THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

ANALYSIS OF BACKWARD ERROR RECOVERY FOR CONCURRENT PROCESSES WITH RECOVERY BLOCKS

K.G. Shin
Y-H. Lee

CRL-TR-9-83

FEBRUARY 1983

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]This work was supported in part by NASA Grant No. NAG 1-296. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

ABSTRACT

Although backward error recovery with recovery blocks(RB's) has received considerable attention from many researchers, no attempt has been made to structure its implementation alternatives and then to evaluate/analyze their effectiveness. In this report we consider three different methods of implementing RB's. These are the asynchronous, synchronous, and the pseudo recovery 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 becomes a serious problem.

In order to completely avoid unbounded rollback, 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 the commitment to establishing a recovery line, leading to inefficiency in time utilization.

As a compromise between asynchronous and synchronous RB's, we propose to insert pseudo recovery points so that unbounded rollback may be avoided while maintaining process autonomy.

We have 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 (i) the interval between two successive recovery lines for asynchronous RB's, (ii) mean loss in computation power for the synchronized method, and (iii) additional overhead and rollback distance in case PRP's are used.

TABLE OF CONTENTS

ANALYSIS OF BACKWARD ERROR RECOVERY FOR CONCURRENT PROCESSES WITH RECOVERY BLOCKS[1]

Kang G. Shin and Yann-Hang Lee

Computing Research Laboratory
Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor, Michigan 48109

ABSTRACT

Although backward error recovery with recovery blocks(RB's) has received considerable attention from many researchers, no attempt has been made to structure its implementation alternatives and then to evaluate/analyze their effectiveness. In this report we consider three different methods of implementing RB's. These are the asynchronous, synchronous, and the pseudo recovery 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 becomes a serious problem.

In order to completely avoid unbounded rollback, 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 the commitment to establishing a recovery line, leading to inefficiency in time utilization.

As a compromise between asynchronous and synchronous RB's, we propose to insert pseudo recovery points so that unbounded rollback may be avoided while maintaining process autonomy.

We have 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 (i) the interval between two successive recovery lines for asynchronous RB's, (ii) mean loss in computation power for the synchronized method, and (iii) additional overhead and rollback distance in case PRP's are used.

[1] *The work reported here is supported in part by National Aeronautics and Space Administration Grant No. NAG 1-296. Any opinions, findings, and conclusions or recommendations in this publication are those of the authors and do not necessarily reflect the view of NASA.*

1. INTRODUCTION

Recent advances in VLSI and communication network technologies have made distributed processing feasible. While distributed processing can theoretically be exploited to provide computation speedup, cost-effectiveness and tolerance of component failure, several problems remain to be solved before its full potential can be realized in practice. In this report, we consider one such problem: that of implementing backward error recovery for concurrent processes with recovery blocks.

The best known technique of backward error recovery, the *recovery block* (RB), was proposed by Horning [1] and Randell [2]. It is a sequential program structure that consists of an acceptance test, a recovery point(RP), and alternative algorithms for a given process. A process saves its state at its recovery point and then enters a recovery region. At the end of a recovery block, the acceptance test is executed 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.

Unfortunately, however, for cooperating concurrent processes the rollback of a process may cause other processes to roll back(this phenomenon is called *rollback propagation*) because of process interactions and imperfect checking of global correctness. Moreover, rollback may propagate to further RP's since recovery points of individual processes may not provide a globally consistent state for all processes involved. This rollback propagation continues until it reaches a *recovery line* at which a globally consistent state does 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 error occurrence.

A detailed description of the domino effect can be found in [3]. For convenience let us consider Figure 1 to visualize rollback propagation. Process P_1 begins to roll back because of unsuccessful acceptance test AT_4^1. This rollback propagates to the other two processes P_2 and P_3. Eventually, the whole system has to restart from recovery line RL_2 and the computation done between RL_2 and AT_4^1 has to be discarded. The interval between the restart point 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 recovery.

The domino effect is the major obstacle in implementing the recovery block scheme for concurrent processes. The designer is able to predict neither the time of the occurrence of process interactions nor that of the appearance of recovery lines. Nonetheless, it is not desirable to randomly place recovery points and acceptance tests without considering process characteristics. Otherwise, it is possible to have a disaster such as unbounded rollback propagations, a large rollback distance, and a great number of largely useless recovery points occupying large amounts of memory space, etc. Furthermore, decision on rollback propagation and determination of recovery lines will become more complex though they can be made in a centralized [4,5] or decentralized manner [6,7,8].

Several refinements have been proposed to overcome the drawbacks in this 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 [10,11,12]. 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 report we propose to employ *pseudo recovery points*[2] (PRP's) to alleviate the rollback propagation problem by allowing a process to restart at a PRP in case the process is forced to roll back by others as a result of rollback propagation. Therefore, we can classify these refinements into two categories, *synchronized recovery blocks* and *pseudo recovery points*, providing a contrast with the third category called *asynchronous recovery blocks.*

To implement the rollback recovery schemes, we have to consider various trade-offs between these three categories and the characteristics of concurrent processes. A satisfactory compromise should include an acceptable delay in process completion due to rollbacks, the preservation of autonomy for each process, and programmer transparency. Therefore, optimal solutions may be a combination of these three categories. A quantitative analysis is necessary to justify the solutions. For example, it is necessary to determine the mean amount of computation undone in case 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 execution proceeds stochastically, accurate modelling is difficult.

In this report, employing standard assumptions in computer performance analysis, we have developed a model to quantitatively describe the characteristics of rollback recovery schemes as well as their effectiveness. In the following sec-

[2] *We call it a pseudo recovery 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 can not be used to recover a failed process.*

tion, 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 3 and 4, the synchronization method and the implantation of pseudo recovery points are evaluated respectively. The report concludes with Section 5.

2. EVALUATION OF ASYNCHRONOUS RECOVERY BLOCKS

Let us consider the history diagram in Figure 1 to illustrate the activities of cooperating concurrent processes P_i, $i=1,2,...n$. 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 $A \subset \{1,...,n\}$, i.e. a subset of concurrent processes. 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 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.

2.1. Modeling Assumptions

We make the following assumptions in our subsequent analyses.

1. *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 there were no processes to interfere with. Thus, a process is executing 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.

2. *Perfect 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 the 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.

3. *Probability Distribution of Interactions:* Usually, process behavior is modeled as an ordered sequence which in turn is specified by the program and 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. Nontheless, for both tractability and simplicity we have adopted here constant reference rates in the multiprocessor and 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 $\lambda_{ij}=\lambda_{ji}$ for all $i,j=1,2,...,n$ and $i \neq j$.

4. *Consistent Communications:* Let two messages m_a and m_b be sent from P_i to P_j. Consistent communications should satisfy : (i) every message sent from P_i to P_j will be received eventually by P_j, and (ii) $m_a$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.

5. *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 process P_i.

2.2. A Model for Asynchronous Recovery Blocks

Since individual recovery points by themselves may not be sufficient in rollback recovery due to the possibility of unbounded rollback propagations, we consider in this report 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, for $i=1,2,\ldots n$, can be stated as follows:

1. Each recovery line has to include one recovery point RP_j^i for every process P_i.

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

The basic idea underlying the model is to trace the occurrence of both recovery points and interactions. Based on the assumptions in Section 2.1, 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 $A = \{1,2,\ldots,n\}$, 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 $S = (x_1, x_2, \ldots, x_n)$, 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 $(1,1,\ldots,1)$. We can establish the following transition rules:

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

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

R3. The system arrives at state $(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n)$ from state $(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n)$ with transition rate $\sum_{j \in B_i} \lambda_{ij}$ 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 $\sum_{k=1}^{n} \mu_k$.

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 number of states for a set of n processes is 2^n+1.

When $\mu_i=\mu_j=\mu$ and $\lambda_{ij}=\lambda$ for all i, $j \in A$, the model can be simplified since all intermediate states $S=(x_1,x_2, \ldots, x_n)$ containing exactly u 1's in $(x_1,x_2, \ldots, x_n)$ can be replaced by a single state $\hat{S}_u$. A simplified model is obtained under the following transition rules and presented in Fig. 3.

R1'. For $u = 0,1,\ldots,n-1$, 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 $u \geq 2$, the system is able to leave state $\hat{S}_u$
for state $\hat{S}_{u-2}$ with rate $\frac{u(u-1)\lambda}{2}$.

R3'. For all $u \geq 1$, 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 $n\mu$.

2.3. 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 for all r becomes stochastically identical. 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.

A. The distribution of X

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

(a). S_r--> state 0,

(b). an intermediate state $(x_1,x_2,\ldots,x_n)$ --> state $(\sum_{i=1}^{n} x_i 2^{i-1}+1)$, and

(c). S_{r+1} --> state m.

Then, the Chapman-Kolmogorov equation becomes

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

where H is the $(m\times m)$ 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 $\pi(0)=[1,0,0\ldots,0]$. 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

$$f_x(t)=\frac{d}{dt}\pi_m(t)$$

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

(a). Convert the previous model to a discrete model:

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

(b). Arrivals at a state $S_u = (x_1,x_2,\ldots,x_i,\ldots,x_n)$ where $x_i=1$ can be grouped 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 than P_i. Accordingly, the state $S_u=(x_1,x_2,\ldots,x_{i,\ldots,}x_n)$ 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'.

Figure 4 shows the conversion and the split of state $S_2 = (1,0,0)$ of the Markov chain for the three concurrent processes in Figure 2. With the new discrete model, Y_d, we can calculate the 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:

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

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 $j=1,2,\ldots,L_i$. The rollback of P_i may propagate to k processes in the process set, $\Omega_A = \{P_l \mid l \in A\}$ where $A=\{1,2,\ldots,n\}$. Let D_j^k be the rollback distance associated with the k processes and RP_j^i for $j=1,2,..,L_i$. Then, X represents the supremum of these random variables, i.e., $D_{L_i}^r$. In Figure 5, the mean values of X are plotted as a function of n. It shows that X increases drastically when there is an increase in the number of processes involved in the rollback recovery. The density function of X, $f_x(t)$, is plotted in Figure 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.

Let $\rho = (\sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \lambda_{ij})/(\sum_{k=1}^{n} \mu_k)$ which represents the relative ratio between the density of interprocess communications and recovery point establishments. 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 1. The minima of X and L_i occur when the distribution of recovery points among these 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.

3. SYNCHRONIZED RECOVERY BLOCKS

The simplest way of avoiding unbounded rollback propagations 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; (2) when the time elapsed since the previous recovery line exceeds a specified value; or (3) when the number of states saved after the previous recovery line is larger than a prespecified number. The implementation of the first strategy is simple since the synchronization request is issued without any knowledge of the state of execution.

Nevertheless, this strategy may become very inefficient since it is possible to make synchronization requests immediately after the formation of recovery lines. For the second and third strategies, rollback distance and the number of saved states are prevented from becoming too large. However, in this case each process must be aware of the occurrence of a recovery line whenever it is established.

Upon the receipt of a synchronization request, every process has to prepare for establishing a recovery line and also has to wait for the commitment (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 $P_{ij}-ready$, for $j=1,2,\ldots,n$, be the flags in process P_i to indicate commitments from P_j. 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.
```

Establishment of recovery lines upon synchronization requests is shown in Figure 7. Synchronization causes the computation power to be diminished because processes have to wait for the commitment (as in step 3). Let 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 2.1, y_i is an exponentially distributed random variable with parameter μ_i.

Let $Z=\max\{y_1, y_2, \ldots, y_n\}$. The total loss in computation power is $CL=\sum_{i=1}^{n}(Z-y_i)$. The mean loss becomes

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

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

4. IMPLANTATION OF PSEUDO RECOVERY POINTS

In the construction of a recovery block, usually, an acceptance test is a number of executable assessments provided by the programmer and then followed by a state saving. Note that process states can also be recorded upon any other requests if they are considered useful in the rollback recovery. A *pseudo recovery 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.

To make every 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 that have to roll back as a result of the rollback propagation from P_i. If such recovery points do not actually exist, a pseudo recovery point, $PRP_j^{ii'}$, has to be inserted in process $P_{i'}$ for a given RP_j^i in process P_i. Further, in order to avoid the need of tracing recovery points at that particular moment, a PRP is established in each of the other processes involved for RP_j^i. 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 $PRP_j^{ii'}$ 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 $PRP_j^{ii'}$. 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 before establishing RP_{j+1}^i and that this error is local to P_i. The recovery line (called a *pseudo recovery line*, PRL_j^i) formed by RP_j^i and all $PRP_j^{ii'}$'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 pseudo recovery line formed by both RP_j^i and all $PRP_j^{ii'}$'s may just reproduce the same error. Therefore, rollback propagation may continue until every process involved has rolled back to a pseudo recovery line past at least one of its recovery points. Most of the processes involved are assured to reach the pseudo recovery line by rolling back past only one recovery point. A few processes may have to roll back past more than one RP due to random interprocesses interactions, and this can not be avoided unless a forced synchronization is employed as discussed in Section 3. Consequently, the pseudo recovery line allows the processes to have the shortest rollback distance for backward error recovery without synchronization. Note that the pseudo recovery line is now guaranteed to contain correct states of all concerned processes. An algorithm of rollback recovery with these pseudo recovery 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 pseudo recovery points $PRP_j^{pi'}$.

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

In Figure 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 pseudo recovery line formed by $(RP_1^1, PRP_1^{12}, PRP_1^{13})$ 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 $(n-1)$ pseudo recovery points are needed for a process to form a pseudo recovery 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 pseudo recovery lines $\{ PRL_j^i \mid i = 1,\ldots,n$, 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 $\{y_1, y_2, \ldots, y_n\}$ 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.

5. CONCLUSION

We have quantitatively evaluated three different recovery blocks employed in backward error recovery for concurrent processing. The recovery block dealt with in this report is defined in software and comprises an acceptance test and a state saving. The environment of concurrent processing considered here is not restricted to any particular method of interprocess communications or system structure.

We have estimated the overhead required to avoid the domino effect when recovery or pseudo recovery points are employed. For both the synchronization method and the implantation of pseudo recovery points, the overheads are largely related to the construction of synchronization, RP's and PRP's. They would become an unacceptable burden when synchronizations and pseudo recovery points are constructed frequently but interprocess communications 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).

In this report, 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, and rollback propagation, etc. However, the interval X does represent an upper bound for the real rollback distance.

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 communications and the distribution of recovery points. Then, we weigh the trade-off between the loss of computation power during normal operation and the increase in response time due to rollback recovery. For instance, the asynchronous method or a longer synchronization 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 pseudo recovery 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.

ACKNOWLEDGEMENT

The authors are grateful to Rick Butler and Milton 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," *Lecture Notes in Computer Science*, Vol. 16, Springer-Verlag, 1974, pp. 171-187.

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

[3]. B. Randell, P. A. Lee and P. C. Treleaven, "Reliability issues in computing system design," *Computing Surveys*, Vol. 10, No. 2, June 1978, pp. 123-165.

[4]. Y. H. Lee and K. G. Shin, "Rollback propagation detection and performance evaluation of $FTMR^2M$ - a fault-tolerant multiprocessor," *Proc. of Int'l Symp. on Computer Architecture*, 1982, pp. 171-180.

[5]. Y. H. Lee and K. G. Shin, "Design and evaluation of fault-tolerant multiprocessor using hardware recovery block," *Technical Report*, Comp. Res. Lab., Dept. of Electrical and Computer Eng., The Univ. of Michigan, CRL-TR-6-82, 1982.

[6]. P. M. Merlin and B. Randell, "State restoration in distributed systems," *Proc. of 8-th Int'l Conf. on Fault-Tolerant Computing*, 1978, pp. 129-134.

[7]. W. G. Wood, "A decentralized recovery control protocol," *Proc. of 11-th Int'l Conf. on Fault-Tolerant Computing*, 1981, pp. 159-164.

[8]. K. Tsuruoka, A. Kaneko and Y. Nishihara, "Dynamic recovery schemes for distributed processes," *Proc. of 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. on Software Eng.*, Vol. SE-8, No.3, May 1982, pp. 189-197.

[10]. J. N. Gray, "Notes on database operating systems, " *Operating Systems: A advanced course*, edited by R. Bayer, et al., Springer-Verlag, 1979, pp.393-481.

[11]. W. H. Kohler, "A survey of techniques for synchronization and recovery in decentralized computer systems," *Computing Surveys*, Vol. 13, No. 2, June 1981, pp. 149-183.

[12]. G. Ferran, "Distributed checkpointing in a distributed data management system," *Proc. of Real Time Systems Symp.*, 1981, pp. 43-49.

[13]. D. L. Russell, "Process backup in producer-consumer systems," *Proc. of 6th ACM Symposium on Operating System Principles*, Nov. 1977, pp. 151-157.

[14]. D. L. Russell, "State restoration in systems of communicating processes," *IEEE Trans. on Software Eng.*, Vol. SE-6, No. 2, March 1980, pp. 183-194.

[15]. K. H. Kim, "An approach to programmer-transparent coordination of recovering parallel processes and its efficient implementation rules," *Proc. of Int'l Conf. on Parallel Processing*, 1978, pp. 58-68.

[16]. K. Kant and A. Silberschatz, "Error recovery in concurrent processes," *Proc. of COMPSAC*, 1980, pp. 608-614.

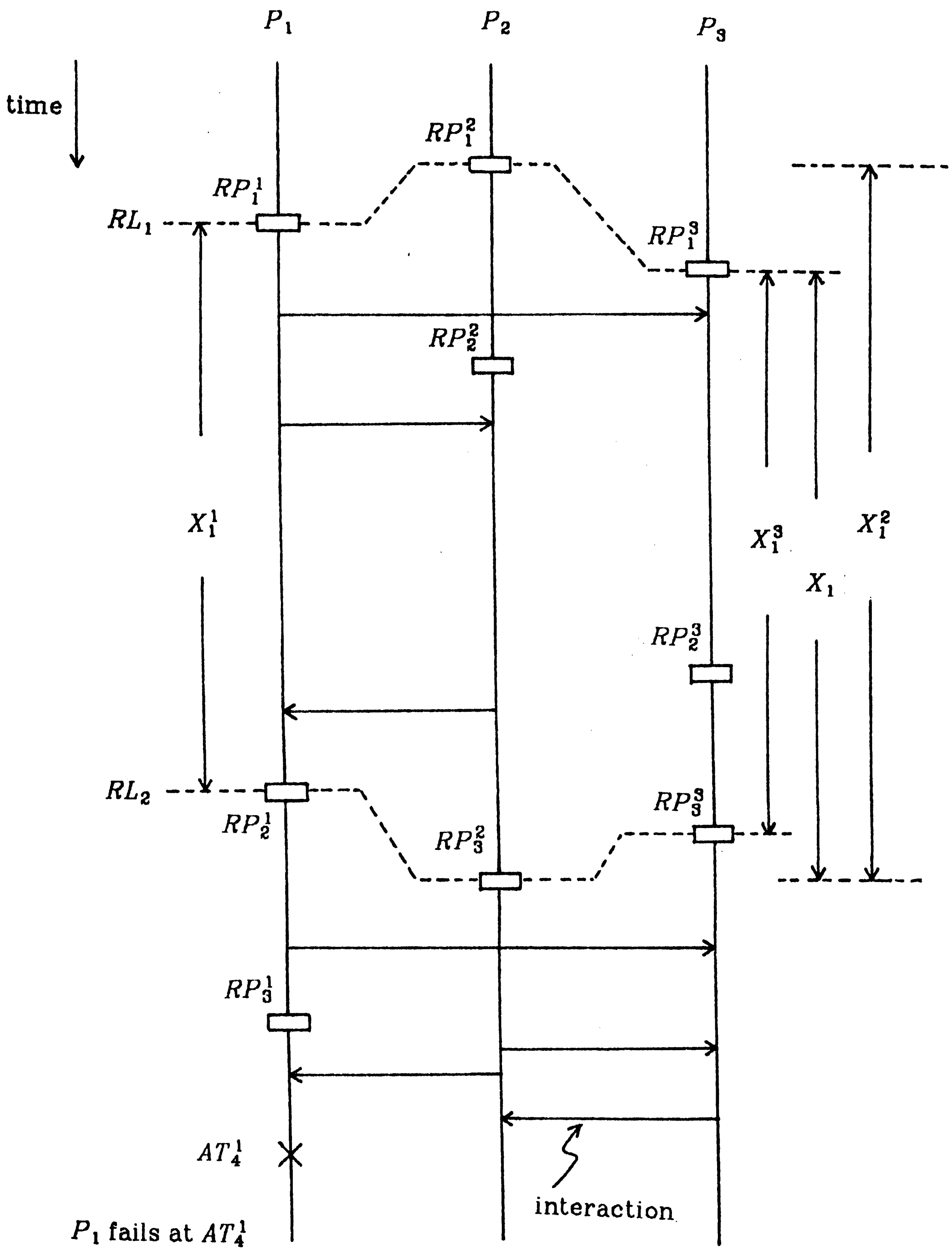


Figure 1. A History Diagram of Occurrence of Interactions and Recovery Points

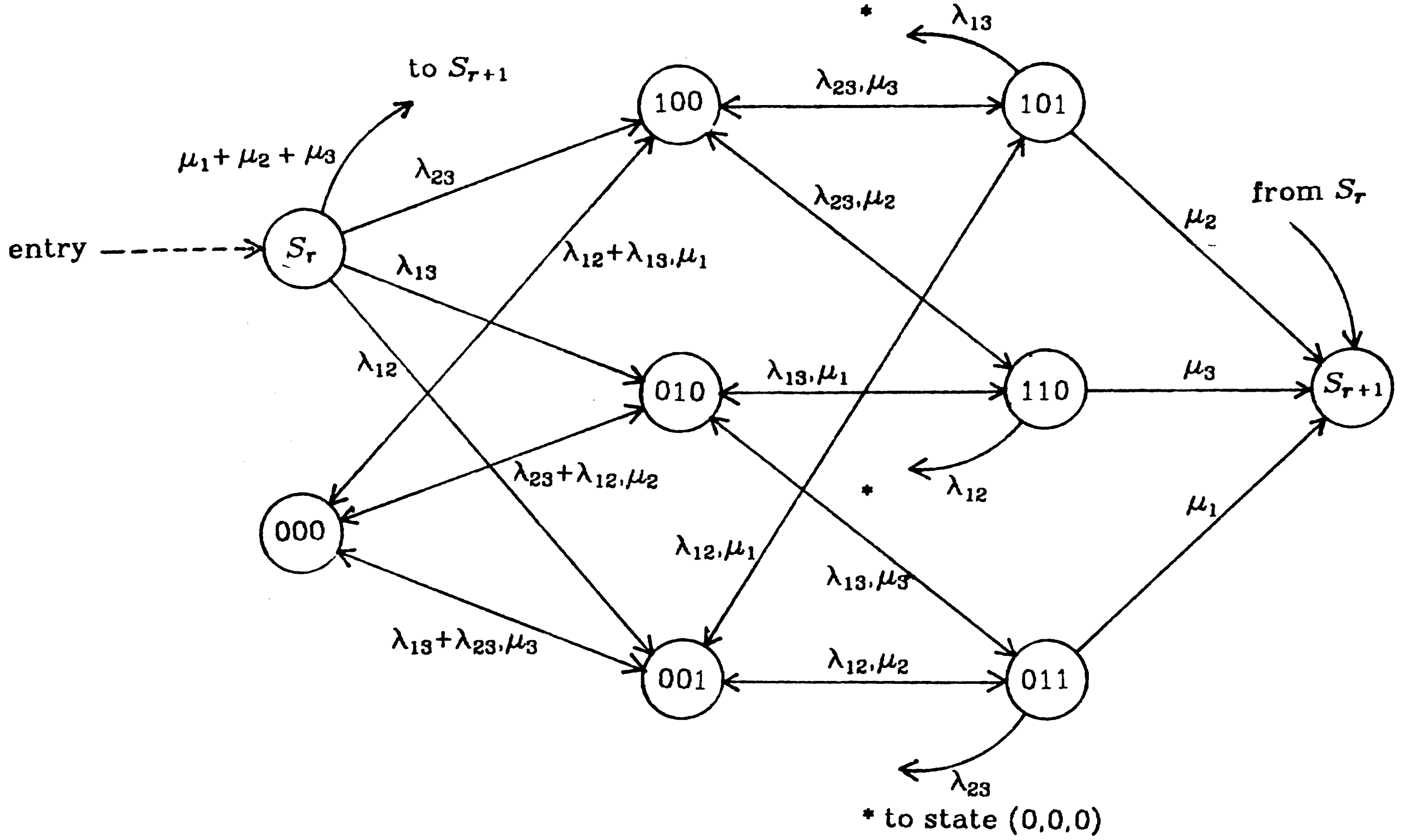


Figure 2. The Model of Asynchronous RB's for 3 Processes

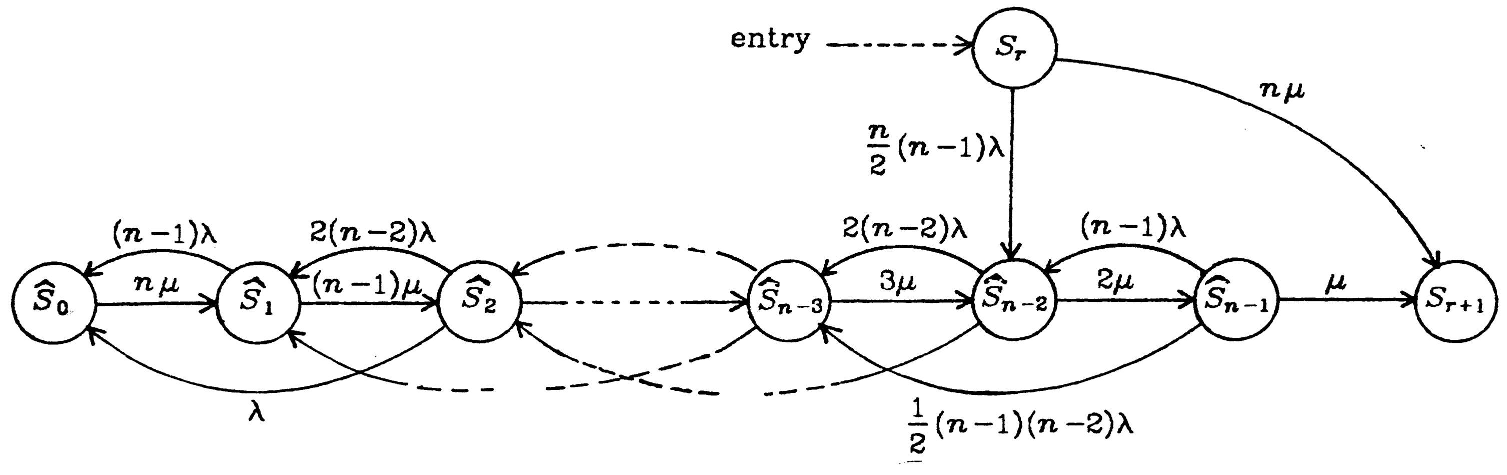
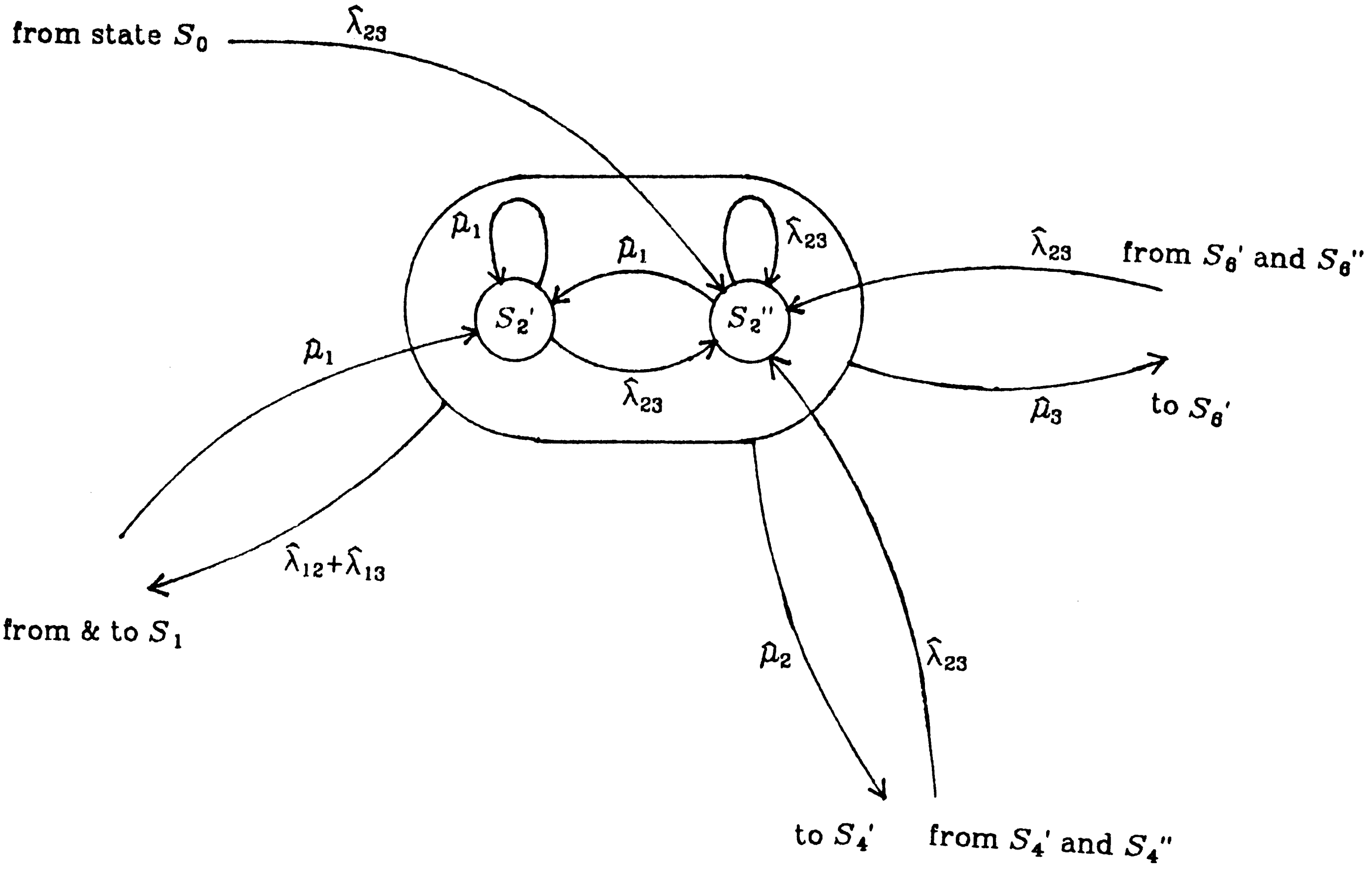
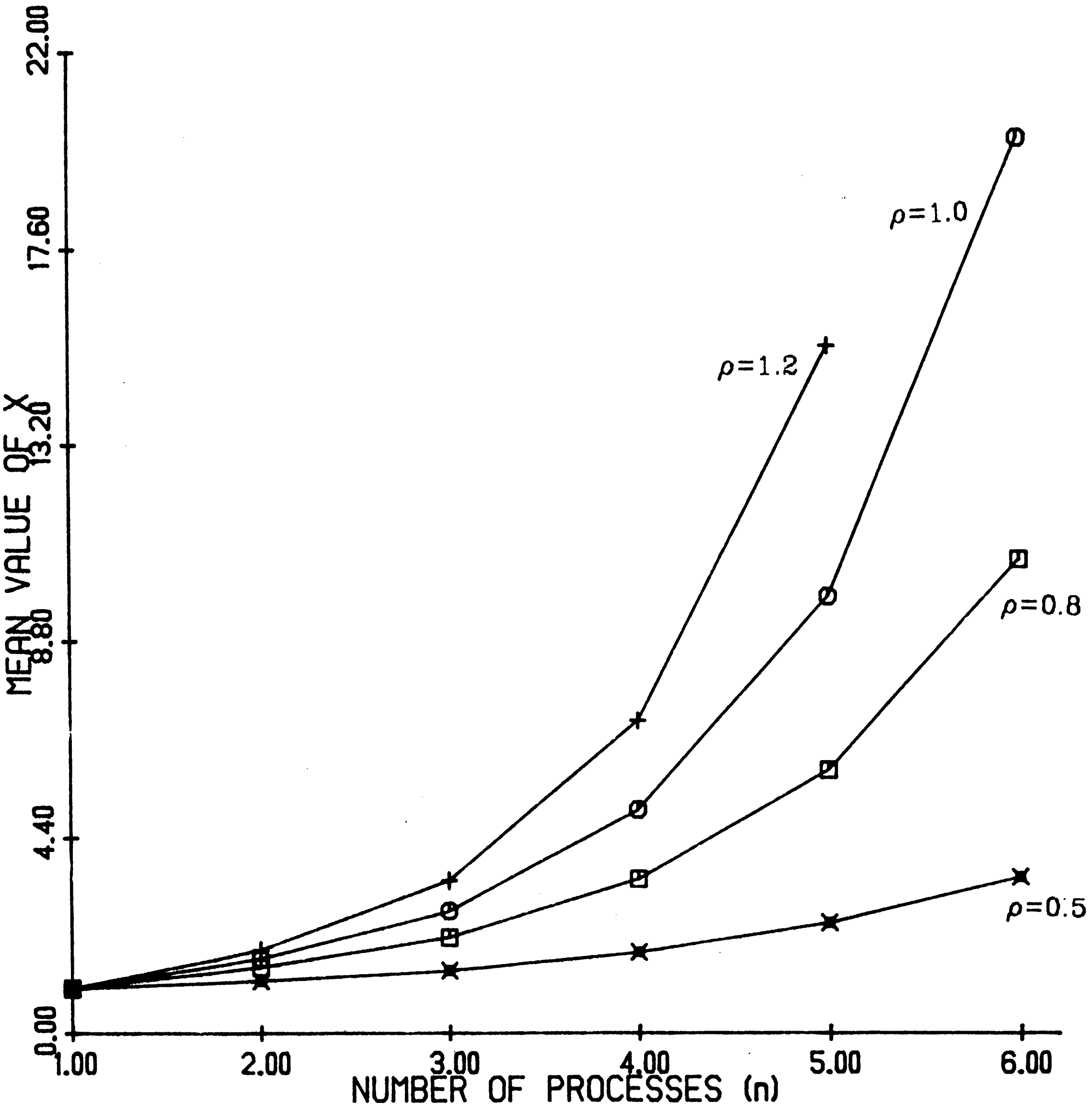


Figure 3. The Simplified Model of Asynchronous RB's for n Processes



$$\hat{\lambda}_{ij} = \frac{\lambda_{ij}}{G},\ \hat{\mu}_k = \frac{\mu_k}{G},\ \text{and}\ G = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \lambda_{ij} + \sum_{k=1}^{n} \mu_k$$

Figure 4. The Construction of State S_2' and S_2'' of Discrete Markov Chain Y_d



$\rho=(\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\lambda_{ij})/(\sum_{k=1}^{n}\mu_k)$

$\lambda_{ij}=\lambda$ for all i,j and $\mu_1=\mu_2=....=\mu=1.0$

Figure 5. Mean value of X vs. the number of processes

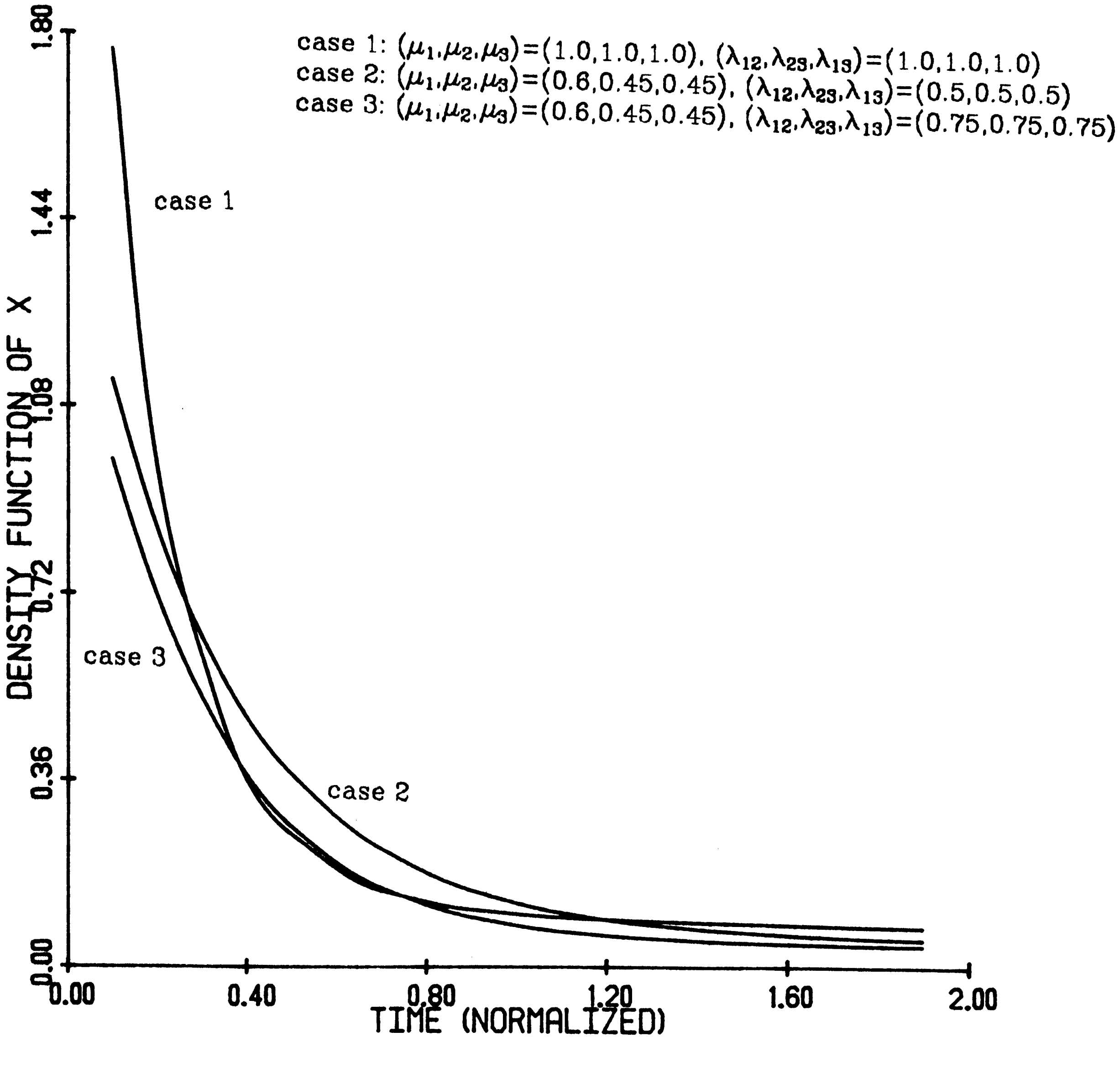


Figure 6. The Density Function of X, $f_x(t)$

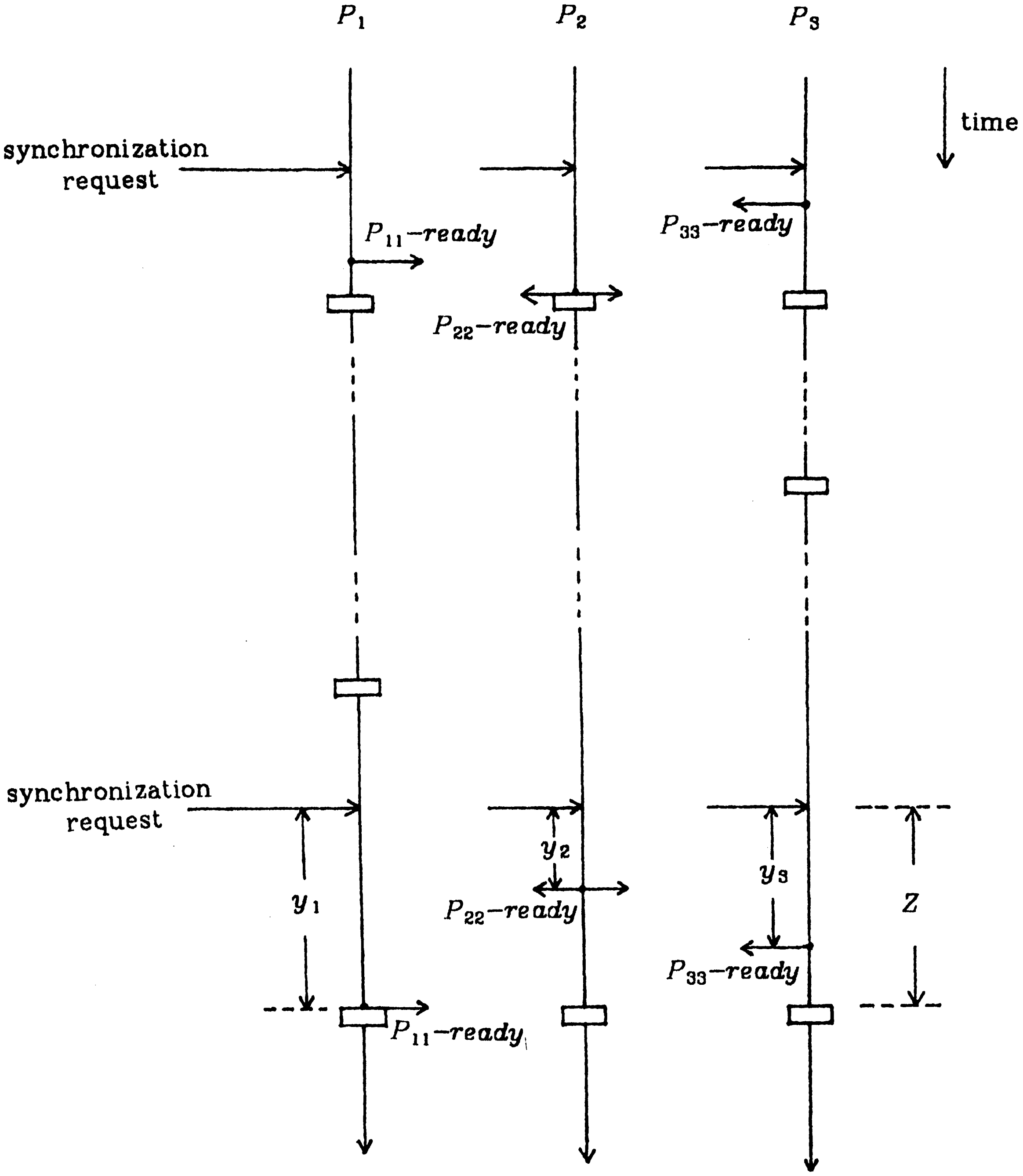
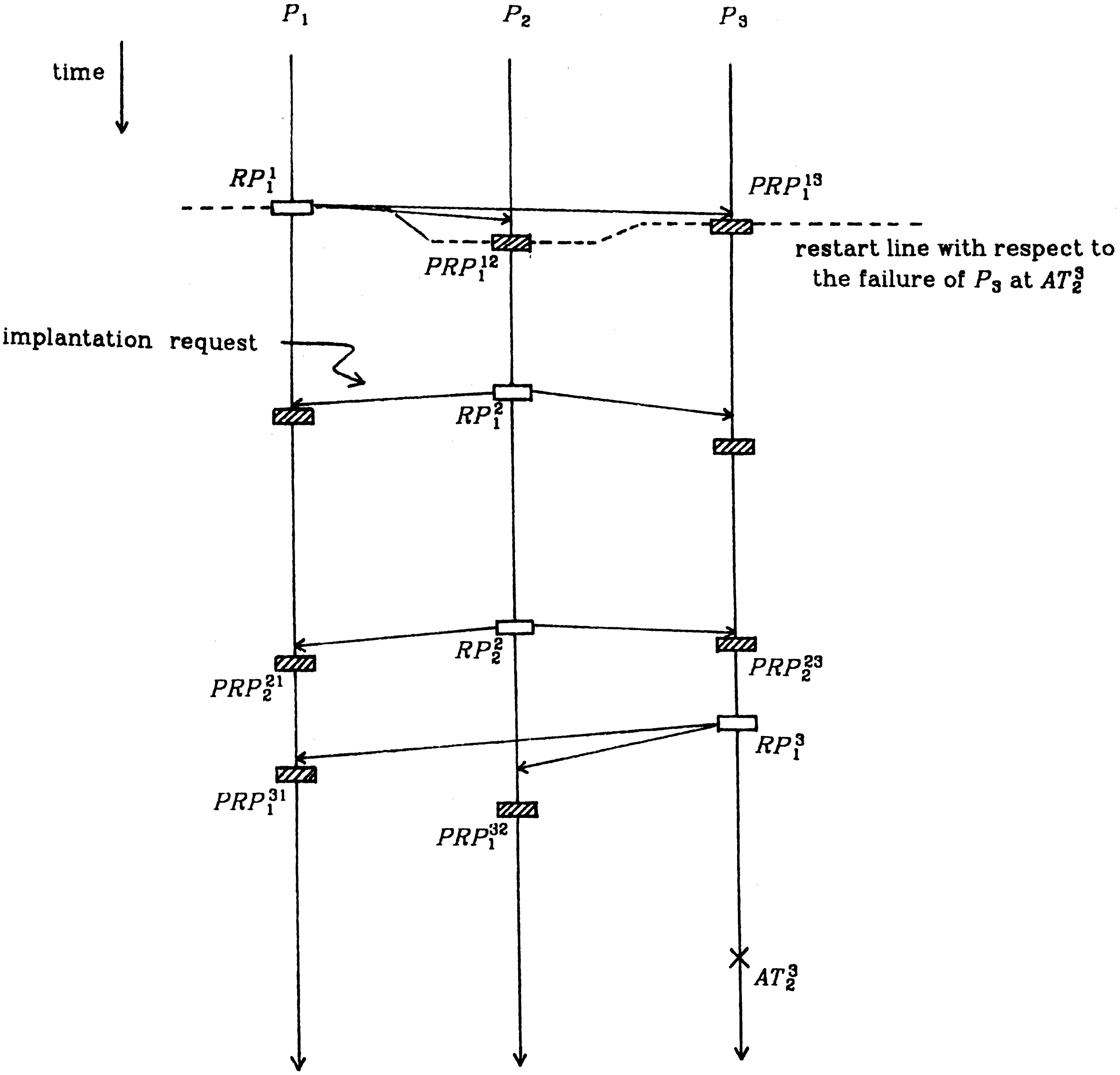


Figure 7. Establishment of Recovery Lines upon Synchronization Requests



▭ : Recovery Point (RP)
▨ : Pseudo Recovery Point (PRP)

Note: all occurrences of interactions are omitted

Figure 8. Establishment of Pseudo Recovery Points for Rollback Error Recovery

case	1	2	3	4	5
(μ_1,μ_2,μ_3) $(\lambda_{12},\lambda_{23},\lambda_{13})$	(1.0,1.0,1.0) (1.0,1.0,1.0)	(1.5,1.0,0.5) (1.0,1.0,1.0)	(1.0,1.0,1.0) (1.5,0.5,1.0)	(1.5,1.0,0.5) (1.5,0.5,1.0)	(1.5,1.0,0.5) (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

Table 1. Mean Values of X and L_i for constant ρ