Evaluation of Error Recovery Blocks Used for Cooperating Processes

KANG G. SHIN, SENIOR MEMBER, IEEE, AND YANN-HANG LEE

Abstract—Three alternatives for implementing recovery blocks (RB's) are conceivable for backward error recovery in concurrent processing. These are the asynchronous, synchronous, and the pseudorecovery point implementations.

Asynchronous RB's are based on the concept of maximum autonomy in each of concurrent processes. Consequently, establishment of RB's in a process is made independently of others and unbounded rollback propagations become a serious problem.

In order to completely avoid unbounded rollback propagations, it is necessary to synchronize the establishment of recovery blocks in all cooperating processes. Process autonomy is sacrificed and processes are forced to wait for commitments from others to establish a recovery line, leading to inefficiency in time utilization.

As a compromise between asynchronous and synchronous RB's we propose to insert pseudorecovery points (PRP's) so that unbounded rollback propagations may be avoided while maintaining process autonomy.

We developed probabilistic models for analyzing these three methods under standard assumptions in computer performance analysis, i.e., exponential distributions for related random variables. With these models we have estimated 1) the interval between two successive recovery lines for asynchronous RB's, 2) mean loss in computation power for the synchronized method, and 3) additional overhead and rollback distance in case PRP's are used.

Index Terms—Backward error recovery, conversation scheme, domino effect, pseudorecovery points and lines(s), recovery block(s), recovery line(s), rollback propagations.

I. INTRODUCTION

THE increasing computation power and rapidly falling cost of microprocessors and memories have given an impetus to the development of distributed computing systems. Potential benefits to be gained through distributed architectures include enhanced system throughput, structural flexibility, reliability, and availability. However, there are several issues to be resolved before the full potential of a distributed processing system can be realized in practice. For example, the multiplicity of physical and logical system components apparently seem to improve reliability, but this becomes less obvious when we consider the issues involved with system reconfiguration, error, recovery, and detection in a distributed environment. In this paper, we consider one such issue: the effectiveness of implementing recovery blocks(RB's) in backward error recovery for a set of cooperating processes.

The *recovery block* (RB), proposed by Horning [1] and Randell [2], has been widely used for backward error recovery. It is a sequential program structure that consists of an acceptance test(AT), a recovery point(RP), and alternative algorithms for a given process. A process saves its state at a recovery point and then enters a recovery region. At the end of a recovery block, the acceptance test is excuted to check correctness of the computation results. In case an error is detected during the normal execution or the computation results fail to pass the acceptance test, the process rolls back to an old state saved at the previous RP and executes one of the other alternatives. This procedure will be repeated until the computation results pass the acceptance test or all alternative algorithms are exhausted. The latter implies the system's inability for tolerating fault(s) whereas the former means the system's capability for fault-tolerance.

Unfortunately, for cooperating concurrent processes the rollback of a process may cause other processes to roll back (this phenomenon is called *rollback propagation*) because of interprocess dependencies and imperfect checking of global correctness. Moreover, rollback may propagate to further RP's since recovery points of individual processes may not provide globally consistent states for all processes involved. This rollback propagation continues until it reaches a *recovery line* at which globally consistent states for all involved processes do exist. In the worst case, an avalanche of rollback propagation, called the *domino effect*, can push the processes back to their beginnings, thus resulting in loss of the entire computation done prior to the occurrence of error.

A detailed description of the domino effect can be found in [3]. For convenience in visualizing rollback propagations let us consider Fig. 1. Process P_1 begins to roll back because of unsuccessful acceptance test AT_4^1. Due to interprocess dependencies the rollback of P_1 propagates to the other two processes P_2 and P_3. Eventually, the whole system has to restart from recovery line RL_2, undoing the entire computation between RL_2 and AT_4^1. The time interval between the restart point following an error recovery and the time point at which an error is detected or the acceptance test fails, called the *rollback distance*, can be used to represent the computation loss in rollback error recovery. The rollback distance may be unbounded in the case of the domino effect.

The domino effect is the major obstacle in implementing the recovery block scheme for concurrent processing. The process designer is able to predict neither the time of the occurrence of process interactions nor that of the appearance of recovery lines. In addition, it is not desirable to randomly place recovery points and acceptance tests without considering process charac-

Manuscript received March 7, 1983; revised October 7, 1983. This work was supported in part by the National Aeronautics and Space Administration under Grant NAG 1-296.

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

0098-5589/84/1100-0692$01.00

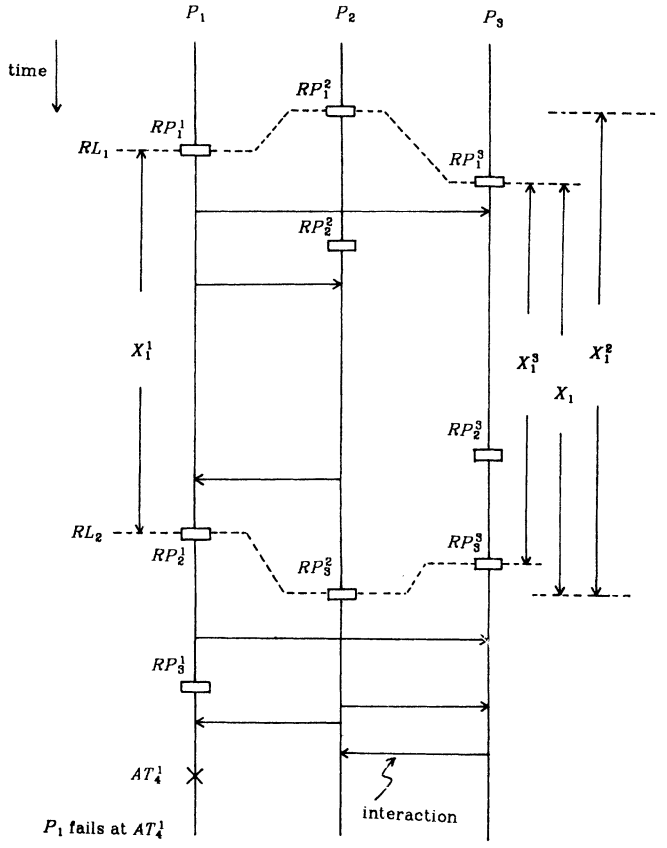


Fig. 1. A history diagram of occurrence of interactions and recovery points.

teristics. Thus, it is impossible to avoid the domino effect only by appropriate placement of recovery blocks and it is possible to have a disaster such as unbounded rollback propagation, a large rollback distance, and a great number of largely useless recovery points occupying large amounts of memory space, etc. Furthermore, detection of rollback propagation and determination of recovery lines will become more complex though they can be made in a centralized [4], [5] or decentralized manner [6]-[8].

Several refinements have been proposed to overcome the drawbacks in the recovery block scheme. One approach is to put concurrent processes into a controlled scope, either to synchronize the occurrence of acceptance tests or to direct process interactions. For the former, Randell [2] has suggested the *conversation scheme* which requests every cooperating concurrent process to leave its acceptance test at the same moment (called *test line*). He has also proposed a language structure in an abstract form for the conversation scheme. Other mechanizations of the conversation scheme on the basis of the same concept but with more flexibility have been devised by Kim [9]. Synchronized rollback recovery schemes for transactions using a two-phase commitment protocol or transaction ordering are also studied in [10]-[12]. Russell has proposed that information be retained for directed interactions from producers to consumers so that rollback propagation can be blocked [13], [14]. Another approach is to save additional states based on the occurrence of interactions; for example, the branch recovery point [15] and the system defined checkpoint (SDCP) [16].

In this paper we propose to employ *pseudorecovery points*[1] (PRP's) to alleviate the rollback propagation problem by allowing a process to restart at a PRP when the process is forced to roll back by others as a result of rollback propagation. Hence, we can classify these refinements into two categories, *synchronized recovery blocks* and *pseudorecovery points*, providing a contrast with the third category called *asynchronous recovery blocks*.

To implement a rollback error recovery scheme, we have to weigh tradeoffs between these three categories and the characteristics of concurrent processes. A satisfactory scheme should have such festures as a low (acceptable) delay in process completion due to rollbacks, the preservation of process autonomy in concurrent processing, and programmer transparency. Therefore, optimal solutions may be a combination of these three categories. A quantitative analysis has to precede any of such optimal solutions. For example, it is necessary to determine the mean amount of computation undone when processes roll back, the optimal interval between two successive synchronizations, the mean size of memory space required to save states, etc. However, because the program behavior is unknown and its execution proceeds stochastically, accurate modeling is in general very difficult if not impossible.

In this paper, employing standard assumptions in computer performance analysis, we develop a model to quantitatively describe the characteristics of rollback recovery schemes as well as their effectiveness. In the following section, several assumptions are discussed and then a model for asynchronous recovery blocks is introduced. Using this model, we employ simulation to present the probability distribution of the interval between two successive recovery lines and the mean number of states recorded during that interval. In Sections III and IV, the synchronization method and the implantation of pseudorecovery points are discussed and evaluated, respectively. The paper concludes with Section V.

II. Evaluation of Asynchronous Recovery Blocks

Let us consider the history diagram in Fig. 1 to illustrate the activities of cooperating concurrent processes $P_i, i = 1, 2, \cdots n$. Process P_i establishes its jth recovery point RP_j^i without synchronizing with other processes. Interprocess communications are represented by arrowed horizontal lines. Let set $A \subset \{1, \cdots, n\}$ be a subset of cooperating process indexes. Then one may find a combination of RP_j^i for all $i \in A$, which forms a recovery line for set A, denoted as RL_r^A for the rth recovery line. For simplicity, superscripts in representing recovery lines will be omitted in the sequel as long as that does not result in ambiguity. The interval between two successive recovery lines RL_r and RL_{r+1} in process P_i is a random variable and denoted by X_r^i. Since a recovery line provides globally consistent states

[1]We call it a pseudorecovery point(PRP) since there is no acceptance test before the saving of process state at a PRP. The states recorded at PRP's may have been contaminated and thus cannot be used to recover a failed process. But PRP's can be used to prevent rollback propagations due to interactions with the faulty process as we shall see in Section IV.

to all members of process set A, it is reasonable to assume that X_r^i is stochastically identical for all $i \in A$. Thus, X_r is used to represent the interval between the rth and $(r+1)$th recovery lines.

A. Modeling Assumptions

We make the following assumptions in our subsequent analyses.

A1) Autonomous Processes: Cooperative autonomy is regarded as the most important requirement in distributed processing. Each process should be executed according to its own program and environment, almost as if no other process existed. In actuality, a process may execute independently of others as long as there is no conflict with others in accessing shared resources. Since synchronization is not enforced in this category of recovery blocks (i.e., asynchronous recovery blocks), processes will transmit messages or establish their recovery points independently of other processes.

A2) Perfect Local Acceptance Test: Acceptance tests should detect all errors within the local process during the execution of recovery blocks and thus ensure the correctness of local execution. It is in general difficult to guarantee complete correctness, but at least the computation results that have passed the acceptance test should be "acceptable" [3]. The local acceptance test may or may not detect external errors or erroneous messages because the local process is not aware of the global system and other processes.

A3) Probability Distribution of Interactions: Usually, process behavior is modeled as an ordered sequence, which in turn is specified by the program and is dependent on the execution condition. Even if the processing sequence is given, the interval between two successive interactions is variable due to conditional branches. Locking and waiting at shared resources make it even more uncertain. Nonetheless, the interaction interval can be modeled by adopting the two assumptions commonly used in the analysis of multiprocessors and computer networks: 1) constant reference rates in the multiprocessor and 2) exponentially distributed intervals between two successive message transmissions in the computer network. The interval for two successive interactions between P_i and P_j is thus assumed to be exponentially distributed with mean $1/\lambda_{ij}$ and $\lambda_{ij} = \lambda_{ij}$ for all $i, j = 1, 2, \cdots, n$ and $i \neq j$.

A4) Consistent Communications: Let two messages m_a and m_b be sent from P_i to P_j. Consistent communications should satisfy: 1) every message sent from P_i to P_j will be received eventually by P_j, and 2) m_a and m_b are received by P_j in the same order as that they are sent. Notice that in some packet-switched computer networks, messages are allowed to be received by the destination out of order. However, the order can be kept easily, for example, by time-stamping messages at the time of transmission.

A5) Distribution of Recovery Points: Because of process independence and the uncertainty of execution conditions, the appearances of recovery points are random and difficult to model. To avoid complexity, establishment of recovery points in a process is assumed to be an independent Poisson process with parameter μ_i for P_i.

B. A Model for Asynchronous Recovery Blocks

Since individual recovery points by themselves may not be sufficient in rollback recovery due to the possibiltiy of rollback propagations, we consider in this paper only the formation of recovery lines for asynchronous recovery blocks instead of separate individual recovery points. The requirements of a recovery line for processes P_i $i = 1, 2, \cdots, n$ can be stated as follows.

1) A recovery line has to include one recovery point RP_j^i for each process P_i.

2) Let the moment of establishment of the jth recovery point in P_i be $t[\mathrm{RP}_j^i]$ and let $t_q^{ii'}$ be the moment of the qth interaction from P_i to $P_{i'}$. For every pair $(\mathrm{RP}_j^i, \mathrm{RP}_{j'}^{i'})$ in a recovery line, there does not exist an integer k such that $t_k^{ii'} \in [t[\mathrm{RP}_j^i], t[\mathrm{RP}_{j'}^{i'}]]$ if $t[\mathrm{RP}_j^i] \leqslant t[\mathrm{RP}_{j'}^{i'}]$, or $t_k^{ii'} \in [t[\mathrm{RP}_{j'}^{i'}], t[\mathrm{RP}_j^i]]$ otherwise. This implies that no interaction from P_i to P_j (and vice versa) can be sandwiched between $t[\mathrm{RP}_j^i]$ and $t[\mathrm{RP}_{j'}^{i'}]$.

The basic idea underlying the model is to trace the occurrence of both recovery points and interactions. Based on the assumptions in Section II-A, random variable X_r can be modeled by a continuous-time Markov process starting from a recovery line (RL_r) and ending at the next recovery line (RL_{r+1}). For a set of processes, $\Omega_A = \{P_i \mid i \in A\}$ where $A = \{1, 2, \cdots, n\}$, two types of states are defined:

a) End states S_r and S_{r+1}: transitions start from S_r where all processes have formed the rth recovery line, and end at S_{r+1} upon establishment of the $(r+1)$th recovery line.

b) Intermediate states $S = (x_1, x_2, \cdots, x_n)$, where $x_i = 0$ if the previous action of P_i was an interaction, and $x_i = 1$ if it was establishment of a recovery point.

Occurrences of interactions and recovery points in a process make the system go through these states. Note that both S_r and S_{r+1} are equivalent to state $(1, 1, \cdots, 1)$. We can establish the following transition rules:

R1) The system goes to state $(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n)$ from state $(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$ with rate μ_i upon establishment of a recovery point in P_i.

R2) The system leaves state $(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_{j-1}, 1, x_{j+1}, \cdots, x_n)$ and enters state $(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_{j-1}, 0, x_{j+1}, \cdots, x_n)$ with rate λ_{ij} if there is an interaction between P_i and P_j.

R3) The system arrives at state $(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n)$ from state $(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)$ with transition rate $\Sigma_{j \in B_i} \lambda_{ij}$ where $B_i = \{j \mid x_j = 0, j \neq i \text{ and } j \in A\}$.

R4) The system can transfer directly from state S_r to state S_{r+1} with transition rate $\Sigma_{k=1}^{n} \mu_k$.

Under these transition rules a Markov model is developed for three processes P_1, P_2, and P_3, and presented in Fig. 2. The single-arrow lines are unidirectional transitions. The double-arrow lines are bidirectional transitions in which left-hand side parameters represent leftward transition rates and right-hand side parameters rightward transition rates. The total number of states for a set of n processes is $2^n + 1$. This implies a quick expansion of state space as n increases, e.g., for 10 cooperating processes there are 1025 states.

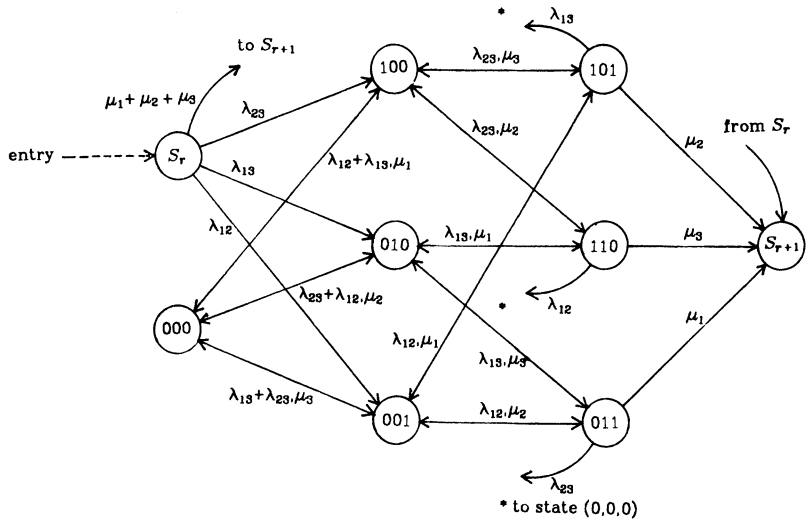


Fig. 2. The model of asynchronous RB's for three processes.

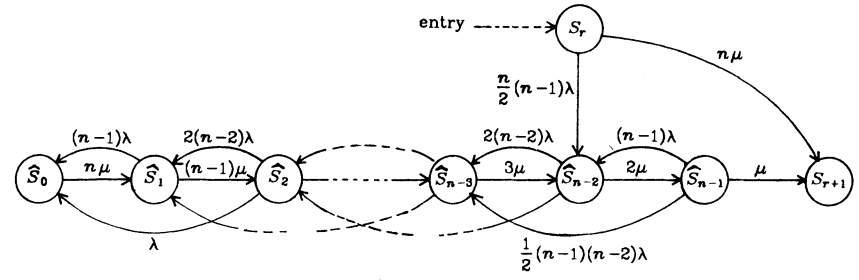


Fig. 3. The simplified model of asynchronous RB's for n processes.

However, the number of states can be reduced drastically when $\mu_i = \mu_j = \mu$ and $\lambda_{ij} = \lambda$ for all $i, j \in A$. In this case the reduction is achievable since all intermediate states $S = (x_1, x_2, \cdots, x_n)$ containing exactly u 1's in $(x_1, x_2, \cdots, x_n)$ can be replaced by a single state $\hat{S}_u$ where $u = 0, 1, 2, \cdots, n-1$. A simplified or reduced model is obtained under the following transition rules and presented in Fig. 3.

R1') For $u = 0, 1, \cdots, n-1$, the system will move to state $\hat{S}_{u+1}$ from state $\hat{S}_u$ with transition rate $(n-u)\mu$ when a new recovery point is formed.

R2') For all $u \geqslant 2$, the system is able to leave state $\hat{S}_u$ for state $\hat{S}_{u-2}$ with rate $(u(u-1)\lambda)/2$.

R3') For all $u \geqslant 1$, there is a transition from state $\hat{S}_u$ to state $\hat{S}_{u-1}$ with rate $u(n-u)\lambda$.

R4') The system can transfer directly from the entry state S_r to the terminal state S_{r+1} with transition rate $n\mu$.

C. The Analysis of Asynchronous Recovery Blocks

With the model developed above, we can characterize the behavior of asynchronous recovery blocks in terms of the degree of interprocess communications and the distribution of recovery points. With the exponentially distributed interprocess communications and recovery points, X_r becomes stochastically identical for all r. Let X denote a random variable representing the interval between two successive recovery lines, L_i the number of states saved in process P_i during interval X. The probability distribution of X and the mean value of L_i are derived below.

1) The distribution of X: Let the state space $\Psi = \{0, 1, 2, \cdots, m\}$ where $m = 2^n$ be the set of states of the preceding continuous-time Markov process with the following convention for numbering states:

a) $S_r \rightarrow$ state 0,
b) an intermediate state $(x_1, x_2, \cdots, x_n) \rightarrow$ state $(\sum_{i=1}^{n} \cdot x_i 2^{i-1} + 1)$, and
c) $S_{r+1} \rightarrow$ state m.

Then, the Chapman-Kolmogorov equation becomes

$$\frac{d}{dt}\pi(t) = \pi(t)H \tag{1}$$

where H is the $((m+1) \times (m+1))$ transition matrix $[h(u, v)]$ in which the (u, v) element is the transition rate from state u to state v, and $\pi(t)$ is a vector whose kth element is the probability that the system is in state k at time t. The initial condition is $\pi(0) = [1, 0, 0, \cdots, 0]$. The interval between two successive recovery lines X is equal to the time needed for transition from state 0 to state m. Therefore, the density function of X, namely $f_X(t)$, is given by

$$f_X(t) = \frac{d}{dt}\pi_m(t). \tag{2}$$

2) The mean value of L_i: Since we are only concerned with the number of recovery points established by process P_i during interval X, a discrete Markov chain is used. To compute the

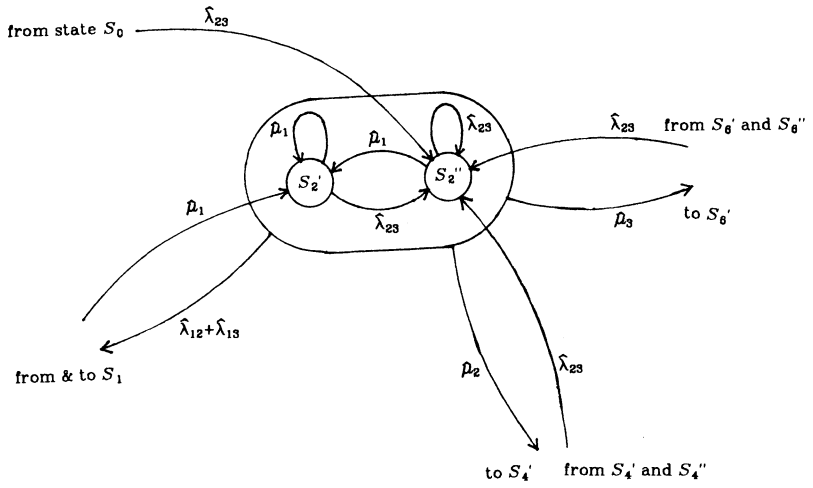


Fig. 4. The construction of states S_2' and S_2'' of discrete chain Y_d.

mean value of L_i,[2] a new Markov chain, denoted by Y_d, is constructed based on the previous model with the following two steps.

S1) Convert the previous model to a discrete model: The new chain Y_d has the same states as the previous Markov process. Let $G = \Sigma_{i=1}^{n} \Sigma_{j=1, j\neq i}^{n} \lambda_{ij} + \Sigma_{k=1}^{n} \mu_k$ be the normalization factor. The transition probability from state u to state v in Y_d is equal to: for $u, v = 0, 1, \cdots, m$, $p(u, v) = h(u, v)/G$ if $u \neq v$, and $p(u, u) = 1 - \Sigma_{v=1, v\neq u}^{n} p(u, v)$.

S2) Decompose states of discrete model: Arrivals at a state $S_u = (x_1, x_2, \cdots, x_i, \cdots, x_n)$ where $x_i = 1$ can be grouped into two classes. One is formed as a result of the occurrences of RP's in P_i and the other is formed as a result of interprocess communications and establishments of RP's in processes other other than P_i. Accordingly, the state $S_u = (x_1, x_2, \cdots, x_i, \cdots, x_n)$ with $x_i = 1$ can be split into two states S_u' and S_u'', representing the two classes, respectively. Both states have the same departure processes as that of S_u. However, all arrivals at state S_u due to the occurrence of recovery points in P_i enter state S_u' whereas all other transitions are made to S_u''. Hence the number of RP's associated with state S_u' is represented by that of arrivals at S_u'.

Fig. 4 shows the conversion and the split of state $S_2 = (1,0,0)$ of the Markov model for the three concurrent processes in Fig. 2. With the new discrete model, Y_d, we can calculate the mean number of visits to state S_u', denoted as $N_{S_u'}$, and the mean value of L_i using the following relationship:

$$E(L_i) = \sum_{S_u' \in \Psi_{Y_d}} E(N_{S_u'}) \tag{3}$$

where Ψ_{Y_d} is the state space of Y_d.

Suppose process P_i detects an error or fails the acceptance test at one of its recovery points RP_j^i where $j = 1, 2, \cdots, L_i$. The rollback of P_i may propagate to k processes in the process set, $\Omega_A = \{P_l | l \in A\}$ where $A = \{1, 2, \cdots, n\}$. Let D_j^k be the rollback distance associated with the k processes and RP_j^i for $j = 1, 2, \cdots, L_i$. Then, X represents the supremum of these

[2] it is possible to have $E(X) \neq 0$, but no recovery point within an inter-recovery line interval. Therefore, $E(L_i) \neq E(X)/\mu_i$.

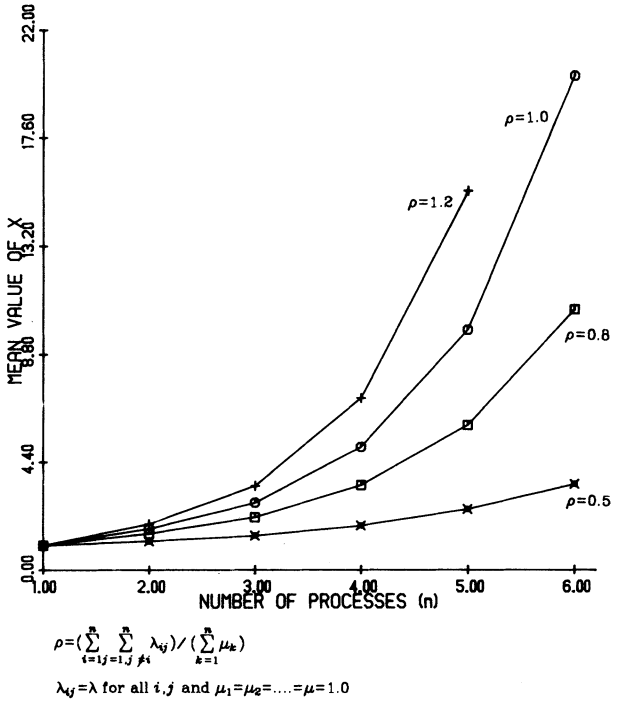


Fig. 5. Mean value of X versus the number of processes.

random variables, i.e., $D_{L_i}^n$. Let $\rho = (\Sigma_{i=1}^{n} \Sigma_{j=1, j\neq i}^{n} \lambda_{ij})/(\Sigma_{k=1}^{n} \mu_k)$ which represents the relative ratio between the density of interprocess communications and recovery point establishments. In Fig. 5, the mean values of X are plotted as a function of n for different values of ρ. It shows that X increases drastically when there is an increase in the number of processes involved in the rollback recovery. The denisty function of X, $f_X(t)$, is plotted in Fig. 6. For all the three cases in Fig. 6, there is a sharp pulse near $t = 0$, which is due to direct transitions between S_r and S_{r+1} and a longer transition time needed once the system enters intermediate states.

With a fixed value of ρ and varying values of μ's and λ's for three processes, we have performed computer simulation and the results are tabulated in Table I. The minima of X and L_i occur when the distribution of recovery points among these

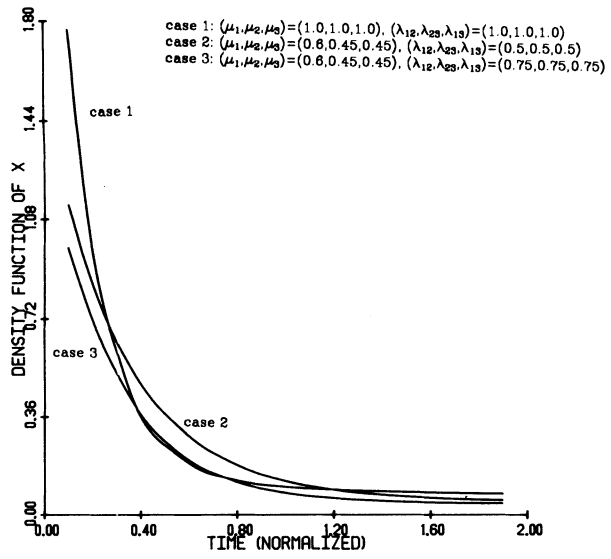


Fig. 6. The density function of X, $f_X(t)$.

TABLE I
MEAN VALUES OF X AND L_i FOR CONSTANT ρ

case	1	2	3	4	5
(μ_1,μ_2,μ_3)	(1.0,1.0,1.0)	(1.5,1.0,0.5)	(1.0,1.0,1.0)	(1.5,1.0,0.5)	(1.5,1.0,0.5)
$(\lambda_{12},\lambda_{23},\lambda_{13})$	(1.0,1.0,1.0)	(1.0,1.0,1.0)	(1.5,0.5,1.0)	(1.5,0.5,1.0)	(0.5,1.5,1.0)
$E(X)$	2.598	3.357	2.600	3.203	3.354
$E(L_1)$	2.500	4.847	2.453	4.533	4.967
$E(L_2)$	2.500	3.231	2.453	3.022	3.111
$E(L_3)$	2.500	1.616	2.453	1.511	1.656
$E(L_1+L_2+L_3)$	7.500	9.693	7.360	9.065	9.933

processes is uniformly balanced (i.e., $\mu_1 = \mu_2 = \mu_3$). The distribution of interprocess communications does play an important role in determining the probability of rollback propagation but has little effect on X and L_i once the set of processes involved in rollback recovery is determined.

III. Synchronized Recovery Blocks

The simplest way of avoiding unbounded rollback propagation is to synchronize the establishment of recovery points during process execution. In this method, interactions are inhibited between any pair of processes during their establishment of recovery points. There are three conceivable strategies in deciding when a synchronization request is to be issued: 1) at a constant interval, denoted as T_s; 2) when the time elapsed since the previous recovery line exceeds a specified value, T_s'; or 3) when the number of states saved after the previous recovery line is larger than a prespecified number M_s. The implementation of the first strategy is simple since the synchronization request is issued without any knowledge of the state of execution. Nevertheless, some synchronization requests may become redundant and unnecessary if they are issued immediatedly after the formation of recovery lines. For the second and third strategies, the rollback distance and the number of saved states are prevented from becoming too large. However, for these two strategies, additional overhead will be required because each process must be aware of the occurrence of a recovery line. Note that the conversation scheme is a special case of the third strategy where $M_s = 1$.

Upon the receipt of a synchronization request, every process has to prepare for establishing a recovery line and also has to wait for commitments (for establishing a recovery line) from other processes before it executes an acceptance test. Thus, all cooperating processes perform their acceptance tests at the same instant upon receiving the commitments from all other processes. Let P_{ij}-*ready* be the flags in process P_i to indicate commitments from P_j for $j = 1, 2, \cdots, n$. The steps for synchronization in each process P_i are described as follows:

1) **execute** "its own normal process" until "acceptance test;"
2) set P_{ii}-*ready* := ON and then broadcast P_{ii}-*ready*;
3) **while** not (all P_{ij}-*ready* = ON) **do**
   receive messages;
   if a message is P_{jj}-*ready* **then** set P_{ij}-*ready* := ON
   else record the message
4) **do** "acceptance test" and record process states.

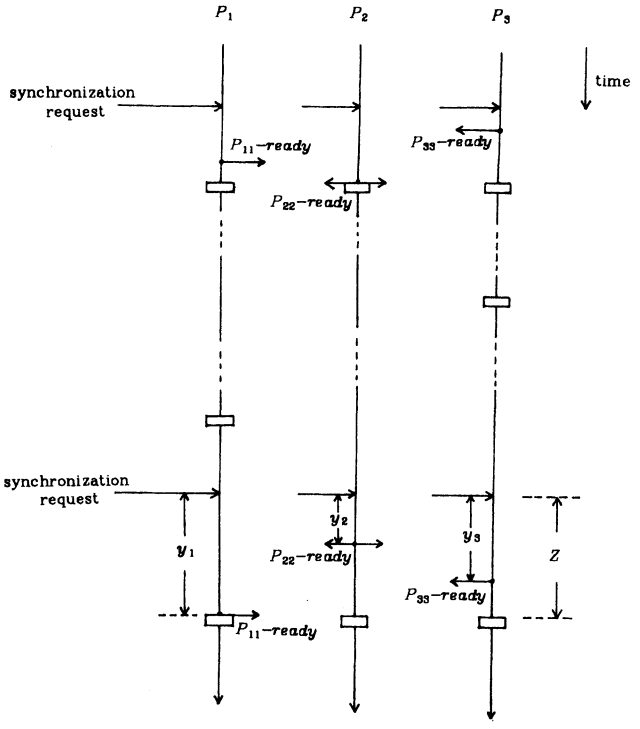


Fig. 7. Establishment of recovery lines upon synchronized requests.

Establishment of recovery lines upon synchronization requests is shown in Fig. 7. Synchronization causes the computation power to be reduced because processes have to wait for commitments (as in Step 3) from other processes. And, process autonomy—a principal characteristic of distributed computing systems—is sacrificed. Ley y_i be the interval between the receiving of a synchronization request and the moment that process P_i reaches its next acceptance test (in Step 1). Then, according to the assumptions in Section II-A, y_i is an exponentially distributed random variable with parameter μ_i. Let $Z = \max\{y_1, y_2, \cdots, y_n\}$. The total loss in computation power is $\mathrm{CL} = \Sigma_{i=1}^{n} (Z - y_i)$. The mean loss becomes

$$\overline{\mathrm{CL}} = n \int_0^\infty (1 - F_z(t))\, dt - \sum_{i=1}^{n} \frac{1}{\mu_i} \tag{4}$$

where $F_z(t)$ is the distribution function of Z and equals $\Pi_{i=1}^{n} \cdot (1 - e^{-\mu_i t})$.

The time interval between two successive recovery lines is a function of the strategy used for issuing synchronization requests as well as characteristics of the processes involved (e.g., patterns of interprocess communications and RP establishments). Let Z' and Z'' be random variables having the same distribution as $Z = \max\{y_1, y_2, \cdots, y_n\}$; then the maximum value of this time interval becomes $mT_s + Z' - Z''$ where m is the smallest integer i such that $Z'' < iT_s$, or $T_s' + Z'$. Observe that Z' and Z'' represent the amount of time required for a process to be ready for establishing an RP after it received a synchronization request. For the third strategy, the maximum number of rollback steps is M_s. Thus, the supremum of this time interval can be expressed as $\max\{z_1, z_2, \cdots, z_n\}$ where $z_i = \sum_{n=1}^{M_s} y_i$.

IV. Implantation of Pseudorecovery Points

In the construction of a recovery block, an acceptance test consists of a number of executable assessments provided by the programmer, followed by a state saving. Note that process states can also be recorded upon any other requests whenever they are considered useful in the rollback recovery. A *pseudorecovery point* (PRP) is defined as a recovery point that is established *without* a preceding acceptance test and is proposed here as an alternative for avoiding the domino effect in a set of cooperating concurrent processes. With a monitor as the interprocess communication means, Kim [15] and Kant and Silberschatz [16] discussed methods for implanting recovery points in a *central* manner. Similarly, we consider a method for implanting PRP's in the set of cooperating concurrent processes in a *decentralized* manner. Also, note that the use of PRP's does not require any particular interprocess communications mechanism (e.g., the implementation does not have to be based on monitors).

To make a recovery point RP_j^i in process P_i maximally useful for rollback error recovery, there should be corresponding recovery points in the other processes affected by the rollback propagation from P_i. If such recovery points do not actually exist, for a given RP_j^i in process P_i a pseudorecovery point $\mathrm{PRP}_j^{ii'}$ has to be inserted in process $P_{i'}$. Further, in order to avoid the need of tracing recovery points at that particular moment, for RP_j^i a PRP is established in each of the other processes involved. An algorithm for implanting PRP's is given below.

1) When P_i establishes a recovery point RP_j^i, it broadcasts a PRP implantation request to other processes.

2) If P_i receives the implantation request, it records its state as $\mathrm{PRP}_j^{ii'}$ upon the completion of the current instruction without an acceptance test. Then $P_{i'}$ broadcasts the commitment $C_{i'}$.

3) Every process executes its own normal task after it establishes RP_j^i or $\mathrm{PRP}_j^{ii'}$. However, the messages sent to a process by $P_{i'}$ prior to $C_{i'}$ have to be retained in the state saved.

Assume that process P_i detects an error at time t_d which is prior to the establishment of RP_{j+1}^i. If this error is local to P_i then the recovery line (called a *pseudorecovery line*, PRL_j^i) formed by RP_j^i and all $\mathrm{PRP}_j^{ii'}$'s is able to recover these processes even if the error has already propagated to other processes. However, when the error detected in P_i is due to error propagation from another process P_l (and therefore not local to P_i), the contents of PRP_j^{il} may have already been contaminated if this error occurred prior to establishing PRP_j^{il}. The restart from the pseudorecovery line formed by both RP_j^i and all $\mathrm{PRP}_j^{ii'}$'s may just reproduce the same error. Therefore, rollback propagation may continue until every process involved has rolled back to a pseudorecovery line, say $\mathrm{PRL}_k^{\hat{i}}$, for which all processes but $P_{\hat{i}}$ have passed at least one of their recovery points. Since there exists an $\mathrm{RP}_j^{i'}$ in $P_{i'}$ for $i' \neq \hat{i}$ between PRL_k^i and t_d, every state belonging to $\mathrm{PRL}_k^{\hat{i}}$ is now guaranteed to contain correct information of the corresponding process.[3] Also, note that this pseudorecovery line renders the shortest rollback distance for backward error recovery in case forced synchronization is not used. An algorithm of rollback recovery with these pseudorecovery points is given by:

1′) If an error is found in process P_i, set $p := i$ where p is a rollback pointer.

2′) P_p rolls back to its previous recovery point RP_j^p. All processes $P_{i'}$ affected by the rollback of P_p roll back to their respective pseudorecovery points $\mathrm{PRP}_j^{pi'}$.

3′) For every affected process P_i', if the rollback has not passed its most recent recovery point, then set $p := i'$ and go back to step 2.

In Fig. 8, the establishment of PRP's in processes P_1, P_2, and P_3 is illustrated. When P_3 fails its acceptance test AT_2^3, all processes have to restart from the pseudorecovery line formed by $(\mathrm{RP}_1^1, \mathrm{PRP}_1^{12}, \mathrm{PRP}_1^{13})$ if P_1 and P_2 are affected by the rollback of P_3.

In the above algorithm, we can find that every process needs to preserve a recovery point for restart in case it fails. Also $(n-1)$ pseudorecovery points are needed for a process to form a pseudorecovery line with other processes where n is the total number of concurrent processes. It is therefore required to save n states for every RP, i.e., one RP and $(n-1)$ PRP's, and all old RP's and PRP's except those in the pseudorecovery lines $\{\mathrm{PRL}_j^i \mid i = 1, \cdots, n$, and RP_j^i is the most recent RP in $P_i\}$ can be purged when a new recovery point is established, thereby reducing storage requirements for saving RP's and PRP's. Note that rollback distance is bounded by the supremum of $\{y_1, y_2, \cdots, y_n\}$ where y_i is the interval between two successive recovery points of process P_i. The additional time overhead for every recovery point is $(n-1)\,t_r$ where t_r is the time needed to record the process state. These overheads should be assessed against the gain of process autonomy and avoidance of unbounded rollback propagations.

V. Conclusion

We have quantitatively evaluated three different recovery blocks employed in backward error recovery for concurrent processing. The recovery block dealt with in this paper is defined in software and includes an acceptance test and a state saving. The environment of concurrent processing considered

[3] If the state saved at $\mathrm{PRP}_k^{\hat{i}i'}$ were contaminated, then the error should have been detected at the subsequent recovery point, $\mathrm{RP}_j^{i'}$. Meanwhile, the state saved at $\mathrm{RP}_k^{\hat{i}}$ is correct by the assumption of perfect local acceptance test.

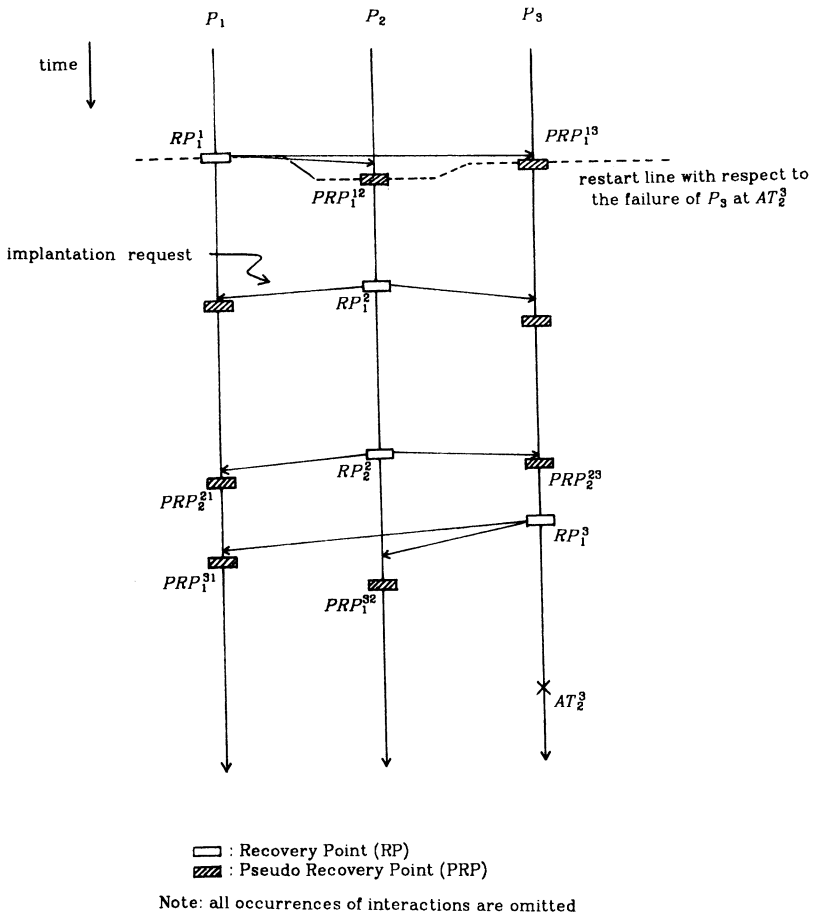


Fig. 8. Establishment of pseudorecovery points for rollback error recovery.

here is not restricted to any particular method of interprocess communications or system structure. However, the occurrence of recovery points and interprocess interactions are assumed to follow exponential distributions. This is principally for tractability: nonexponential interaction patterns would be extremely difficult to analyze if not impossible.

In this paper, we have considered the distribution of the interval between two successive recovery lines instead of the actual rollback distance. The rollback distance after an error is detected is related to the probability of error occurrence, error detection, rollback propagation, etc. However, the interval X does represent an upper bound for the real rollback distance. We have also estimated the overhead required to avoid the domino effect when recovery or pseudorecovery points are employed. For both the synchronization method and the implantation of pseudorecovery points, the overheads are largely related to the construction of synchronization, RP's and PRP's. They would become unacceptably inefficient when synchronizations and pseudorecovery points are constructed frequently but interprocess communications do rarely occur. At the other extreme, i.e., asynchronous recovery blocks, it may result in a longer rollback distance due to unlimited rollback propagations (in place of synchronization and PRP insertion overheads).

To select a suitable strategy or a combination of these three methods, we have to first examine the properties of concurrent processes such as the amount of interprocess communication and the distribution of recovery points. Then, we weigh the tradeoffs between the loss of computation power during normal operation and the increase in response time due to rollback error recovery. For instance, the asynchronous method or a longer sychronization period is not acceptable for time-critical tasks in which a delay in system response beyond a certain value, the *system deadline*, leads to a catastrophic failure. The implantation of pseudorecovery points is also inefficient for concurrent processes when they establish recovery points frequently (thus requiring many PRP's to be implanted) and rarely communicate with each other. In general, if more knowledge of the execution state in concurrent processes can be obtained, a better strategy for implementing recovery blocks can be derived.

Acknowledgment

The authors are grateful to R. Butler and M. Holt at NASA Langley Research Center for both financial and technical support and C. M. Krishna at The University of Michigan for technical discussions.

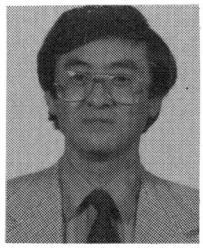
References

[1] J. Horning, H. C. Lauer, P. M. Melliar-Smith, and B. Randell, "A program structure for error detection and recovery," in *Lecture Notes in Computer Science*, vol. 16. New York: Springer-Verlag, 1974, pp. 171–187.

[2] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 220–232, June 1975.

[3] B. Randell, P. A. Lee, and P. C. Treleaven, "Reliability issues in

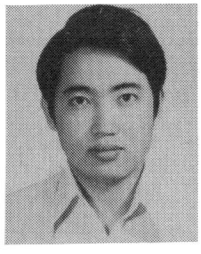
computing system design," *Comput. Surveys*, vol. 10, no. 2, pp. 123-165, June 1978.
[4] Y. H. Lee and K. G. Shin, "Rollback propagation detection and performance evaluation of *FTMR*^{2}*M*–A fault-tolerant multiprocessor," in *Proc. Int. Symp. Computer Architecture*, 1982, pp. 171-180.
[5] Y. H. Lee and K. G. Shin, "Design and evaluation of fault-tolerant multiprocessor using hardware recovery blocks," Comput. Res. Lab., Dep. Electrical and Computer Eng., Univ. Michigan, 1982, Tech. Rep. CRL-TR-6-82; see also *IEEE Trans. Comput.*, vol. C-33, no. 2, pp. 113-124, Feb. 1984.
[6] P. M. Merlin and B. Randell, "State restoration in distributed systems," in *Proc. 8th Int. Conf. Fault-Tolerant Computing*, 1978, pp. 129-134.
[7] W. G. Wood, "A decentralized recovery control protocol," in *Proc. 11th Int. Conf. Fault-Tolerant Computing*, 1981, pp. 159-164.
[8] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic recovery schemes for distributed processes," in *Proc. Reliability in Distributed Software and Database Systems*, 1981, pp. 124-130.
[9] K. H. Kim, "Approaches to mechanizations of the conversation scheme based on monitors," *IEEE Trans. Software Eng.*, vol. SE-8, no. 3, pp. 189-197, May 1982.
[10] J. N. Gray, "Notes on database operating systems," in *Operating Systems: An Advanced Course*, R. Bayer *et al.*, Eds. New York: Springer-Verlag, 1979, pp. 393-481.
[11] W. H. Kohler, "A survey of techniques for synchronization and recovery in decentralized computer systems," *Comput. Surveys*, vol. 13, no. 2, pp. 149-183, June 1981.
[12] G. Ferran, "Distributed checkpointing in a distributed data management system," in *Proc. Real Time Systems Symp.*, 1981, pp. 43-49.
[13] D. L. Russell, "Process backup in producer-consumer systems," in *Proc. 6th ACM Symp. Operating System Principles*, Nov. 1977, pp. 151-157.
[14] ——, "State restoration in systems of communicating processes," *IEEE Trans. Software Eng.*, vol. SE-6, no. 2, pp. 183-194, Mar. 1980.
[15] K. H. Kim, "An approach to programmer-transparent coordination of recovering parallel processes and its efficient implementation rules," in *Proc. Int. Conf. Parallel Processing*, 1978, pp. 58-68.
[16] K. Kant and A. Silberschatz, "Error recovery in concurrent processes," in *Proc. COMPSAC*, 1980, pp. 608-614.



Kang G. Shin (S'75-M'78-SM'83) was born in the province of Choongbuk, Korea, on October 20, 1946. He received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

From 1970 to 1972 he served in the Korean Army as an ROTC officer and from 1972 to 1974 he was on the research staff of the Korea Institute of Science and Technology, Seoul, working on the design of VHF/UHF communication systems. From 1974 to 1978 he was a Teaching/Reserach Assistant and then an Instructor in the School of Electrical Engineering, Cornell University. From 1978 to 1982 he was an Assistant Professor at Rensselaer Polytechnic Institute, Troy, NY. He was also a visiting scientist at the U.S. Air Force Flight Dynamics Laboratory in the summer of 1979 and at Bell Laboratories, Holmdel, NJ, in the summer of 1980 where his work was concerned with distributed airborne computing and cache memory architecture, respectively. He also taught short courses for the IBM Computer Science Series in the area of computer architecture. Since September 1982, he has been with the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, MI, where he is currently an Associate Professor. His current teaching and research interests are in the areas of distributed and fault-tolerant computing, computer architecture, and robotics and automation.

Dr. Shin is a member of the Association for Computing Machinery, Sigma Xi, and Phi Kappa Phi.



Yann-Hang Lee received the B.S. degree in engineering science and the M.S. degree in electrical engineering from National Cheng Kung University, Taiwan, R.O.C., in 1973 and 1978, respectively.

Currently he is working towards the Ph.D. degree in computer, information and control engineering at the University of Michigan, Ann Arbor, MI. His research interests include distributed computer systems, multiprocessor, performance evaluation, and fault-tolerant computing.