Proceedings of the 9th WSEAS International Conference on Applied Mathematics, Istanbul, Turkey, May 27-29, 2006 (pp650-655)

# Adaptive Checkpointing Schemes for Fault Tolerance in Real-Time Systems with Task Duplication

Zhongwen Li[1,2], Hong Chen[1]

[1]Information Science and Technology College,
Xiamen University,
Xiamen 361005, China

[2]Zhongshan Institute of UESTC,
Zhongshan, 528402, China

*Abstract:* Dynamic adaptation techniques based on checkpointing is studied in this paper. Placing store-checkpoints and compare-checkpoints between CSCP (store-and-compare-checkpoint), we first present adaptive checkpointing schemes in which the checkpointing interval for a task is dynamically adjusted on line. Introducing the overheads of comparison and storage, the average execution times to complete a task for proposed schemes are obtained, using renewal equations. Further, we have discussed analytically the optimal numbers of checkpoints that minimize the average execution times. We then extend proposed schemes to a set of multiple tasks in real-time systems. Simulation results show that compared to previous method, the proposed approach significantly increases the likelihood of timely task completion.
*Key words:* Fault-tolerant computing, Checkpointing interval, SCP, CCP, Multiple tasks, Task Duplication

## 1 Introduction

Fault tolerance is typically achieved in real-time systems through checkpointing and task duplication. Checkpointing enables reducing the time to recover from a fault by saving intermediate states of the task in a secure storage. Generally speaking, each checkpoint serves two purposes. The first is to save the processor state and to reduce the fault-recovery time by supplying an intermediate correct state, thus avoiding rollback to the beginning of the task. The second purpose is fault-detection, which is achieved by executing and comparing the processors' states at each checkpoint in task duplication systems. In task duplication systems, such as DMR (Double Modular Redundancy), TMR-F, DMR-F-1 and RFCS[1], the task is executed on more than one processor and the states of the processors are compared to detect faults. Several papers describe schemes that combine checkpointing and task duplication[2-6]. When there is a big difference between the time to store the processors' states and the time to compare these states, the overhead time is determined mainly by the operation that takes a longer time. In order to avoided unnecessary overhead, some people have researched SCP and CCP checkpoint[2,4,6].

In this paper, we present adaptive schemes to dynamically adjust checkpointing interval to reduce task execution time for single and multiple real-time tasks. These schemes are also designed to tolerate up to $k$ fault occurrences. The proposed adaptive checkpointing is consisted of two schemes in which we place SCP or CCP between consecutive CSCPs, respectively. For the sake of simplicity, we use a double modular redundancy (DMR) in which a task is executed on two processors.

we assume that the time for a failure to occur

Foundation item: Fujian natural science foundation (A0410004), NCETXMU program (0000-X07116) and Xiamen University research foundation (0630-E23011).

is exponentially distributed with constant failure rate $\lambda$. Some notation used in our paper is as following:

$t_s$: the time to store the states of processors.

$t_{cp}$: the time to compare their states.

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

$R_t$: remaining execution time ( not including checkpointing and recovery).

$R_d$: time left before the deadline.

$R_f$: an upper bound on the remaining number of faults that must be tolerated.

## 2 Adaptive checkpointing scheme for single real-time task

We assume that shared memory is fault tolerant and that operation system kernel has adequate redundancy and uses robust data structures to enable fault containment. Assume task $\tau$ 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 $\lambda$, the average execution time for the task is minimum, if a constant checkpoint interval of $\sqrt{2C/\lambda}$ is used [7]. 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 $\sqrt{NC/k}$ [8]. This is the $k$-fault-tolerant approach. In addition, we assume that task $\tau$ is divided equally into $n$ intervals of length $T=\left\lceil \frac{N}{n} \right\rceil$, and at the end of each interval, CSCP is always placed.

### 2.1 SCPs and CCPs

Each CSCP interval is divided equally into $m$ intervals of length $T_1=\left\lceil \frac{T}{m} \right\rceil$. The SCP are placed 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. Two processors are rolled back to $(i\text{-}1)T_1$ because some errors have occurred during $((i\text{-}1)T_1, iT_1)$, and repeat the execution from $(i\text{-}1)T_1$. The average execution time $R_1(m)$ for a CSCP interval $((j\text{-}1)T, jT)$ is given by a renewal-equation [5, 9]:

$$R_1(m) = (mT_1 + mt_s + t_{cp})e^{-2\lambda mT_1}$$

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

$$= mT_1 + mt_s + t_{cp} + [\frac{1}{2}m(m+1)(T_1+t_s) + m(t_{cp}+t_r)](e^{2\lambda T_1}-1)$$

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 1.

```
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. 1 Procedure for calculating the $\tilde{m}$

The adaptive checkpointing with SCPs, adapchp-SCP ($D$,$E$,$C$,$k$, $\lambda$ ), is described in Figure 2. A check is performed 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 ($R_d$, $R_t$, $C$, $R_f$, $\lambda$ ) [10] to calculate the checkpoint interval.

Now we place CCP between the CSCPs.

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

Procedure adapchp-SCP ($D$, $N$, $C$, $k$, $\lambda$ ){
1. $R_t$=$N$; $R_d$=$D$; $R_f$=$k$;
2. $Itv$=interval($R_d$, $R_t$, $C$, $R_f$, $\lambda$ );
3. $m$=num_SCP($Itv$); $itv$=$\lceil Itv/m \rceil$;
4. while ($R_t$>0) do {
5. if ($R_t$> $R_d$) break with task failure;
6. Insert SCP with interval length $itv$;
7. Insert CSCP with interval length $Itv$;
8. Update $R_t$, $R_d$;
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. $R_f$= $R_f$-1;
14. $Itv$=interval($R_d$, $R_t$, $C$, $R_f$, $\lambda$ );
15. $m$=num_SCP($Itv$); $itv$=$\lceil Itv/m \rceil$;
16. Resume execution;}}

Fig. 2 Adaptive checkpointing with SCPs

some errors have occurred during this interval, the two processors will be rolled back to ($j$-1)$T$. The average execution time $R_2(m)$ for an interval (($j$-1)$T$, $jT$) is given by a renewal-equation:

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

Therefore, the average execution time $R_{CCP}(n)$=$nR_2(m)$. Replacing $m = T / T_2$ , we have:

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

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 1 to calculate $\tilde{m}$ which minimize $R_2(\tilde{m})$ . In figure 2, let "Insert CCP with interval length $itv$;" replace line 6, let "if (no error has been detected at CCP/CSCP);" replace line 9, and let "Rollback to last CSCP;" replace line 12. Then, we obtain the adaptive checkpointing with CCP, namely adapchp-CCP ($D$,$N$,$C$,$k$, $\lambda$ ) procedure.

## 2.2 Simulation results

We carried out a set of simulation experiments to evaluate our adaptive checkpointing schemes (referred to as ADTSCPs and ADTCCPs) 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[10]. Faults are injected into system using a Poisson process with various values for the arrival rate $\lambda$ . 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, either on or before the stipulated deadline. As in [10], we use the term task utilization $U$ to refer to the ratio $N/D$. In order to compared with results of [10], we let $t_r$=0.

### 2.2.1 SCPs

As said above, additional SCPs scheme fits systems which overhead time is determined mainly by the time to compare processor' states. Therefore, the parameters is as following: $D$=10000, $t_s$=1, $t_{cp}$=10, $C$=11.

Table 1 Comparison between ADTSCPs with other schemes

| $U$ | $2\lambda(\times10^{-2})$ | Probability of timely completion of tasks, $P$ | | | |
|---|---|---|---|---|---|
| | | Poisson | $k$-f-t | ADT | ADTSCPs |
| 0.76 | 0.13 | 0.704 | 0.701 | 0.716 | 0.9649 |
| | 0.15 | 0.532 | 0.512 | 0.536 | 0.9319 |
| 0.78 | 0.13 | 0.468 | 0.468 | 0.470 | 0.8818 |
| | 0.15 | 0.385 | 0.280 | 0.409 | 0.8065 |

(a) $k$=5

| $U$ | $2\lambda(\times10^{-4})$ | Probability of timely completion of tasks, $P$ | | | |
|---|---|---|---|---|---|
| | | Poisson | $k$-f-t | ADT | ADTSCPs |
| 0.92 | 1.0 | 0.7560 | 0.7592 | 0.7575 | 0.7715 |
| | 2.0 | 0.4387 | 0.4412 | 0.4780 | 0.5310 |
| 0.95 | 1.0 | 0.3843 | 0.3852 | 0.3785 | 0.3942 |
| | 2.0 | 0.1449 | 0.1449 | 0.2777 | 0.2797 |

(b) $k$=1

### 2.2.2 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, $t_s$=10, $t_{cp}$=1, $C$=11.

Table 2. Comparison between ADTCCPs with other schemes

| $U$ | $2\lambda(\times10^{-2})$ | Probability of timely completion of tasks, $P$ | | | |
|---|---|---|---|---|---|
| | | Poisson | k-f-t | ADT | ADTCCPs |
| 0.76 | 0.13 | 0.7071 | 0.7027 | 0.7191 | 0.9693 |
| | 0.15 | 0.5298 | 0.5068 | 0.5337 | 0.9306 |
| 0.78 | 0.13 | 0.4623 | 0.4623 | 0.4671 | 0.8938 |
| | 0.15 | 0.3843 | 0.2780 | 0.4062 | 0.8058 |

(a) $k$=5

| $U$ | $2\lambda(\times10^{-4})$ | Probability of timely completion of tasks, $P$ | | | |
|---|---|---|---|---|---|
| | | Poisson | k-f-t | ADT | ADTCCPs |
| 0.92 | 1.0 | 0.7528 | 0.7579 | 0.7581 | 0.7650 |
| | 2.0 | 0.4497 | 0.4490 | 0.4849 | 0.5416 |
| 0.95 | 1.0 | 0.3941 | 0.4007 | 0.3887 | 0.4030 |
| | 2.0 | 0.1544 | 0.1567 | 0.2919 | 0.2934 |

(b) $k$=1

The performances of the four schemes, measured by the probability of timely completion of the task, are comparable. For $2\lambda > 0.001$ and $U$>0.7(high fault arrival rate and relatively high task utilization), ADTSCPs and ADTCCPs schemes clearly outperforms the other three schemes, the results are shown in table 1(a), and table 2(a). For $2\lambda < 0.001$ and $U$>0.9 (low fault arrival rate and relatively high task utilization), we can get the same result in the case of $2\lambda > 0.001$ and $U$>0.7, see table 1(b) and table 2(b).

# 3 Adaptive checkpointing scheme for multiple real-time tasks

## 3.1 Extensions of adaptive checkpointing to multiple tasks

Now, we extent adaptive checkpointing procedures, namely adapchp_SCP(D,E,C,$k$, $\lambda$) and adapchp_CCP(D,E,C,$k$, $\lambda$), to a set of multiple real-time tasks. We assume that there is a set $\Psi=\{\tau_1, \tau_2, \cdots, \tau_n\}$ of $n$ tasks that must be scheduled by the system in the absence of errors. Any task $\tau_i$ in $\Psi$ has a period $T_i$, a deadline $D_i$, a computation time $E_i$ under fault-free conditions ($E_i \leqslant D_i \leqslant T_i$), and a priority level $p_i$.

For each hyper-period, we can obtained a sequence of $m$ instance $\Phi=\{\theta_1, \theta_2, \ldots, \theta_m\}$. In addition, any instance $\theta_i$ ($1 \le i \le m$) has a starting time $a_i$, an execution time $b_i$, and a deadline $c_i$. According to EDF, we have $c_1 \le c_2 \le \cdots \le c_m$. Now, we develop a checkpointing scheme that inserts checkpoints to each instance by exploiting the slacks in a hyper-period. It must determine an appropriate value for the deadline that can be provided as an input parameter to the adapchp_SCP(D,E,C,$k$, $\lambda$) and adapchp_CCP(D,E,C,$k$, $\lambda$) procedures. Of course, in order to meet the timing constraint and ensure time redundancy can be exploited for fault-tolerance, this deadline must be no greater than the exact instance deadline and no less than the instance execution time. In order to compute this parameter, we employ the method of [11], which incorporates a preprocessing step immediately after the offline EDF scheduling is carried out and the resulting values are then provided for subsequent online adaptive checkpointing procedure. We let $h_i$ denote the slack time for instance $\theta_i$ ($1 \le i \le m$). Like paper [11], we add a pseudo instance $\theta_0$, which has parameters $a_0 = b_0 = c_0 = h_0 = 0$. According to the deadline constraints, we have $\max\{a_i + b_i + h_i, a_{i+1}\} + b_{i+1} + h_{i+1} \le c_{i+1}$, where $0 \le i \le m-1$. To ensure each job has the minimum time-redundancy for fault-tolerance, paper [11] require the slack of each job to be greater than a constant threshold value $Q$, which is defined as a given number of checkpoints. Then we have $h_i \ge Q$, where $1 \le i \le m$ In this paper, we us Matlab to calculate $h_i$ that maximize the sum of all slacks $\sum_{i=1}^{m} h_i$.

Let $v_i$ represent checkpointing deadline, which is the deadline parameter provided to adapchp_SCP(D,E,C,$k$, $\lambda$) and adapchp_CCP(D,E,C,$k$, $\lambda$) procedures. Assume the actual starting time of $\theta_i$ is $a_i'$, and the actual execution time is $b_i'$. Than the actual starting time $a_{i+1}'$ of the next instance $\theta_{i+1}$ can be calculated as $a_{i+1}' = \max\{a_{i+1}, a_i' + b_i'\}$, the actual slack time $h_{i+1}'$ of $\theta_{i+1}$ is adjusted as $h_{i+1}' = h_{i+1} - (a_{i+1}' - a_{i+1})$, and the actual checkpointing deadline $v_{i+1}'$ is adjusted as

$$v_{i+1}' = \begin{cases} b_{i+1} & if \quad hi+1 < 0 \\ b_{i+1} + h_{i+1} & if \quad hi+1 \ge 0. \end{cases}$$ [11].

## 3.2 Simulation results

Like paper [11], we consider two tasks $\Psi$ ={ $\tau_1$ , $\tau_2$ }, $\tau_1$ =(5000, 7000,12000) and $\tau_2$=(4000, 11000, 18000). After offline scheduling is carried out using EDF, we obtain the sequence of instances of tasks. Note that $\theta_1$, $\theta_3$, and $\theta_5$ are instances of $\tau_1$, $\theta_2$ and $\theta_4$ are instances of $\tau_2$. Here we assume that the CSCP checkpoint cost is 11, and we require that at least 20 CSCP checkpoints are inserted for each slack. Then $Q$ equals 220. The slack values generated by Matlab are $h_1 = 899, h_2 = 1101, h_3 = 1476, h_4 = 2854, h_5 = 670$ . The parameters of instances are listed in Table 3.

Table 3 Instances parameters of the two-task example

| | $a_i$ | $b_i$ | $c_i$ | $h_i$ |
|---|---|---|---|---|
| $\theta_1$ | 0 | 5000 | 7000 | 899 |
| $\theta_2$ | 5000 | 4000 | 11000 | 1101 |
| $\theta_3$ | 12000 | 5000 | 19000 | 1476 |
| $\theta_4$ | 18000 | 4000 | 29000 | 2854 |
| $\theta_5$ | 24000 | 5000 | 31000 | 670 |

As said above, additional SCPs scheme fits systems which overhead time is determined mainly by the time to compare processor' states. Therefore, the parameters is as following: $t_s = 1, t_{cp} = 10, C = 11$ . Of course, for additional CCPs scheme, we let $t_s = 10, t_{cp} = 1, C = 11$ .

We carried out simulation experiments to evaluate our multitask adaptive checkpointing schemes, named as ADTSCPs_MUL and ADTCCPs_MUL, with the Poisson-arrival (referred to as Poisson), the *k*-fault-tolerant (referred to as *k*-f-t) checkpointing schemes. Faults are injected into system using a Poisson process with various values for the arrival rate $\lambda$ . The experiment results are shown in Table 4 and Table 5.

Table 4 Comparison between ADTSCPs_MUL with other schemes

| $2\lambda(\times 10^{-4})$ | Probability of timely completion of tasks, *P* | | |
|---|---|---|---|
| | Poisson | k-f-t | ADTSCPs_MUL |
| 0.5 | 0.4891 | 0.8841 | 0.9567 |
| 1 | 0.3811 | 0.5959 | 0.8342 |
| 2 | 0.1769 | 0.1842 | 0.4901 |
| 3 | 0.0423 | 0.0414 | 0.3050 |
| 4 | 0.0081 | 0.0079 | 0.1794 |
| 5 | 0.0008 | 0.0020 | 0.0980 |
| 6 | 0.0001 | 0.0003 | 0.0484 |

$k$=2

Table 5 Comparison between ADTCCPs_MUL with other schemes

| $2\lambda(\times 10^{-4})$ | Probability of timely completion of tasks, *P* | | |
|---|---|---|---|
| | Poisson | k-f-t | ADTCCPs_MUL |
| 0.5 | 0.4951 | 0.8698 | 0.9558 |
| 1 | 0.3765 | 0.6055 | 0.8125 |
| 2 | 0.1698 | 0.1792 | 0.4629 |
| 3 | 0.0394 | 0.0405 | 0.2974 |
| 4 | 0.0071 | 0.0081 | 0.2023 |
| 5 | 0.0016 | 0.0009 | 0.1143 |
| 6 | 0.0001 | 0.0000 | 0.0389 |

$k$=2

These results show that adaptive schemes provide a higher probability of timely completion for multitask systems than the other two schemes.

# 4 Conclusion

We develop adaptive checkpointing schemes for a set of multiple tasks in real-time systems with two processors. We use two types of checkpoints (SCP and CCP) to increase the likelihood of timely task completion in the presence of faults. Separating the comparison and store operations enables choosing the optimal interval for each operation, without concern about the other. Further, we have discussed analytically the optimal numbers of checkpoints that minimize the mean times. Based above, we present the dynamically set the checkpoints interval algorithm. Simulation results show that adaptive schemes provide a higher probability of timely completion for multitask systems than the other two schemes.

*References*

[1] Ziv A, Bruck J. "Analysis of Checkpointing Schemes with Task Duplication", *IEEE Transactions on Computers* [J], 1998, 47(2):222-227

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

[3] Kimura M, Yasui K, Nakagawa T, *et al*. Optimal checkpointing interval of a communication system with rollback recovery, *Mathematical and computer modeling*, 2003, 38:1303-1311

[4] Li Kai-Yuan, Yang Xiao-Zong. Improving the performance of checkpointing scheme with task duplication, *ACTA ELECTRONICA SINICA*(in chinese), 2000,28（5）：33-35

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

[6] Vaidya N H. A Case for two-level distributed recovery schemes, Proce. *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, 1995: 64-73

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

[8] 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

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

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

[11] Ying Z, Krishnendu C. Dynamic Adaptation for Fault Tolerance and Power Management in Embedded Real-Time Systems, *ACM Transactions on Embedded Computing Systems*, 2004,3(2):336-360