

## Short Paper

---

DOI:10.1688/JISE.2000.16.1.4

### A Cost-Effective Forward Recovery Checkpointing Scheme in Multiprocessor Systems\*

KUOCHEN WANG AND CHIEN-CHUN WANG<sup>+</sup>

*Department of Computer and Information Science*

*National Chiao Tung University*

*Hsinchu, Taiwan 300, R.O.C.*

*E-mail: kwang@cis.nctu.edu.tw*

*<sup>+</sup>Central Telecommunications Administration Station*

*Directorate General of Telecommunications*

*Taichung, Taiwan 408, R.O.C.*

This paper proposes a novel and cost-effective forward recovery checkpointing scheme for multiprocessor systems with duplex modular redundancy. In our scheme, one processing module is selected to retry the questionable checkpoint, and the other processing module executes toward the next checkpoint if a mismatched comparison between the two processing modules occurs at any checkpoint. Those schemes using a spare module to retry need much time to initiate the module, and the extra cost is high. Although the traditional rollback scheme retries the questionable checkpoint without any spare module, it has longer average completion time than our scheme for a job under any fault distribution. In our scheme, besides transient faults, permanent faults can be located as well. Experimental results based on our mathematical models demonstrate that, under burst errors, the average completion time of our scheme is reduced by 50% compared with that of the traditional rollback and is comparable with that of the scheme using a spare module to retry. In addition, our scheme has the least total execution time (the most cost-effectiveness) among the three schemes under any fault distribution.

**Keywords:** forward recovery, multiprocessor system, cost-effective, checkpointing scheme, transient fault, permanent fault

## 1. INTRODUCTION

A multiprocessor system with duplex modular redundancy (DMR) may employ two processing modules (PMs) to execute the same job concurrently. To reduce the number of restarts and to roll back to the last checkpointing state instead, the execution of a job is divided into  $n$  useful computation intervals, and each interval is followed by checkpointing [1-6]. The overhead for a checkpointing scheme is the time needed to handle these

---

Received October 16, 1997; revised June 3, 1998; accepted September 9, 1998.

Communicated by Lionel M. Ni.

\*This work was supported in part by the National Science Council, ROC under Grant NSC87-2213-E-009-030.

checkpointing states [7]. Some fault tolerant schemes are based on *fault masking*, which uses more redundant PMs to reduce the average completion time, such as in a triple module redundancy (TMR) system [8, 9] or voting [10]. Fault masking used by TMR is not suitable for a DMR system because it requires three PMs. Therefore, *fault retry* is used instead of fault masking. At each checkpoint, the states of two PMs are compared [7, 11, 12], and a retry is performed to locate a faulty checkpoint if the comparison mismatches. Two major types of retry are *retry on a spare PM* and *retry on an original PM*. The former can reduce the average completion time of a job, but the system cost is high because a spare module is required, and more time to initiate it before a retry can begin. In transaction systems, this is a serious problem because system availability is important [13]. To reduce the system cost, the latter may be preferred. However, it has problems related to *retry validation* and *longer average completion time*. To avoid an invalid retry on a faulty PM, the state of a PM needs to be made consistent with the last checkpoint at the beginning of the retry. In this paper, we propose a novel forward recovery checkpointing scheme (FCS) to provide a trade-off between the average completion time of a job and the number of PMs required. Two existing checkpointing schemes will be reviewed and compared: (1) the roll-forward checkpointing scheme (RFCS) and (2) the traditional rollback scheme (RB) [2, 7, 14]. RFCS needs a spare module to retry when a fault is detected, and RB uses both PMs simultaneously to retry the questionable checkpoint. FCS is different from the schemes in [7, 11] in terms of forward recovery without using a spare module and is unlike the traditional rollback scheme in terms of retry on a single PM, not on two PMs.

In digital computer systems, the occurrence probability of transient faults is larger than that of permanent faults [1, 15, 16]. Therefore, our analysis focuses on transient faults. We assume that there is no identical checkpointing state on two PMs when they have transient faults at the same checkpoint interval [7]. Nevertheless, our scheme can handle permanent faults, such as fail-stop faults and crash faults, as well.

The remainder of this paper is organized as follows. A DMR architecture is presented in section 2.1. Some notations are introduced in section 2.2. The basic concept of FCS is given in section 2.3. RB and RFCS schemes are reviewed in section 2.4. Section 3 illustrates in detail the FCS scheme in each fault case. Mathematical models and experimental results obtained using FCS, RB, and RFCS are presented in section 4. Section 5 depicts implementation issues. Some concluding remarks are given in section 6.

## 2. PRELIMINARIES

### 2.1 DMR Architecture for FCS

A DMR architecture for FCS is shown in Fig. 1. It consists of two PMs ( $P_1$  and  $P_2$ ), two reliable storages (RSs), and one reliable checkpoint processor (RCP). These components are connected by an interconnection network (e.g., a bus) in a multiprocessor system. To recover transient faults, no spare module is needed. Both  $P_1$  and  $P_2$  concurrently execute each replica of a job. At each checkpoint, both checkpointing states of  $P_1$  and  $P_2$  are sent to the RCP to be compared for fault detection. The fault history of each PM is recorded in the RCP. To improve the reliability of RCP, it may be configured as a TMR structure

[17]. Each PM has an RS used to save checkpointing states during fault recovery. The contents of an RS can be accessed by other processors (PMs and RCP). For brevity, we use checkpoint instead of checkpointing state in the following discussion.

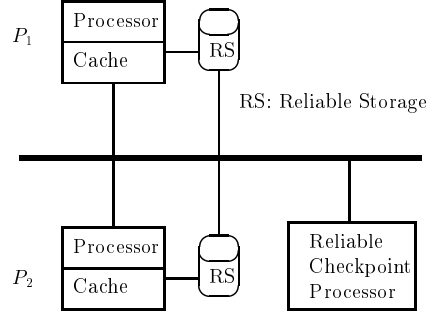
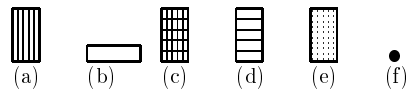


Fig. 1. DMR architecture for FCS.

## 2.2 Notations

We use six box notations for FCS, as shown in Fig. 2: (a) The time required for checkpointing is denoted by  $t_{ch}$ , which also includes the time needed for comparing the checkpoints of  $P_1$  and  $P_2$ . (b) The time required for synchronization is denoted by  $t_w$ , where the faster PM waits for the slower PM. (c) The time required to compare the retrieved checkpoint  $\phi_{j3}$  with two previously saved checkpoints  $\phi_{j1}$  and  $\phi_{j2}$  is denoted  $t_{cc}$ . (d) The time required to make the state of a faulty PM consistent with the state of the other PM is denoted  $t_{cp}$ . (e) The time required to make the states of both PMs consistent with a previous checkpoint is denoted  $t_r$ . (f) A fault is represented by a black dot. A box shaded with the same pattern represents the same operation that takes the same amount of time.



- (a) Duplex checkpointing -  $t_{ch}$
- (b) Idle -  $t_w$
- (c) Comparing the retrieved checkpoint with two saved checkpoints -  $t_{cc}$
- (d) Restoring checkpoint -  $t_{cp}$
- (e) Rollback -  $t_r$
- (f) A fault

Fig. 2. Box notations for FCS.

## 2.3 Basic Concept

This section introduces the basic concept of fault detection, location, and recovery in FCS. Fault detection is achieved through comparison of a pair of checkpoints [7, 11, 12]. In Fig. 3, one PM is selected to retry the questionable checkpoint, and the other PM ex-

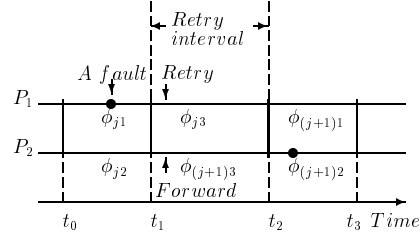


Fig. 3. FCS for a DMR system.

ecutes toward the next checkpoint continuously. Assume that the last checkpoint  $\phi_{j-1}$  is completed at time  $t_0$ . The current checkpoint is denoted  $\phi_j$ , and its following interval is denoted  $\phi_{j+1}$ . In the  $j$ th interval ( $I_j$ ), both checkpoints of  $P_1$  and  $P_2$  are denoted  $\phi_{j1}$  and  $\phi_{j2}$ , respectively. Suppose that there is a mismatch between  $\phi_{j1}$  and  $\phi_{j2}$ , that one PM has a transient fault, and that the other PM is fault-free during interval  $I_j$ . Assume  $P_1$  has a fault in interval  $I_j$ . At the beginning of the retry ( $t_1$ ),  $P_1$  is selected to retry the questionable checkpoint  $\phi_j$ , and  $P_2$  executes the next checkpoint  $\phi_{j+1}$  forward. At the end of the retry interval ( $t_2$ ), the checkpoint produced by  $P_1$  is denoted  $\phi_{j3}$ , the other checkpoint completed by  $P_2$  is denoted a  $\phi_{(j+1)3}$ ,  $\phi_{j3}$  is compared with the two previously saved checkpoints  $\phi_{j1}$  and  $\phi_{j2}$ , and  $\phi_{(j+1)3}$  is retained on the corresponding RS. The result of this comparison indicates that  $\phi_{j3}$  is identical to  $\phi_{j2}$  and different from  $\phi_{j1}$ . The correct checkpoint  $\phi_j$  for interval  $I_j$  can be derived from either  $\phi_{j2}$  or  $\phi_{j3}$ .  $\phi_j$  is then retained on the RS, and the previous checkpoint  $\phi_{j-1}$  is replaced to reduce the size of the RS. Thus, the fault in  $P_1$  is recovered successfully. Then  $P_1$  and  $P_2$  execute at the next interval  $I_{j+1}$  continuously. At time  $t_3$ , the checkpoint  $\phi_{(j+1)1}$  of  $P_1$  and the checkpoint  $\phi_{(j+1)2}$  of  $P_2$  have been completed. If another fault happens in interval  $I_{j+1}$ , either on  $P_1$  or  $P_2$ , the comparison between  $\phi_{(j+1)1}$  and  $\phi_{(j+1)2}$  will show them not to be identical. Then  $\phi_{(j+1)3}$  is compared with  $\phi_{(j+1)1}$  and  $\phi_{(j+1)2}$  immediately. Note that  $\phi_{(j+1)3}$  has completed at time  $t_2$ . The faulty  $\phi_{(j+1)2}$  can be correctly located without the need for an additional retry interval.

## 2.4 RB and RFCS Schemes

For comparison with the other two schemes, the RFCS and RB schemes are illustrated in Fig. 4 and Fig. 5, respectively [7, 11]. Note that the retry for RFCS is performed on a spare module. As to RB, both PMs are rolled back to the previous checkpoint upon failure at the same time.

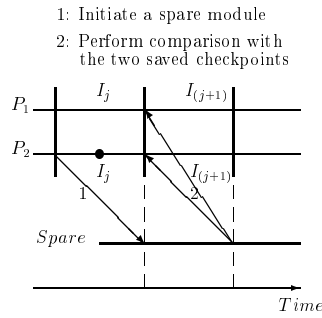


Fig. 4. Roll forward checkpointing scheme (RFCS).

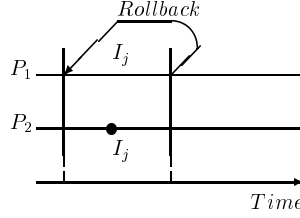


Fig. 5. Rollback scheme (RB).

### 3. FORWARD RECOVERY CHECKPOINTING SCHEME

#### 3.1 Transient Faults

Now we will begin to describe our FCS under transient faults in detail. If there is a mismatched comparison at an interval, the second interval is needed to obtain a retried checkpoint and a forward checkpoint on two PMs, respectively. A forward checkpoint is valid if its previous checkpointing state is fault-free. Here the correctness of the forward checkpoint has not yet been determined. An additional interval is required to make sure whether the forward checkpoint is correct. In the following analysis, the states of at most three consecutive intervals are considered. That is, the scheme has three possible situations — (1) no retry interval, (2) one retry interval and the forward checkpoint is invalid, and (3) one retry interval and the forward checkpoint is valid. Depending on how faults occur, there are six possible fault cases. In the following, we will discuss these fault cases denoted (A) through (F).

(A) *No failure*: Both  $P_1$  and  $P_2$  are fault-free in interval  $I_j$  (see Fig. 6). At the end of the current interval  $I_j$ ,  $\phi_{j1}$  and  $\phi_{j2}$  are found to be identical. The correct checkpoint  $\phi_j$  is derived from either  $\phi_{j1}$  or  $\phi_{j2}$ . Then  $\phi_j$  is saved in each RS, and the previous checkpoint  $\phi_{j-1}$  is replaced. At time  $t_1$ , the current interval  $I_j$  has completed, and  $P_1$  and  $P_2$  are ready to execute at the next interval  $I_{j+1}$ . The time required for this case is  $t_A = T = t_u + t_{ch}$ , where  $t_u$  is the time required for useful computation in a checkpoint interval. The average completion times in this case are all the same for the three schemes.

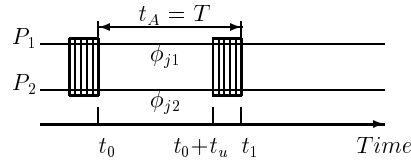


Fig. 6. Case (A) — No failure.

(B) *Recovering a failure, followed by a state restoration*: In this case, the current interval  $I_j$  has a fault, the retry checkpoint  $\phi_{j3}$  is fault-free, and the forward checkpoint  $\phi_{(j+1)3}$  will be invalid. Suppose that  $\phi_{j1}$  is fault-free, and that  $\phi_{j2}$  has a fault (see Fig. 7). A retry interval is inserted between intervals  $I_j$  and  $I_{j+1}$ . Suppose that  $P_1$  is selected to retry due to a

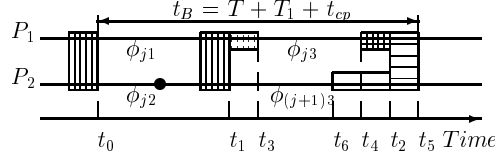


Fig. 7. Case (B) — Recovering a failure, followed by a state restoration.

faulty module selection miss. The forward checkpoint  $\phi_{(j+1)3}$  is invalid because it is infected by the fault in  $\phi_{j2}$ . We will discuss the situation of a selection hit in cases (D) through (F). At time  $t_1$ ,  $\phi_{j1}$  and  $\phi_{j2}$  are found to be different. A retry interval is initiated. First, both  $\phi_{j1}$  and  $\phi_{j2}$  are saved in the respective RS. Then, the state of  $P_1$  is made consistent with the state of the previous checkpoint  $\phi_{j-1}$ . The time required for this retry is  $t_r = t_3 - t_1$ . At time  $t_3$ ,  $P_1$  is ready to retry. Concurrently,  $P_2$  continues to execute toward the forward checkpoint. At time  $t_6$ , this forward checkpoint is completed first. Then  $P_2$  waits until  $P_1$  completes the retry. The idle time is  $t_w = t_2 - t_6$ . At time  $t_4$ ,  $P_1$  completes the retry, and  $\phi_{j3}$  is compared with the two saved checkpoints  $\phi_{j1}$  and  $\phi_{j2}$ . At time  $t_2$ , it is found that  $\phi_{j3}$  is identical to  $\phi_{j1}$  and different from  $\phi_{j2}$ . The time required for this comparison is  $t_{cc} = t_2 - t_4$ . At the same time, the forward checkpoint  $\phi_{(j+1)3}$  is determined to be invalid due to a fault in its previous interval. The state of  $P_2$  is incorrect and must be made consistent with the correct state from  $P_1$ . The time required for this state restoration is  $t_{cp} = t_5 - t_2$ . At time  $t_5$ ,  $P_1$  and  $P_2$  are ready to execute at the next interval  $I_{j+1}$ . FCS can recover a transient failure that appears in  $P_2$  without the need for a spare module, but RFCS, which needs a spare module to retry, will fail if no spare module is available. The time required for this case is  $t_B = T + T_1 + t_{cp}$ , where  $T_1 = t_r + t_u + t_{cc}$ .

(C) *Rollback due to retry interval failure*: The retry interval may not succeed if at least two faults occur. There are three possible scenarios as listed in Table 1. In this case, at least two faults happen in interval  $I_j$  and in the retry interval. The number of faulty checkpoints is at least two among  $\phi_{j1}$ ,  $\phi_{j2}$ , and  $\phi_{j3}$ . RCP can not find the correct checkpoint  $\phi_j$ . Both PMs are rolled back to the previous checkpoint  $\phi_{j-1}$ . For the sake of illustration, consider scenario  $C_1$  illustrated in Fig. 8. Here two faults happen in interval  $I_{j2}$  and retry interval  $I_{j3}$ , respectively. First,  $\phi_{j1}$  and  $\phi_{j2}$  are found to be different at time  $t_1$ . A retry interval is initiated. The retry is the same as that in case (B) except that the comparison mismatches at time  $t_2$ . RCP can not find two identical checkpoints because  $\phi_{j2}$  and  $\phi_{j3}$  are incorrect. Then the states of  $P_1$  and  $P_2$  are made consistent with the previous checkpoint  $\phi_{j-1}$ . The time required for this rollback is  $t_r = t_5 - t_2$ . After the rollback is completed at time  $t_5$ ,  $P_1$  and  $P_2$  are in a state identical to their respective state at  $t_0$ . No checkpoint is completed in this case. The time required for this case is  $t_C = T + T_1 + t_r$ .

Table 1. Three possible fault scenarios in case (C).

Scenario	Status in interval $I_j$		Status in the retry interval	
	$\phi_{j1}$	$\phi_{j2}$	$\phi_{j3}$	$\phi_{(j+1)3}$
$C_1$	fault-free	faulty	faulty	don't-care
$C_2$	faulty	fault-free	faulty	don't-care
$C_3$	faulty	faulty	don't-care	don't-care

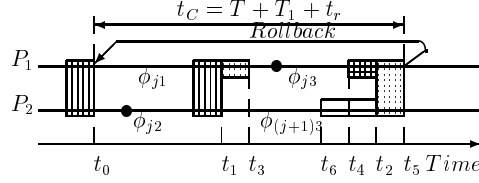


Fig. 8. Case (C1) — Rollback due to retry interval failure.

(D) *Retry successful and no more failure in the next interval*: In this case and the following two cases, (E) and (F), the retry mechanism succeeds to recover a failure in the current interval  $I_j$ . The forward checkpoint  $\phi_{(j+1)3}$  is also valid due to a selection hit. No more failures happen in the next interval  $I_{j+1}$  in this case. As shown in Fig. 9, there is a fault in  $P_1$ , and  $P_1$  is selected to retry. At time  $t_1$ ,  $\phi_{j1}$  and  $\phi_{j2}$  are found to be different. A retry interval is inserted between the current interval  $I_j$  and the next interval  $I_{j+1}$ . Up to time  $t_2$ , the case is similar to case (B). The forward checkpoint  $\phi_{(j+1)3}$  is first saved in the corresponding RS. Both  $P_1$  and  $P_2$  continue to execute at the next interval  $I_{j+1}$ . After one interval of time, both checkpoints  $\phi_{(j+1)1}$  and  $\phi_{(j+1)2}$  are found to be identical at time  $t_5$ . The forward checkpoint  $\phi_{(j+1)3}$  will not be used in this case. At time  $t_5$ , both  $P_1$  and  $P_2$  are ready to execute the following interval  $I_{j+2}$ . Thus, a single failure in  $P_1$  is recovered successfully. The time required for this case is  $t_D = 2T + T_1$ .

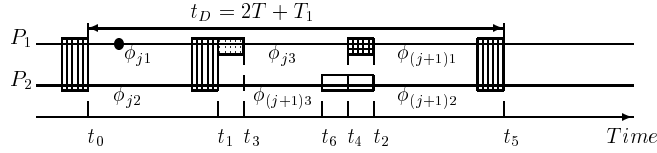


Fig. 9. Case (D) — Retry successful and no more failure in the next interval.

(E) *Retry successful and forward recovery of a failure in the next interval*: In this case, there are two faults in intervals  $I_j$  and  $I_{j+1}$ , respectively. The forward checkpoint  $\phi_{(j+1)3}$  will be valid due to a selection hit. The infection of the fault is only one interval, but it is two intervals in RFCS. There are two possible fault scenarios as listed in Table 2. To illustrate, scenario  $E_1$  is shown in Fig. 10. Since this case has been described in section 2.3, we will only make some observations. Up to time  $t_5$ , this case is similar to case (D) except that a fault occurs in  $I_{(j+1)2}$ . Note that both failures in  $\phi_{j1}$  and  $\phi_{(j+1)1}$  are recovered successfully. In this case, FCS needs only one retry interval to recover two failures, but RB needs at least two rollback intervals. The time required for this case is  $t_E = 2T + T_1 + t_{cc} + t_{cp}$ .

Table 2. Two possible fault scenarios in case (E).

Scenario	Status in interval $I_j$		Status in the retry interval		Status in interval $I_{j+1}$	
	$\phi_{j1}$	$\phi_{j2}$	$\phi_{j3}$	$\phi_{(j+1)3}$	$\phi_{(j+1)1}$	$\phi_{(j+1)2}$
$E_1$	faulty	fault-free	fault-free	fault-free	fault-free	faulty
$E_2$	faulty	fault-free	fault-free	fault-free	faulty	fault-free

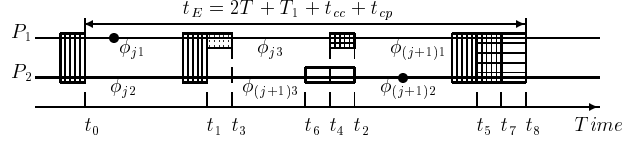


Fig. 10. Case ( $E_1$ ) — Retry successful and forward recovery of a single failure in the next interval.

(F) *Retry successful and rollback in the next interval*: There are three possible fault scenarios in this case (see Table 3). To illustrate, consider scenario  $F_1$  as shown in Fig. 11, where  $\phi_{j1}$ ,  $\phi_{(j+1)3}$ , and  $\phi_{(j+1)2}$  are faulty. At time  $t_1$ , a fault is detected, and a retry interval is inserted between interval  $I_j$  and interval  $I_{j+1}$ . Up to time  $t_7$ , this case is similar to case (E) except that two faults occur in  $\phi_{(j+1)3}$  and  $\phi_{(j+1)2}$ , respectively. RCP can not find the correct checkpoint  $\phi_{j+1}$  in the next interval  $I_{j+1}$ . So both PMs are rolled back to the previous checkpoint  $\phi_j$ . This rollback is similar to that of case (C). The time required for the rollback is  $t_r = t_8 - t_7$ . At time  $t_8$ , the states of  $P_1$  and  $P_2$  are the same as those at time  $t_2$ . In this case, a failure in  $I_j$  is recovered successfully. But the other two failures, in  $\phi_{(j+1)2}$  and  $\phi_{(j+1)3}$ , can not be recovered. The time required for this case is  $t_F = 2T + T_1 + t_{cc} + t_r$ .

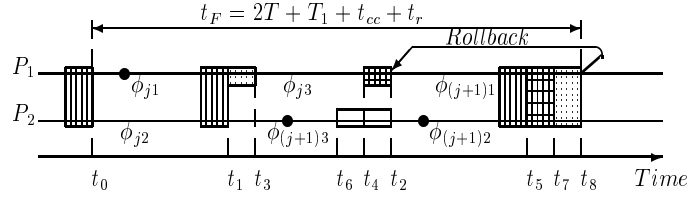


Fig. 11. Case ( $F_1$ ) — Retry successful and rollback in the next interval.

**Table 3. Three possible fault scenarios in case (F).**

Scenario	Status in interval $I_j$		Status in the retry interval		Status in interval $I_{j+1}$	
	$\phi_{j1}$	$\phi_{j2}$	$\phi_{j3}$	$\phi_{(j+1)3}$	$\phi_{(j+1)1}$	$\phi_{(j+1)2}$
$F_1$	faulty	fault-free	fault-free	faulty	fault-free	faulty
$F_2$	faulty	fault-free	fault-free	faulty	faulty	fault-free
$F_3$	faulty	fault-free	fault-free	don't-care	faulty	faulty

### 3.2 Permanent Faults

The scheme presented above can also detect and locate permanent faults in a PM. Remember that if a PM encounters failures in several consecutive intervals during a period of time, then the PM is assumed to have a permanent fault. We assume that  $P_1$  and  $P_2$  will not have permanent faults at the same time. Suppose  $P_1$  has a permanent fault, and that  $P_2$  is fault-free. As shown in Fig. 12, at time  $t_1$ , a fault is detected.  $P_1$  is first selected to retry.



At time  $t_2$ , the retry fails, and  $P_1$  and  $P_2$  are rolled back to the previous checkpoint  $\phi_{j-1}$  as in case (C). At time  $t_3$ , the execution still fails. Therefore, the retry is switched to  $P_2$ . At time  $t_4$ , this retry succeeds, and  $P_1$  is identified as being faulty. The failure history is updated to indicate that  $P_1$  is faulty. Then both  $P_1$  and  $P_2$  continuously execute in the next interval  $I_{j+1}$ . At time  $t_5$ , the comparison between the two checkpoints  $\phi_{(j+1)1}$  and  $\phi_{(j+1)2}$  mismatches. A retry is performed again.  $P_2$  is selected to retry again. At time  $t_6$ , the correct checkpoint  $\phi_{j+1}$  is found, and  $P_1$  is found to have a fault once again. The results of the subsequent execution during  $t_4$  to  $t_6$  are similar to those during  $t_2$  to  $t_4$ , which are similar to case (B) except that two consecutive faults occur in  $P_1$ . The results always indicate that  $P_1$  is faulty. Since  $P_1$  encounters several consecutive failures in a period of time, it is determined to have a permanent fault.

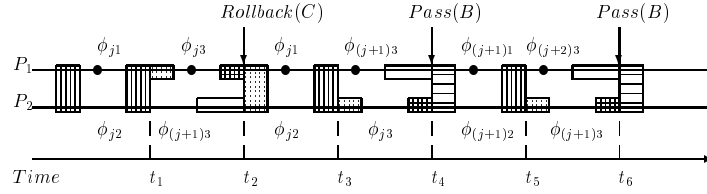


Fig. 12. Locating a module with a permanent fault: an example.

## 4. PERFORMANCE EVALUATION

In this section, we will evaluate the performance of FCS, RFCS, and RB quantitatively. Two performance measures will be examined: *average completion time* and *total execution time*. The total execution time can be used as a measure of cost-effectiveness for each scheme.

### 4.1 Poisson Distributed Faults

In section 3, we have described the six possible fault cases in detail. Now we will evaluate the occurrence probability for each case. The analysis presented here focuses on transient faults. Occurrence of transient faults in a module is first assumed to be a Poisson process with a constant failure rate  $\lambda$  [13, 18]. Therefore, the failure probability of a module is  $1 - e^{-\lambda T}$  during time period  $T$ . Based on the probability and the time required for each case (see Table 4, where  $m$  is the selection miss rate), we can derive the average completion time and the total execution time for FCS. Now we will calculate the average completion time for a job with  $k$  checkpoints. If  $k = 1$ , then FCS is identical to RB. When a fault is detected, a DMR system will roll back instead of performing a retry. The occurrence probability for case (A),  $P_A$ , is derived under the no failure condition.  $P_{roll}$  is the probability of a rollback with  $k = 1$ . Thus  $P_{roll} = 1 - P_A = 1 - e^{-2\lambda T}$ . The average completion time ( $\bar{\tau}_k$ ) for  $k = 1$  is derived as follows [7]:

$$\bar{\tau}_1 = P_A t_A + P_{roll} (\bar{\tau}_1 + t_{roll}), \quad (1)$$

**Table 4. The likelihood of the six cases.**

Case	Probability	required time
(A)	$P_A = e^{-2\lambda T}$	$t_A = T$
(B)	$P_B = 2me^{-\lambda(T+T_1)}(1 - e^{-\lambda T})$	$t_B = T + T_1 + t_{cp}$
(C)	$P_C = 2e^{-\lambda T}(1 - e^{-\lambda T})(1 - e^{-\lambda T_1}) + (1 - e^{-\lambda T})^2$	$t_C = T + T_1 + t_r$
(D)	$P_D = 2(1 - m)e^{-\lambda(3T+T_1)}(1 - e^{-\lambda T})$	$t_D = 2T + T_1$
(E)	$P_E = 4(1 - m)e^{-2\lambda(T+T_1)}(1 - e^{-\lambda T})^2$	$t_E = 2T + T_1 + t_{cc} + t_{cp}$
(F)	$P_F = 2(1 - m)e^{-\lambda(T+T_1)}(1 - e^{-\lambda T})^2(1 + e^{-\lambda T} - 2e^{-\lambda(T+T_1)})$	$t_F = 2T + T_1 + t_{cc} + t_r$

where  $t_{roll} = T + t_r$ .

If  $k = 2$ , then FCS remains the same as RB. Therefore,

$$\overline{\tau}_2 = 2\overline{\tau}_1. \quad (2)$$

For  $k \geq 3$ , a retry procedure is initiated if a fault is detected. Note that  $P_A + P_B + P_C + P_D + P_E + P_F = 1$ . The recursion for  $\overline{\tau}_k$  is derived as follows:

$$\begin{aligned} \overline{\tau}_k = & P_A(\overline{\tau}_{k-1} + t_A) + P_B(\overline{\tau}_{k-1} + t_B) + P_C(\overline{\tau}_k + t_C) + P_D(\overline{\tau}_{k-2} + t_D) + \\ & P_E(\overline{\tau}_{k-2} + t_E) + P_F(\overline{\tau}_{k-1} + t_F). \end{aligned} \quad (3)$$

The DMR system is rolled back when a fault occurs in the last two intervals. The initial conditions are  $\overline{\tau}_1$  and  $\overline{\tau}_2$ . Starting with the above recursive equation (3), the closed form of  $\overline{\tau}_n$  is shown in equation (4):

$$\overline{\tau}_n = \frac{q_{DE}}{1 + q_{DE}} \left( \begin{aligned} & (q_A t_A + q_B t_B + q_C t_C + q_D t_D + q_E t_E + q_F t_F)((n-2)q_{DE}^{-1}) \\ & + \frac{1 - (-q_{DE})^{n-2}}{1 + q_{DE}} + \overline{\tau}_1(q_{DE}^{-1} + (-q_{DE})^{n-1}) \\ & + \overline{\tau}_2(2 - (q_A + q_B + q_F))(q_{DE}^{-1} + (-q_{DE})^{n-2}) \end{aligned} \right) \quad (4)$$

where  $q_X = \frac{P_X}{1 - P_C}$ , for  $X = A, B, C, D, E, F$ ,

$$q_{DE} = q_D + q_E,$$

$$\overline{\tau}_1 = \frac{T + t_r}{e^{-2\lambda T}} - t_r, \text{ and}$$

$$\overline{\tau}_2 = 2\overline{\tau}_1.$$

The average completion time  $\overline{\tau}_n|f$  of a job can then be derived. Note that  $\overline{\tau}_n|f$  is equal to  $\overline{\tau}_n$  except that at least one Poisson distributed fault ( $f$ ) occurs during the execution of a job. If there are  $n$  checkpoints, the probability that there will be no fault in each execution interval is  $P_A^n$ . The value of  $\overline{\tau}_n|f$  can be derived as follows [7]:

$$\overline{\tau}_n|f = \frac{\overline{\tau}_n - P_A^n nT}{1 - P_A^n}. \quad (5)$$

## 4.2 Burst Faults

The average completion time  $\overline{\tau_n|f_b}$  when a DMR system encounters burst faults ( $f_b$ ) can be derived as follows. Note that there are no faults in case (A), single faults in cases (B) and (D), multiple faults in cases (C), (E) and (F). Under burst faults, assume that cases (B) and (D) will not happen, and that their occurrence probabilities are distributed to cases (C), (E) and (F) based on their occurrence weights. Thus, we have

$$\begin{aligned} P_C' &= P_C + W_C \times (P_B + P_D), \\ P_E' &= P_E + W_E \times (P_B + P_D), \\ P_F' &= P_F + W_F \times (P_B + P_D), \end{aligned}$$

where  $W_i = \frac{P_i}{P_C + P_E + P_F}$ ,  $i = C, E$ , or  $F$ .

Therefore,

$$\overline{\tau_n|f_b} = \overline{\tau_n}$$

with  $P_A$  retained,  $P_B = P_D = 0$ , and  $P_C$ ,  $P_E$  and  $P_F$  replaced by  $P_C'$ ,  $P_E'$  and  $P_F'$ , respectively.

## 4.3 Analysis of RB and RFCS

We will derive the average completion time of RB and RFCS under Poisson distributed faults and burst faults. Since  $\overline{\tau_n|f}$  can be derived from  $\overline{\tau_n}$  by means of equation (5), only  $\overline{\tau_n}$  for RB and RFCS are given here. Under burst faults, RB needs at least two intervals to roll back. Therefore,  $\overline{\tau_n|f_b}$  of RB can be derived from  $\overline{\tau_n}$  by multiplying the time of rollback interval by 2. In the case of RFCS under burst faults, situation B will not happen. Therefore, the occurrence probability of situation B is distributed to situations C and D based on their occurrence weights. Thus, we have

*RB:*

$$\overline{\tau_n} = n \left( \frac{T + t_r}{e^{-2\lambda T}} - t_r \right), \quad (7)$$

$$\overline{\tau_n|f_b} = n \left( \frac{2 \times (T + t_r)}{e^{-2\lambda T}} - t_r \right); \quad (8)$$

*RFCS:*

$$\overline{\tau_n} = \frac{q_B}{1 + q_B} \left[ \frac{(q_A t_A + q_B t_B + q_C t_C + q_D t_D)(n-2)q_B^{-1}}{1 - (-q_B)^{n-2}} + \overline{\tau_1}(q_B^{-1} + (-q_B)^{n-1}) + (\overline{\tau_2} - q_{AD} \overline{\tau_1})(q_B^{-1} + (-q_B)^{n-2}) \right], \quad (9)$$

$$\overline{\tau_n|f_b} = q_{AD}^{n-1} \overline{\tau_1} + (q_A t_A + q_C t_C + q_D t_D) \left( \frac{1 - q_{AD}^{n-1}}{1 - q_{AD}} \right). \quad (10)$$

#### 4.4 Performance Comparison Among FCS, RB and RFCS

To conduct performance comparisons among these three schemes, we will use a hypothetical job called *job 1*. Parameters for job 1 are listed in Table 5, where  $T_u$  is defined as the *total useful execution time* of a job,  $t_s$  is the time needed to initialize a job, and  $t_{pr}$  is the time needed to initiate a spare module to retry [7]. The other notations have been defined in section 2. These parameters of checkpointing and fault recovery are very sensitive with the system environment [13], including the network delay and synchronization problem. However, they will not affect the performance comparisons.

**Table 5. Parameters for job 1.**

$T_u$	$t_{ch}$	$t_r$	$t_s$	$t_{cc}$	$t_{cp}$	$t_{pr}$
50	0.50	0.50	0.30	0.07	0.05	0.80

*Comparison of the average completion time under Poisson distributed faults:* In Fig. 13,  $\overline{\tau_n|f}$  of each scheme is shown. The  $x$ -axis represents the failure rates from  $10^{-2}$  to  $10^{-8}$ , and the  $y$ -axis represents the average completion time with the optimal checkpoint number [19]. We use three miss rates to illustrate our FCS scheme. FCS\_1, FCS\_0.5, and FCS\_0 represent the cases of  $m = 1$ ,  $m = 0.5$ , and  $m = 0$ , respectively. RFCS has the shortest average completion time among the three schemes because it uses a spare module during a fault retry. Note that neither FCS nor RB employ any spare modules, and FCS performs better than RB. In addition, the miss rate has little impact on the average completion time under low failure rates, unless the failure rate is high. This means that the performance of our scheme is very stable.

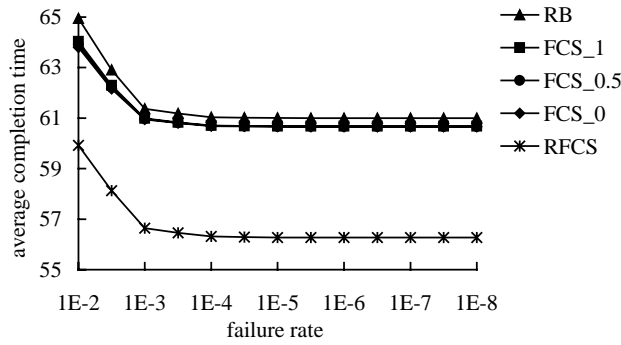
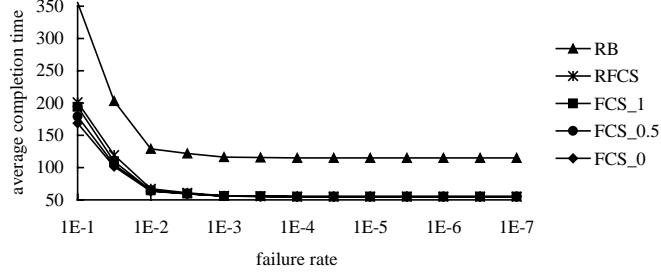


Fig. 13.  $\overline{\tau_n|f}$  for job 1 with optimal checkpoint number.

*Comparison of the average completion time under burst faults:* In Fig. 14,  $\overline{\tau_n|f_b}$  of each scheme is shown. The  $x$ -axis represents the failure rates from  $10^{-1}$  to  $10^{-7}$ , and the  $y$ -axis represents the average completion time with 10 checkpoints. The results show that FCS reduces by 50% the average completion time compared with RB and is comparable with RFCS in this respect. Therefore, our scheme is very suitable for situations with burst errors.

Fig. 14.  $\overline{\tau_n|f_b}$  for job 1.

*Comparison of the total execution time  $T_e$  (or cost-effectiveness):* Without loss of generality, we only consider Poisson distributed faults. In a DMR system without a spare module for fault recovery, the total execution time is equal to twice the average completion time  $\overline{\tau_n|f}$ . FCS and RB are such cases because they use only two PMs. The execution time of the spare module must be considered in RFCS. That is, it uses two modules when no failures occur and three modules when a failure occurs. Four situations are considered [7]. In situation (A), there is no failure, and the number of PMs is two. In situations (B) through (D), RFCS uses a spare module to retry, and the number of PMs is three. The execution time of RFCS for each situation is listed in Table 6. For RFCS, derivation of the total execution time is similar to the derivation of  $\overline{\tau_n|f}$  except that the required time for each situation is replaced by the corresponding execution time in Table 6. The total execution time for each of the three schemes (FCS, RB and RFCS) is equal to twice  $\overline{\tau_n|f}$ , summarized as follows:

**Table 6. The execution time for each situation in RFCS.**

Situation	Required execution time
(A)	$t'_A = 2T$
(B)	$t'_B = 4T + 2t_w + 2t_{cc} + 2t_u + 2t_{cp} + t_{pr}$
(C)	$t'_C = 4T + 2t_w + t_{cc} + t_u + 2t_r + t_{pr}$
(D)	$t'_D = 4T + 2t_w + 4t_{cc} + 4t_u + 2t_r + t_{pr}$

$$T_e(\text{FCS}) = 2 \times \overline{\tau_n|f}(\text{FCS}),$$

$$T_e(\text{RB}) = 2 \times \overline{\tau_n|f}(\text{RB}),$$

$$T_e(\text{RFCS}) = 2 \times \overline{\tau_n|f}(\text{RFCS}), \text{ with } \{t'_A, t'_B, t'_C, t'_D\} \text{ instead of } \{t_A, t_B, t_C, t_D\}.$$

In Fig.15, the  $x$ -axis represents the failure rates from  $10^{-2}$  to  $10^{-8}$ , and the  $y$ -axis represents the total execution time under Poisson distributed faults with the optimal checkpoint number [19]. In sum, the total execution time of our FCS is the smallest while that of RFCS is the largest among the three schemes under any fault distribution.

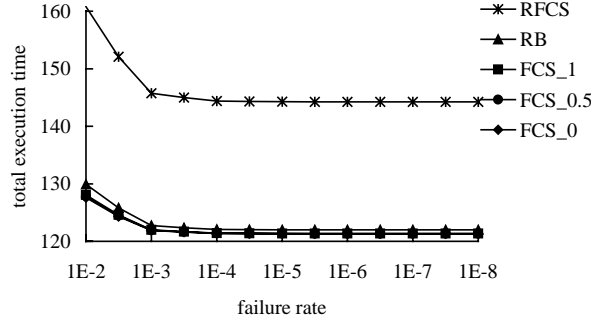


Fig. 15.  $T_e$  for job 1 with optimal checkpoint number.

## 5. DISCUSSION

Some implementation issues are considered here. All checkpointing schemes need reliable storages to save some checkpoints during execution of a job. Reliable storages are very important components of the proposed scheme. We assume that reliable storages are very dependable. The papers in [7, 14, 15, 17, 20] all assumed that reliable storages are available for fault recovery. The RFCS scheme requires that at most five checkpointing states be stored in the reliable storages when a failure occurs while FCS requires that at most three checkpointing states be stored for each case. In this respect, our scheme is better. Each checkpointing scheme needs a reliable checkpoint processor for comparison and synchronization. To avoid a single point of failure, the reliable checkpoint processor can be implemented by using a masking redundancy or by a distributed self-checking approach.

## 6. CONCLUSIONS

A very cost-effective forward recovery checkpointing scheme for multiprocessor systems with duplex modular redundancy has been described in this paper. We have compared our scheme (FCS) with the other two representative schemes: RB and RFCS. FCS only needs one retry interval to recover two transient faults at two consecutive checkpoints. RB needs at least two rollback intervals in the same fault situation. RFCS performs state restoration at the end of a retry, and execution on a faulty module is useless during the retry interval. Therefore, our scheme may be able to avoid wasting time on a faulty module during a retry interval. In addition, our scheme, without a spare module, has less system cost than those schemes which have a spare module in terms of hardware expense and the initialization time of the spare module. Mathematical models have been derived to evaluate the average completion time and the cost-effectiveness (total execution time) of the three schemes. Experimental results show that FCS can provide a better trade-off between average completion time and cost-effectiveness than the other two schemes under any fault distribution. In addition, under burst errors, the average completion time of FCS is de-

creased by 50% compared with that of RB and is comparable with that of RFCS. Furthermore, a processing module with a permanent fault can be located in FCS, and the size of the required reliable storage is smaller than that for RFCS.

## REFERENCES

1. G. Muller, M. Banâtre, N. Peyrouze and B. Rochat, "Lessons from FTM: an experiment in the design & implementation of a low-cost fault-tolerant system," *IEEE Transactions on Reliability*, Vol. 45, No. 2, 1996, pp. 332-340.
2. Y. Deng and E. K. Park, "Checkpointing and rollback-recovery algorithms in distributed systems," *Journal of Systems Software*, Vol. 25, No. 1, 1994, pp. 59-71.
3. K. Saleh, I. Ahmad and K. Al-Saqabi, "An efficient recovery procedure for fault tolerance in distributed systems," *Journal of Systems Software*, Vol. 25, No. 1, 1994, pp. 39-50.
4. J. L. Kim and T. Park, "An efficient protocol for checkpointing recovery in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 8, 1993, pp. 955-960.
5. E. N. Elnozahy, D. B. Johnson and W. Zwaenepoel, "The performance of consistent checkpointing," in *Proceeding of Symposium on Reliable Distributed Systems*, 1992, pp. 39-47.
6. R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Transactions on Computer Systems*, Vol. 3, No. 3, 1985, pp. 204-226.
7. D. K. Pradhan and N. H. Vaidya, "Roll-forward checkpointing scheme: a novel fault-tolerant architecture," *IEEE Transactions on Computers*, Vol. 43, No. 10, 1994, pp. 1163-1174.
8. K. G. Shin and H. Kim, "A time redundancy approach to TMR failures using fault-state likelihoods," *IEEE Transactions on Computers*, Vol. 43, No. 10, 1994, pp. 626-636.
9. M. Kameyama and T. Higuchi, "Design of dependent failure tolerant microcomputer systems using triple modular redundancy," *IEEE Transactions on Computers*, Vol. c-29, No. 2, 1980, pp. 202-205.
10. T. Takano, T. Yamada, K. Shutoh and N. Kanekawa, "In-orbit experiment on the fault-tolerant space computer aboard the satellite Hiten," *IEEE Transactions on Reliability*, Vol. 45, No. 4, 1996, pp. 624-631.
11. J. Long, W. K. Fuchs and J. A. Abraham, "Forward recovery using checkpointing in parallel systems," *International Conference on Parallel Processing*, Vol. 1, 1990, pp. 272-275.
12. P. Agrawal, "Fault tolerance in multiprocessor systems without dedicated redundancy," *IEEE Transactions on Computers*, Vol. 37, No. 3, 1988, pp. 358-362.
13. A. Zvi and J. Bruck, "Analysis of checkpointing schemes for multiprocessor systems," *IEEE 13th Symposium on Reliable Distributed Systems*, 1994, pp. 52-61.
14. P. A. Bernstein, "Sequoia: a fault-tolerant tightly coupled multiprocessor for transactions processing," *Computer*, Vol. 21, No. 2, 1988, pp. 37-45.
15. K. G. Shin and Y. H. Lee, "Error detection process – model, design, and its impact on computer performance," *IEEE Transactions on Computers*, Vol. c-33, No. 6, 1984, pp.

- 529-540.
16. S. R. McConnel, D. P. Siewiorek and M. M. Tsao, "The measurement and analysis of transient errors in digital computer systems," *The 9th International Symposium Fault-Tolerant Computing*, 1979, pp. 67-70.
  17. P. K. Chande, A. K. Ramani and P. C. Sharma, "Modular TMR multiprocessor systems," *IEEE Transactions on Industrial Electronics*, Vol. 36, No. 1, 1989, pp. 34-41.
  18. R. A. Howard, *Dynamic Probabilistic Systems, Vol. II: Semi Markov and Decision Processes*, John Wiley, 1971.
  19. A. Zvi and J. Bruck, "Optimal number of checkpoints in checkpointing schemes," Manuscript, 1993.
  20. T. H. Lin and K. G. Shin, "An optimal retry policy based on fault classification," *IEEE Transactions on Computers*, Vol. 43, No. 9, 1994, pp. 1014-1025.

**KuoChen Wang (王國禎)** received the B.S. degree in control engineering from National Chiao Tung University, Taiwan, in 1978, and the M.S. and Ph.D. degrees in electrical engineering from the University of Arizona in 1986 and 1991, respectively. He is currently a Professor in the Department of Computer and Information Science, National Chiao Tung University. From 1980 to 1984, he worked on network management, and the design and implementation of the Toll Trunk Information System at the Directorate General of Telecommunications in Taiwan. He served in the army as a second lieutenant communication platoon leader from 1978 to 1980. His research interests include computer networks, mobile computing and wireless networks, fault-tolerant computing, and computer-aided VLSI design.

**Chien-Chun Wang (王建春)** received the B.S. degree in electronic engineering from Taipei University of Science and Technology, Taiwan, in 1985 and the M.S. degree in computer and information science from National Chiao Tung University in 1995. He is currently a specialist at the Central Telecommunications Administration Station, Directorate General of Telecommunications in Taiwan. His research interests include mobile computing, fault-tolerant computing, and telecommunication networks.