Proceedings of International Conference on Distributed
Systems, Software Engineering and Database Systems

An Efficient Forward Recovery Checkpointing Scheme in Distributed Systems

Kuochen Wang and Chien-Chun Wang

Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.
Email: kwang@cis.nctu.edu.tw

Abstract

In this paper, we present a cost-effective forward recovery checkpointing scheme (FRCS) for distributed systems without a spare module, where a job is executed simultaneously on two processors (or processing modules). For those schemes with a spare module, the time required for initiating the spare module is long and the cost is high for an extra module. Mathematical models have been developed to demonstrate that our scheme is always better than the traditional rollback scheme without any spare module and is better than other schemes with a spare module in terms of number of processing modules required and the total execution time (cost-effectiveness).

1. Introduction

Checkpointing schemes [1] [2] [3] are widely used for fault tolerance in distributed systems. At each checkpoint, the states of two processing modules (PMs) are compared in order to detect a fault, and a recovery procedure is performed to locate a faulty module when a mismatch occurs. Those schemes using spare modules [4] [5] to perform the recovery procedure can reduce the average completion time of a task, but the system cost is high due to the need of a spare module and a long time to initiate it. In transaction systems, the system availability is important [6]. For reducing the system cost, retry is performed on the same module, which maybe the faulty module. For avoiding the invalid retry on the faulty module, the module needs to be rolled back to the last checkpoint at the begin of the retry. However, the average completion time is longer than the schemes using spare modules. In this paper, we propose a novel checkpointing scheme to provide a trade-off between the average completion time of a task and the number of modules required.

In our forward recovery checkpointing scheme (FR CS), no spare module is required and the retry for an additional checkpoint is conducted in one of the PMs. Two other checkpointing schemes are reviewed here:

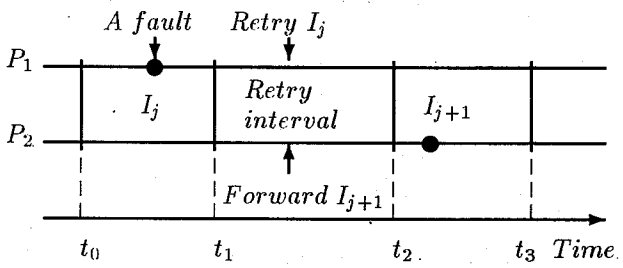


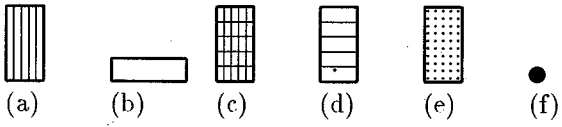
Figure 1: FRCS for a duplex system.

(1) the roll-forward checkpointing scheme (RFCS) and (2) the rollback (RB) scheme [1] [5] [8], to compare with our FRCS scheme. The RFCS scheme needs to employ a spare module to do the retry when a fault is detected. As to the RB scheme, when the two checkpoints mismatch, both PMs reexecute the questionable checkpoint simultaneously.

The remainder of this paper is organized as follows. The basic concept of FRCS is introduced in section 2. Section 3 illustrates our cost-effective checkpointing scheme in detail. The evaluation among the FRCS, RB, and RFCS schemes are presented in section 4. Some concluding remarks are shown in section 5.

2. Basic concept

This section introduces the basic concept of fault detection, location, and recovery in our FRCS scheme. The fault detection is achieved by a comparison on a pair of checkpoints [4] [5]. At the beginning of the retry interval (time t_1), the states CP_{j1} and CP_{j2} are saved on the corresponding *reliable storage* (see Figure 1). The CP_{j1} and CP_{j2} are the states of P_1 and P_2, respectively. If P_1 is selected to retry the current checkpoint I_j and P_2 executes in the next checkpoint I_{j+1} forward. At the end of the retry interval (time t_2), the retried checkpoint of P_1, denoted CP_{j3}, is used to compare with CP_1 and CP_2 in order to locate the faulty checkpoint. Assume that a single fault occurs



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

Figure 2: Box notations for FRCS.

on P_1 and P_2 is fault-free during the interval I_j. The result of comparison will indicate that CP_{j3} is identical with CP_{j2} and different from CP_{j1}. Now, we can assume that P_1 was faulty during the interval I_j. At the same time, the forward checkpoint of P_2, denoted $CP_{(j+1)3}$, will complete and be saved on the *reliable storage*. It is used to recover an additional fault in the next interval, if any. Then P_1 and P_2 execute in the next interval $I_{(j+1)}$. Suppose another single fault happens on P_2 during the next interval I_{j+1}. At time t_3, the comparison of $CP_{(j+1)1}$ and $CP_{(j+1)2}$ will not be identical. They can be compared with $CP_{(j+1)3}$ from the corresponding reliable storage immediately. By comparison, another fault on P_2 during the interval $I_{(j+1)}$ can be recovered without requiring another retry interval. However, in the RB scheme, this overhead is at least two rollback intervals for the same fault case.

3. Forward recovery checkpointing scheme

There are six box notations used in this paper (see Figure 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) For synchronization, the time is denoted by t_w, where the faster PM waits for the slower PM. (c) The time required for comparing the retried checkpoint CP_{j3} with two previous saved checkpoints CP_{j1} and CP_{j2} is named t_{cc}. (d) The time needed for making the state of a faulty PM consistent with the state of the other PM is named t_{cp}. (e) The time needed for making the states of both PMs consistent with a previous checkpoint is named t_r. (f) A fault is represented by a black dot. The length of one *useful computation* is denoted by t_u, and let $T = t_u + t_{ch}$ and $T_1 = t_r + t_u + t_{cc}$. Depending on how the faults occur, there are six possible fault cases, denoted as (A) through (F).

(A) *No failure*: Both P_1 and P_2 are fault-free in the interval I_j (see Figure 3). The probability of occurrence for this case is $P_A = e^{-2\lambda T}$ [5] [9]. The execution time is $t_A = T$.

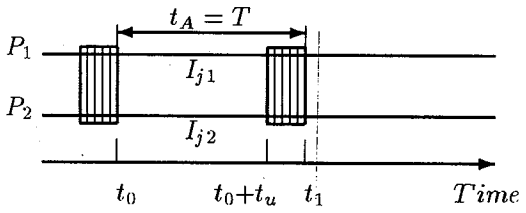


Figure 3: Case (A) — No failure.

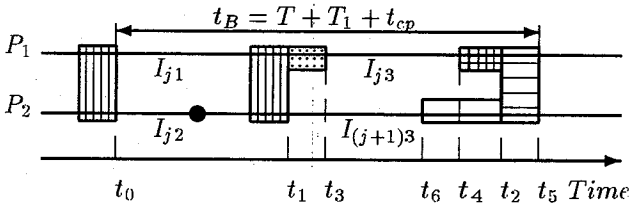


Figure 4: Case (B) — Recovering a single failure and followed by a state restoration.

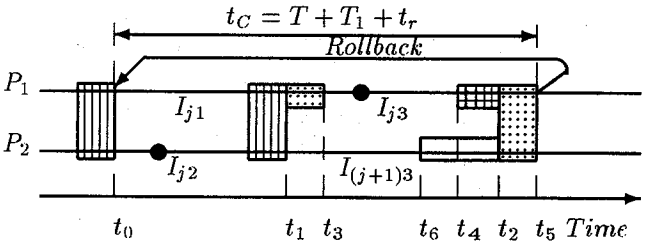


Figure 5: Case (C_1) — Rollback due to the retry interval failure.

(B) *Recovering a single failure and followed by a state restoration*: In this case, the current interval I_j has a single fault and the retry checkpoint CP_{j3} is fault-free (see Figure 4). At time t_6, this forward checkpoint is completed first. Then P_2 waits until P_1 completes the retry procedure. This idle time is named $t_w = t_2 - t_6$. At time t_4, P_1 finishes the retry and its state CP_{j3} is compared with the two saved checkpoints CP_{j1} and CP_{j2}. At time t_2, it is found that CP_{j3} is identical with CP_{j1} and different from CP_{j2}. The time required for this comparison is denoted by $t_{cc} = t_2 - t_4$. Then P_2 is assumed faulty and it must be made consistent with the correct state from P_1. The time required for this state restoration is denoted by $t_{cp} = t_5 - t_2$. The probability of occurrence for this case is $P_B = e^{-\lambda(T+T_1)}(1 - e^{-\lambda T})$. The execution time is denoted by $t_B = T + T_1 + t_{cp}$.

(C) *Rollback due to the retry interval failure*: The number of faulty checkpoints is at least two among CP_{j1}, CP_{j2}, and CP_{j3}. There are three possible scenarios as listed in Table 1. For the sake of illustration, consider scenario C_1 illustrated in Figure 5. Here two faults happened in the interval I_{j2} and the retry interval I_{j3}. Both PMs are rolled back to the previous

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

Scenario	Status in the interval I_j		Status in the retry interval	
	I_{j1}	I_{j2}	I_{j3}	$I_{(j+1)3}$
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

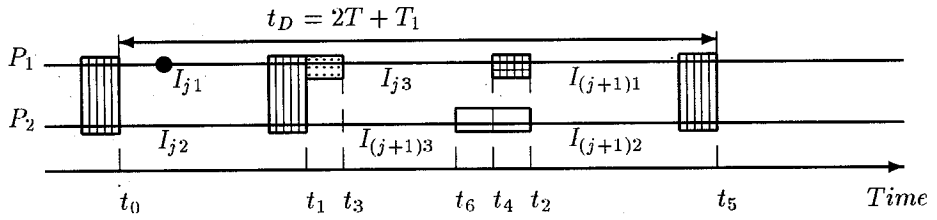


Figure 6: Case (D) — Retry successful and no more failure in the next interval.

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

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

checkpoint CP_{j-1}. The time required for this state restoration is denoted by $t_r = t_5 - t_2$. The probability of occurrence for this case is $P_C = 2e^{-\lambda T}(1 - e^{-\lambda T})(1 - e^{-\lambda T_1}) + (1 - e^{-\lambda T})^2$. The execution time is denoted by $t_C = T + T_1 + t_r$.

(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 single failure in the current interval I_j. The forward checkpoint $CP_{(j+1)3}$ is also valid due to a faulty module *selection hit*. There is only a single fault during the interval I_j and the remainder is fault-free (see Figure 6). The probability of occurrence for this case is $P_D = e^{-\lambda(3T+T_1)}(1 - e^{-\lambda T})$. The execution time is denoted by $t_D = 2T + T_1$.

(E) *Retry successful and forward recovery a single failure in the next interval*: There are two possible fault scenarios listed in Table 2. For illustration, consider scenario E_1 as shown in Figure 7. The intervals I_{j1} and $I_{(j+1)1}$ are both faulty. At time t_7, the correct checkpoint CP_{j+1} is derived since $CP_{(j+1)3}$ and $CP_{(j+1)2}$ are identical. The time required for this comparison, $t_7 - t_5$, is similar to that between t_4 and t_2, and is also denoted by t_{cc}. Both failures in I_{j1} and $I_{(j+1)1}$ are recovered successfully. Note that it needs only one retry interval to recover two failures in our scheme. For the RB scheme, it needs at least two rollback intervals. The probability of occurrence for this case is $P_E = 2e^{-2\lambda(T+T_1)}(1 - e^{-\lambda T})^2$. The execution time is denoted by $t_E = 2T + T_1 + t_{cc} + t_{cp}$.

(F) *Retry successful and rollback in the next interval*: There are three possible fault scenarios in this case (see Table 3). For illustration, consider scenario F_1 as shown in Figure 8. The failure in I_{j1} is recovered successfully. But the other two failures, in $I_{(j+1)2}$ and $I_{(j+1)3}$, can not be recovered successfully. The probability of occurrence for this case is $P_F = e^{-\lambda(T+T_1)}(1 - e^{-\lambda T})^2(1 + e^{-\lambda T} - 2e^{-\lambda(T+T_1)})$. The execution time is denoted by $t_F = 2T + T_1 + t_{cc} + t_r$.

4. Performance evaluation

Here we will evaluate the performance among FRCS, RB, and RFCS schemes quantitatively. The performance measures examined are: *average completion time* and *total execution time*.

$\overline{\tau_n}$ = the average completion time for a job

$\overline{\tau_n | f}$ = same as $\overline{\tau_n}$, except that at least one failure occurs during the execution of a job

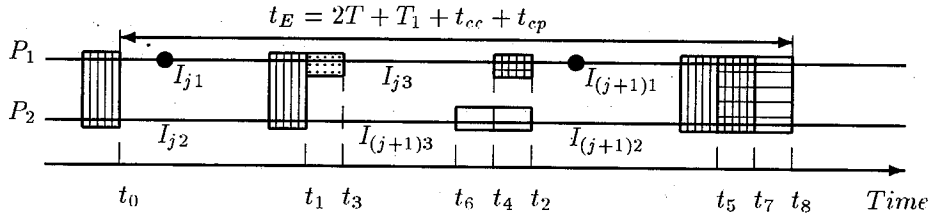


Figure 7: Case (E_1) — Retry successful and forward recovery a single failure in the next interval.

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

Scenario	Status in the interval I_j		Status in the retry interval		Status in interval I_{j+1}	
	I_{j1}	I_{j2}	I_{j3}	$I_{(j+1)3}$	$I_{(j+1)1}$	$I_{(j+1)2}$
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

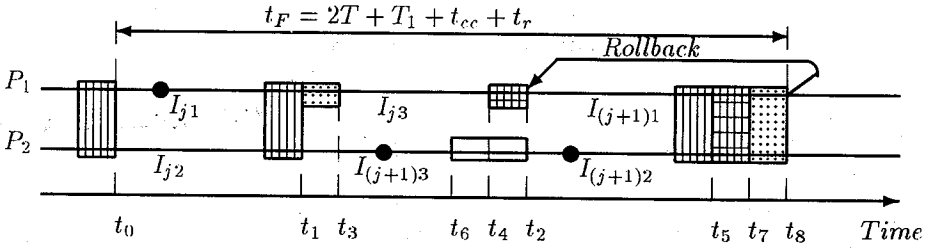


Figure 8: Case (F_1) — Retry successful and rollback in the next interval.

Table 4: Parameters for job 1.

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

T_e = total execution time of a job which is the summation of the execution time of each PM, including that of a spare module if used.

The total execution time can be used as a measure of cost-effectiveness for each scheme. The analysis presented here focuses on transient faults. Occurrence of a transient fault in a module is assumed to be a Poisson process with a constant failure rate λ [6] [9]. So the failure probability of a module is $1 - e^{-\lambda T}$ during the time period T. Based on the probability and required time for each case, the *average completion time* of three schemes are shown in equations (1), (2), and (3).

If there are n checkpoints, the probability of no failure in every execution interval is ${P_A}^n$. The value of $\overline{\tau_n|f}$ can be derived as follows [5]:

$$\overline{\tau_n|f} = \frac{\overline{\tau_n} - {P_A}^n nT}{1 - {P_A}^n}$$

To conduct the performance comparison among these three schemes, we use a hypothetical job called job 1. Parameters for job 1 are listed in Table 4.

Comparison of the average completion time $\overline{\tau_n|f}$: In Figure 9, $\overline{\tau_n|f}$ of the three schemes are shown, respectively. 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 [7]. The RFCS scheme has the shortest time among the three schemes, because it uses a spare module during a fault retry interval. Note that both FRCS and RB do not employ any spare module and our FRCS performs better than RB.

Comparison of the total execution time T_e (or cost-effectiveness): In a duplex system without using a spare module for fault recovery, the total execution time is equal to twice of the completion time $\overline{\tau_n|f}$. FRCS and RB are belong to this case, because they

$FRCS$:

$$\overline{\tau_n} = \frac{q_{DE}}{1+q_{DE}} \left(\begin{array}{l} (q_A t_A + q_B t_B + q_C t_C + q_D t_D + q_E t_E + q_F t_F)\left((n-2){q_{DE}}^{-1} + \frac{1-(-q_{DE})^{n-2}}{1+q_{DE}}\right) \\ +\overline{\tau_1}\left({q_{DE}}^{-1} + (-q_{DE})^{n-1}\right) + \overline{\tau_1}\left(2-(q_A+q_B+q_F)\right)\left({q_{DE}}^{-1} + (-q_{DE})^{n-2}\right) \end{array} \right) \quad (1)$$

where $q_X = \frac{P_X}{1-P_C}$, for $X = A, B, C, D, E, F$, $q_{DE} = q_D + q_E$, $\overline{\tau_1} = \frac{T+t_r}{e^{-2\lambda T}} - t_r$, and $\overline{\tau_2} = 2\overline{\tau_1}$.

RB :

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

$RFCS$:

$$\overline{\tau_n} = \frac{q_B}{1+q_B} \left(\begin{array}{l} (q_A t_A + q_B t_B + q_C t_C + q_D t_D)\left((n-2){q_B}^{-1} + \frac{1-(-q_B)^{n-2}}{1+q_B}\right) \\ +\overline{\tau_1}\left({q_B}^{-1} + (-q_B)^{n-1}\right) + (\overline{\tau_2} - q_{AD}\overline{\tau_1})\left({q_B}^{-1} + (-q_B)^{n-2}\right) \end{array} \right) \quad (3)$$

Table 5: The execution time for each situation in RFCS scheme.

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)	$t_D{}' = 4T + 2t_w + 4t_{cc} + 4t_u + 2t_r + t_{pr}$

always use only two PMs to retry. For RFCS, the execution time of the spare module must be considered as well. That is, it uses two modules when no failure occurs and three modules when a failure occurs. We derive the execution time of RFCS for each situation in Table 5 [5]. The total execution time T_e for each of the three schemes (FRCS, RB, and RFCS) equal to twice of $\overline{\tau_n|f}$ is summarized as follows:

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

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

$T_e(\text{RFCS}) = 2 \times \overline{\tau_n|f}$ (RFCS) with $\{ t_A{}', t_B{}', t_C{}', t_D{}' \}$ instead of $\{ t_A, t_B, t_C, t_D \}$

In Figure 10, the x-axis represents the failure rates from 10^{-2} to 10^{-8} and the y-axis represents the total execution time with the optimal checkpoint number [7]. The total execution time of our FRCS is the best while that of RFCS is the worst among the three schemes.

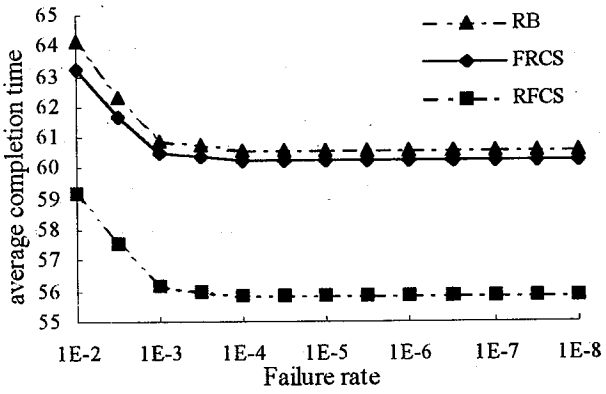


Figure 9: $\overline{\tau_n|f}$ for job 1 with optimal checkpoint number.

5. Conclusions

We have presented a very cost-effective forward recovery checkpointing scheme for distributed systems without using spares. Our scheme without a spare module has less system cost than those schemes with a spare module in terms the hardware expense and the initialization time of the spare module. During the retry interval, one processing module is doing the retry checkpoint and the other processing module is doing the next checkpoint for recovering a possible fault in the next interval. Thus, our scheme only needs one retry interval to recover two transient faults at two consecutive checkpoints. In the rollback scheme, it needs at least two rollback intervals at the same fault situation. Mathematical models have been derived to evaluate the average completion time and the cost-effectiveness (total execution time) of the different schemes. Simulation results shows that our scheme

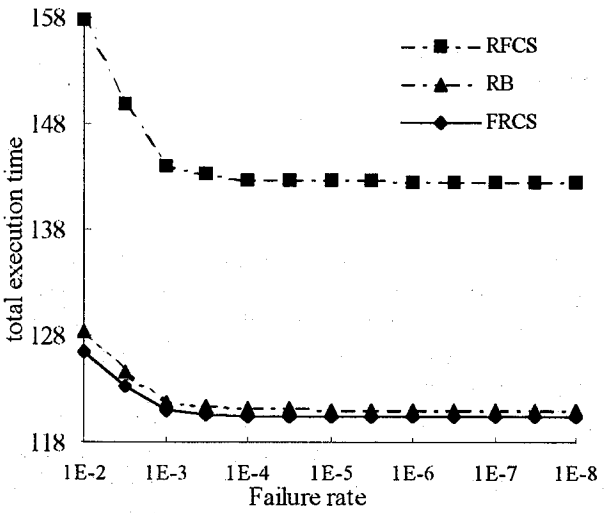


Figure 10: T_e for job 1 with optimal checkpoint number.

can provide a better trade-off between average completion time and cost (or number of processing modules required) than the other two schemes. That is, our FRCS scheme is the most cost-effective among the three schemes in terms of the total execution time under a wide range of failure rates.

Acknowledgment

This work was supported in part by the National Science Council of the Republic of China under Grant NSC86-2213-E-009-085.

References

[1] Yong Deng and E. K. Park, "Checkpointing and Rollback-Recovery Algorithms in Distributed Systems," *Journal of Systems Software*, vol. 25, 1994, pp. 59-71.

[2] Kassem Saleh, Imtiaz Ahmad, and Khaled Al-Saqabi, "An Efficient Recovery Procedure for Fault Tolerance in Distributed Systems," *Journal of Systems Software*, vol. 25, 1994, pp. 39-50.

[3] Junguk L. Kim and Taesoon Park, "An Efficient Protocol for Checkpointing Recovery in Distributed Systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 8, Aug. 1993, pp. 955-960.

[4] Junsheng Long, W. Kent Fuchs, and Jacob A. Abraham, "Forward Recovery using Checkpointing in Parallel Systems," *International Conference on Parallel Processing*, vol. 1, Aug. 1990, pp. 272-275.

[5] Dhiraj K. Pradhan and Nitin H. Vaidya, "Roll-Forward Checkpointing Scheme: A Novel Fault-Tolerant Architecture," *IEEE Trans. on Computers*, vol. 43, no. 10, Oct. 1994, pp. 1163-1174.

[6] Avi Zvi and Jehoshua Bruck, "Analysis of Checkpointing Schemes for Multiprocessor Systems," *IEEE 13th Symposium on Reliable Distributed Systems*, 1994, pp. 52-61.

[7] Avi Zvi and Jehoshua Bruck, "Optimal Number of Checkpoints in Checkpointing Schemes," *Manuscript*, 1993.

[8] P. A. Bernstein, "Sequoia: A Fault-Tolerant Tightly Coupled Multiprocessor for Transaction Processing," *Computer*, Feb. 1988, pp. 37-45.

[9] R. A. Howard, "Dynamic Probabilistic Systems Vol II: Semi Markov and Decision Processes," *John Wiley*, 1971.