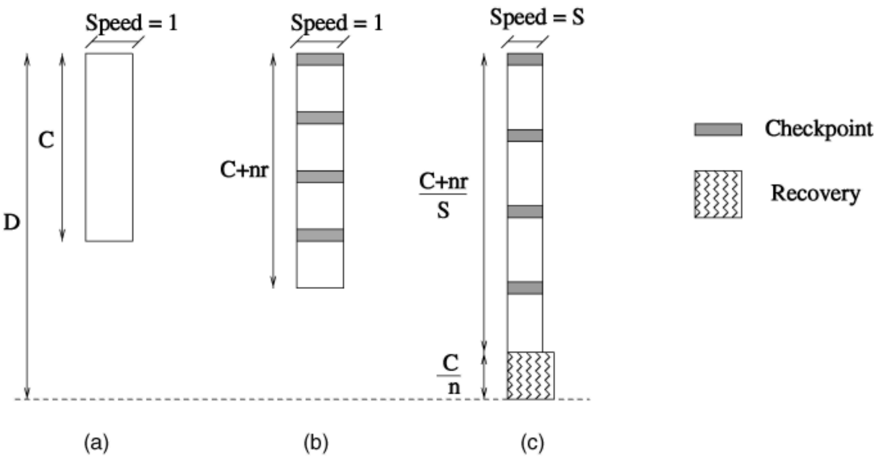

Fig. 1. Equally spaced checkpoints.

checkpoint cannot be discarded before ensuring that the new checkpoint is complete and that it represents a correct state of the system. Therefore, it will be necessary to run diagnosis and self-tests after taking each checkpoint and, thus, one can include the cost of these tests as part of the checkpointing overhead.

Although main memory checkpointing cannot recover from a total system failure, the timing constraints of real-time systems make traditional disk-based checkpointing impractical. Battery-backed RAM, shielded, or stable semiconductor memory could be used if a restart from a total system failure or crash is desired. Either way, for the purpose of this presentation, it suffices to assume that the system can recover from transient faults. We do not address, in this paper, the engineering aspects of checkpointing, but we remark that, due to the real-time nature of these applications, the checkpointing mechanism must be *predictable*.

In the absence of power consumption considerations, the checkpoints must be placed such that the tasks can recover from failures and still meet their deadlines. Thus, if there is slack t in the schedule within a interval T, checkpoints must be placed such that no more than t amount of execution is at risk. This simple view becomes more complex when exploiting the slack for power management. In the next section, we discuss how the placement of checkpoints is affected by the competing desires for reducing power consumption and tolerating failures.

3 Power Management under Uniformly Distributed Checkpoints

We study the placement of checkpoints to guarantee recovery from failures without missing deadlines, while reducing the energy by lowering the processor speed. In this section, we derive the optimal number of checkpoints in the case where tasks take regular checkpoints.

3.1 Description

Assume that task τ takes n periodic checkpoints to enable recovery from failures and that at most one fault can occur during the execution of τ. If a fault occurs, the task rolls back to the most recent checkpoint and reexecutes. Let r be the number of cycles needed to create a checkpoint and run self-tests and diagnosis, where, typically, r is much smaller than C, the maximum number of cycles needed to execute τ. Our goal is to try to use the time slack between the deadline, D, and the completion time of τ for both fault tolerance and energy management.

Fig. 1 illustrates this reasoning: A computation is represented by a rectangle whose area is the number of CPU cycles needed for execution. The width of the rectangle represents the CPU execution speed and its length represents the time taken for execution. Fig. 1a and Fig. 1b show the execution of τ, at the maximum speed, without checkpoints and with $n = 4$ checkpoints, respectively. Fig. 1c shows that we can reduce the speed of executing τ (with the four checkpoints) to S as long as the difference between the deadline, D, and the time for executing τ at speed S is at least enough for the overhead of checkpoints (nr) and the time necessary for a potential rollback, ($\frac{C}{n}$), at maximum speed. Note that, since failures are not frequent, recovery from a fault proceeds at the maximum speed rather than at speed S, thus leaving more slack to be used for speed reduction.

More specifically, the speed, S, that allows τ to finish on time while minimizing energy consumption and allowing for the checkpointing overhead and the time to roll back from a fault, should satisfy the following:

$$D \geq \frac{C + nr}{S} + \frac{C}{n}. \tag{1}$$

To simplify the presentation, denote $\frac{C}{D}$ and $\frac{r}{D}$ by σ and ρ, respectively. The higher the value of σ, the less the amount of slack available. Similarly, ρ represents the overhead of checkpointing relative to the total available time to execute. Simple algebraic manipulation of (1) leads to the following solution for the speed, S:

$$S \geq \max\left\{\frac{n\sigma + n^2\rho}{n - \sigma}, S_{min}\right\}. \tag{2}$$

This equation eliminates the impractical solutions where the operating voltage cannot be reduced below a given minimum. Since the energy, E, consumed during the execution of τ is proportional to the time it takes to execute

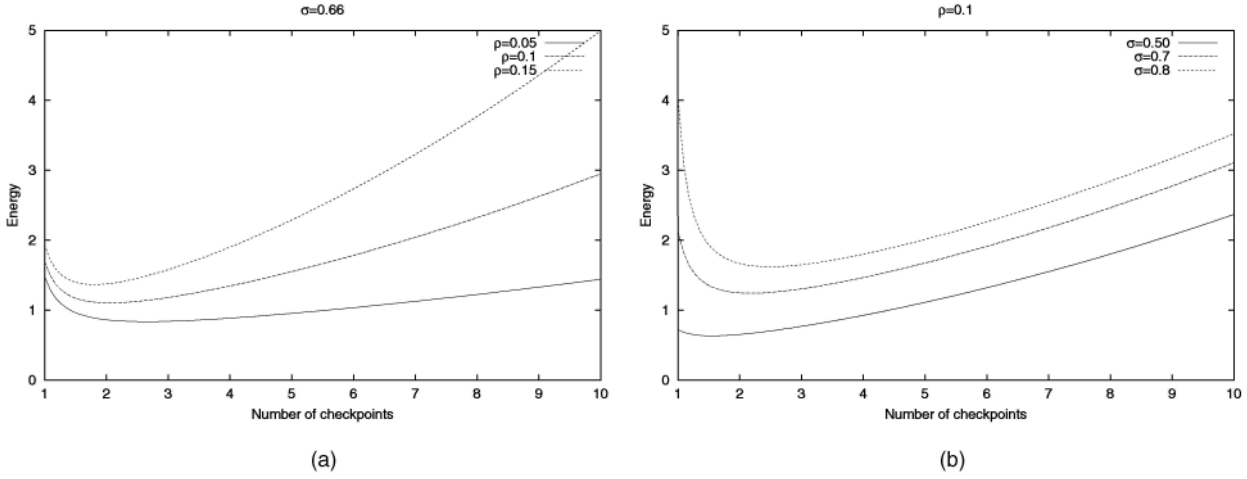

Fig. 2. Energy consumption versus the number of checkpoints, for fixed $\sigma = 0.66$ (a) and fixed $\rho = 0.1$ (b).

the task and to the square of the speed during execution [10], we obtain:

$$E = cS^2 \frac{C + nr}{S} = cD \frac{n(\sigma + n\rho)^2}{n - \sigma}, \tag{3}$$

where c is a proportionality constant and the lower bound for S is temporarily ignored.

We can derive the value of n which minimizes E by differentiating (3) with respect to n and equating the result to zero. This gives a cubic equation in n with two nonpositive solutions (which we ignore) and one optimal positive value of n, given by:

$$n = \frac{\sigma}{4}\left(3 + \sqrt{9 + \frac{8}{\rho}}\right). \tag{4}$$

The value obtained for n should result in a speed, S, which is larger than S_{min}. If not, then n should be chosen such that $S_{min} = (n\sigma + n^2\rho)/(n - \sigma)$. Also, if (4) gives a noninteger value for n, then either the floor or the ceiling of this value should be taken, according to which number would produce a lower energy consumption value. This point is further discussed next.

3.2 Analysis

Intuitively, the more checkpoints are taken, the less work is at risk and, therefore, the portion of the slack reserved for rollback-recovery is smaller, allowing the remaining slack to be used for further reduction of processor speed. However, the overhead of checkpointing consumes a part of the available slack that would be used to reduce speed. Consequently, the more checkpoints are taken, the less the opportunity to reduce speed. Fig. 2 shows the consumed energy as it varies with the number of checkpoints.[1] The figure reveals a couple of interesting points. First, the minimum energy consumption is usually obtained at noninteger values of n, which means that the solution of (4) is not usually an integer. This is true for most cases and, therefore, an approximation to an integer value must be computed by checking the integer neighboring values, which gives an optimal solution due to the convexity of the energy function given by (3). Second, the optimal number of checkpoints is typically low and there is a large increase in the energy consumption if the number of checkpoints increases.

In Fig. 3, we plot the energy level for the optimal number of checkpoints derived from (4). The figure shows that the energy consumed increases as σ increases since there is less slack in the system and therefore fewer opportunities to slow down the processor. Moreover, the energy consumption increases as the overhead of checkpointing increases. Note that some of the curves (with higher energy consumption) are not "complete": This is because the higher values of ρ and σ yield infeasible solutions. The figure also shows the energy consumption assuming that no checkpoints are taken (NoFT). This curve is given in order to compare the energy consumption with and without fault tolerance.

Table 1 shows the number of checkpoints taken under two scenarios and for various combinations of values for σ and ρ. In the first scenario, denoted by `FT-Only`, checkpoints are taken only to support recovery, but no power management takes place. In the second, denoted by `FT + EC`, checkpoints are taken to support reliability and speed is controlled to conserve energy. Additionally, the table shows the percentage energy savings that is obtained from the second scenario compared to the first. The number of checkpoints computed for the first scenario is the minimum number of checkpoints that can allow recovery, from (1) with $S = 1$. The number of checkpoints shown for the second scenario is obtained by computing the optimal value n in (4), then computing the energies for both the floor and the ceiling of the result, and choosing the one that gives lower energy consumption. Although the choice of $\rho = 0.005$ may appear too low, for small values of σ (e.g., $\sigma = 0.3$), this corresponds to a reasonable checkpointing overhead (when $\sigma = 0.3$, the overhead is actually 1.5 percent of the computation time, C). This is in line with published numbers in the literature [5]. Last, empty entries within the table indicate situations where checkpointing cannot be used because the combination of available slack and checkpointing overhead makes it

[1] All figures in this paper are drawn for $cD = 1$.

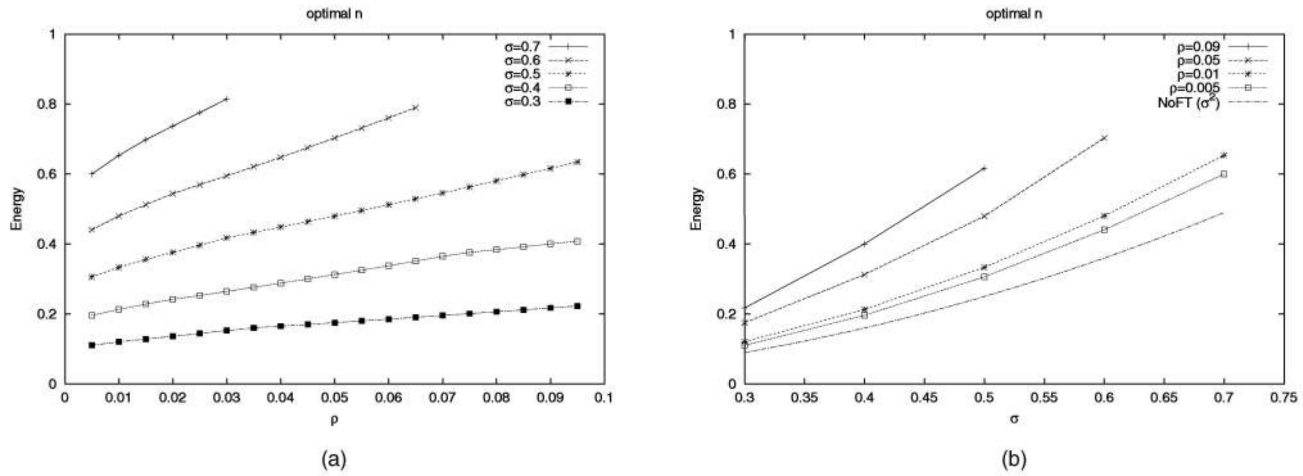

Fig. 3. Energy consumption for the optimal number of checkpoints, as a function of ρ (a) and σ (b).

impossible to guarantee recovery by the deadline, even when executing at maximum speed.

Table 1 shows two predictable trends. As expected, when the slack is relatively large (e.g., $\sigma = 0.3$), substantial energy saving can be obtained by adding checkpoints. However, as the slack becomes relatively smaller, the speed reduction yields predictably smaller energy saving until the savings practically disappear at $\sigma = 0.8$. This is not due to the ineffectiveness of the speed reduction, but rather because the slack does not even accommodate checkpointing for the purpose of recovery alone, much less energy savings. The second trend is that the efficiency of checkpointing is extremely important. When the overhead is high (ρ is 0.10 or larger), the resulting energy saving is very low and disappears quickly with larger values of σ.

4 Power Management with Nonuniform Distribution of Checkpoints

4.1 Motivation and Intuition

In this section, we explore whether further speed reduction is possible through an alternative checkpoint placement policy. The intuition behind this exploration takes the optimistic view that failures are not frequent. Therefore, if a

TABLE 1
Number of Checkpoints When They Are Taken Only for Reliability (FT-Only) and When They Are Also Taken for Energy Management (FT+EC)

		$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$	$\sigma = 0.6$	$\sigma = 0.7$	$\sigma = 0.8$
$\rho = 0.005$	FT-Only	1	1	2	2	3	5
	FT + EC	3	4	5	6	8	9
	Energy Saving	64%	52%	40%	28%	16%	5%
$\rho = 0.01$	FT-Only	1	1	2	2	3	6
	FT + EC	2	3	4	5	6	6
	Energy Saving	61%	48%	35%	22%	10%	0%
$\rho = 0.03$	FT-Only	1	1	2	2	4	-
	FT + EC	2	2	3	3	4	-
	Energy Saving	57%	38%	25%	10%	0%	-
$\rho = 0.05$	FT-Only	1	1	2	2	-	-
	FT + EC	1	2	2	2	-	-
	Energy Saving	50%	30%	20%	0%	-	-
$\rho = 0.07$	FT-Only	1	1	2	-	-	-
	FT + EC	1	2	2	-	-	-
	Energy Saving	47%	22%	15%	-	-	-
$\rho = 0.10$	FT-Only	1	1	2	-	-	-
	FT + EC	1	1	2	-	-	-
	Energy Saving	42%	17%	7%	-	-	-

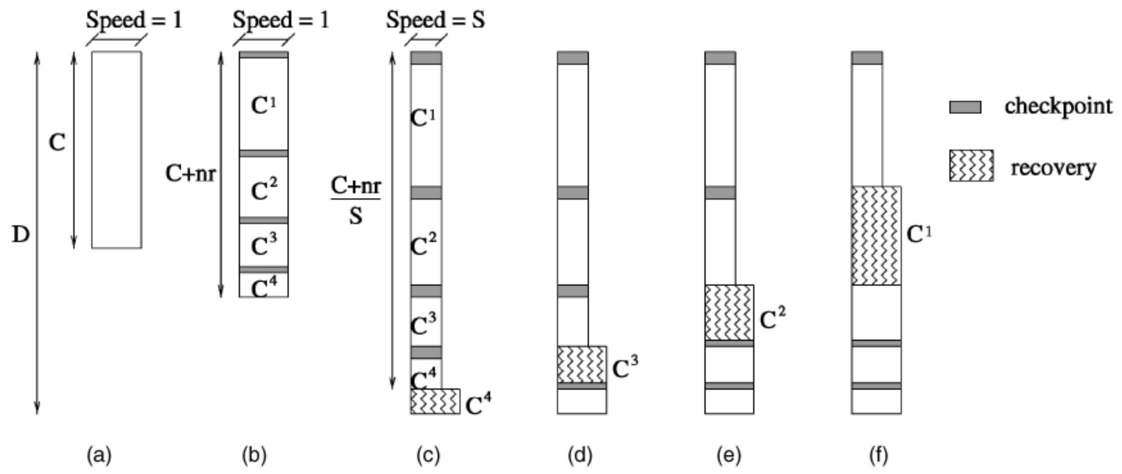

Fig. 4. Nonequally spaced checkpoints.

failure occurs, it is conceivable that we let the task execute at maximum speed not only during rollback, as in Section 3, but also following the rollback and recovery when the task executes the remainder of the computation. Of course, execution under maximum speed is not optimal for energy consumption, but we force the task to run at maximum speed only following a presumably rare failure and only until the deadline expires. This scheme, however, requires nonuniform checkpoint placement and allows execution at a lower speed than in the case of uniform checkpointing. To see why, consider that the task executes at a low speed, exploiting the slack. Yet, at the beginning of execution, the task hasn't "consumed" much of "available" slack and, therefore, low frequency of checkpoints is possible because most of the slack is available to accommodate a large amount of work at risk if we need to recover at maximum speed. As the task continues to execute, however, it slowly "consumes" the slack available. Gradually, the remaining slack can accommodate decreasing amounts of work at risk and, hence, there is a need for increasing the frequency of the checkpointing to accommodate the decreasing ability of the remaining slack to handle work at risk as the task approaches the deadline.

4.2 Technical Description

Assume that the n checkpoints are placed in τ such that the C cycles of τ are divided into n sections, $P^{(1)}, \ldots, P^{(n)}$ with each $P^{(k)}$ requiring $C^{(k)}$ CPU cycles to execute. Note that, in the previous section, we have assumed that $C^{(k)} = \frac{C}{n}$, for $k = 1, \ldots, n$.

When a fault is detected in $P^{(k)}$, then the reexecution of $P^{(k)}$ as well as the execution of $P^{(k+1)}, \ldots, P^{(n)}$ can proceed at the maximum processor speed, as described in Section 4.1. This means that the minimum speed, S, should satisfy the following, for $k = 1, \ldots, n$:

$$D \geq \sum_{i=1}^{k} \frac{(r + C^{(i)})}{S} + C^{(k)} + \sum_{i=k+1}^{n} (r + C^{(i)}). \tag{5}$$

The term $\sum_{i=1}^{k} \frac{(r+C^{(i)})}{S}$ represents the execution at a reduced speed up to a failure in section $P^{(k)}$. The term $C^{(k)}$ represents the time it takes to reexecute section $P^{(k)}$, while $\sum_{i=k+1}^{n} (r + C^{(i)})$ represents the time to execute the remainder of the task at maximum speed after a failure in section $P^{(k)}$. The goal is thus to find $C^{(1)}, \ldots, C^{(n)}$ and the minimum value of S such that (5) is satisfied and

$$\sum_{i=1}^{n} C^{(i)} = C. \tag{6}$$

The minimum value of S that satisfies (5) and (6) simultaneously is obtained if we can find a solution for the system of (5) and (6) with the inequality in (5) replaced by an equality. The next lemma, which is proven in the Appendix, sets the stage for finding such a solution.

Lemma 1. *When $\sum_{i=1}^{n} C^{(i)} = C$, the n equations*

$$D = \sum_{i=1}^{k} \frac{(r + C^{(i)})}{S} + C^{(k)} + \sum_{i=k+1}^{n} (r + C^{(i)}) \quad k = 1, \ldots, n, \tag{7}$$

are equivalent to the following n equations:

$$C^{(n)} = D - \frac{C + nr}{S} \tag{8}$$

$$C^{(k)} + r = \frac{C^{(k+1)} + r}{S} \qquad k = 1, \ldots, n-1. \tag{9}$$

We illustrate in Fig. 4 the rationale used to write (5) assuming $n = 4$. Fig. 4a and Fig. 4b show the execution of τ under the maximum speed without and with checkpoints, respectively. Fig. 4c shows that the minimum speed, S, that can be used for executing τ should allow it to finish at or before $D - C^{(4)}$ to allow for the recovery, at maximum speed, from a fault in $C^{(4)}$. Fig. 4d shows that if a fault is detected at the end of the execution of $C^{(3)}$ (at speed S), then the remaining time should be large enough to allow the reexecution of $C^{(3)}$ and the execution of $C^{(4)}$, both at maximum speed. Fig. 4e and Fig. 4f show the recovery from faults at the end of $C^{(2)}$ and $C^{(1)}$, respectively.

We need to find the value of S that simultaneously satisfies the $n + 1$ ((6), (8), and (9)). For this, we first observe that the repeated application of (9) gives

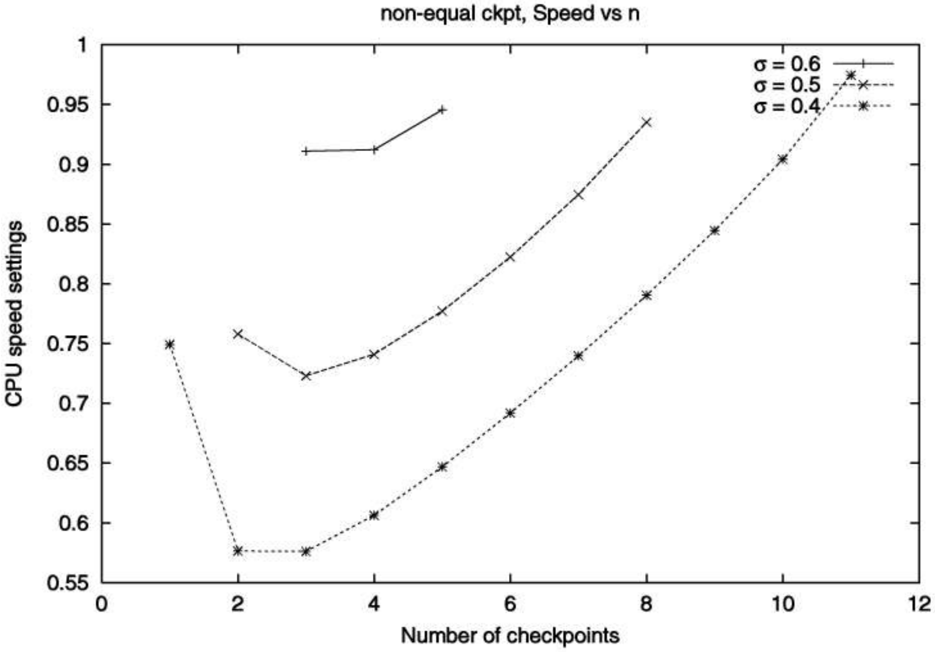

Fig. 5. Speed settings as a function of number of checkpoints, for fixed $\rho = 0.05$.

$$C^{(k)} + r = \frac{C^{(n)} + r}{S^{n-k}} \qquad k = 1, \ldots, n-1,$$

which, when substituted in (6), gives

$$(C^{(n)} + r)\left(\frac{1}{S^{n-1}} + \ldots + \frac{1}{S} + 1\right) = C + nr$$

$$(C^{(n)} + r)\frac{1 - S^n}{S^{n-1} - S^n} = C + nr.$$

By substituting for $C^{(n)}$ from (8), using $\sigma = \frac{C}{D}$ and $\rho = \frac{r}{D}$, and, after simple algebraic manipulation, we obtain

$$S = \left(1 - \frac{\sigma + n\rho}{1 + \rho}\right)S^{n+1} + \frac{\sigma + n\rho}{1 + \rho}. \qquad (10)$$

Hence, given σ, ρ, and n, we can solve (10) for S to find the minimum allowable CPU speed that guarantees completion by the deadline, even with a recovery from a single fault. In the absence of faults and recovery, the energy consumed during the execution of τ at speed S is given by:

$$E = cS^2\frac{C + nr}{S} = cDS(\sigma + n\rho), \qquad (11)$$

where c is again a proportionality constant.

Similar to the case described in Section 3, for a given σ and ρ, the optimum value of E depends on the number of checkpoints n. Given the form of (10), however, it is very difficult to find the optimum n analytically for a given σ and ρ. Nevertheless, it is straightforward to iteratively solve (10) and find S for any given n. The optimum n that minimizes E can then be found by searching the possible values of n. This is reasonable because the number of checkpoints is typically small. For example, for $cD = 1$, $\rho = 0.05$, and $\sigma = 0.5$ (the same case considered in Section 3), (10) gives the solutions $S = 0.75$, $S = 0.72$, $S = 0.74$, $S = 0.77$, and $S = 0.82$ for $n = 2$, $n = 3$, $n = 4$, and $n = 5$, respectively (the case for $n = 1$ is not feasible). Using (11), these S values yield $E = 0.45$, $E = 0.47$, $E = 0.51$, and $E = 0.58$, respectively, which shows that the energy consumption is minimized at $n = 2$.

Using the computed values of n and S, determining the checkpoint placements is straightforward. Using Lemma 1, one can compute the value of $C^{(n)}$, then recursively compute the values $C^{(n-1)}, \ldots, C^{(1)}$.

4.3 Analysis

In this section, we analyze the nonuniform checkpoint placement scheme. First, we show the CPU speed settings as a function of the number of checkpoints (see Fig. 5). As we can see, in this scheme, when the slack offered by the system is low (i.e., high value of σ), the CPU is typically set at a high speed. This is because there is a need to compensate for the checkpoints, which are done during slack. On the other hand, as the slack increases, the CPU speed decreases as a function of the number of checkpoints and then increases very quickly; intuition leads us to expect the biggest gains to be during the minimum speed settings. We can also see that the more slack there is in the system, the more checkpoints *can be* taken (not necessarily *should be* taken). To illustrate this point, when $\sigma = 0.6$, 0.5, and 0.4, the maximum number of checkpoints that maintains feasibility of the schedule is 5, 8, and 11, respectively (that is, the curves do not go beyond those points). Similarly, the smallest number of checkpoints is also restricted.

As can be seen from Fig. 6, the intuitive behavior alluded to above is not observed: The biggest energy savings are not always when the CPU speed is set to the smallest value. The top line corresponds to the smallest slack in the system and it can be seen that the energy consumption increases almost linearly with the number of checkpoints. This is because there is a need to increase the speed of the CPU if more checkpoints are being taken since there will be less time for the computations to use the CPU, within the required deadlines. Other values of σ behave differently, which can

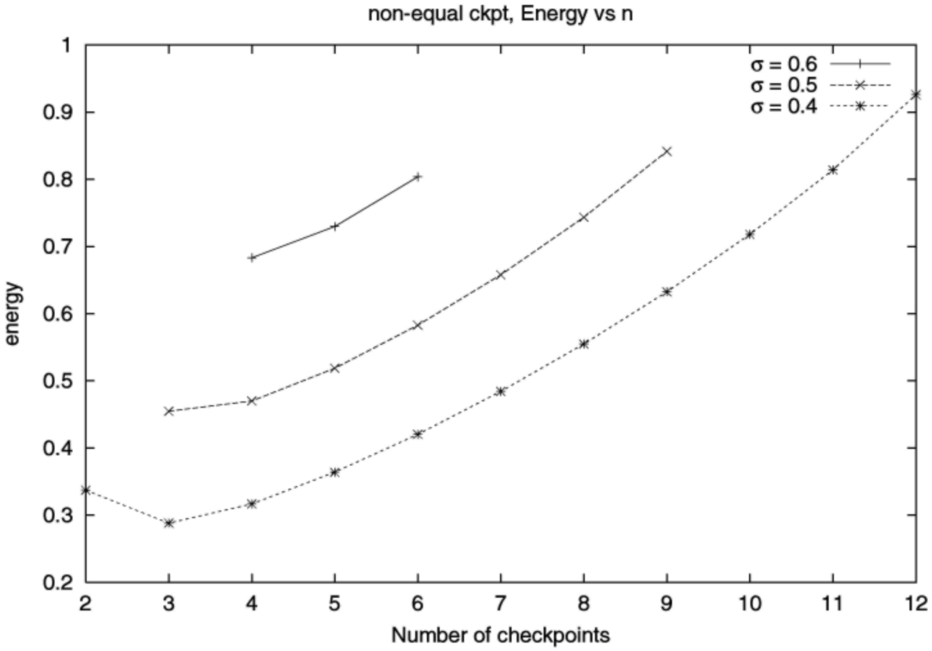

Fig. 6. Resulting energy as a function of number of checkpoints, for fixed $\rho = 0.05$.

be accounted for by observing that the formulas depend not only on n, but also on σ and ρ. Of particular interest is the curve for $\sigma = 0.5$, which has smaller speed when $n = 3$, but smaller energy when $n = 2$ due to the overhead of checkpointing.

Next, we compare the uniform checkpoint distribution scheme of Section 3 with the nonuniform scheme presented in this section. Table 2 shows a comparison between the two checkpointing placement policies. The table shows the number of checkpoints under the nonuniform and the `FT-Only` schemes. The table also shows the energy savings that results from the nonuniform checkpointing compared to `FT-Only` and the additional savings that are obtained from the nonuniform scheme compared to the uniform placement. The table shows that the nonuniform scheme is very effective in reducing energy, reaching a reduction of up to 68 percent. The comparison shows that the nonuniform checkpointing policy is also more effective in conserving energy than uniform checkpointing. The additional savings reached about 8 percent in the best case, *in addition to* the savings that can be obtained by the uniform checkpointing policy itself. Therefore, we conclude that relying on the low probability of faults and taking a more aggressive stance yield better conservation than a more passive approach.

5 Power Management and Error Recovery in Periodic Tasks

Consider the case of N periodic tasks, $\tau_1, \ldots, \tau_N$, where each task, τ_i is specified by the number of CPU cycles, C_i, needed to execute the task and the period, T_i, of the task. Without loss of generality, we assume that $T_1 \leq T_2 \leq \ldots \leq T_N$. We also assume that $U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$, as defined in Section 2. In other words, EDF scheduling is able to complete execution of each instance of every task before the end of its period, thus meeting its deadline. In this section, we show that if U is *sufficiently* less than 1, then the slack, $1 - U$, can be used for fault recovery and, at the same time, for slowing down the processor speed to save energy. We again consider two checkpointing schemes. In the first scheme, the checkpoints are uniformly placed within each task in the system, while, in the second scheme, the checkpoints are not uniformly placed.

5.1 Uniformly Distributed Checkpoints

In this section, we assume that checkpoints are uniformly inserted in each task every γ interval (measured in CPU cycles). That is, for each task, τ_i, $\lceil \frac{C_i}{\gamma} \rceil$ checkpoints are inserted, one every γ of the C_i cycles of τ_i (including one at the beginning of the task). With the addition of these checkpoints, the execution of τ_i takes $C_i + \lceil \frac{C_i}{\gamma} \rceil r_i$ cycles every period in the absence of faults, where r_i is the overhead of checkpointing for task τ_i. If τ_i executes at a speed S, then the execution will take $\frac{C_i + \lceil \frac{C_i}{\gamma} \rceil r_i}{S}$ time and the effective utilization of τ_i is

$$\frac{C_i + \left\lceil \frac{C_i}{\gamma} \right\rceil r_i}{T_i S}.$$

In order to allow for rollback after a fault, a slack of γ cycles will be reserved for error recovery every interval T_1. As stated in the following lemma (proven in the Appendix), this slack will guarantee that each task finishes at least γ time units before its deadline and, thus, has time for recovery from a fault in any task.

Lemma 2. *Given a set of tasks, $\tau_1, \ldots, \tau_N$, where each task τ_i has a worst-case execution time of C_i and a period of T_i, if $\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 - \frac{\gamma}{T_1}$, then, in EDF execution of the tasks, each task finishes at least γ time units before the end of its period.*

TABLE 2
Number of Checkpoints under Uniform versus Nonuniform Checkpointing and Percentage Improvement in Energy Saving When Using Nonuniform Instead of Uniform Checkpointing

		$\sigma = 0.3$	$\sigma = 0.4$	$\sigma = 0.5$	$\sigma = 0.6$	$\sigma = 0.7$	$\sigma = 0.8$
$\rho = 0.005$	FT-Only	1	1	2	2	3	5
	Non-Uniform	3	4	5	6	8	9
	Energy Saving	68%	56%	45%	33%	24%	13%
	Diff. w/ Uniform	+5%	+5%	+5%	+4%	+8%	+8%
$\rho = 0.01$	FT-Only	1	1	2	2	3	6
	Non-Uniform	2	3	4	5	6	7
	Energy Saving	65%	53%	42%	29%	16%	3%
	Diff. w/ Uniform	+4%	+5%	+6%	+6%	+6%	+3%
$\rho = 0.03$	FT-Only	1	1	2	2	4	-
	Non-Uniform	2	2	3	3	4	-
	Energy Saving	58%	44%	32%	15%	0%	-
	Diff. w/ Uniform	+4%	+6%	+7%	+5%	+0%	-
$\rho = 0.05$	FT-Only	1	1	2	2	-	-
	Non-Uniform	2	2	2	2	-	-
	Energy Saving	51%	36%	25%	3%	-	-
	Diff. w/ Uniform	+1%	+6%	+5%	+3%	-	-
$\rho = 0.07$	FT-Only	1	1	2	-	-	-
	Non-Uniform	1	2	2	-	-	-
	Energy Saving	47%	28%	19%	-	-	-
	Diff. w/ Uniform	+0%	+6%	+4%	-	-	-
$\rho = 0.10$	FT-Only	1	1	2	-	-	-
	Non-Uniform	1	1	2	-	-	-
	Energy Saving	43%	18%	10%	-	-	-
	Diff. w/ Uniform	+0%	+2%	+3%	-	-	-

If the speed of the system in the absence of faults is set to S and recovery is allowed at the maximum speed, then the utilization of the system is given by

$$\frac{\gamma}{T_1} + \sum_{i=1}^{N} \frac{C_i + \left\lceil \frac{C_i}{\gamma} \right\rceil r_i}{T_i S}.$$

In order to fully use the available slack, $1 - U$, for error recovery and power management, we can set the above utilization to 1, from which we find that

$$S = \sum_{i=1}^{N} \frac{C_i + \left\lceil \frac{C_i}{\gamma} \right\rceil r_i}{T_i} \frac{1}{1 - \frac{\gamma}{T_1}}. \tag{12}$$

From the above equation, it is clear that the minimum processor speed that guarantees that the deadlines are met depends on the checkpoint separation, γ. The choice of γ affects the speed S and, thus, the energy consumption. The following lemma, which is proven in the Appendix, specifies how to chose γ so as to minimize the energy consumption.

Lemma 3. *If, in each task in the system, checkpoints are separated by γ, then the total energy consumption is minimized when γ is given by*

$$\gamma = \frac{-3b + \sqrt{9b^2 + 8abT_1}}{2a}, \tag{13}$$

where, $a = \sum_{i=1}^{N} \frac{C_i + r_i}{T_i}$ and $b = \sum_{i=1}^{N} \frac{C_i}{T_i} r_i$.

In other words, to minimize power consumption, checkpoints should be placed in each τ_i separated by γ as given by (13). Of course, if C_i is not a multiple of γ, then the number of cycles between the last checkpoint and the end of τ_i will be less than γ. After finding the optimal checkpoint interval, γ, the CPU speed during fault-free operation can be found from (12).

Note that the reservation of a slack of γ every period T_1 is sufficient to guarantee that, when a fault occurs in a task τ_i, there will be enough time reserved for recovery. However, observe that, when a fault occurs in τ_i, recovery consumes γ time units, which is equivalent to τ_i executing for $C_i + \gamma$ time, rather than C_i. Thus, until the end of the period T_i, the conditions of Lemma 2 are not satisfied. After the end of the period T_i, each instance of τ_i will execute again for only C_i.

Furthermore, the γ units of slack will get replenished after T_1 units of time. Hence, the slack needed for recovery will be available and the conditions of Lemma 2 will be satisfied at most $T_N = \max\{T_i\}$ after the occurrence of a fault. In other words, the system can tolerate faults that are separated by T_N.

5.2 Nonuniformly Distributed Checkpoints

The scheme applied in this section is simple and relies on distributing the utilization slack to the N tasks such that the time allocated for each instance of τ_i is increased from C_i (under the maximum speed) to a quantity D_i (recall that D_i can be interpreted as the deadline of a task or the interval within which the task is allowed to execute). The slack for a task is then the difference $D_i - C_i$, which will be used to add checkpoints to τ_i and slow down the CPU speed for the execution of each instance of τ_i in a manner identical to the one described in Section 4. Assuming that, after the addition of checkpoints and with the reduced speed, each instance of τ_i is allocated a time D_i, then the EDF schedulability theory guarantees that each instance will meet its deadline if and only if $\sum_{i=1}^{N} \frac{D_i}{T_i} \leq 1$. By defining $\sigma_i = \frac{C_i}{D_i}$ and setting $\sigma_i = U$ for $i = 1, \ldots, N$, we guarantee that

$$\sum_{i=1}^{N} \frac{D_i}{T_i} = \frac{1}{U} \sum_{i=1}^{N} \frac{C_i}{T_i} = 1. \tag{14}$$

With the above distribution of the slack $1 - U$ to all the tasks, we may apply (10) and (11) to find the optimal number of checkpoints and the optimum speed for the execution of each task τ_i. According to (10) and (11), the optimum processor speed depends on the values of σ and ρ. The value of $\sigma_i = U$ is equal for all tasks, while the value of ρ_i for task τ_i is equal to $\frac{r_i}{D_i}$, where r_i is the number of cycles needed for performing a checkpoint in task τ_i. In the general case, where ρ_i depends on the task τ_i, the optimal number of checkpoints and the optimum speed will be different for the different tasks. However, we consider, in this section, two special cases.

First, we consider the special case where $\rho_i = \rho$ independent of the particular task τ_i. That is, the time to take a checkpoint in task τ_i is proportional to the computation time, C_i, of τ_i. Although this is not true for general tasks, it is true for many applications in which the space requirements of the application are proportional to its computational requirements. Examples of such applications are matrix and image processing operations.

If $\rho_i = \rho$ for $i = 1, \ldots N$, the optimum speed obtained from (10) and (11) will be identical for all the tasks. In this case, the convexity of the power function implies that this speed is the one that minimizes the energy consumption for the entire system and this implies that our decision to distribute the slack $1 - U$ to the tasks in proportion to the utilization, $\frac{C_i}{T_i}$, of each task was an optimal decision. Specifically, consider two tasks, τ_i and τ_j. If the proportional distribution of slack leads to the selection of the same optimum speed, S, for executing within D_i and D_j, then any different distribution of slack that would result in $D'_i = D_i + \delta > D_i$ and $D'_j = D_j - \delta < D_j$ (or vice versa), with $D'_i + D'_j = D_i + D_j$, will lead to two speeds $S_i = S\frac{D_i}{D'_i} < S$ and $S_j = S\frac{D_j}{D'_j} > S$ for execution within D'_i and D'_j. When the energy is a function of the square of the speed and the time of execution of the task, this execution at two different speeds increases the energy consumption since it can be shown that

$$S_i^2 D'_i + S_j^2 D'_j > S^2 (D_i + D_j).$$

The second case that we consider is the one in which the time for a checkpoint, r_i, is constant for any task, which makes $\rho_i = \frac{r_i}{D_i}$ different for different tasks. In this case, to facilitate the analysis, we use a more conservative fixed value for ρ_i, $i = 1, \ldots, N$, to obtain the same speed in all the tasks (this has the advantage of avoiding CPU speed changes during fault-free operation). We use the conservative value of $\rho = \max_{i=1}^{N} \rho_i$ to obtain the same number of checkpoints, n, and speed, S, for all N tasks.

The energy consumed during the execution of each instance of a task τ_i during fault-free operation is given from (11) by

$$c D_i S (\sigma + n\rho_i).$$

Hence, the energy consumed during an LCM period is

$$E = \sum_{i=1}^{N} c D_i S (\sigma + n\rho_i) \left(\frac{LCM}{T_i} \right) = cS(\sigma + n\rho) LCM \sum_{i=1}^{N} \frac{D_i}{T_i},$$

which, by virtue of the fact that we set $\sum_{i=1}^{N} \frac{D_i}{T_i} = 1$, gives

$$E = cS\left(\sigma + n \sum_{i=1} N\rho_i \right) LCM. \tag{15}$$

5.3 Analysis

The scheme described in Section 5.2 allows for the recovery from a fault in each instance of a task, which is different from the scheme described in Section 5.1, which allows for the recovery from a fault in each period T_N. Given that each of the two schemes result in different error recovery capabilities, it is not useful to statistically compare the power consumption of the two schemes. In fact, there is a trade off between the power consumption and the error recovery capability and it is not fruitful to statistically analyze this trade off for general task systems because of the interplay between the many parameters that may affect the outcome $(N, T_1, \ldots, T_N, C_1, \ldots, C_N, r)$. However, for a given task system (for example, one resulting from an embedded system), it is possible to compare the two schemes and pick the most suitable one according to the required reliability and power consumption. In this section, we demonstrate the application of the two schemes to the example task set shown in Fig. 7a. The example is chosen to be very simple to simplify the illustration of the two schemes.

Consider a set of two periodic tasks, τ_1 and τ_2, with $C_1 = 4$, $T_1 = 10$, $C_2 = 3$, and $T_2 = 15$. At maximum speed, this task set has a utilization $U = 0.6$ and, thus, there is

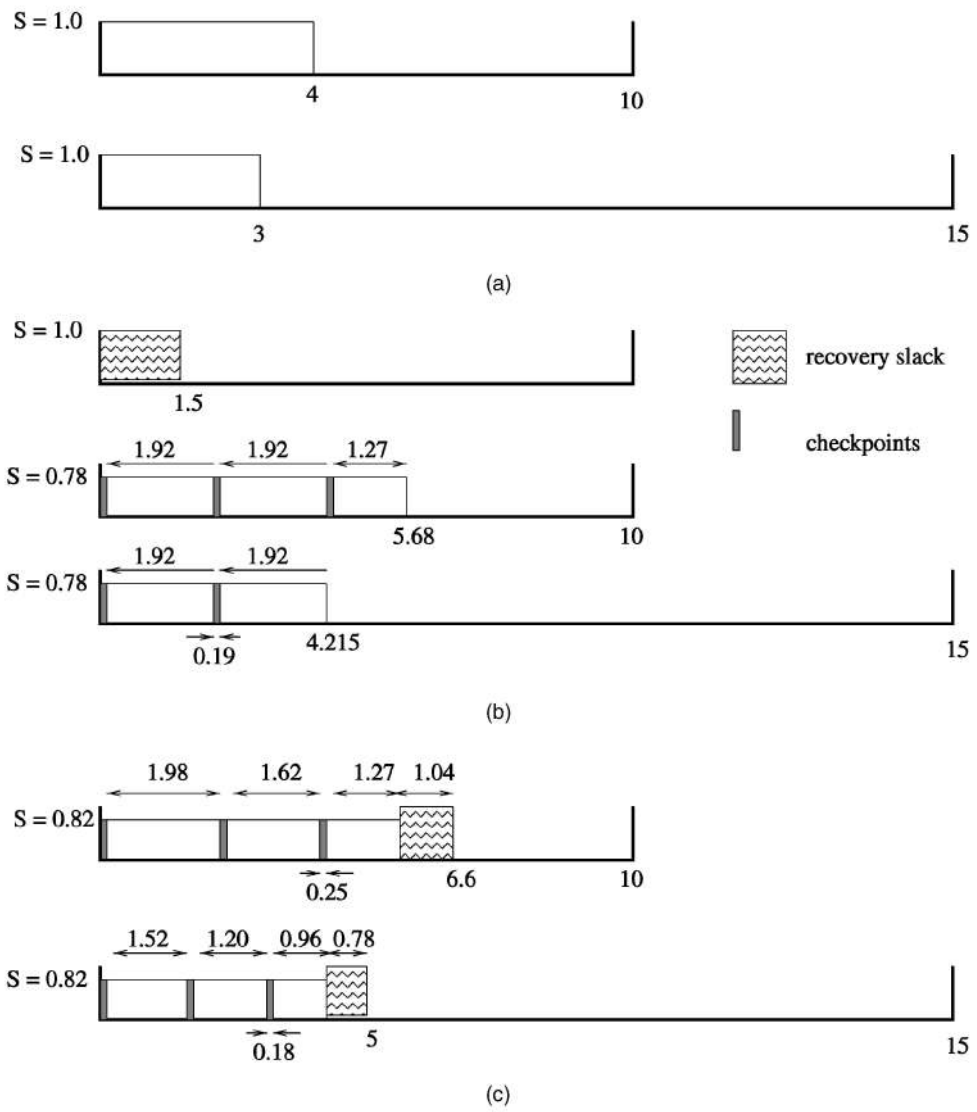

Fig. 7. An example of a two-task system. (a) The task instances at maximum speeds and without checkpoints. (b) The task instances with uniform checkpointing. (c) The task instances with nonuniform checkpointing.

40 percent slack in the system, which can be used to guarantee error recovery and/or save power consumption. Let us assume that the cost of a checkpoint is fixed at $r = 0.15$.

If we are to apply the scheme of Section 5.1, we get $a = 0.625$ and the solution found from (13) is $\gamma = 1.5$. That is, three checkpoints should be inserted in C_1 and two in C_2, as shown in Fig. 7b. The optimum speed is found from (12) to be $S = 0.783$ and the energy consumption during the LCM is found from (17) to be $E = 0.52LCM$. Note that, at $S = 0.783$, the checkpoints are separated in time by $\frac{\gamma}{S} = 1.92$ time units and the checkpoints consume $\frac{r}{S} = 0.192$ time units. After adding the checkpoints, the utilization of τ_1 is $\frac{4.45}{10}\frac{1}{0.783} = 0.568$ and that of τ_2 is $\frac{3.3}{15}\frac{1}{0.783} = \frac{4.215}{15} = 0.281$. Adding the utilization of the slack reserved for recovery at maximum speed, namely, $\frac{1.5}{10} = 0.15$, the total system utilization is 0.999, which guarantees that the two tasks are schedulable by EDF and that a slack of 1.5 is available every period T_1 for recovery.

On the other hand, if we apply the scheme of Section 5.2, we would increase the original utilizations of τ_1 and τ_2 by a factor of $\frac{1}{U} = 1.66$. That is, the allocation of τ_1 is increased from 4 to 6.66 time units every period $T_1 = 10$ and the allocation of τ_2 is increased from 3 to 5 time units every $T_2 = 15$ time units. The new allocation of each task is used to add its own recovery slack and to slow down its own computation. Given that $\rho_1 = \frac{0.15}{6.66} = 0.02$ and $\rho_2 = \frac{0.15}{5} = 0.03$, we use $\rho = 0.03$ and $\sigma = 0.6$ for both tasks to derive the number of checkpoints and speed, which leads to the same speed for both tasks and, thus, avoids speed changes in the absence of faults. From Table 2, we find that three checkpoints should be used for each task and, from (10), we find that the speed S should be set to 0.817. For τ_1, (8) and (9) with $r = \rho D_1 = 0.3 * 6.66 = 0.2$ give $C_1^{(3)} = 1.04$, $C_1^{(2)} = 1.32$, and $C_1^{(1)} = 1.64$. Thus, at speed $S = 0.817$, $\frac{C_1^{(3)}}{S} = 1.27$, $\frac{C_1^{(2)}}{S} = 1.62$ and $\frac{C_1^{(1)}}{S} = 1.98$ (see Fig. 7c).

TABLE 3
Energy Consumptions (per LCM) and Recovery Capabilities for the Two-Task Example

	Energy	Recovery capability
No checkpoints and no speed management	0.6 LCM	none
No checkpoints and speed management	0.36 LCM	none
Checkpointing and no speed management	0.65 LCM	one fault per task instance
Uniform checkpoints and speed management	0.52 LCM	one fault every 15 time units
Non-uniform checkpoints and speed management	0.56 LCM	one fault per task instance

For τ_2, the same equations give $C_2^{(3)} = 0.78$, $C_2^{(2)} = 0.98$, and $C_2^{(1)} = 1.24$. Thus, at speed $S = 0.817$, $\frac{C_2^{(3)}}{S} = 0.96$, $\frac{C_2^{(2)}}{S} = 1.20$, and $\frac{C_2^{(1)}}{S} = 1.52$. Over the LCM, the energy consumption during fault free operation is thus obtained from (15) to be $E = 0.56LCM$, which is higher than the first scheme, but allows for the recovery from one fault in every task instance.

Finally, for comparison purposes, consider that, with no power management and with no checkpoints, the energy consumption during an LCM period is $0.6LCM$. If two checkpoints are added to each task, which will allow for rollback after a fault in each task instance, and the system is run at the maximum speed (no power management), then the energy consumption during an LCM period increases to $0.65LCM$. If no checkpoints are taken and the entire slack is used to reduce the CPU speed, then the energy consumption during an LCM period is reduced to $0.36LCM$. Table 3 summarizes these results.

6 Implementation Issues

6.1 Error Detection and Recovery

Our assumptions about the failure model are standard and have been used by other researchers [33]. We address here some of the engineering issues in realizing this model in practice. First, we have focused on the common case where it is assumed that there is at most a single failure within a period of a periodic task or by the deadline of an aperiodic task. In practice, the period or deadline is usually short, making this assumption a realistic one. Nevertheless, it is straightforward to extend the results to k faults per period or per deadline task. The slack in this case must be enough for k rollbacks and the equations given in the paper may be easily modified to take multiple rollbacks into account. One can conjecture, though, that the maximum potential benefit in energy savings will be significantly reduced as the number of tolerated failures increases.

The second aspect of the fault model concerns the issues of detecting failures and ensuring the integrity of the checkpoints. We assume that, as part of the checkpointing process, some self-test and diagnosis are run before the checkpoint is committed. The time it takes to perform these tests can be considered as part of the overhead ρ. Several references exist on how to perform such tests and we do not address them here [2]. We also assume that the checkpoints will be stored in main memory to ensure predictability and fast response time, which are required in any real-time system. This can subject the checkpoints to contamination due to transient failures. Again, good engineering solutions exist for protecting the checkpoints while they are stored in main memory. These range from duplex memory to using sophisticated encoding schemes that can detect and correct one or more errors [17]. Of course, these methods differ in the degree of protection that they offer to the data in the checkpoints. Therefore, there is a trade off between cost, performance, and the desired degree of reliability that must be considered in any practical situation. Our techniques for power management are orthogonal to, and compatible with, these fault tolerance mechanisms.

The third aspect of the fault model is the time it takes to detect a failure and start recovery. Realistically, the failure will be detected either through time-outs or during the self-test prior to taking the checkpoint. In either case, restoring the state from a previous checkpoint and starting recovery cannot be done in zero time. We have not included this overhead in our analysis, but it can be incorporated into an implementation very easily, by moving the deadline forward. That is, if the time it takes to restore a checkpoint is t_r, then we set the deadline to be $D_r = D - t_r$ and use D_r in the analysis without change. This engineering change accommodates the time to perform recovery without affecting the fundamental conclusions of the theoretical analysis. If no failure occurs, the extra cycles available between D_r and D can be used for other computations or to shut down the processor completely if desired.

6.2 Voltage and Frequency Scaling

An assumption made in this paper is that the CPU voltage can be changed to any value within a continuum bracketed by the maximum and minimum voltages. In practice, however, the CPU voltage is usually set to a discrete number of values to simplify the implementation and circuitry. A simple approach to apply our results to the case of discrete CPU voltage setting is to apply the techniques described in this paper to find the optimum speed, S and then to set the CPU voltage to the closest available that can allow a speed larger than S. Such an approach may slightly increase energy consumption, but is necessary since selecting a speed smaller than S cannot guarantee that deadlines can be met. Note that the effect of having discrete speeds decreases when the granularity of the discretization is small. Current variable speed processors allow the CPU speed to change in increments of 33 MHz [7]. Smaller speed increments should be possible in the near future.

Finally, in the analysis presented in this paper, the overhead of changing the CPU speed is ignored. Nevertheless, we change the speed only when recovery starts and, therefore, the time overhead of changing the CPU speed can be easily incorporated into the recovery time. The power overhead of changing the speed will not affect the analysis in the paper since the goal of that analysis is to minimize the energy consumption during fault-free operation. In practice, the use of our techniques will be precluded in situations where the time to change the voltage and frequency is a substantial portion of the deadline or period. This typically occurs when the deadline or the period is extremely short. Note that, in such situations, the limitation is not confined to our work—power management may not be feasible in the first place, regardless of the desire to implement fault tolerance or not.

6.3 Dynamic Slack Reclamation

The power management considered in this paper follows a standard practice of considering the worst-case parameters for the workload involved. The values of σ and ρ, respectively, represent the maximum number of cycles to execute a task and the maximum number of cycles to perform diagnosis and checkpointing. This practice ensures the feasibility of the schedule under any circumstances and is commonly followed among engineers who implement real-time systems. During operation, however, it is usually the case that tasks do not need the estimated worst-case execution time, yielding more slack to the system. The management of such a dynamically created slack is not discussed in this paper, but the techniques that have been proposed for managing this slack to reduce power consumption may be applied at runtime to further reduce energy consumption beyond the reduction obtained in this paper [14], [19], [35].

7 Related Work

7.1 Power Management

Research in power management has picked up some momentum, driven by the needs of portable devices that operate on low power supply. There have been a number of studies of specific power management mechanisms and policies and a set of standards has been developed for the mechanisms, specifying the interfaces between power-management software and hardware. Examples of such architectures include the industry standard Advanced Configuration and Power Interface or ACPI [6] and Microsoft's OnNow initiative [29]. Many of these are directly aimed at laptop environments, but the mechanisms provided should prove useful to a system that uses the processor speed adjustment as described here.

Modern microprocessors and microarchitectures incorporate power-saving features. Examples include the mobile processors available from Intel with its SpeedStep [15] technology and the Transmeta Crusoe processor with LongRun [7]. The physical underpinnings and fundamental issues with these techniques can be found in [10], [25].

In the realm of real-time systems, variable voltage scheduling focuses on minimizing energy consumption of the system, while still meeting the deadlines. The seminal work by Yao et al. [37] provided a static offline scheduling algorithm, assuming aperiodic tasks and worst-case execution times. Heuristics for online scheduling of aperiodic tasks while not hurting the feasibility of offline periodic requests are proposed in [13]. Concentrating on a periodic task set with *identical periods*, the effects of having an upper bound on the voltage change rate are examined in [14], along with a heuristic to solve the problem.

Recent work in variable voltage scheduling includes the exploitation of idle intervals by slowing down the processor whenever there is a single task eligible for execution and its worst-case completion time is earlier than the first future arrival [35]. Although this *One Task Extension technique* was originally proposed in the context of Rate Monotonic Scheduling, it is easy to see that the idea can be applied to any periodic scheduling discipline. Cyclic and EDF scheduling of periodic hard real-time tasks on systems with multiple voltage levels, including dynamic energy reclaiming heuristics, have been investigated in [1], [19].

7.2 Fault Tolerance

To the best of our knowledge, this paper is the first work that attempts to bring together concepts from real-time systems, power management, and fault tolerance. Fault tolerance has been studied extensively in real-time systems and an encompassing literature survey is not in the scope of this paper. In some real-time systems such as satellites and space shuttles, transient faults occur at a much higher frequency than in general purpose systems [2]. In [2], an orbiting satellite containing a microelectronics test system was used to measure error rates in various semiconductor devices, including microprocessor systems. The number of errors caused by protons and cosmic ray ions mostly ranged between 1 and 15 in 15-minute intervals and was measured to be as high as 35 in such intervals. Physical redundancy [9], [31] and temporal redundancy [30] have been used to tolerate permanent and transient faults in real-time systems.

Transient faults in real-time systems are generally tolerated using time redundancy, which involves the retry or reexecution of any task running during the occurrence of a transient fault [9], [18], [34]. This is a relatively inexpensive method of providing fault tolerance since not much extra hardware is required. In space and aviation applications, reducing the hardware is important since that decreases weight, size, power consumption, and cost. Other studies have dealt with real-time scheduling providing tolerance to transient faults using a timeline and a primary-backup approach [20], [21], [26].

8 Conclusions

We have presented two checkpointing policies that allow a real-time system to recover from failure and reduce power consumption. Both policies enable the reduction of the processor speed to the level that yields minimum energy consumption during failure-free operation. If a failure occurs, the processor reexecutes the lost work at maximum speed to guarantee recovery and meet the task's deadlines.

The first policy places checkpoints uniformly within a task, as in traditional, periodic checkpointing schemes. We derived the optimal number of checkpoints that yields the lowest energy consumption, subject to the constraints of recovering from one fault and meeting the task's deadline. The analysis shows that, compared to a system that takes checkpoints only for the purpose of recovery, our policy can substantially reduce energy consumption and yet guarantee the same level of fault tolerance and timeliness.

The second policy takes the position that, since failures are rare, it is acceptable to put the processor at maximum speed once a failure occurs and until the deadline expires, and not just during failure recovery. This yields additional energy savings for some workloads, but requires nonuniform placement of checkpoints. We derived the optimal number of checkpoints for minimum energy consumption under this policy and derived the necessary formulas for checkpoint placement.

The results show that standard, periodic checkpointing is not the best policy for checkpoint placement when energy consumption is to be reduced. We then discussed how these results could also be applicable to periodic tasks. To our knowledge, this is the first work that combines the aspects of real-time scheduling, power consumption, and reliability. Future work will include how these ideas could be carried to replication-based systems and the engineering aspects that must be addressed for actual implementations of these theoretical concepts.

APPENDIX

In this Appendix, we present the proofs of the lemmas used in the paper.

Proof of Lemma 1. First, (8) is directly obtained by considering $k = n$ in (7) and using $\sum_{i=1}^{n} C^{(i)} = C$.

Next, consider $k = n - 1$ in (7), which gives

$$\begin{aligned} D &= \sum_{i=1}^{n-1} \frac{(r + C^{(i)})}{S} + C^{(n-1)} + r + C^{(n)} \\ &= \frac{(nr + C)}{S} - \frac{(r + C^{(n)})}{S} + C^{(n-1)} + r + C^{(n)}. \end{aligned}$$

By using (8) in the above equation, we conclude that (9) is correct for $k = n - 1$. The proof of (9) for $k < n - 1$ proceeds by induction. Specifically, assuming that

$$C^{(u)} + r = \frac{C^{(u+1)} + r}{S}, \tag{16}$$

we prove that (9) is true for $u - 1$. Starting from (7), we get

$$\begin{aligned} D &= \sum_{i=1}^{u-1} \frac{(r + C^{(i)})}{S} + C^{(u-1)} + \sum_{i=u}^{n} (r + C^{(i)}) \\ &= \sum_{i=1}^{n} \frac{(r + C^{(i)})}{S} + C^{(u-1)} + \sum_{i=u}^{n} \left(r + C^{(i)} - \frac{(r + C^{(i)})}{S} \right) \\ &= \frac{(nr + C)}{S} + C^{(u-1)} - \frac{(r + C^{(u)})}{S} + (r + C^{(n)}). \end{aligned}$$

The last step is obtained by expanding the last summation term and using (16). By using (8), we obtain

$$C^{(u-1)} + r = \frac{C^{(u)} + r}{S},$$

which completes the induction proof and, thus, the proof of the lemma. □

Proof of Lemma 2. Consider a modified task set in which task τ_i, for some i, is replaced by another task, $\bar{\tau}_i$ with the same period, T_i, but with computation time $C_i + \gamma$. Since $T_1 \leq T_i$, the utilization of the new task set is $\sum_{i=1}^{N} \frac{C_i}{T_i} + \frac{\gamma}{T_i} \leq 1$. Hence, in EDF execution, every task in the set, including $\bar{\tau}_i$, will finish before the end of its period. If $\bar{\tau}_i$ executes only for C_i rather than $C_i + \gamma$, then it will finish γ units before the end of the period T_i. In other words, in the EDF execution of $\tau_1, \ldots, \tau_N$, task τ_i finishes execution γ units before its deadline. This argument is true for any i between 1 and N and, thus, every task will finish γ time units before its deadline. □

Proof of Lemma 3. In the absence of faults, the energy consumed during the execution of each instance of a task τ_i is proportional to the product of S^2 and the time to execute that instance. That is,

$$cS^2 \frac{\left(C_i + \left\lceil \frac{C_i}{\gamma} \right\rceil r_i \right)}{S},$$

where, as before, c is a proportionality constant. Denoting by LCM the least common multiple of all the periods, $T_1, \ldots, T_N$, the energy consumed during an LCM period is

$$E = \sum_{i=1}^{N} cS \left(C_i + \left\lceil \frac{C_i}{\gamma} \right\rceil r_i \right) \frac{LCM}{T_i},$$

which, combined with (12), yields

$$E = c \left(\sum_{i=1}^{N} \frac{C_i + \left\lceil \frac{C_i}{\gamma} \right\rceil r_i}{T_i} \right)^2 \frac{1}{1 - \frac{\gamma}{T_1}} LCM. \tag{17}$$

Now, to find the checkpoint interval, γ which minimizes the energy consumption, we need to differentiate (17) with respect to γ. For this, we first bound $\lceil \frac{C_i}{\gamma} \rceil$ by $\frac{C_i}{\gamma} + 1$ and use $a = \sum_{i=1}^{N} \frac{C_i + r_i}{T_i}$ and $b = \sum_{i=1}^{N} \frac{C_i}{T_i} r_i$ to rewrite (17) as

$$E = c \left(a + \frac{b}{\gamma} \right)^2 \frac{1}{1 - \frac{\gamma}{T_1}} LCM,$$

which, when differentiated with respect to γ and equated to zero, gives

$$a\gamma^2 + 3b\gamma - 2bT_1 = 0$$

and, thus, after ignoring the negative root, results in the optimal value of γ given by (13). □

ACKNOWLEDGMENTS

This work has been supported by the US Defense Advanced Research Projects Agency through the PARTS (Power-Aware Real-Time Systems) project under Contract F33615-00-C-1736.

REFERENCES

[1] H. Aydin, R. Melhem, D. Mossé, and P. Alvarez, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS '99),* Dec. 1999.

[2] A. Campbell, P. McDonald, and K. Ray, "Single Event Upset Rates in Space," *IEEE Trans. Nuclear Science,* vol. 39, no. 6, pp. 1828-1835, 1992.

[3] X. Castillo, S. McConnel, and D. Siewiorek, "Derivation and Calibration of a Transient Error Reliability Model," *IEEE Trans. Computers,* vol. 31, no. 7, pp. 658-671, July 1982.

[4] The Standard Performance Evaluation Corp., http://www.specbench.org, 2003.

[5] E. Elnozahy, L. Alvisi, Y.-M. Wang, and D. Johnson, "A Survey of Rollback-Recovery Protocols in Message Passing Systems," technical report, Carnegie Mellon Univ., 1999.

[6] Compaq et al., "ACPI Specification, Version 2.0," 2000.

[7] M. Fleischmann, "Crusoe Power Management: Cutting x86 Operating Power through LongRun," *Embedded Processor Forum,* June 2000.

[8] S. Ghosh, R. Melhem, D. Mossé, and J. Sarma, "Fault-Tolerant Rate-Monotonic Scheduling," *J. Real-Time Systems,* vol. 15, no. 2, Sept. 1998.

[9] S. Ghosh, D. Mossé, and R. Melhem, "Implementation and Analysis of a Fault-Tolerant Scheduling Algorithm," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 3, Mar. 1997.

[10] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE J. Solid-State Circuits,* vol. 31, no. 9, Sept. 1996.

[11] F. Gruian, "Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 46-51, 2001.

[12] V. Gutnik and A. Chandrakasan, "An Efficient Controller for Variable Supply Voltage Low Power Processing," *Proc. Symp. VLSI Circuits,* pp. 158-159, 1996.

[13] I. Hong, M. Potkonjak, and M. Srivastava, "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor," *Proc. Computer-Aided Design (ICCAD '98),* pp. 653-656, 1998.

[14] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava, "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors," *Proc. 19th IEEE Real-Time Systems Symp. (RTSS '98),* Dec. 1998.

[15] Intel Corp, "SpeedStep," http://developer.intel.com/mobile/PentiumIII, 2003.

[16] R. Iyer, D. Rossetti, and M. Hsueh, "Measurement and Modeling of Computer Reliability as Affected by System Activity," *ACM Trans. Computer Systems,* vol. 4, no. 3, pp. 214-237, Aug. 1986.

[17] B. Johnson, *Design and Analysis of Fault Tolerant Digital Systems.* Addison Wesley, 1989.

[18] H. Kopetz, H. Kantz, G. Grunsteidl, P. Puschner, and J. Reisinger, "Tolerating Transient Faults in MARS," *Digest of Papers, 20th Ann. Int'l Symp. Fault-Tolerant Computing (FTCS-20),* pp. 466-473, June 1990.

[19] C. Krishna and Y. Lee, "Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems," *Proc. Sixth IEEE Real-Time Technology and Applications Symp. (RTAS '00),* May 2000.

[20] C. Krishna and K. Shin, "On Scheduling Tasks with a Quick Recovery from Failure," *IEEE Trans. Computers,* vol. 35, no. 5, pp. 448-455, May 1986.

[21] A. Liestman and R. Campbell, "A Fault-Tolerant Scheduling Problem," *IEEE Trans. Software Eng.,* vol. 12, no. 11, pp. 1089-1095, Nov. 1986.

[22] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, pp. 46-61, 1973.

[23] J. Lorch and A. Smith, "Improving Dynamic Voltage Scaling Algorithms with PACE," *Proc. ACM SIGMETRICS 2001,* June 2001.

[24] T. Ma and K. Shin, "A User-Customizable Energy-Adaptive Combined Static/Dynamic Scheduler for Mobile Applications," *Proc. 21st IEEE Real-Time Systems Symp. (RTSS '00),* pp. 227-236, 2000.

[25] A. Martin, A. Lines, R. Manohar, M. Nystrm, P. Penzes, R. Southworth, U. Cummings, and T. Lee, "The Design of an Asynchronous MIPS R3000 Microprocessor," *Proc. 17th Conf. Advanced Research in VLSI,* Sept. 1997.

[26] A. Mehra, J. Rexford, H. Ang, and F. Jahanian, "Design and Evaluation of a Window-Consistent Replication Service," *Proc. Real-Time Technology and Applications Symp.,* 1995.

[27] H. Mehta, R. Owens, M. Irwin, R. Chen, and D. Ghosh, "Techniques for Low Energy Software," *Proc. 1997 Int'l Symp. Low Power Electronics,* 1997.

[28] R. Melhem, N. AbouGhazaleh, H. Aydin, and D. Mosse, "Power Management Points in Power-Aware Real-Time Systems," *Power-Aware Computing,* R. Graybill and R. Melhem, eds., Kluwer/Plenum Series in Computer Science, Jan. 2002.

[29] Microsoft Corp., *PC99 System Design Guide.* Microsoft Press, 1999.

[30] Y. Oh and S. Son, "Enhancing Fault-Tolerance in Rate-Monotonic Scheduling," *J. Real-Time Systems,* vol. 7, no. 3, pp. 315-329, Nov. 1994.

[31] Y. Oh and S. Son, "Scheduling Hard Real-Time Tasks with Tolerance of Multiprocessor Failures," *Microprocessing and Microprogramming,* pp. 193-206, 1994.

[32] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. Design Automation of Electronics Systems,* vol. 1, no. 1, pp. 3-56, Jan. 1996.

[33] S. Punnekkat, A. Burns, and R. Davis, "Analysis of Checkpointing for Real-Time Systems," *Real-Time Systems J.,* vol. 20, no. 1, pp. 83-102, Jan. 2001.

[34] S. Ramos-Thuel and J. Strosnider, "Scheduling Fault Recovery Operations for Time-Critical Applications," *Proc. Fourth IFIP Conf. Dependable Computing for Critical Applications,* Jan. 1995.

[35] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. 36th Design Automation Conf. (DAC '99),* pp. 134-139, 1999.

[36] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, "Energy-Aware Runtime Scheduling for Embedded-Multiprocessor SOCs," *IEEE Design and Test of Computers,* vol. 18, no. 5, pp. 46-58, Sept./Oct. 2001.

[37] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," *Proc. IEEE Ann. Foundations of Computer Science,* pp. 374-382, 1995.

Rami Melhem received the BE degree in electrical engineering from Cairo University in 1976, the MA degree in mathematics and the MS degree in computer science from the University of Pittsburgh in 1981, and the PhD degree in computer science from the University of Pittsburgh in 1983. He was an assistant professor at Purdue University prior to joining the faculty of the University of Pittsburgh in 1986, where he is currently a professor of computer science and electrical engineering and the chair of the Computer Science Department. His research interests include real-time and fault-tolerant systems, optical interconnection networks, high performance computing, and parallel computer architectures. He serves on the advisory boards of the IEEE technical committees on Parallel Processing and on Computer Architecture. He is the editor for the Kluwer/Plenum Book Series in Computer Science and is on the editorial board of *Computer Architecture Letters*. He is a fellow of the IEEE and a member of the ACM.

Daniel Mossé received the BS degree in mathematics from the University of Brasilia in 1986 and the MS and PhD degrees in computer science from the University of Maryland in 1990 and 1993, respectively. He joined the faculty of the University of Pittsburgh in 1992, where he is currently an associate professor. His research interests include fault-tolerant and real-time systems, as well as networking. The major thrust of his research in the new millenium is power-aware computing and security. He is a member of the editorial board of the *IEEE Transactions on Computers* and a member of the IEEE, IEEE Computer Society, and ACM.

Elmootazbellah (Mootaz) Elnozahy received the BSc degree in electrical engineering from Cairo University in 1984 and the MS and PhD degrees in computer science from Rice University in 1990 and 1994, respectively. From 1993 until 1997, he was on the faculty at the School of Computer Science at Carnegie Mellon University, where he received a US National Science Foundation (NSF) CAREER award. Since 1997, he has been with the IBM Austin Research Lab, where he leads the Systems Software Department. He is also an adjunct associate professor at the University of Texas at Austin. He currently serves as an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*. In 2002, he was designated an IBM Research Master Inventor. His research interests include distributed systems, operating systems, computer architecture, and fault tolerance. He has published more than 25 regularly cited articles in these areas and obtained 12 patents. He is a member of the IEEE.