Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems*

Ying Zhang and Krishnendu Chakrabarty
Department of Electrical & Computer Engineering
Duke University
Durham, NC 27708, USA
E-mail: {*yingzh, krish*}*@ee.duke.edu*

Abstract

We present an integrated approach that provides fault tolerance and dynamic power management for a real-time task executing in an embedded system. Fault tolerance is achieved through an adaptive checkpointing scheme that dynamically adjusts the checkpointing interval during task execution. Adaptive checkpointing is then combined with a dynamic voltage scaling scheme to achieve power reduction. The resulting energy-aware adaptive checkpointing scheme uses a dynamic voltage scaling criterion that is based not only on the slack in task execution but also on the occurrences of faults during task execution. Simulation results show that compared to previous methods, the proposed approach significantly reduces power consumption and increases the likelihood of timely task completion in the presence of faults.

1. Introduction

Embedded systems often operate in harsh environmental conditions that necessitate the use of fault-tolerant computing techniques to ensure dependability. These systems are also severely energy-constrained since system lifetime is determined to a large extent by the battery lifetime. In addition, many embedded systems execute real-time applications that require strict adherence to task deadlines [1]. In this paper, we present an integrated approach that provides fault tolerance and dynamic power management for a real-time task executing in an embedded system.

Dynamic voltage scaling (DVS) has emerged as a popular solution to the problem of reducing power consumption during system operation [2, 3, 4]. Many embedded processors are now equipped with the ability to dynamically scale the operating voltage. Since a reduction in voltage results in a corresponding drop in the processor speed, a number of techniques have been proposed recently to balance real-time responsiveness with low-energy task execution.

Fault tolerance is typically achieved in real-time systems through on-line fault detection [5], checkpointing and rollback recovery [6]. Figure 1 illustrates checkpointing and rollback recovery. At each checkpoint, the system saves its state in a secure device. When a fault is detected, the system rolls back to the most recent checkpoint and resumes normal execution.

Checkpointing increases task execution time and in the absence of faults, it might cause a missed deadline for a task that completes on time without checkpointing. In the presence of faults however, checkpointing can increase the likelihood of a task completing on time with the correct result. Without checkpointing, a fault necessitates the restart of the task. Frequent checkpointing reduces re-computation time due to faults, but it increases task execution time. On the other hand, infrequent checkpointing has less impact on task execution in the absence of faults, but it increases the amount of rollback that must be performed after a fault is detected. Therefore, the checkpointing interval, i.e., duration between two consecutive checkpoints, must be carefully chosen to balance checkpointing cost (the time needed to perform a single checkpoint) with the rollback time.

Dynamic power management and fault tolerance for embedded real-time systems have been studied as separate problems in the literature. DVS techniques for power management do not consider fault tolerance [2, 3, 4], and checkpoint placement strategies for fault tolerance do not address dynamic power management [7, 8, 9]. We present an integrated approach that facilitates fault tolerance through checkpointing and power management through DVS. To the best of our knowledge, this is the first approach that addresses these two issues in conjunction. The main contributions of this paper are as follows.

- We introduce an adaptive checkpointing scheme that dynamically adjusts the checkpointing interval during task execution, based on the frequency of fault occurrences and the amount of time remaining before the task deadline.
- The proposed adaptive checkpointing scheme is tailored to handle not only a random fault-arrival process, but it is also designed to tolerate up to *k* fault occurrences.
- Adaptive checkpointing is then combined with a DVS scheme to achieve power reduction and fault tolerance simultaneously. The resulting energy-aware adaptive checkpointing scheme uses a dynamic speed scaling criterion that is based not only on the slack in task

*Supported in part by DARPA and Army Research Office under Award No. DAAD19-01-1-0504. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the view of the DARPA and ARO agencies.

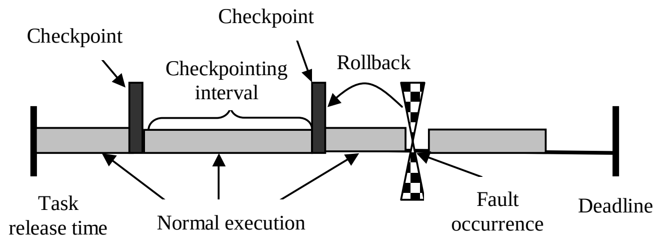

Figure 1: Checkpointing and rollback recovery.

execution but also on the occurrences of faults during task execution.

We assume throughout that faults are intermittent or transient in nature, and that permanent faults are handled through manufacturing testing or field-testing techniques [10]. We also assume a "soft" real-time system in which even though it is important to meet task deadlines, missed deadlines do not lead to catastrophic consequences [11].

The rest of the paper is organized as follows. Section 2 introduces some relevant background material on checkpointing. Section 3 presents our adaptive checkpointing scheme for real-time systems. In Section 4, we describe how dynamic voltage scaling (DVS) is incorporated into the adaptive checkpointing scheme. Conclusions and directions for future work are presented in Section 5.

2. Checkpointing in real-time systems

In this section, we present a classification of checkpointing schemes for real-time systems that have been presented in the literature.

2.1 On-line scheme versus off-line schemes

An off-line checkpointing scheme determines the checkpointing interval for a task *a priori*, i.e., before task execution. Most known checkpointing schemes for real-time systems belong to this category [9, 12, 13]. A drawback here is that the checkpointing interval cannot be adapted to the actual fault occurrence during task execution. An on-line scheme in which the checkpointing interval can be adapted to fault occurrences is therefore more desirable. However, current on-line checkpointing schemes [8] provide only probabilistic guarantees on the timely completion of tasks, as described next.

2.2 Probabilistic versus deterministic guarantees

Some checkpointing schemes, e.g. [9, 12], assume that faults occur as a Poisson process with arrival rate λ. These schemes use a checkpointing interval that maximizes the probability that a task completes on time for a given fault arrival rate λ. Hence the real-time guarantees in these schemes are probabilistic. Other checkpointing schemes, e.g., [13], offer deterministic real-time guarantees under the condition that at most k faults occur during task execution. A drawback of these k-fault-tolerant schemes is that they cannot adapt to actual fault occurrences during task execution.

2.3 Equidistant versus variable checkpointing interval

Equidistant checkpointing, as the term implies, relies on the use of a constant checkpointing interval during task execution. This is typically used with off-line checkpointing schemes. It has been shown in the literature that if the checkpointing cost is C and faults arrive as a Poisson process with rate λ, the mean execution time for the task is minimum if a constant checkpointing interval of $\sqrt{2C/\lambda}$ is used [12]. We refer to this as the Poisson-arrival approach. However, the minimum execution time does not guarantee timely completion of a task under real-time deadlines. It has also been shown that if the fault-free execution time for a task is E, the worst-case execution time for up to k faults is minimum if the constant checkpointing interval is set to $\sqrt{EC/k}$ [13]. We refer to this as the k-fault-tolerant approach. A drawback with these equidistant schemes is that they cannot adapt to actual fault arrivals. For example, due to the random nature of fault occurrences, the checkpointing interval can conceivably be increased halfway through task execution if only a few faults occur during the first half of task execution. Another drawback of equidistant checkpointing schemes is that they do not exploit the advantages offered by DVS for dynamic power management. For these reasons, we consider on-line checkpointing with variable checkpointing intervals.

2.4 Constant versus variable checkpointing cost

Most prior work has been based on the assumption that all checkpoints take the same amount of time, i.e., the checkpointing cost is constant. An alternative approach, taken in [8], but less well understood, is to assume that the checkpointing cost depends on the time at which it is taken. We use the constant checkpointing cost model in our work because of its inherent simplicity.

Our goal in this paper is to develop a two-priority energy-aware checkpointing scheme for real-time systems. The first priority is to meet the real-time deadline under faulty conditions. The second priority is to save energy consumption for real-time execution in faulty conditions.

3. Adaptive checkpointing

We are given the following parameters for a real-time task: deadline D; execution time E when there are no fault in the system ($E < D$); an upper limit k on the number of fault occurrences that must be tolerated; checkpointing cost C. We make the following assumptions related to task execution and fault arrivals:

- Faults arrive as a Poisson process with rate λ.
- The task starts execution at time $t = 0$.
- The times for rollback and state restoration are zero.
- Faults are detected as soon as they occur, and no faults occur during checkpointing and rollback recovery.

We next determine the maximum value of E for the Poisson-arrival and the k-fault-tolerant schemes beyond which these schemes will always miss the task deadline. Our proposed adaptive checkpointing scheme is more likely to meet the task deadline even when E exceeds these threshold values. If the Poisson-arrival scheme is used, the effective task execution time in the absence of faults must be less than

```
Procedure interval($R_d$,$R_t$,$C$,$R_f$,$\lambda$)
1. $Exp\_fault = \lambda R_t$;
2. if ($Exp\_fault \le R_f$) {
3.     if ($Rt > Th_\lambda(Rd,\lambda,C)$) then
           $chk\_interval = I_3(R_t,R_d,C)$;
4.     else if ($R_t > Th(Rd,Rf,C)$) then
           $chk\_interval = I_2(R_t,Exp\_fault,C)$;
5.     else  $chk\_interval = I_2(R_t,R_f,C)$;}
6. else {if ($R_t > Th_\lambda(R_d,\lambda,C)$) then
           $chk\_interval = I_3(R_t,R_d,C)$;
7.     else  $chk\_interval = I_1(C, \lambda)$;}
8. return $chk\_interval$;
```

Figure 2: Procedure for calculating the checkpointing interval.

the deadline D if the probability of timely completion of the task in the presence of faults is to be nonzero. This implies that $E+(E/(\sqrt{2C/\lambda})-1)C \le D$, from which we get the threshold:

$$E_{\lambda th} = (D+C)/(1+\sqrt{\lambda C/2}) \quad (1)$$

Here $(E/(\sqrt{2C/\lambda})-1)$ refers to the number of checkpoints. The re-execution time due to rollback is not included in the formula for $E_{\lambda th}$. If E exceeds $E_{\lambda th}$ for the Poisson-arrival approach, the probability of timely completion of the task is simply zero. Therefore, beyond this threshold, the checkpointing interval must be set by exploiting the slack time instead of utilizing the optimum checkpointing interval for the Poisson-arrival approach. The checkpointing interval I_m that barely allows timely completion in the fault-free case is given by $E+(E/I_m-1)C = D$, from which it follows that $I_m = EC/(D+C-E)$. To decrease the checkpointing cost, we set the checkpointing interval to $2I_m$ in our adaptive scheme (details are given in Section 3.1).

A similar threshold on the execution time can easily be calculated for the k-fault-tolerant scheme. In order to satisfy the k-fault-tolerant requirement, the worst-case re-execution time is incorporated. The following inequality must hold:

$$E+(E/(\sqrt{EC/k})-1)C+k\sqrt{EC/k} \le D.$$

This implies the following threshold on E:

$$E_{kth} = ((D+C)+2kC)-2\sqrt{kC(D+C)+(kC)^2} \quad (2)$$

If the execution time E exceeds E_{kth}, the k-fault-tolerant checkpointing scheme cannot provide a deterministic guarantee to tolerate k faults.

3.1 Checkpointing algorithm

The adaptive checkpointing algorithm attempts to maximize the probability that the task completes before its deadline despite the arrival of faults as a Poisson process with rate λ. A secondary goal is to tolerate, as for a possible, up to k faults. In this way, the algorithm accommodates a pre-defined fault-tolerance requirement (handle up to k faults) as well as dynamic fault arrivals modeled by the Poisson process. We list below some notation that we use in our description of the algorithm:

```
Procedure adapchp($D$,$E$,$C$, $k$,$\lambda$)
1. $R_t = E$; $R_d = D$; $R_f = k$; $Itv$ = interval($R_d$,,$R_t$,$C$,$R_f$, $\lambda$);
2. while ($R_t > 0$) do{
3.   if ($R_t > R_d$)  break;
4.   Case 1: During normal execution, do{
       4.1 Insert checkpoints with interval length $Itv$;
       4.2 Update $R_t$, $R_d$;}
5.   Case 2:  Upon fault occurrence, do{
       5.1 Roll back and restore status;
       5.2 $R_f = R_f - 1$;
       5.3 $Itv$ = interval($R_d$,$R_t$,$C$,$R_f$,$\lambda$);
       5.4 Resume execution;}}
```

Figure 3: Adaptive checkpointing procedure.

1) $I_1(C,\lambda)=\sqrt{2C/\lambda}$ denotes the checkpointing interval for the Poisson-arrival approach.
2) $I_2(E,k,C)=\sqrt{EC/k}$ denotes the checkpointing interval for the k-fault-tolerant approach.
3) $I_3(E,D,C) = 2I_m = 2EC/(D+C-E)$ denotes the checkpointing interval if the Poisson-arrival approach is not feasible for timely task completion.
4) R_t denotes the remaining execution time. It is obtained by subtracting from E the amount of time the task has executed (not including checkpointing and recovery).
5) R_d denotes the time left before the deadline. It is obtained by subtracting the current time from D.
6) R_f denotes an upper bound on the remaining number of faults that must be tolerated.
7) The threshold $Th_\lambda(R_d,\lambda,C)$ is obtained by replacing D with R_d in (1).
8) The threshold $Th(R_d,R_f,C)$ is obtained by replacing D with R_d and k with R_f in (2).

The procedure *interval*(R_d,R_t,C,R_f,λ) for calculating the checkpointing interval is described in Figure 2, and the adaptive checkpointing scheme *adapchp*(D,E,C,k,λ) is described in Figure 3. The adaptive checkpointing procedure is event-driven and the checkpointing interval is adjusted when a fault occurs and rollback recovery is performed.

In Figure 2, we first calculate the number of faults *Exp_fault* that are expected to occur in the remaining time R_t (Line 1). If *Exp_fault* is less than or equal to R_f, the k-fault-tolerant requirement is deemed to be more stringent than the Poisson-arrival criterion (Line 2). In Line 3, a check is performed to see if R_t exceeds the threshold $Th_\lambda(R_d,\lambda,C)$. If this condition is satisfied, the checkpointing interval is set to $I_3(R_t,R_d,C)$. In Line 4, a check is performed to see if R_t exceeds threshold $Th(R_d,R_f,C)$ but is below $Th_\lambda(R_d,\lambda,C)$. If this condition is satisfied, the checkpointing interval is set to $I_2(Rt,Exp\_fault,C)$. If the k-fault-tolerant threshold is met, the checkpointing interval is set to $I_2(R_t,R_f,C)$ in Line 5. Lines 6-7 handle the case when the k-fault-tolerant requirement is deemed to be less stringent than the Poisson-arrival criterion.

In Figure 3, Line 1 initializes the parameters. In Line 2, a check is performed to see if the task has been completed. Line 3 checks for the deadline constraint. Line 4 handles the case for normal execution. It inserts checkpoints and updates R_d and R_t. Line 5 handles the case for fault occurrences.

3.2 Simulation results on adaptive checkpointing

We carried out a set of simulation experiments to evaluate the adaptive checkpointing scheme (referred to as ADT) and to compare it with the Poisson-arrival and the *k*-fault-tolerant checkpointing schemes. Faults are injected into the system using a Poisson process with various values for the arrival rate λ. Due to the stochastic nature of the fault arrival process, the experiment is repeated 10,000 times for the same task and the results are averaged over these runs. We are interested here in the probability P that the task completes on time, i.e., either on or before the stipulated deadline. As in [11], we use the term task utilization U to refer to the ratio E/D.

For $\lambda < 0.002$ and $U < 0.7$ (low fault arrival rate and low task utilization), the performances of the three schemes, measured by the probability of timely completion of the task, are comparable. For $\lambda > 0.002$ and $U > 0.7$ (relatively high fault arrival rate as well as high task utilization), the adaptive checkpointing scheme clearly outperforms the other two schemes; the results are shown in Table 1. The value of P is as much as 30% higher for the ADT scheme. Note that even though the results are reported only for $D = 10000$, $C = 10$, and $k = 10$, similar trends were observed for other values of D, C, and k. For $\lambda < 0.002$ and $U \geq 0.9$ (low fault arrival rate and high task utilization), the ADT scheme outperforms the other two schemes; see Table 2.

To further illustrate the advantage of the ADT scheme, we note that if we set $U = 0.99$ and $k = 1$ (using the values of D and C as before), the value of P drops to zero for both the Poisson-arrival and the *k*-fault-tolerant schemes if $\lambda > 3\times10^{-5}$. In contrast, the proposed ADT scheme continues to provide significant higher value of P as λ increases (Table 3).

For $\lambda > 0.002$ and $U \geq 0.9$ (high fault arrival rate and high task utilization), the ADT scheme again outperforms the other two schemes. The results are not shown here due to lack of space.

In conclusion, we note that the ADT scheme is more likely to meet task deadlines when the task utilization is high and the fault arrival rate is not very low. In many cases, up to 50% increase is obtained in the probability of timely task completion.

4. ADT-DVS: Adaptive checkpointing with DVS

We next show how adaptive checkpointing scheme can be combined with DVS to obtain fault tolerance and power savings in real-time systems. We consider adaptive intra-task voltage scaling, wherein the processor speed is scaled

$U = E/D$	Fault arrival rate λ ($\times10^{-2}$)	Probability of timely completion of tasks, P: Poisson-arrival	*k*-fault-tolerant	ADT
0.80	0.24	0.505	0.476	0.532
	0.28	0.229	0.243	0.273
0.82	0.24	0.204	0.168	0.235
	0.28	0.052	0.042	0.092

(a)

λ ($\times10^{-2}$)	$U = E/D$	Probability of timely completion of tasks, P: Poisson-arrival	*k*-fault-tolerant	ADT
0.26	0.76	0.887	0.888	0.909
	0.78	0.655	0.666	0.715
0.30	0.76	0.864	0.823	0.872
	0.78	0.589	0.597	0.626

(b)

Table 1: (a) Variation of P with λ (b) Variation of P with U, for $D = 10000$, $C = 10$, and $k = 10$.

$U = E/D$	Fault arrival rate λ ($\times10^{-4}$)	Probability of timely completion of tasks, P: Poisson-arrival	*k*-fault-tolerant	ADT
0.92	1.0	0.902	0.945	0.947
	2.0	0.770	0.786	0.831
0.95	1.0	0.659	0.649	0.774
	2.0	0.372	0.387	0.513

(a)

λ ($\times10^{-4}$)	$U = E/D$	Probability of timely completion of tasks, P: Poisson-arrival	*k*-fault-tolerant	ADT
1.0	0.92	0.902	0.945	0.947
	0.94	0.747	0.818	0.852
2.0	0.92	0.770	0.786	0.831
	0.94	0.573	0.558	0.643

(b)

Table 2: (a) Variation of P with λ (b) Variation of P with U, for $D = 10000$, $C = 10$, and $k = 1$.

$U = E/D$	Fault arrival rate λ ($\times10^{-5}$)	Probability of timely completion of tasks, P: Poisson-arrival	*k*-fault-tolerant	ADT
0.99	1.0	0.893	0.000	0.907
	3.0	0.000	0.000	0.732
	5.0	0.000	0.000	0.515

Table 3: Variation of P with λ, for $D = 10000$, $C = 10$, and k=1.

up in response to a need for increased slack for checkpointing, and scaled down to save power if the slack for a lower speed is adequate. We consider a two-speed processor here—the extension to more than two speeds appears to be straightforward and it is left for future work. For the sake of simplicity, we use the terms processor speed and processor frequency interchangeably.

We use the same notation as described in Section 3.1. In addition, we are given the following:

1) A single processor with two speeds f_1 and f_2. Without loss of generality, we assume that $f_2 = 2f_1$.
2) The processor can switch its speed in a negligible amount of time (relative to the task execution time).
3) The number of computation cycles N for the task in the fault-free condition.

The objective here consists of two priorities. The first priority is to maximize the probability that the task meets its deadline in the presence of faults. The second priority is to reduce energy consumption through DVS.

We note that if supply voltage V_{dd} is used for a task with N single-cycle instructions, the energy consumption is proportional to NV^2_{dd}. We also note that the clock period is proportional to $V_{dd}/(V_{dd} - V_t)^2$, where V_t is the transistor threshold voltage. We assume here without loss of generality that $V_t = 0.8$ V, and the supply voltage V_{dd1} corresponding to speed f_1 is 2.0 V. Using the formula for the clock period, we find that the supply voltage V_{dd2} corresponding to speed f_2 is 2.8 V.

Let R_c be the number of instructions of the task that remain to be executed at the time of the voltage scaling decision. Let c be the number of clock cycles that a single checkpoint takes. We first determine if processor frequency f can be used to complete the task before the deadline. As before, let R_d be the amount of time left before the task deadline. The checkpointing cost C at frequency f is given by: $C = c/f$. Let t_{est} be an estimate of the time that the task has to execute in the presence of faults and with checkpointing. The expected number of faults for the duration t_{est} is λt_{est}. We are assuming here that the checkpointing cost is negligible compared to the time for forward execution and rollback recovery, hence even though no faults occur during checkpointing, the expected number of faults is λt_{est}.To ensure λt_{est}-fault-tolerance during task execution, the checkpointing interval must be set to $\sqrt{t_{est}C/(\lambda t_{est})} = \sqrt{C/\lambda} = \sqrt{c/(\lambda f)}$. Now, the parameter t_{est} can be expressed as follows:

$$t_{est} = \frac{R_c}{f} + \lambda t_{est}\sqrt{\frac{c}{\lambda f}} + \frac{c}{f}\frac{R_c/f}{\sqrt{c/(\lambda f)}} \qquad (3)$$

The first term on the right-hand side of (3) denotes the time for forward execution, the second term denotes the recovery cost for λt_{est} faults, and the third term denotes the checkpointing cost. From (3), we get

$$t_{est} = \frac{R_c(1+\sqrt{\lambda c/f})}{f(1-\sqrt{\lambda c/f})}.$$

We consider the voltage scaling (to frequency f) to be feasible if $t_{est} \le R_d$. This forms the basis of the energy-aware adaptive checkpointing procedure *adap_dvs* described in Figure 4. At every DVS decision point, an attempt is made to run the task at the lowest-possible speed.

```
Procedure adap_dvs($D,N,c,k,\lambda$)
1. $R_c = N$; $R_d = D$; $R_f = k$;
2 .if ($t_{est}(R_c, f_1) \le R_d$ ) $f = f_1$; else $f = f_2$;
3. $Itv = interval(R_d,R_c/f, c/f,R_f,\lambda)$;
4. while ($R_t>0$) do{
5.   if ($R_t > R_d / f$) break;
6.   Case 1: During normal execution, do{
       6.1 Insert checkpoints with interval length $Itv$;
       6.2 Update $R_c$, $R_d$ according to speed $f$;}
7.   Case 2: Upon fault occurrence, do{
       7.1 Roll back and restore status;
       7.2 $R_f = R_f - 1$;
       7.3 if ($t_{est}(R_c) \le R_d$) $f = f_1$; else $f = f_2$;
       7.4 $Itv$ = interva($R_d,R_c/f,c/f,R_f,\lambda$);
       7.5 Resume execution; }}
```
Figure 4: Energy-aware adaptive checkpointing procedure.

4.1 Simulation results on ADT_DVS

We compare the adaptive DVS scheme, denoted by ADT_DVS, with the Poisson-arrival and k-fault-tolerant schemes in terms of the probability of timely completion and energy consumption. We use the same experimental set-up as in Section 3.2. In addition, we consider the normalized frequency values $f_1 = 1$ and $f_2 = 2$. First we assume that both the Poisson-arrival and the k-fault-tolerant schemes use the lower speed f_1. The task execution time at speed f_1 is chosen to be less than D, i.e., $N/f_1 < D$. The task utilization U in this case is simply $N/(f_1D)$. Our experimental results are shown in Table 4. The ADT_DVS scheme always leads to timely completion of the task by appropriately choosing segments of time when the higher frequency f_2 is used. The other two schemes provide a rather low value for P, and for larger values of λ and U, P drops to zero. The energy consumption for the ADT_DVS scheme is slightly higher than that for the other two schemes; however, on average, the task runs at the lower speed f_1 for as much as 90% of the time. The combination of adaptive checkpointing and DVS utilizes the slack effectively and stretches the task completion time to as close to the deadline as possible.

Next we assume that both the Poisson-arrival and the k-fault-tolerant schemes use the higher speed f_2. The task execution time at speed f_2 is chosen to be less than D, i.e., $N/f_2 < D$, and the task utilization here is $N/(f_2D)$. Table 5 shows that since even though ADT_DVS uses both f_1 and f_2, adaptive checkpointing allows it to provide a higher value for P than the other two methods that use only the higher speed f_2. The energy consumption for ADT_DVS is up to 50% less than for the other two methods for low to moderate values of λ and U; see Table 6. When either λ or U is high, the energy consumption of ADT_DVS is comparable to that of the other two schemes. (Energy is measured by summing the product of the square of the voltage and the number of computation cycles over all the segments of the task.) This is expected, since ADT_DVS attempts to meet the task deadline as the first priority and if either λ or U is high, ADT_DVS seldom scales down the processor speed.

U	Fault arrival rate λ ($\times 10^{-4}$)	Probability of timely completion of tasks, P		
		Poisson-arrival	k-fault-tolerant	ADT_DVS
0.95	0.5	0.790	0.704	1.000
	1.0	0.648	0.508	1.000
	1.5	0.501	0.367	1.000
	2.0	0.385	0.244	1.000

(a)

λ ($\times 10^{-4}$)	U	Probability of timely completion of tasks, P		
		Poisson-arrival	k-fault-tolerant	ADT_DVS
1.0	0.92	0.924	0.960	1.000
	0.96	0.549	0.000	1.000
	1.00	0.000	0.000	1.000
2.0	0.92	0.799	0.849	1.000
	0.96	0.229	0.000	1.000
	1.00	0.000	0.000	1.000

(b)

Table 4: (a) Variation of P with λ (b) Variation of P with U, for $D = 10000$, $c = 10$, and $k = 2$.

U	Fault arrival rate ($\times 10^{-4}$)	Probability of timely completion of tasks, P		
		Poisson-arrival	k-fault-tolerant	ADT_DVS
0.95	0.8	0.898	0.939	0.965
	1.2	0.841	0.868	0.912
	1.6	0.754	0.785	0.871
	2.0	0.706	0.695	0.791

Table 5: Variation of P with λ, for $D = 10000$, $c = 10$, and $k = 1$.

U	Fault arrival rate λ ($\times 10^{-4}$)	Energy consumption		
		Poisson-arrival	k-fault-tolerant	ADT_DVS
0.60	2.0	25067	26327	21568
	4.0	25574	26477	21642
	6.0	25915	26635	21714
	8.0	26277	26806	22611

(a)

λ ($\times 10^{-4}$)	U	Energy consumption		
		Poisson-arrival	k-fault-tolerant	ADT_DVS
5.0	0.10	4295	4909	2508
	0.20	8567	9335	4791
	0.30	12862	13862	7026
	0.40	17138	17990	9223
	0.50	21474	22300	15333

(b)

Table 6: (a) Variation of energy consumption with λ (b) Variation of energy consumption with U, for $D = 10000$, $c = 10$, and $k = 10$.

5. Conclusions

We have presented a unified approach for adaptive checkpointing and dynamic voltage scaling for a real-time task executing in an embedded system. This approach provides fault tolerance and facilitates dynamic power management. The proposed energy-aware adaptive checkpointing scheme uses a dynamic voltage scaling criterion that is based not only on the slack in task execution but also on the occurrences of faults during task execution. We have presented simulation results to show that the proposed approach significantly reduces power consumption and increases the probability of tasks completing correctly on time despite the occurrences of faults.

We are currently extending the proposed approach to a set of multiple periodic tasks. We are also examining ways to relax the restrictions of zero rollback and state restoration costs, as well as the assumption of no fault occurrence during checkpointing and rollback recovery.

References

[1] P. Pop, P. Eles and Z. Peng, “Schedulability analysis for systems with data and control dependencies”, *Proc. Euromicro RTS*, pp. 201-208, June 2000.

[2] T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors”, *Proc. Int. Symp. Low Power Electronics and Desig*n, August 1998.

[3] Y. Shin, K. Choi, and T. Sakurai, “Power optimization of real-time embedded systems on variable speed processors”, *Proc. Int. Conf. Computer-Aided Design*, pp. 365-368, June 2000.

[4] G. Quan and X. Hu, “Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors”, *Proc. Design Automation Conference*, pp. 828-833, June 2001.

[5] K. G. Shin and Y.-H. Lee, “Error detection process—Model, design and its impact on computer performance”, *IEEE Trans. on Computers*, vol. C-33, ppp. 529-540, June 1984.

[6] K. M. Chandy, J. C. Browne, C. W. Dissly, and W. R. Uhrig, “Analytic Models for Rollback and Recovery Strategies in Data Base Systems”, *IEEE Trans. Software Eng*., vol. 1, pp. 100-110, March 1975.

[7] K. Shin, T. Lin and Y. Lee, “Optimal Checkpointing of Real-Time Tasks”, *IEEE Trans. Computer*s, vol. 36, no. 11, pp. 1328-1341, November 1987.

[8] A. Ziv and J. Bruck, “An on-line algorithm for checkpoint placement”, *IEEE Trans. Computer*s, vol. 46, no. 9, pp. 976-985, September 1997.

[9] S. W. Kwak, B. J. Choi and B. K. Kim, “An optimal checkpointing-strategy for real-time control systems under transient faults”, *IEEE Trans. Reliability*, vol. 50, no. 3, pp. 293-301, September 2001.

[10] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing*, Kluwer Academic Publishers, Norwell, MA, 2000.

[11] J. W. Liu, *Real-Time Systems*, Prentice Hall, Upper Saddle River, NJ, 2000.

[12] A. Duda, “The effects of checkpointing on program execution time”, *Information Processing Letter*s, vol. 16, pp. 221-229, June 1983.

[13] H. Lee, H. Shin and S. Min, “Worst case timing requirement of real-time tasks with time redundancy”, *Proc. Real-Time Computing Systems and Applications*, pp. 410-414, 1999.