Performance Optimization for Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems*

Zhongwen Li,
Information Science and Technology College,
Xiamen University, China
lizw@xmu.edu.cn

Hong Chen,
Information Science and Technology College,
Xiamen University, China
chenhon103@hotmail.com

Shui Yu
School of Information Technology,
Deakin University, Australia
syu@deakin.edu.au

Abstract

Using additional store-checkpoinsts (SCPs) and compare-checkpoints (CCPs), we present an adaptive checkpointing for double modular redundancy (DMR) in this paper. The proposed approach can dynamically adjust the checkpoint intervals. We also design methods to calculate the optimal numbers of checkpoints, which can minimize the average execution time of tasks. Further, the adaptive checkpointing is combined with the DVS (dynamic voltage scaling) scheme to achieve energy reduction. Simulation results show that, compared with the previous methods, the proposed approach significantly increases the likelihood of timely task completion and reduces energy consumption in the presence of faults.

1. Introduction

Checkpointing is an important method for fault-tolerance in real-time systems in the condition of harsh environment. The following three types of checkpoints are well known: CSCP, SCP and CCP [1-3]. CCPs are used to compare the states of the processors without storing them, while, the processors store their states without comparison in SCPs. If the two operations are used together in the same checkpoint, we call it CSCP. Using CCP and SCP, Ziv and Bruck have shown numerically that the task execution time is significantly reduced [1,4]. Using additional CCPs and SCPs, Nakagawa and Fukumoto have used a triple modular redundancy and double modular redundancy to analyze the optimal checkpoint intervals that can minimize a task execution time, respectively [5].

In addition, many real-time systems are often energy-constrained since system lifetime is determined to a large extent by the battery lifetime [2]. For example, autonomous airborne and sea-borne systems working on limited battery supply, space systems working on a limited combination of solar and battery power supply, time-sensitive systems deployed in remote locations where a steady power supply is not available [3,6]. DVS has emerged as a popular solution to the problem of reducing power consumption during system operations. The DVS become possible on the availability of embedded processors that can dynamically scale the frequency by adjusting the operation voltage [2,3]. Many embedded processors have the ability to dynamically scale the operation voltage currently. Such as, the mobile processors from Intel with its SpeedStep [7] technology. In the realm of real-time systems, the DVS techniques focuse on minimizing energy consumption of the system under the condition of meeting the deadlines. The DVS and fault tolerance for real-time systems have been studied as separate problems. It is only recently that an attempt has been made to combine fault tolerance with the DVS [3].

The combination of DVS, CSCPs (CCPs or SCPs) can be used to satisfy system's DVS requirement and improve the performance of real-time systems. However, none of the mentioned papers addressed these issues in terms of conjunction. Using additional SCPs and CCPs, we modify the methods of [3] in the double modular redundancy (DMR) in this paper. Different from the existing methods, our approach is to tune the scheme to the specific system which it is implemented on, and use both the comparison and storage operations efficiently, the performance of checkpoint schemes is improved.

Some notations used in our paper are listed below:

t_s: the time to store the states of processors.

t_{cp}: the time to compare processors' states.

t_r: the time to roll back the processors to a consistent state.

R_t : remaining execution time.

R_d : time left before the deadline.

* This work is supported in part of Fujian natural science grant (A0410004), Fujian young science & technology innovation grant (2003J020), NCETXMU 2004 program, Program of 985 Innovation on Information in Xiamen Univ.(2004-2007) and Xiamen Univ. research grant (0630-E23011).

R_f: upper boundary on the remaining number of faults that can be tolerated by the system.

2 Adaptive checkpointing scheme

Assume task τ has a period T, a deadline D, a worst-case computation time N when there are no fault in the system. An upper boundary k represents the number of fault occurrences that have to be tolerated. C is the overhead of a checkpoint. Faults arrive as a Poisson process with parameter λ, the average execution time for the task is minimum, if a constant checkpoint interval of $\sqrt{2C/\lambda}$ is used [8]. We refer to this as the Poisson-arrival approach. If the Poisson-arrival scheme is used, the effective task execution time in the absence of faults must be less than the deadline D. Assume the fault-free execution time for a task is N, the worst-case execution time for up to k faults is minimum, if the constant checkpoint interval is set to $\sqrt{NC/k}$ [9]. This is the k-fault-tolerant approach.

In addition, we assume that task τ is divided equally into n intervals of length $T=\left\lfloor \frac{N}{n} \right\rfloor$, and at the end of each interval, CSCP is always placed.

2.1 Additional SCPs

Each CSCP interval is divided equally into m intervals of length $T_1=\left\lfloor \frac{T}{m} \right\rfloor$(figure 1). The SCPs are placed

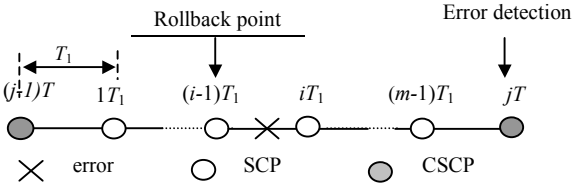

Fig. 1 Task execution with SCPs

between the CSCPs, the states of two processors are stored at iT_1 and jT (i=1,2,…, m-1). If two states do not get an agreement at time jT, then, we need to find the most recent SCP with identical states and roll back to it. As shown in figure 1, two processors are rolled back to $(i-1)T_1$ because some errors have occurred during $((i-1)T_1, iT_1)$, and repeat the execution from $(i-1)T_1$. The average execution time $R_1(m)$ for a CSCP interval $((j-1)T, jT)$ is given by a renewal-equation[4, 10]:

$$R_1(m) = (mT_1 + mt_s + t_{cp})e^{-2\lambda mT_1}$$
$$+ \sum_{i=1}^{m} \int_{(i-1)T_1}^{iT_1} [mT_1 + mt_s + t_{cp} + t_r + R_1(m-(i-1))]d(1-e^{-2\lambda t})$$
$$= mT_1 + mt_s + t_{cp} + [\frac{1}{2}m(m+1)(T_1+t_s) + m(t_{cp}+t_r)](e^{2\lambda T_1} - 1)$$

Therefore, the average execution time of a task $R_{SCP}(n)=nR_1(m)$.

Replace $m = T / T_1$, we have

$$R_1(T_1) = T + \frac{T}{T_1}t_s + t_{cp} + [(T + \frac{T}{T_1}t_s)\frac{(T+T_1)}{2T_1} + \frac{T}{T_1}t_{cp}](e^{2\lambda T_1} - 1).....(1)$$

If $T_1 \to 0^+$, then $R_1(T_1)= +\infty$. Let $T_1=T$, we have $R_1(T_1) = (T + t_s + t_{cp})e^{2\lambda T}$. Thus, there exists a finite $\tilde{T}_1 \in ((j-1)T, jT]$ which minimizes $R_1(T_1)$. Differentiating equation (1) with respect to T_1 and setting it equal to zero, we get $\tilde{T}_1$. Procedure num_SCP(T) for calculating $\tilde{m}$ which minimize $R_1(\tilde{m})$ is described in Figure 2.

The adaptive checkpointing with SCPs, adapchp-SCP (D,E,C,k, λ), is described in Figure 3. A check is per-

```
Procedure num_SCP(T){
1.  Find  T̃1  which minimizes R1(m);
2.  if ( T̃1 <T) {
3.     m=⌊T / T̃1⌋;
4.     if (R1(m) ≤ R1(m+1)) then
5.           m̃ = m;
6.           else  m̃ = m+1;
7.    } else  m̃ = 1;
8.   return  m̃; }
```

Fig. 2 Procedure for calculating the $\tilde{m}$

```
Procedure adapchp-SCP (D, N, C, k, λ ){
1.  Rt=N; Rd=D; Rf=k;
2.  Itv=interval(Rd, Rt, C, Rf, λ );
3.  m=num_SCP(Itv);  itv=⌈Itv/m⌉;
4.  while (Rt>0) do {
5.     if (Rt> Rd) break with task failure;
6.     Insert SCP with interval length itv;
7.     Insert CSCP with interval length Itv;
8.     Update Rt, Rd;
9.     if (no error has been detected at CSCP)
10.       Resume execution;
11.  else{
12.       Rollback to the most recent SCP with identical
      states;
13.       Rf= Rf-1;
14.       Itv=interval(Rd, Rt, C, Rf, λ );
15.       m=num_SCP(Itv);  itv=⌈Itv/m⌉;
16.       Resume execution;}}
```

Fig. 3 Adaptive checkpointing with SCPs

formed to see if the task has been completed in line 4, and line 5 checks for the deadline constraint. The length of SCP and CSCP interval is set in line 6 and line 7, respectively. In line 9, a check is performed to see if fault is detected. If there is no fault, then continue to run task, otherwise, roll back to previous SCP with identical states and continues execution, which are described from line 12 to

line 16. In line 2 and 14, we use procedure interval (R_d, R_t, C, R_f, λ) [3] (figure 4) to calculate the checkpoint interval.

In figure 4, $I_1(C,\lambda)=\sqrt{2C/\lambda}$ is the checkpoint interval of the Poisson-arrival approach,

```
Procedure interval(R_d, R_t, C, R_f, λ ){
1   .exp_error= λ R_t;
2.    if (exp_error ≤ R_f) {
3.       if (R_t>Th_λ(R_d, λ ,C)) then
4.            chk_interval=I_3(R_t, R_d, C);
5.       else if (R_t>Th(R_d, R_f, C)) then
6.               chk_interval= I_2(R_t, exp_error,C);
7.            else chk_interval= I_2(R_t, R_f ,C);}
8.       else{ if(R_t> Th_λ(R_d, λ ,C)) then
9.            chk_interval= I_3(R_t, R_d, C);
10.           else chk_interval=I_1(C, λ );}
11.   return chk_interval; }
```

Fig. 4 Calculating checkpointing interval

$I_2(N,k,C)=\sqrt{NC/k}$ is the checkpoint interval of the k-fault-tolerant approach. In addition, [3] defined some quations:

$$Th_\lambda(R_d,\lambda,C)=(R_d+C)/(1+\sqrt{\lambda C/2})$$

$$Th(R_d,R_f,C)=R_d+C+2R_fC-2\sqrt{R_fC(R_d+C)+(R_fC)^2}$$

$$I_3(N,D,C)=2NC/(D+C-N)$$

Line 1 of figure 4 calculates the number of faults *Exp-fault* that are expected to occur in the remaining time R_t. 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. 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 checkpoint interval is set to $I_3(R_t, R_d,C)$. In line5, 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 $I_2(R_t, Exp\text{-}fault,C)$. If the k-fault-tolerant threshold is met, the checkpoint interval is set to $I_2(R_t, R_f,C)$ in line 7. Line 8-10 handle the case when the k-fault-tolerant requirement is deemed to be less stringent than the Poisson-arrival criterion.

2.2 Additional CCPs

Each CSCP interval is divided equally into m intervals of length $T_2=\left\lceil\frac{T}{m}\right\rceil$. The CCPs are placed between CSCPs, and the states of the two processors are compared at iT_2 and jT (i=1,2,…, m-1). If two states do not reach to an agreement at iT_2 and jT, that means some errors have occurred during this interval, the two processors will be rolled back to $(j\text{-}1)T$ (Figure 5).

The average execution time $R_2(m)$ for an interval

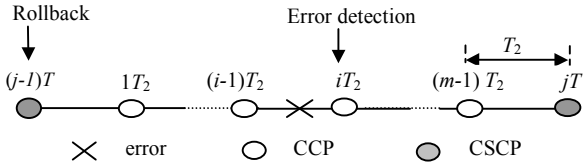

Fig. 5 Task execution with ICCPs

$((j\text{-}1)T, jT)$ is given by a renewal-equation:

$$\begin{aligned}R_2(m)&=(mT_2+mt_{cp}+t_s)e^{-2\lambda mT_2}\\&+\sum_{i=1}^{m}\int_{(i-1)T_2}^{iT_2}[iT_2+it_{cp}+t_r+R_2(m)]d(1-e^{-2\lambda t})\\&+\int_{(m-1)T_2}^{mT_2}t_s d(1-e^{-2\lambda t})\\&=t_s e^{2\lambda T_2}+(e^{2\lambda mT_2}-1)\frac{T_2+t_{cp}}{1-e^{2\lambda T_2}}\end{aligned}$$

Therefore, the average execution time $R_{CCP}(n)=nR_2(m)$.

Replacing $m=T/T_2$, we have:

$$R_2(T_2)=t_s e^{2\lambda_0 T_2}+(e^{2\lambda_0 T}-1)\frac{T_2+t_{cp}}{1-e^{-2\lambda_0 T_2}}\quad\ldots\ldots(2)$$

If $T_2\to 0^+$, then $R_2(T_2)=+\infty$. If $T_2=T$, then $R_2(T_2)=(T+t_s+t_{cp})e^{2\lambda T}$. Therefore, there exists a finite $\tilde{T}_2\in((j-1)T, jT]$, which minimizes $R_2(T_2)$. Differentiating equation (2) with respect to T_2 and setting it to zero, we can get $\tilde{T}_2$. We can use the similar approach described in figure 2 to calculate $\tilde{m}$ which minimize $R_2(\tilde{m})$.

3 Adaptive checkpointing with DVS

With additional SCPs and CCPs, we show how adaptive checkpointing scheme can be combined with the DVS to obtain fault tolerance and power savings in real-time systems. In the one hand, our approach is to maximize the probability that the task meets its deadline in the presence of faults. In another hand, our approach is to reduce energy consumption through the DVS.

Assume that task τ has a fixed quantity of computation cycles N in the fault-free condition. Because the variable voltage CPUs are available, the time to execute task τ depends on the processor speed. We therefore characterize τ by a fixed quantity N, namely, its worst-case number of CPU cycles, needed to execute the task at the minimum processor speed. For the rest of this paper, we normalize the units of N such that the minimum processor speed is 1. That is, if the minimum processor speed is S cycles per second, then we express the number of cycles in units of S cycles and thus normalize the minimum processor speed to S_{min}=1. Of course, period T and deadline D are expressed in terms of the number of CPU cycles at the

minimum processor speed.

To simplify the analysis and to allow for the derivation of analytical formulas, we would like to assume that a single processor with two speeds f_1 and f_2, and f_1 is the minimum processor speed, namely, $f_1 = S_{min}=1$. Moreover, the processor can switch its speed in a negligible amount of time.

Additional notations we use is below:

R_c : the number of instructions of the task that remain to be executed at the time of the voltage scaling decision.

c : the numbe of clock cycles that a single checkpoint takes.

t_{est}: 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}.

The checkpointing cost C at frequency f is given by $C=c/f$.

To ensure λt_{est} fault tolerance during task execution, the checkpointing interval must be set to $\sqrt{t_{est}C/(\lambda t_{est})} = \sqrt{C/\lambda} = \sqrt{c/(\lambda f)}$. In addition, we have $t_{est} = \dfrac{R_c(1+\sqrt{\lambda c / f})}{f(1-\sqrt{\lambda c / f})}$ [3].

We consider the voltage scaling to be feasible if $t_{est} \le R_d$. This forms the basis of the energy-aware adaptive checkpointing that are described in procedure adapchp_dvs_SCPs and adapchp_dvs_CCPs (Figure 6 and Figure 7).

```
Procedure adapchp_dvs_SCP(D, N, c, k, λ){
1.    R_c = N; R_d = D; R_f = k;
2.    if( t_est(R_c, f_1) ≤ R_d )f = f_1; else f = f_2;
3.    Itv = interval(R_d, R_c / f, c / f, R_f, λ);
4.    m = num_SCP(Itv); itv = Itv / m;
5.    while( (R_t = R_c / f) > 0 )do{
6.       if(R_t > R_d) break with task failure;
7.       Insert SCP with interval length itv;
8.       Insert CSCP with interval length Itv;
9.       Update R_c, R_d according to speed f;
10.       if(no error has been detected at CSCP)
11.          Resume execution;
12.       else{
13.          Roll back to the most recent SCP/CSCP with identical states;
14.          R_f = R_f -1;
15.          if( t_est(R_c, f_1) ≤ R_d )f = f_1; else f = f_2;
16.          Itv = interval(R_d, R_c / f, c / f, R_f, λ);
17.          m = num_SCP(Itv); itv = Itv / m;
18.          Resume execution;}
}}
```

Fig. 6 adapchp_dvs_SCPs

```
Procedure adapchp_dvs_CCP(D, E, C, k, λ){
1.    R_t = E; R_d = D; R_f = k;
2.    if( t_est(R_c, f_1) ≤ R_d )f = f_1; else f = f_2;
3.    Itv = interval(R_d, R_c / f, c / f, R_f, λ);
4.    m = num_CCP(Itv); itv = Itv / m;
5.    while( (R_t = R_c / f) > 0 )do{
6.       if(R_t > R_d) break with task failure;
7.       Insert CCP with interval length itv;
8.       Insert CSCP with interval length Itv;
9.       Update R_c, R_d according to speed f;
10.       if(no error has been detected at CCP/CSCP)
11.          Resume execution;
12.       else{
13.          Roll back to the last CSCP;
14.          R_f = R_f -1;
15.          if( t_est(R_c, f_1) ≤ R_d )f = f_1; else f = f_2;
16.          Itv = interval(R_d, R_c / f, c / f, R_f, λ);
17.          m = num_CCP(Itv); itv = Itv / m;
18.          Resume execution;}
}}
```

Fig. 7 adapchp_dvs_CCPs

4 Simulation results

We carried out a set of simulation experiments to evaluate our adaptive checkpointing schemes adapchp_dvs_CCPs and adapchp_dvs_SCPs (referred to as A_D_C and A_D_S) and to compare it with the Poisson-arrival (referred to as Poisson), the k-fault-tolerant (referred to as k-f-t) checkpointing schemes and ADT_DVS[3] (referred to as A_D). Faults are injected into 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 that the task completes on time, and the energy consumption. Energy consumption 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 [3]. As in [3], we use the term task utilization U to refer to the ratio N/D. In order to compare with results of ADT_DVS scheme, we let $t_r=0$ and $f_2=2f_1$. Moreover, let P and E represent the probability of timely completion of tasks and energy consumption, respectively.

4.1 Additional SCPs

As mentioned previously, additional SCPs scheme fits systems, in which time overhead is determined mainly by the time to compare processor's states. Therefore, the parameters are as following: D=10000, t_s=2, t_{cp}=20, c=22.

First, we let the Poisson-arrival and the k-fault-tolerant schemes use the lower speed f_1. The task

utilization U in this case is $N/(f_1D)$. Our experimental results are shown in table 1. In table 1(a), for $\lambda > 0.001$ and $0.7<U<0.9$(high fault arrival rate and relatively high task utilization), the experimental results show that adapchp-dvs-SCPs scheme clearly outperforms the ADT_DVS scheme. Although Poisson-arrival and the k-fault-tolerant schemes have lower energy consumption, their probability of timely completion of the task are lower that 0.2. In table 1(b), for $\lambda < 0.001$ and $0.9 < U \le 1$ (low fault arrival rate and high task utilization), we draw the similar conclusions described above.

We assume that both Poisson-arrival and the k-fault-tolerant schemes use the higher speed f_2. Then the task utilization U in this case is $N/(f_2D)$. Our experimental results are shown in table 2. We also can draw a conclusion that our scheme outperforms the other three schemes.

Tab. 1 The comparison between adapchp-dvs-SCPs and other algorithms, both the Poisson-arrival and the k-fault-tolerant schemes use the lower speed f_1.

U	λ ($\times10^{-2}$)	P / E			
		Poisson	k-f-t	A_D	A_D_S
0.76	0.14	0.1185 39015	0.1115 38940	0.9991 57564	0.9999 52863
	0.16	0.0489 39183	0.0466 39153	0.9992 59765	0.9999 54176
0.78	0.14	0.0504 39358	0.0496 39350	0.9990 60441	0.9999 55520
	0.16	0.0181 39443	0.0182 39396	0.9993 62687	0.9999 56814
0.80	0.14	0.0091 38951	0.0204 39507	0.9993 63039	0.9999 58079
	0.16	0.0021 39217	0.0062 39684	0.9990 65233	0.9998 59344
0.82	0.14	0.0013 39161	0.0018 39122	0.9995 65778	1.0000 60731
	0.16	0.0005 39290	0.0003 39200	0.9990 67987	0.9999 62091

(a) $k = 5$

U	λ ($\times10^{-4}$)	P / E			
		Poisson	k-f-t	A_D	A_D_S
0.92	1.0	0.3914 38032	0.3965 38665	0.9229 74193	0.9549 72862
	2.0	0.1650 38623	0.1628 38681	0.9793 76444	0.9985 72566
0.95	1.0	0.3851 39316	0.3852 39844	0.9188 77097	0.9516 75743
	2.0	0.1520 39844	0.1510 39844	0.9462 80414	0.9944 76841
1.00	1.0	0.0000 NaN	0.0000 NaN	0.9146 81572	0.9557 81047
	2.0	0.0000 NaN	0.0000 NaN	0.9204 84371	0.9892 82499

(b) $k = 1$

Tab. 2 The comparison between adapchp-dvs-SCPs and other algorithms, both the Poisson-arrival and the k-fault-tolerant schemes use the higher speed f_2.

U	λ ($\times10^{-2}$)	P / E			
		Poisson	k-f-t	A_D	A_D_S
0.76	0.14	0.6159 149458	0.6121 149682	0.6486 149599	0.9462 146097
	0.16	0.5369 151339	0.4258 150911	0.5451 151264	0.9006 147873
0.78	0.14	0.4659 151964	0.3593 150851	0.4699 151935	0.8385 149415
	0.16	0.3007 152371	0.2055 151581	0.3227 152552	0.7389 150742
0.80	0.14	0.2355 152698	0.2305 152918	0.2672 153124	0.6491 151905
	0.16	0.1264 153007	0.1207 153495	0.1617 153695	0.4864 152742
0.82	0.14	0.0921 153077	0.0838 153103	0.0992 153320	0.3843 153562
	0.16	0.0285 153494	0.0271 153619	0.0388 154288	0.2242 154279

(a) $k = 5$

U	λ ($\times10^{-4}$)	P / E			
		Poisson	k-f-t	A_D	A_D_S
0.92	1.0	0.7609 151255	0.7638 151722	0.7640 150583	0.7776 150583
	2.0	0.4365 152453	0.4384 152554	0.4737 152444	0.5334 152452
0.95	1.0	0.3847 152589	0.3924 154140	0.3799 149117	0.3941 150259
	2.0	0.1498 153946	0.1498 154167	0.2816 155147	0.2842 155612

(b) $k = 1$

4.2 Additional CCPs

Additional CCPs scheme fits systems which overhead time is determined mainly by the time to store processor' states. Therefore, the parameters is as following: D=10000, t_s=20, t_{cp}=2, c=22.

Tab. 3 The comparison between adapchp-dvs-CCPs and other algorithms, both the Poisson-arrival and the k-fault-tolerant schemes use the lower speed f_1.

U	λ ($\times10^{-2}$)	P / E			
		Poisson	k-f-t	A_D	A_D_S
0.76	0.14	0.1104 38942	0.1070 38953	0.9990 57662	1.0000 52862
	0.16	0.0505 39141	0.0479 39128	0.9989 59736	0.9999 54036
0.78	0.14	0.0530 39374	0.0534 39345	0.9989 60435	1.0000 55520
	0.16	0.0190 39422	0.0210 39362	0.9989 62477	0.9998 56719

0.80	0.14	0.0085 39030	0.0209 39500	0.9989 63040	1.0000 58042
	0.16	0.0022 39103	0.0057 39530	0.9992 65230	1.0000 59274
0.82	0.14	0.0021 39266	0.0020 39031	0.9990 65731	1.0000 60573
	0.16	0.0005 39658	0.0005 39350	0.9989 68038	1.0000 61935

(a) $k = 5$

U	λ $(\times 10^{-4})$	PE			
		Poisson	k-f-t	A_D	A_D_S
0.92	1.0	0.3887 38032	0.3984 38667	0.9241 74350	0.9800 73547
	2.0	0.1634 38619	0.1635 38685	0.9783 77021	0.9994 72669
0.95	1.0	0.3775 39316	0.3772 39844	0.9116 77266	0.9812 76756
	2.0	0.1498 39844	0.1480 39844	0.9519 80540	0.9978 76614
1.00	1.0	0.0000 NaN	0.0000 NaN	0.9074 81397	0.9831 81675
	2.0	0.0000 NaN	0.0000 NaN	0.9202 84379	0.9959 82254

(b) $k = 1$

Tab. 4 The comparison between adapchp-dvs-CCPs and other algorithms, both the Poisson-arrival and the k-fault-tolerant schemes use the higher speed f_2.

U	λ $(\times 10^{-2})$	PE			
		Poisson	k-f-t	A_D	A_D_S
0.76	0.14	0.6130 149575	0.6063 149738	0.6456 149694	0.9544 146237
	0.16	0.5252 151286	0.4147 150869	0.5336 151206	0.9104 148058
0.78	0.14	0.4731 151926	0.3641 150860	0.4804 151917	0.8519 149493
	0.16	0.3016 152389	0.2061 151610	0.3277 152618	0.7546 150926
0.80	0.14	0.2356 152662	0.2283 152988	0.2664 153111	0.6540 152034
	0.16	0.1279 153171	0.1195 153558	0.1629 153834	0.4942 152927
0.82	0.14	0.0873 153081	0.0849 153118	0.0950 153365	0.3758 153731
	0.16	0.0321 153207	0.0319 153394	0.0418 153946	0.2115 154400

(a) $k = 5$

U	λ $(\times 10^{-4})$	PE			
		Poisson	k-f-t	A_D	A_D_S
0.92	1.0	0.7559 151220	0.7570 151703	0.7583 150564	0.7657 150564
	2.0	0.4409 152537	0.4398 152623	0.4715 152479	0.5327 152546
0.95	1.0	0.3946 152591	0.3984 154155	0.3878 149117	0.3995 150239
	2.0	0.1479 153946	0.1488 154171	0.2775 155132	0.2850 155597

(b) $k = 1$

Our experimental results are shown in table 3 and table 4. Similar to section 4.1, simulation results show that compared to ADT_DVS scheme, the proposed scheme significantly increases the likelihood of timely task completion and reduces power consumption in the present of faults.

5. Conclusion

In this paper, we presented an adaptive checkpointing, using a DMR with two processors, and tuning the scheme to the specific system which it is implemented on. The proposed scheme is done by inserting two types of checkpoints (CCP and SCP) between CSCP. Separating the comparison and store operations enables choosing the optimal interval for each operation, without concerning about the other. We also discussed the optimal numbers of checkpoints that minimize the average times. Based on that, we combined the adaptive checkpoiting with the DVS schemes to achieve energy reduction. We presented simulation results which showed the advantages of our scheme. We will extend the proposed scheme to other task duplication systems with security needs as a future work.

References

[1] Ziv A. Analysis of checkpointing schemes with task duplication, IEEE Trans. Computers, 1998,47(2):222-227

[2] Ying Z, Crishnendu C. Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems, Proc. Of the design, automation and test in Europe conference and exhibition (DATE'04)

[3] Ying Z, Crishnendu C. Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems, Proc. of the design, automation and test in Europe conference and exhibition (DATE'03), 2003

[4] Ziv A, Bruck J. Performance Optimization of Checkpointing Schemes with Task Duplication IEEE Transactions on Computers, 1997, 46(2):1381-1386

[5] Sayori N, Satoshi F, Naohiro I. Optimal Checkpointing Intervals of Three Error Detection Schemes by a Double Modular Redundancy, Mathematical and Computer Modelling, 2003,38:1357-1363

[6] Melhem R, Mosse D, Elnozahy E. The interplay of power management and fault recovery in real-time systems, IEEE Tran. On computers, 2004, 53(2):217-231

[7] Intel Corp, speedstep, http://developer.inte.com/mobile/pentiumIII, 2003

[8] Duda A. The effects of checkpointing on program execution time, Information Processing Letters, 1983 (16):221-229

[9] Lee H, Shin H, Min S. Worst case timing requirement of real-time tasks with time redundancy, Processing Real-Time computing systems and Applications, 1999: 410-414

[10] Osaki S. Applied stochastic system modeling, Springer-Verlag, 1992