Damage Assessment for Optimal Rollback Recovery

Tein-Hsiang Lin, *Member*, *IEEE Computer Society*, and Kang G. Shin, *Fellow*, *IEEE*

Abstract—Conventional schemes of rollback recovery with checkpointing for concurrent processes have overlooked an important problem: contamination of checkpoints as a result of error propagation among the cooperating processes. Error propagation is unavoidable due to imperfect detection mechanisms and random interprocess communications, and it could give rise to contaminated checkpoints which, in turn, result in unsuccessful rollbacks. To counter the problem of error propagation, a *damage assessment* model is developed to estimate the correctness of saved checkpoints under various circumstances. Using the result of damage assessment, determination of the "optimal" checkpoints for rollback recovery—which minimize the average total recovery overhead—is formulated and solved as a nonlinear integer programming problem. Integration of damage assessment into existing recovery schemes is also discussed.

Index Terms—Damage assessment, error propagation, rollback recovery, checkpointing, nonlinear integer programming.

1 Introduction

Considerable research effort has been directed toward rollback error recovery with checkpointing for concurrent processes [1], [2], [3], which requires each process to save its intermediate state, called a *checkpoint*, several times during the execution so that, upon detection of an error, it may roll back to, and resume execution from, one of the saved checkpoints. A major problem in such rollback recovery is the rollback propagation (or *domino effect*) that results from uncoordinated/asynchronous checkpoints and/or random communications among the concurrent processes [4], [5], [6]. Two types of solution have been proposed: *synchronous* and *optimistic* approaches. A synchronous approach eliminates rollback propagation by synchronizing both checkpoint establishments and interprocess communications among the concurrent processes, whereas an optimistic approach minimizes the effect of rollback propagation by recording interprocess messages (*message logging*) and replaying them during rollback recovery.

The numerous synchronous approaches proposed over the last decade or so include rollback propagation detection [7], [8], redundant recovery points insertion [9], [10], two-phase commitment protocols [11], [12], [13], and pseudorecovery points [14]. The main disadvantage of these approaches is the coordination overhead incurred during normal operation, but their advantage is fast rollback recovery upon detection of an error.

Most optimistic approaches [3], [15], [16], [17], [18], [19], [20] assume optimistic message logging in which each process establishes its own checkpoints independently and saves the checkpoints and the received messages asynchronously with others. Rollback propagation in these schemes may originate from those messages which have not been recorded by the receiver processes because they need to be regenerated, thus causing the sender processes to roll back. Rollback propagation could also occur if the system rolled back to an *inconsistent* system state where some messages have been recorded and will be replayed by the receiver processes but may not be generated again by the rolled-back sender processes due to the nondeterministic nature of distributed systems. In such cases, the receiver processes would have to roll back further until they reach a consistent system state. To identify consistent system states, a dependency vector is attached to every message indicating the state of the sender process at the time of message transmission. Whenever a recorded message satisfies a certain criterion, it can be discarded to free storage space since the message won't be needed for any future rollback recovery. Because these schemes do not eliminate rollback propagation completely, their disadvantage is a slower recovery than the synchronous approach in case an error is detected. Their advantage is the small overhead during normal operation, because checkpointing and message logging can be done asynchronously.

Besides rollback propagation, another major problem which has been overlooked in the existing approaches is *error propagation* as a result of random interprocess communications and incomplete detection coverage [21]. All the schemes mentioned above are based on the assumption that errors are detected immediately upon their occurrence. Under this assumption, the information contained in each checkpoint is always correct, because an error would otherwise have been detected before the checkpoint is saved. The benefits of this assumption (perfect detection coverage) are twofold: less secure storage space and less recovery overhead. A checkpoint can be discarded immediately as soon as the most recent consistent recovery line goes past it.

- *T.-H. Lin is with Microtec/Mentor Graphics, 2350 Mission College Blvd., Santa Clara, CA 95054. E-mail: Michael.Lin@mri.com.*
- *K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI" 48109-2122. E-mail: kgshin@eecs.umich.edu.*

Manuscript received 24 June 1994; revised 8 Jan. 1998.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 106498.

However, it is practically impossible to achieve perfect coverage of error detection. The penalty for making such an unrealistic assumption is the *global restart*—restart from the very beginning—required when a saved checkpoint is found to be incorrect. This procedure is acceptable only if the global restart is not very expensive and/or the cost of keeping several checkpoints is very high.

There are, however, a class of applications where the global restart is very expensive and the cost of keeping more than one checkpoint is low. For such applications, one must minimize the occurrence of global restarts by keeping multiple checkpoints. Under imperfect error detection, the key issues are to assess damages (caused by errors before their detection) and determine "optimal" rollback points using the results of damage assessment.

Damage assessment refers to the evaluation of the correctness of checkpoints in each process. Whether a checkpoint is correct or not can never be known for sure until it is actually used for error recovery. But, it is possible to estimate the time when a process became contaminated based on the information obtained from error detection and fault diagnosis mechanisms. The probability of a checkpoint being incorrect can then be determined under the assumption that the checkpoints established in a process after it became contaminated are incorrect. We develop a new method for damage assessment on the basis of our earlier results on error propagation [21] and fault diagnosis [22], [23]. Because of the probabilistic nature of damage assessment, the success of any rollback recovery cannot always be guaranteed and, thus, other recovery mechanisms, such as the global restart, should always be provided in case of unsuccessful rollbacks. Using the result of damage assessment, one can determine the optimal rollback points by minimizing the average recovery overhead, which is based on the overheads of the rollback recovery and the global restart, and the probabilities of incorrect checkpoints.

The paper is organized as follows. In Section 2, damage assessment is carried out by deriving the distributions of contamination times of individual processes. Three different cases are considered for this derivation. In Section 3, the problem of determining the optimal rollback points is formulated and solved as a nonlinear programming problem. In Section 4, we discuss how to integrate damage assessment into an existing optimistic scheme of rollback recovery. The paper concludes with Section 5.

2 Damage Assessment

In this section, a method of damage assessment is developed by probabilistically characterizing the interval between the occurrence and the detection of an error using parameters associated with faults and errors.

A *fault* is defined as any defect capable of causing potential damage, or any deviation from the normal state of a computing system. An *error* is deviation from the specification of a program running on a computing system. Consider a multimodule computing system, where the processes communicate with one another via message passing. Each process is assumed to run on a separate module and, thus, the term "module" will mean a hardware module or the process running on it. The computing system can be represented by a digraph, $D = (V, E)$, where $V = \{v_1, \ldots, v_N\}$ denotes the set of nodes, and $E = \{e_{ij}, 1 \le i, j \le N\}$ denotes the set of directed edges. Each node in V represents a module in the system, and a directed edge e_{ij} represents the communication channel via which v_i can send messages to v_j. A module is said to be *faulty* if it contains faults, and *contaminated* if it contains errors. Let T_i^F, the v_i's *faulty time*, denote the time instant a fault occurs in v_i. Let T_i^C, the v_i's *contamination time*, denote the time instant the first error occurs in v_i as a result of either the manifestation of a fault within v_i or the propagation of error(s) from other module(s). Then, damage assessment can be viewed as the estimation of the distributions of all modules' contamination times.

Faults are detected directly by periodic diagnostics or fault detection mechanisms, such as self-checking circuits. Errors, on the other hand, are detected by error detection mechanisms, such as acceptance tests, capability checks, and time-outs. Upon detection of an error, a fault location procedure is called for to identify the faulty module. (Note that, in the absence of error propagation, the fault location procedure is not needed.) In the discussions below, we assume that there is only one faulty module to simplify mathematical derivation. Based on when and how a detection is made, damage assessment is carried out separately in the following three cases:

Case 1: An error is detected and the faulty module is identified.

Case 2: An error is detected but the faulty module is not yet identified.

Case 3: A fault is detected by periodic diagnostics.

When a fault is detected by a self-checking circuit, instruction retry is a better recovery scheme than rollback [24], [25].

The accuracy of damage assessment depends on the information collected from detection and diagnosis mechanisms. This information includes the location of the faulty module and the error syndrome S, expressed as

$$S = \left[v_{b_1}, t_1; \ldots; v_{b_s}, t_s\right],$$

where $v_{b_1}, \ldots, v_{b_s}$ are the modules which have detected error(s), and $t_1, \ldots, t_s$ are the times at which the respective modules detected their first error. The modules which have not detected any error will be denoted by $v_{w_1}, \ldots, v_{w_{N-s}}$. In Case 1, both the error syndrome and the faulty module are known. In Case 2, only the error syndrome is available. In Case 3, the faulty module is known, but the error syndrome is not.

Let $\Delta_i(t)$ denote the conditional probability that v_i's contamination time is no later than t, given the information necessary for damage assessment, and let $\delta_i(t)$ denote the density function of $\Delta_i(t)$. Damage assessment is actually the derivation of $\delta_i(t)$, $1 \le i \le N$. These functions will be used in the next section to determine the optimal rollback points by minimizing the average total recovery time.

2.1 Modeling Error Propagation

We briefly describe an error propagation model which is instrumental in the derivation of $\delta_i(t)$'s. The parameters defined in this model are random variables with certain distributions unless explicitly defined otherwise. For a detailed account of this model, see [21] and [22].

The rate of fault occurrence in module v_i is characterized by its *fault cycle*, denoted by Y_i, which is the time interval between two consecutive fault arrivals at v_i. If v_i is the faulty module, then the *fault latency* of v_i, denoted by L_i, is the interval between v_i's faulty time and contamination time.

Errors in one module can propagate to other modules via *propagation paths*, which are communication paths from the source to the destination with *distinct* intermediate modules. Errors propagating into an already contaminated module are assumed to have negligible effects on its error propagation property (see [21] for a justification of this assumption). The *error propagation time* from v_i to v_j, denoted by X_{ij}, is defined as the time interval between the contamination times of v_i and v_j. The error propagation times will be derived from B_{ij}, which is defined as the time for an error to propagate from v_i to one of its neighbors, v_j, via a direct communication channel between them. B_{ij}'s are assumed to be independent of each other. The relationship between X_{ij}s and B_{ij}s is obtained as follows: First, identify all propagation paths and calculate the error propagation time of each path by summing up the B_{ij}s along the path. Then, X_{ij} is the minimum propagation time among all the paths from v_i to v_j. For example, for a system represented by graph D1 in Fig. 1,

$$X_{13} = \min\,(B_{12} + B_{23}, B_{14} + B_{45} + B_{53}, B_{14} + B_{45} + B_{52} + B_{23},\; B_{12} + B_{24} + B_{45} + B_{53}).$$

The distributions of X_{ij}s can be derived from those of B_{ij}s systematically and efficiently, as shown in [21].

Based on the types of detection mechanism used, faults in a module are classified into three categories:

1) *FD-detectable* if they can be uncovered by signal-level fault detection mechanisms [26],
2) *PD-detectable* if they are not FD-detectable but are detectable by periodic diagnostics, and
3) *undetectable* if they are neither FD-detectable nor PD-detectable.

A signal-level fault detection mechanism has the property that faults are detected immediately upon their occurrence [26]. If a fault is detected in a module during periodic diagnostics, errors might already have been induced and propagated to other modules. An undetectable fault can be captured only during the fault diagnosis after the errors induced by this fault are detected by some detection mechanisms. The probabilities for any fault to be FD-detectable and PD-detectable, denoted by C_i^F and C_i^P, respectively, are assumed to be fixed and known.

Error detection in a module v_i is characterized by K_i, the *detection latency* of v_i, defined as the time interval from the v_i's contamination time to its detection time, which is the time the first error detection is made. The distribution of K_i depends on the error detection mechanisms used in v_i.

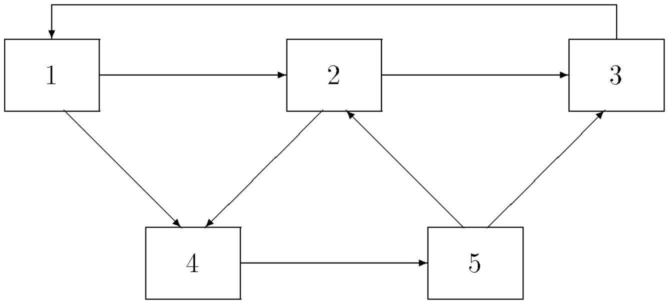

Fig. 1. System graph D1.

2.2 Damage Assessment for Case 1

Both S and the faulty module v_k are known in this case. Damage assessment will start from v_k, the source of errors. To derive $\delta_k(t)$, it is essential to calculate

1) $f_k^{T^C}$, the density function of T_k^C without considering the error syndrome, and
2) the error syndrome's conditional likelihood $\mathcal{L}_k(S, \tau)$, which is the conditional probability of S given that v_k is the faulty module and $T_k^C = \tau$.

Let T_k^D denote the time of v_k's last complete diagnosis and T_k^P denote the time of v_k's last periodic diagnostic. A complete diagnosis is assumed to have 100 percent coverage so that v_k should be fault-free immediately after T_k^D, which can be as early as T_k^Y, the last faulty time, if no complete diagnosis has been applied since then. In v_k, only undetectable faults could occur between T_k^D and T_k^P, but both PD-detectable and undetectable faults could occur between T_k^P and t_1, the time of the first error detection. Therefore, the density function of T_k^F is expressed as

$$f_k^{T^F}(t) = \begin{cases} \frac{\left(1-C_k^F-C_k^P\right)}{W_k} f_k^Y\left(t - T_k^Y\right) & \text{if } T_k^D \le t < T_k^P \\ \frac{\left(1-C_k^F\right)}{W_k} f_k^Y\left(t - T_k^Y\right) & \text{if } T_k^P \le t < t_1, \end{cases}$$

where $f_k^Y(\cdot)$ is the density function of Y_k and W_k is the normalizing constant. The density function of T_k^C is evaluated as

$$f_k^{T^C}(t) = f_k^{T^F}(t) * f_k^L(t),$$

where "$*$" denotes the convolution, since $T_k^C = T_k^F + L_k$.

Define E_{kj}, the error latency from v_k to v_j, as the time interval from the v_k's contamination time to the v_j's detection time, i.e.,

$$E_{kj} = X_{kj} + K_j.$$

If $k = j$, then $E_{kj} = K_j$. Propagation of errors from a faulty module v_k into other modules is characterized by the joint distribution of E_{k1}, E_{k2}, …, and E_{kN} and, thus, the error syndrome's conditional likelihood can be calculated as

$$\mathcal{L}_i(S,\tau) = \mathrm{Prob}\Big[E_{ib_1} = t_1 - \tau, \ldots, E_{ib_s} = t_s - \tau, E_{iw_1} > T - \tau, \ldots, E_{iw_{N-s}} > T - \tau\Big],$$

where T is the current time instance at which damage assessment is done. Using the Bayes's equation, $\delta_k(t)$ is derived as

$$\delta_k(t) = \frac{\mathcal{L}_k(S,t) f_k^{T^C}(t)}{\int_{T_k^D}^{t_1} \mathcal{L}_k(S,\tau) f_k^{T^C}(\tau) d\tau}.$$

To estimate the damage on module v_j, $j \neq k$, first calculate $\mathcal{L}_{kj}(S,\tau_k,\tau_j)$, the likelihood of S when $T_k^C = \tau_k$ and $T_j^C = \tau_j$. Obviously, $\mathcal{L}_{kj}(S,\tau_k,\tau_j) = 0$ if $\min(t_1,\tau_j) < \tau_k < T_k^D$, and for $T_k^D < \tau_k < \min(t_1,\tau_j)$,

$$\begin{aligned}\mathcal{L}_{kj}\big(S,\tau_k,\tau_j\big) = \mathrm{Prob}\Big[X_{kj} &= \tau_j - \tau_k,\\ E^j_{kb_1}\big(\tau_j-\tau_k\big) &= t_1 - \tau_k, \ldots,\\ E^j_{kb_s}\big(\tau_j-\tau_k\big) &= t_s - \tau_k,\\ E^j_{kw_1}\big(\tau_j-\tau_k\big) &> T - \tau_k, \ldots,\\ E^j_{kw_{N-s}}\big(\tau_j-\tau_k\big) &> T - \tau_k\Big],\end{aligned}$$

where $E^j_{ki}(s) = X^j_{ki}(s) + K_i$ and $X^j_{ki}(s)$ is the error propagation time from v_k to v_i under the condition that $X_{kj} = s$. In other words, $X^j_{ki}(s)$ can be viewed as a special case of X_{ki} with two sources of errors. One source is v_k, from which errors propagate to v_i via all possible paths between v_k and v_i except for those passing through v_j. The other source is v_j, which starts the propagation of error to v_i via all possible paths between v_j and v_i at s time units after v_k became faulty. This interpretation simplifies the evaluation of $X^j_{ki}(s)$. For example, in graph D_1 of Fig. 1, $X^4_{13}(s)$ can be evaluated as

$$\begin{aligned}X^4_{13}(s) &= \min\{B_{12} + B_{23}, s + X_{43}\}\\ &= \min\{B_{12} + B_{23}, s + B_{45} + B_{53}, s + B_{45} + B_{52} + B_{23}\}.\end{aligned}$$

With the knowledge of $\mathcal{L}_{kj}(S,\tau_k,\tau_j)$ for all τ_k and τ_j,

$$\begin{aligned}\delta_j(t) &= \frac{\int_{T_k^D}^{t} \mathcal{L}_{kj}\big(S,\tau_k,t\big) f_k^{T^C}\big(\tau_k\big) d\tau_k}{\int_{T_k^D}^{\infty}\int_{T_k^D}^{t_1} \mathcal{L}_{kj}(S,\tau_k,\tau_j) f_k^{T^C}\big(\tau_k\big) d\tau_k\, d\tau_j}\\ &= \frac{\int_{T_k^D}^{t} \mathcal{L}_{kj}\big(S,\tau_k,t\big) f_k^{T^C}\big(\tau_k\big) d\tau_k}{\int_{T_k^D}^{t_1} \mathcal{L}_k\big(S,\tau_k\big) f_k^{T^C}\big(\tau_k\big) d\tau_k}.\end{aligned}$$

Note that the expressions of $\delta_k(t)$ and $\delta_j(t)$ have the same denominator.

As an example, consider a system represented by D1 in Fig. 1 and a error syndrome $S = [v_2, 0]$. We carried out simulation to determine all the likelihoods and the functions Δ_is for this syndrome under different assumptions of the faulty module. All random parameters, except B_{ij}s, are assumed to be exponentially distributed. B_{ij} is assumed to have a bimodal normal distribution, i.e.,

TABLE 1
DISTRIBUTION OF RANDOM PARAMETERS FOR ALL i AND j

Variable	Distribution	Parameters
Y_i	Exponential	Mean = 100,000
L_i	Exponential	Mean = 40
B_{ij}	Bimodal Normal	$\eta_{ij} = 0.6$ $\mu^1_{ij} = 40, \sigma^1_{ij} = 25$ $\mu^2_{ij} = 80, \sigma^2_{ij} = 60$

TABLE 2
PARAMETERS FOR THE FAULTY MODULE v_k

Module	C_k^F	C_k^P	T_k^Y	T_k^D	T_k^P
1	0.4	0.4	–100,000	-50,000	–600
2	0.4	0.4	–100,000	-40,000	–700
3	0.4	0.4	–100,000	-30,000	–800
4	0.4	0.4	–100,000	-20,000	–900
5	0.4	0.4	–100,000	-10,000	–1,000

$$B_{ij} = \begin{cases}\mathcal{N}\big(\mu^1_{ij},\sigma^1_{ij}\big) & \text{with probability } \eta_{ij}\\ \mathcal{N}\big(\mu^2_{ij},\sigma^2_{ij}\big) & \text{with probability } 1-\eta_{ij},\end{cases}$$

where $\mathcal{N}(\mu, \sigma)$ denotes a normally-distributed random variable with mean μ and standard deviation σ. The specification and derivation of K_i's distribution were discussed in [22] and, thus, will not be repeated here. The parameters used in the simulation are tabulated in Tables 1 and 2, where all time variables are assumed to have the same unit with the current error detection time set to zero.

The details of the simulation were reported in [22]. Some results of Δ_is are plotted in Figs. 2 and 3, where the faulty module is v_1 and v_2, respectively. In Fig. 3, Δ_3 and Δ_4 are almost identical because the respective parameters are identical and both v_3 and v_4 are only one-hop away from v_2. The same can be said about Δ_1 and Δ_5.

2.3 Damage Assessment for Case 2

In this case, damage assessment must be made without any knowledge on the location of the faulty module. One example is when error recovery and fault location are carried out in parallel. Another example is when the fault diagnosis routine fails to locate the faulty module due to either the occurrence of a transient fault or insufficient coverage of the diagnosis routine.

Let π_i represent the v_i's faulty probability without considering the error syndrome. These π_is are our subjective belief in locating the faulty module. If damage assessment is performed upon detection of an error, π_i is the same as the prior faulty probability π'_i determined by

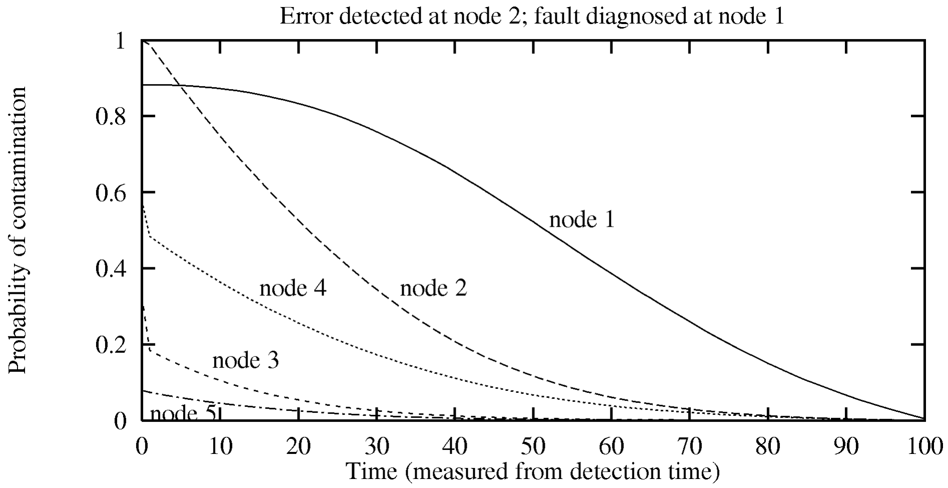

Fig. 2. Δ functions when $S = [v_2, 0]$ and $v_k = v_1$.

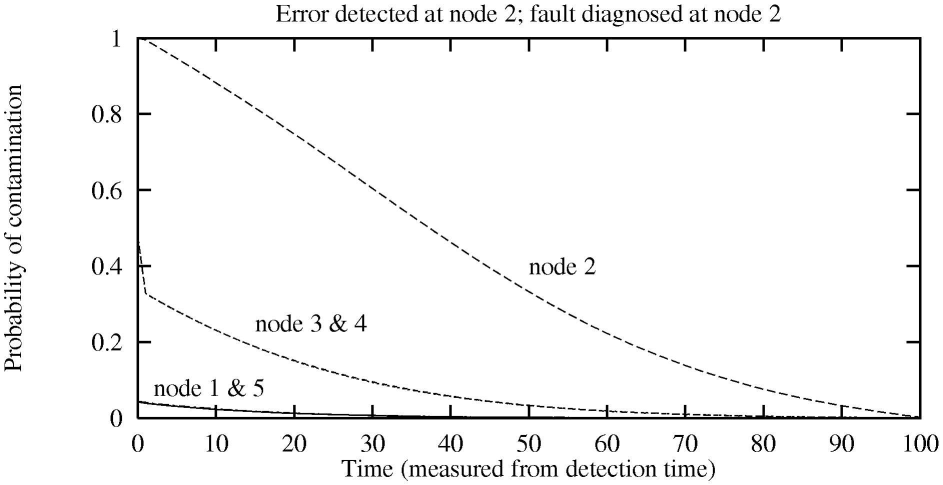

Fig. 3. Δ functions when $S = [v_2, 0]$ and $v_k = v_2$.

$$\pi_i' = \frac{F_i^Y\left(T - T_i^Y\right) \prod_{j=1, j\neq i}^{N} \left(1 - F_j^Y\left(T - T_j^Y\right)\right)}{\sum_{j=1}^{N} \left(F_j^Y\left(T - T_j^Y\right) \prod_{j=1, j\neq i}^{N} \left(1 - F_j^Y\left(T - T_j^Y\right)\right)\right)},$$

where $F_i^Y(T - T_i^Y)\prod_{j=1,j\neq i}^{N}(1 - F_j^Y(T - T_j^Y))$ is our relative suspicion of v_i being the faulty module.

On the other hand, if damage assessment is performed after fault diagnosis, π_i is determined by

$$\pi_i - \pi_i'(1 - \Phi_i) \Big/ \sum_{j=1}^{N} \pi_j'(1 - \Phi_j),$$

where Φ_i is the coverage of the diagnosis routine applied to v_i. $(1 - \Phi_i)$ can be interpreted as the likelihood of v_i being faulty but misdiagnosed to be nonfaulty. Using π_is as the weighting factors, $\delta_j(t)$, $1 \leq j \leq N$, is derived as

$$\delta_j(t) = \frac{\sum_{k=1}^{N} \pi_k \int_{T_k^D}^{t} \mathcal{L}_{kj}(S, \tau_k, t) f_k^{T^C}(\tau_k) d\tau_k}{\sum_{k=1}^{N} \pi_k \int_{T_k^D}^{t_1} \mathcal{L}_k(S, \tau_k) f_k^{T^C}(\tau_k) d\tau_k}.$$

2.4 Damage Assessment for Case 3

This case arises when a periodic diagnostic detects a fault in v_k. It implies that the fault may have occurred any time between the present and the previous periodic diagnostic completed at T_k^P. If the fault had occurred before T_k^P, it would have been detected by the previous periodic diagnostic. Using the Bayes' equation, the distribution of v_k's faulty time in this case is derived as

$$f_k^{T^F}(t) = \frac{f_k^Y\left(t - T_k^Y\right)\left(1 - F_k^Y(T - t)\right)}{\int_{T_k^P}^{T} f_k^Y\left(\tau - T_k^Y\right)\left(1 - F_k^Y(T - \tau)\right) d\tau}$$

$$\text{for } T_k^P \leq t \leq T,$$

where T is the fault detection time. If Y_k is exponentially distributed, T_k^F can be shown to be distributed uniformly over the interval $[T_k^P, T]$. The function $f_k^{T^C}(t)$ is then calculated as the convolution of $f_k^{T^F}(t)$ and $f_k^L(t)$, since $T_k^C = T_k^F + L_k$.

Though no error syndrome is available in this case, the fact that no errors have been detected so far is useful information. Let $\overline{\mathcal{L}}(T, \tau_k, \tau_j)$ be the likelihood of no error detection up to time T under the condition that $T_k^C = \tau_k$ and $T_j^C = \tau_j$, where v_k is the faulty module and $\tau_k \le \tau_j$. It is easy to see that $\overline{\mathcal{L}}_{kj}(T, \tau_k, \tau_j) = 1$ if $\tau_k > T$. For $\tau_k \le T$,

$$\overline{\mathcal{L}}_{kj}(T, \tau_k, \tau_j) = \mathrm{Prob}\Big[X_{kj} = \tau_j - \tau_k, E_{k1}^j(\tau_j - \tau_k) > T - \tau_k, \ldots, E_{kN}^j(\tau_j - \tau_k) > T - \tau_k\Big].$$

The function $\delta_k(t)$ for $T_i^P \le t \le T$ is, thus, derived as

$$\delta_k(t) = \frac{\overline{\mathcal{L}}_{kk}(T,t,t) f_k^{T^C}(t)}{\int_{T_k^P}^{\infty}\int_{T_k^P}^{\infty} \overline{\mathcal{L}}_{kk}(T,\tau_k,\tau_k) f_k^{T^C}(\tau_k)\, d\tau_k} = \frac{\overline{\mathcal{L}}_{kk}(T,t,t) f_k^{T^C}(t)}{\left(1 - F_k^{T^C}(T)\right) + \int_{T_k^P}^{T} \overline{\mathcal{L}}_{kk}(T,\tau_k,\tau_k) f_k^{T^C}(\tau_k)\, d\tau_k}.$$

And, for $1 \le j \le N, j \ne k$,

$$\delta_j(t) = \frac{\int_{T_k^P}^{\infty} \overline{\mathcal{L}}_{kj}(T,\tau_k,t) f_k^{T^C}(\tau_k)\, d\tau_k}{\int_{T_k^P}^{\infty}\int_{T_k^P}^{t} \overline{\mathcal{L}}_{kj}(T,\tau_k,\tau_j) f_k^{T^C}(\tau_k)\, d\tau_k\, d\tau_j} = \frac{\left(1 - F_k^{T^C}(T)\right) + \int_{T_k^D}^{T} \overline{\mathcal{L}}_{kj}(T,\tau_k,t) f_k^{T^C}(\tau_k)\, d\tau_k}{\left(1 - F_k^{T^C}(T)\right) + \int_{T_k^P}^{T} \overline{\mathcal{L}}_{kk}(T,\tau_k,\tau_k) f_k^{T^C}(\tau_k)\, d\tau_k}.$$

2.5 Remarks

The derivation of $\delta_i(t)$s in all three cases requires the knowledge of $f_i^{T^C}(t)$ and some likelihood functions such as $\mathcal{L}_k$, $\mathcal{L}_{kj}$, or $\overline{\mathcal{L}}_{kj}$. It is very complex to derive these likelihoods analytically and, thus, these likelihood functions are usually calculated numerically. Since even the numerical solution requires an excessive amount of time, the likelihood functions are computed off-line. However, the function $\delta_i(t)$ has to be derived on-line since $f_i^{T^C}(t)$ depends upon on-line information, such as the elapsed time from a previous fault and the elapsed time from the previous periodic diagnostic. If the situation does not allow for on-line derivation of $\delta_i(t)$, $\delta_i(t)$ can be determined off-line using the noninformative prior function in place of $f_i^{T^C}(t)$. The noninformative prior function is usually a uniform density function, i.e., a faulty module could be contaminated any time prior to the detection time with an equal probability.

2.5.1 List of Symbols

$F_i^V(f_i^V)$ Distribution (density) function of the random variable V_i.

f_i^V Density function of the random variable V_i.

Y_i Fault cycle of v_i.

L_i Fault latency of v_i.

C_i^F Coverage of fault detection in v_i.

C_i^P Coverage of periodic diagnostics in v_i.

B_{ij} Direct propagation time from v_i to v_j.

X_{ij} Error propagation time from v_i to v_j.

K_i Detection latency in v_i.

H_i Hypothesis of v_i being the faulty module.

$\pi_i'(\pi_i)$ Prior (posterior) Prob$[H_i]$.

π_i Posterior Prob$[H_i]$.

S Fault syndrome.

$\mathcal{L}_i(S)$ Fault syndrome's likelihood function if H_i is true.

E_{ij} Error latency for the error originated from v_i and detected in v_j.

T_i^Y Last time a repair was done on v_i.

T_i^D Last time a thorough diagnostic was applied to v_i.

T_i^P Last time a periodic diagnostic was applied to v_i.

T_i^C The contaminating time of v_i.

T_i^F The faulty time of v_i.

Φ_i Percentage of faults that the diagnosis routine can uncover in v_i and $\phi_i = d\Phi_i/dt$.

3 Optimal Rollback Points

In this section, we will determine the optimal rollback points for rollback recovery using the results from damage assessment.

Let T denote the current time. For the convenience of problem formulation, T is designated as the origin of the time axis and all events are enumerated backward from T. For example, the kth checkpoint (message) means the kth previous checkpoint (message) from T. Let cp_i^k be the time v_i established the kth checkpoint, and r_i^k be the time v_i received the kth message. If v_i rolls back to the kth checkpoint, then v_i's rollback distance is defined as

$$d_i(k) = T - cp_i^k. \tag{3.1}$$

The probability that v_i can successfully roll back to the kth checkpoint is denoted by $p_i(k)$. Note that $p_i(k)$ is the same as the probability that v_i is contaminated after cp_i^k. Hence,

$$p_i(k) = \int_{cp_i^k}^{\infty} \delta_i(\tau)\, d\tau = 1 - \Delta_i\left(cp_i^k\right). \tag{3.2}$$

The above equation holds only when v_i is the faulty module. If v_i is not the faulty module, its contamination is caused by incoming message(s) from other module(s), i.e., error propagation. Since v_i receives messages only at r_i^n, $n \geq 1$, if v_i is not the faulty module and $r_i^n < cp_i^k < r_i^{n+1}$, then

$$p_i(k) = 1 - \Delta_i\left(r_i^n\right).$$

There is one exception, however; if an error has been detected in v_i and v_i has received no messages since cp_i^k, then $p_i(k) = 0$.

Define the overhead of rollback recovery as the sum of the rollback distances (d_is) in all modules, which is the total computation to be redone. Our object is to minimize the mean recovery overhead, denoted by O. Let Z denote the total overhead for a global restart which will be invoked in case rollback recovery fails. Rollback recovery will eventually fail if any module of the system rolls back to an incorrect checkpoint. Thus, the probability of successful rollback recovery is the product of the probability that each module's rollback point is correct. Hence, O can be expressed as

$$\begin{aligned} O &= \left(\prod_{i=1}^{N} p_i(k_i)\right)\sum_{i=1}^{N} d_i(k_i) + \left(1 - \prod_{i=1}^{N} p_i(k_i)\right)\left(\sum_{i=1}^{N} d_i(k_i) + Z\right) \\ &= \sum_{i=1}^{N} d_i(k_i) + \left(1 - \prod_{i=1}^{N} p_i(k_i)\right)Z, \end{aligned} \tag{3.3}$$

where k_i is a nonnegative integer indicating the rollback point of v_i. There is an upper limit, say C_i, for k_i, $1 \leq i \leq N$, because each secure storage will have a limited capacity. The optimal rollback problem is then stated as follows:

$$\begin{aligned} &\text{Minimize } \sum_{i=1}^{N} d_i(k_i) + \left(1 - \prod_{i=1}^{N} p_i(k_i)\right)Z \\ &\text{subject to } k_i \leq C_i \quad \text{for} \quad 1 \leq i \leq N. \end{aligned}$$

This is a nonlinear integer programming problem.

Integer programming problems are in general very difficult to solve. Most of existing algorithms, such as cutting plane methods and enumerative search methods, are applicable only if the objective function is a linear function. To develop an algorithm for a nonlinear integer programming problem, we have to find and utilize some special properties of the object function O. One useful property is the convexity of O. If p_i is expressed as in (3.2), O can be viewed as a continuous function of d_is. The following lemma then provides the necessary and sufficient condition for the convexity of O.

LEMMA 1. *O is convex with respect to d_i if and only if $\delta_i(t)$, $t = T - d_i$, is monotonically increasing.*

PROOF. From (3.3),

$$\frac{\partial O}{\partial d_i} = 1 - \frac{\partial p_i}{\partial d_i} Z \prod_{j=1, j\neq i}^{N} p_j$$

and

$$\frac{\partial^2 O}{\partial d_i^2} = -\frac{\partial^2 p_i}{\partial d_i^2} Z \prod_{j=1, j\neq i}^{N} p_j .$$

O is convex with respect to d_i if and only if $\frac{\partial^2 p_i}{\partial d_i^2} < 0$, since both Z and p_j are positive. From (3.1) and (3.2),

$$\frac{\partial p_i}{\partial d_i} = -\delta_i(t)\frac{\partial t}{\partial d_i} = \delta_i(t)$$

and

$$\frac{\partial^2 p_i}{\partial d_i^2} = -\delta_i'(t).$$

Thus, $\frac{\partial^2 p_i}{\partial d_i^2} < 0$ if and only if $\delta_i'(t) > 0$, i.e., if $\delta_i(t)$ is monotonically increasing. □

In general, $\delta_i(t)$ may not be monotonically increasing for $t \in [-\infty, \infty]$. But, there must exist T_i such that $\delta_i(t)$ is monotonically increasing for $t \in [-\infty, T_i]$, since $\delta_i(t) \to 0$ as $t \to -\infty$.

Based on the convexity of O, the following algorithm is developed to solve the optimal rollback problem.

Algorithm RB:

1) For $1 \leq i \leq N$, let k_i be the smallest integer with the property $cp_i^{k_i} < T_i$.
   Let $k_i := C_i$ if $k_i > C_i$, and $Y := \left(\prod_{i=1}^{N} p_i(k_i)\right)Z$.
2) Let $S := \{i : k_i < C_i\}$.
   If $k_i = C_i$ $\forall i \in S$, then go to Step 6.
3) $\forall i \in S$, let $y_i := \delta d_i - (\rho_i - 1)Y$ where
$$\begin{aligned} \delta d_i &:= d_i(k_i + 1) - d_i(k_i), \\ \rho_i &:= 1 + \frac{\delta p_i}{p_i(k_i)}, \\ \delta p_i &:= p_i(k_i + 1) - p_i(k_i). \end{aligned}$$
4) If $y_i \geq 0$ $\forall i \in S$, then go to Step 5.
   Otherwise, $\forall i \in S$ where $y_i < 0$, let $k_i := k_i + 1$ and $Y := \rho_i Y$.
   Go to Step 2.
5) For any $A \subset S$, let $D(A) := \sum_{i\in A} \delta d_i - \left(\prod_{i\in A} \rho_i - 1\right)Y$.
   Find a set $S^* \subset S$ such that $D(S^*) < 0$.
   If such an S^* does not exist, then go to Step 6.
   Otherwise, $\forall i \in S^*$, let $k_i := k_i + 1$ and $Y := \rho_i Y$.
   Go to Step 2.
6) Terminate the algorithm.
   The current value of k_i, $1 \leq i \leq N$, is the optimal recovery point for v_i.

RB is essentially an algorithm which searches from smaller k_is toward larger k_is. In Step 1, k_i is initialized to the smallest value that will put the corresponding cp_i^k inside the convex region of the function O with respect to k_i. By Lemma 1, the optimal solution for k_is is obtained when O cannot be reduced any further by incrementing any subset of k_is. If k_i reaches its limit C_i before the minimum O is reached, the optimal solution for k_i will be C_i. If all k_is reach their limits during the search, **RB** terminates immediately (Step 2).

The search in **RB** is conducted in two levels. In the first level (Steps 3 and 4), we check whether incrementing a legitimate k_i (i.e., $k_i < C_i$) would result in a smaller mean overhead. The set S defined in Step 2 is the set of all legitimate k_i's at that moment. The variable y_i is actually the difference in O with k_i and $k_i + 1$, i.e.,

$$y_i = O(\ldots, k_i + 1, \ldots) - O(\ldots, k_i, \ldots).$$

If all y_is are greater than zero, the next level of search is performed in Step 5. If at least one y_i is negative, all k_is with a negative y_i will be incremented by one, which will yield an even smaller mean overhead because of the following lemma.

LEMMA 2. *Let y_i, $1 \le i \le N$, be defined as in Step 3 of* **RB**. *If $y_i < 0$ and $y_j < 0$, then*

$$O(\ldots, k_i + 1, \ldots, k_j + 1, \ldots) < O(\ldots, k_i, \ldots, k_j, \ldots).$$

PROOF. Let $\delta d_i = d_i(k_i + 1) - d_i(k_i)$ and $\delta p_i = p_i(k_i + 1) - p_i(k_i)$ for $1 \le i \le N$. Then,

$$\begin{aligned} &O(\ldots, k_i + 1, \ldots, k_j + 1, \ldots) \\ &= \sum_{i=1}^{N} d_i(k_i) + \delta d_i + \delta d_j + \\ &\qquad \left(1 - \left(p_i(k_i) + \delta p_i\right)\left(p_j(k_j) + \delta p_j\right) \prod_{n=1, n\ne i,j}^{N} p_n(k_n)\right) Z \\ &= O(\ldots, k_i, \ldots, k_j, \ldots) + y_i + y_j - \delta p_i \delta p_j \prod_{n=1, n\ne i,j}^{N} p_n(k_n)\, Z \\ &< O(\ldots, k_i, \ldots, k_j, \ldots), \end{aligned}$$

since $y_i < 0$, $y_j < 0$, and $\delta p_i \delta p_j \prod_{n=1, n\ne i,j}^{N} p_n(k_n)\, Z > 0$. □

Even if O cannot be reduced any further by incrementing any single legitimate k_i, it is still possible to reduce O further by incrementing a subset of S. If such a subset (i.e., S^*) exists, it would be found in Step 5. The function $D(A)$ in Step 5 is the difference in O with k_is and $k_i + 1$s, $\forall i \in A$. Finding an S^* is not a trivial task because the number of subsets increases exponentially with the number of legitimate k_is. A branch-and-bound (B&B) algorithm is, thus, developed below either to find an S^*, if it exists, or to show that it does not exist.

Let $S_n(A)$ represent any set with n elements which is both a subset of S and superset of A. In other words, $\|S_n(A)\| = n$ and $A \subset S_n(A) \subset S$, where $\|X\|$ denotes the cardinality of the set X. The elements of S will be sorted in two lists: one based on δd_i and the other based on ρ_i so that the sets defined below can be easily determined. For any $A \subset S$ and $n \le \|S\|$, let $S^{\rho}_{max}(A, n)$ be defined such that

(1) $A \subset S^{\rho}_{max}(A, n)$ and $\left\|S^{\rho}_{max}(A, n)\right\| = n$,

(2) $\rho_i \ge \rho_j$ for any $i \in S^{\rho}_{max}(A, n) - A$ and $j \in S - S^{\rho}_{max}(A, n)$,

and let $S^{d}_{min}(A, n)$ be defined such that

(i) $A \subset S^{d}_{min}(A, n)$ and $\left\|S^{d}_{min}(A, n)\right\| = n$,

(ii) $\delta d_i \le \delta d_j$ for any $i \in S^{d}_{min}(A, n) - A$ and $j \in S - S^{d}_{min}(A, n)$.

For $A \subset S$ and $n \ge \|A\|$, define

$$w(A, n) = \sum_{i \in S^{d}_{min}(A,n)} \delta d_i - \left(\prod_{i \in S^{\rho}_{max}(A,n)} \rho_i - 1\right) Y,$$

and it can be shown easily that $w(A, n) \le D(S_n(A))$. With the above definitions, we now present the following B&B algorithm.

Algorithm BB:

1) Let $n := \|S\|$.
2) If $n = 1$, the algorithm terminates and it is concluded that S^* does not exist.
3) Let $A_n := S^{d}_{min}(\varnothing, n) \cap S^{\rho}_{max}(\varnothing, n)$, where $\varnothing$ denotes the empty set.
4) If $w(A_n, n) \ge 0$, let $n := n - 1$ and go to Step 2.
5) If $\|A_n\| = n$, the algorithm terminates with $S^* := A_n$.
6) Examine all subsets of $S - A_n$ with $n - \|A_n\|$ elements using a depth-first search to find a set R with $D(A_n \cup R) < 0$.
   Whenever such an R is found, the algorithm terminates with $S^* := A_n \cup R$.
   Otherwise, let $n := n - 1$ and go to Step 2.

The basic strategy in finding an S^* is to classify subsets of S based on the cardinality of each subset and, then, search within each subset class for a possible solution. The subset class with the most number of elements is searched first. If no solutions are found in all subset classes, we conclude that S^* does not exist. In searching the subset class with n elements, $A_n \subset S^*$ can be assumed to shorten the search according to the following lemma.

LEMMA 3. *If an S^* exists and $\|S^*\| = n$, then there must exist a set S' with n elements such that $D(S') < 0$ and*

$$A_n = S^{d}_{min}(\varnothing, n) \cap S^{\rho}_{max}(\varnothing, n) \subset S'.$$

PROOF. If $A_n \subset S^*$, the proof is trivial since we can let $S' = S^*$. If $A_n \not\subset S^*$, then, by the definitions of S^{d}_{min} and S^{ρ}_{max}, we can find a set $A' \subset S^*$ with $\|A_n\|$ elements such that $\sum_{i \in A} \delta d_i \le \sum_{j \in A'} \delta d_j$ and $\prod_{i \in A} \rho_i \ge \prod_{j \in A'} \rho_j$. The lemma is proved by letting $S' = (S^* - A') \cup A$, since

$$\begin{aligned} 0 > D(S^*) &= \sum_{i \in S^*-A'} \delta d_i \sum_{j \in A'} \delta d_j - \left(\prod_{i \in S^*-A'} \rho_i \prod_{j \in A'} \rho_j - 1\right) Y \\ &\ge \sum_{i \in S^*-A'} \delta d_i \sum_{j \in A} \delta d_j - \left(\prod_{i \in S^*-A'} \rho_i \prod_{j \in A} \rho_j - 1\right) Y \\ &= D\left((S^* - A') \cup A\right). \end{aligned}$$

□

Since $w(A_n, n)$ is the lower bound of $D(S_n(A_n))$ $\forall S_n(A_n)$, S^* cannot exist in the subset class with n elements if $w(A_n, n) \ge 0$. This property will be applied during the depth-first search to minimize the search time. To illustrate this, suppose $n = 5$, $\|A_n\| = 2$, and $S - A_n = \{1, 2, 3, 4, 5\}$. The objective is to

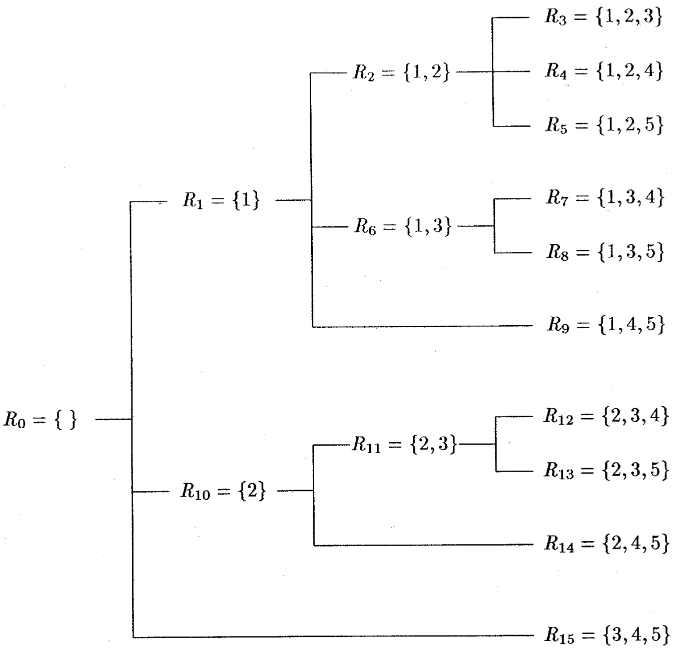

Fig. 4. A depth-first search tree.

find a set R with 3 elements and $D(A_n \cup R) < 0$. The depth-first search tree in this case is shown in Fig. 4, where the subscript of R denotes the search sequence. If for any non-leaf set R_i in the tree $w(A_n \cup R_i, n) \geq 0$, then it is not necessary to search the children sets of R_i. For example, if $w(A_n \cup R_1, n) \geq 0$, then the next set to examine would be R_{10}, skipping all the sets between them.

We now present an example of the optimal rollback recovery problem using the results of damage assessment in Figs. 2 and 3.

EXAMPLE. Consider the system represented by D1 in Fig. 1. Let $S = \{v_2, 0\}$ and the total global restart overhead be 1,200. The rollback distances of all recovery points are tabulated in Table 3. We assume that there are only six recovery points for each module, so $C_i = 6\ \forall i$. Each module's probability of successful rollback is obtained from Figs. 2 and 3. Results obtained from applying Algorithms **RB** and **BB** are summarized in Table 4.[1] Note that if the faulty module and the error-detecting module are farther apart, the mean overhead will generally be higher because modules' contaminating probabilities are usually larger. Another interesting observation is that we have yet to discover a case where **BB** finds an S^*. It appears that, in most cases, if O cannot be reduced by incrementing any single legitimate k_i, it will not be reduced by incrementing a subset of k_i. Therefore, using the bounding function $w(A, n)$ becomes very beneficial, since it can speed up the termination of **BB**.

The optimal recovery points obtained in the above example produce a much larger total rollback distance than what other optimistic approaches would have produced. But, in view of the facts that

1) detection mechanisms are imperfect, and
2) checkpoints that may have been contaminated are considered,

our solution should outperform other optimistic approaches in the sense of minimizing the mean recovery overhead.[2] Actually, our solution will be identical to an optimistic solution if the detection mechanisms in every module are perfect.

4 Damage Assessment for Optimistic Rollback

Many rollback recovery schemes have been proposed for distributed systems over the last decade or so, but none of them have considered corrupted checkpoints and damage assessment. Thus, the focus of this section is to show how to integrate damage assessment with existing checkpointing schemes, especially the optimistic message logging and checkpointing schemes such as the one proposed by Johnson and Zwaenepoel [18].

In general, any rollback recovery scheme equipped with damage assessment should comply with the following steps after detecting an error.

1) *Damage assessment*: Using the algorithms for determining the optimal rollback points as derived in the previous section, compute a set of checkpoints which are safe to rollback to, while considering the overall overhead.
2) *Cancellation of corrupted messages*: All checkpoints established after the set of safe checkpoints are considered to have been corrupted. Hence, all messages sent after a corrupted checkpoint are also corrupted and should be removed from the receiver's stable storage if they have been logged. Furthermore, all messages sent after the receipt of a corrupted message are considered corrupted and must be canceled as well. The messages sent between a safe checkpoint and its next checkpoint are considered correct if they have not been canceled. In some rare event, rollback may propagate from a safe checkpoint if one of the messages received before the establishment of the safe checkpoint has been cancelled.
3) *Search for a recoverable system state*: The system will likely be in a recoverable state after canceling all corrupted messages, since most messages would have been logged if they can survive the cancellation. If not, a new recoverable system state must be determined from the current state.
4) *Recovery*: Roll back to the derived recoverable system state. The system will recover successfully if this system state is indeed not corrupted as predicted in the damage assessment.

In Johnson and Zwaenepoel's scheme [18], a search algorithm, *FIND_REC*, based on a known recoverable state, is proposed to check if a new state developed afterwards is a recoverable system state. The initial state of the system is obviously recoverable. Since then, whenever the system

1. The data for the case of v_5 being the faulty module are obtained by simulation similarly to the other two cases.

2. Direct comparison between our solution and others is not possible, though, because none of the others considered these two facts.

TABLE 3
ROLLBACK DISTANCES OF ALL RECOVERY POINTS

$d_i(k)$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
$i = 1$	5	20	45	70	85	99
$i = 2$	15	21	35	68	74	98
$i = 3$	10	20	30	55	80	92
$i = 4$	12	30	52	65	77	92
$i = 5$	20	39	50	73	86	98

TABLE 4
THE OPTIMAL ROLLBACK POINTS OF ALL MODULES

Faulty module	Optimal recovery points					Mean overhead
	k_1	k_2	k_3	k_4	k_5	
v_1	6	5	4	5	2	401
v_2	2	6	4	4	1	334
v_5	1	6	6	3	6	388

moves into a new state, *FIND_REC* is called in to determine if it is recoverable. If the new state is recoverable, it becomes the latest recoverable system state, replacing the previous one. Thus, a series of recoverable system states will be derived during normal operation and can be stored in stable storage. Later, when an error is detected, this knowledge can greatly simplify the search for the latest recoverable system state before the current state is established by damage assessment and message cancellation. The *FIND_REC* algorithm can be invoked here using one of known past recoverable system states closest to the current state as the base.

Damage assessment can be integrated differently into an existing recovery scheme. However, the problem of finding the optimal checkpoints in the previous section will have to be reformulated under different assumptions and constraints. For example, a more efficient damage assessment for Johnson and Zwaenepoel's scheme [18] can take advantage of the series of recoverable system states derived during normal operation. Instead of rolling back to any combination of checkpoints, we need to evaluate only the success rate and the overhead of rolling back to these past recoverable system states. The optimization is much simpler and there is no need for canceling messages and searching for a new recoverable state.

The main difficulty of integrating damage assessment into an existing rollback recovery scheme is the size of stable storage. Without considering error propagation, whenever a checkpoint becomes part of a recoverable system state, all messages received before the checkpoint can be safely discarded from the stable storage. With damage assessment, however, checkpoints and messages must be kept in the stable storage for a longer period, thus requiring a larger stable storage. This is the cost one has to pay instead of using more expensive (both in time and resource) error detection mechanisms to enhance the coverage and thus remove the need for damage assessment.

5 CONCLUSION

Checkpointing and rollback recovery for concurrent processes have received considerable attention. Two crucial problems which must be resolved are the problems of rollback propagation (or the domino effect) and error propagation. All previous work focused on solving the rollback propagation problem and avoided the error propagation problem by assuming detection mechanisms to be perfect so that all checkpoints are correct. However, it is practically impossible to make error detection perfect and, thus, a checkpoint may be incorrect if it is established after the process became erroneous. These contaminated checkpoints will lead to unsuccessful rollback recovery. In this paper, we have developed a method of damage assessment to handle the error propagation problem, based on our earlier work on error propagation and fault location. Using the results of damage assessment, we formulated and solved the problem of determining the optimal rollback points by minimizing the average total recovery overhead. Integration of damage assessment with existing recovery schemes is discussed using an optimistic message logging and checkpointing scheme as an example.

It is difficult to compare our approach directly with the previous work because the assumptions and the performance criteria are quite different. All previous work assumes perfect acceptance tests so that error propagation is bounded by the recovery points and the success of rollback recovery is guaranteed if proper rollback points are selected. The performance of a rollback recovery scheme is judged by the overhead of a successful rollback. By contrast, we assume that rollback recovery may fail under certain circumstances since any recovery point could be contaminated by error propagation due to imperfect coverage of acceptance tests. We have to resort to an alternative recovery scheme, such as a global restart should rollback recovery fail. The rollback points determined in this paper is thus to minimize the *expected* overhead taking into consideration the overhead of the rollback recovery and the alternative scheme, and the probability of successful rollback recovery.

The damage assessment model developed in this paper can also be used to solve other problems, such as the scheduling of periodic diagnostics and the determination of optimal inter-checkpoint intervals. These problems are matters of our future inquiry.

ACKNOWLEDGMENTS

The work reported here was supported in part by the U.S. Office of Naval Research under Grant N000-14-94-1-0229, and by the U.S. National Science Foundation under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] B. Randell, P.A. Lee, and P.C. Treleaven, "Reliability Issues in Computing System Design," *Computing Surveys*, vol. 10, pp. 123-165, June 1978.
[2] P.M. Merlin and B. Randell, "State Restoration in Distributed Systems," *Digest of Papers, FTCS-8*, pp. 129-134, June 1978.
[3] R.E. Strom and S. Yemini, "Optimistic Recovery in Distributed Systems," *ACM Trans. Computer Systems*, vol. 3, pp. 204-226, Aug. 1985.
[4] B. Randell, "System Structures for Software Fault Tolerance," *IEEE Trans. Software Eng.*, pp. 220-232, June 1975.
[5] D.L. Russell, "Process Backup in Producer-Consumer Systems," *Proc. Sixth ACM Symp. Operating System Principles*, pp. 151-157, Nov. 1977.
[6] D.L. Russell, "State Restoration in Systems of Communicating Processes," *IEEE Trans. Software Eng.*, vol. 6, pp. 183-194, Mar. 1980.
[7] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic Recovery Schemes for Distributed Processes," *Proc. IEEE Reliability in Distributed Software and Database Systems*, pp. 124-130, July 1981.
[8] W.G. Wood, "A Decentralized Recovery Control Protocol," *Digest of Papers, FTCS-11*, pp. 159-164, June 1981.
[9] K. Kant and A. Silberschatz, "Error Recovery in Concurrent Processes," *Proc. COMPSAC*, pp. 608-614, 1980.
[10] K. Venkatesh, T. Radhakrishnan, and H.F. Li, "Optimal Checkpointing and Local Recording for Domino-Free Rollback Recovery," *Information Processing Letters*, vol. 25, pp. 295-303, July 1977.
[11] G. Ferran, "Distributed Checkpointing in a Distributed Data Management System," *Proc. Real Time Systems Symp.*, pp. 43-49, 1981.
[12] W.H. Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," *Computing Surveys*, vol. 13, pp. 149-183, June 1981.
[13] R. Koo and S. Toueg, "Checkpointing and Rollback-Recovery for Distributed Systems," *IEEE Trans. Software Eng.*, vol. 13, no. 1, pp. 23-31, Jan. 1987.
[14] K.G. Shin and Y.-H. Lee, "Evaluation of Error Recovery Blocks Used for Cooperating Processes," *IEEE Trans. Software Eng.*, vol. 10, no. 11, pp. 692-700, Nov. 1984.
[15] B. Bhargava and S.R. Lian, "Independent Checkpointing and Concurrent Rollback for Recovery—An Optimistic Approach," *Proc. IEEE Symp. Reliable Distributed Systems*, pp. 3-12, 1988.
[16] Y.M. Wang and W.K. Fuchs, "Optimistic Message Logging for Independent Checkpointing in Message-Passing Systems," *Proc. IEEE Symp. Reliable Distributed Systems*, Oct. 1992.
[17] A.P. Sistla and J.L. Welch, "Efficient Distributed Recovery Using Message Logging," *Proc. Eighth ACM Symp. Principles of Distributed Computing*, pp. 223-238, 1989.
[18] D.B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing," *J. Algorithms*, vol. 11, pp. 462-491, 1990.
[19] T.T.-Y. Juang and S. Venkatesan, "Crash Recovery with Little Overhead," *Proc. IEEE Int'l Conf. Distributed Computing Systems*, pp. 454-461, 1991.
[20] A. Lowry, J.R. Russell, and A.P. Goldberg, "Optimistic Failure Recovery for Very Large Networks," *Proc. IEEE Symp. Reliable Distributed Systems*, pp. 66-75, 1991.
[21] K.G. Shin and T.-H. Lin, "Modeling and Measurement of Error Propagation in a Multi-Module Computing System," *IEEE Trans. Computers*, vol. 37, no. 9, pp. 1,053-1,066, Sept. 1988.
[22] T.-H. Lin and K.G. Shin, "Location of Faulty Module in a Computing System," *IEEE Trans. Computers*, vol. 39, no. 2, pp. 182-194, Feb. 1990.
[23] T.-H. Lin and K.G. Shin, "A Bayesian Approach to Fault Classification," *Performance Evaluation Review*, vol. 18, no. 1, pp. 58-66, 1990.
[24] Y.-H. Lee and K.G. Shin, "Optimal Design and Use of Retry in Fault-Tolerant Computer Systems," *J. ACM*, vol. 35, pp. 45-69, Jan. 1988.
[25] T.-H. Lin and K.G. Shin, "An Optimal Retry Policy Based on Fault Classification," *IEEE Trans. Computers*, vol. 43, no. 9, pp. 1,014-1,025, Sept. 1994
[26] K.G. Shin and Y.-H. Lee, "Error Detection Process—Model, Design, and Its Impact on Computer Performance," *IEEE Trans. Computers*, vol. 33, no. 6, pp. 529-540, June 1984.

Tein-Hsiang Lin received a PhD degree from the University of Michigan, Ann Arbor. He is a principle engineer in the Microtec Division of Mentor Graphics Corporation, where he is the system architect of Microtec's Spectra Cross Development tool suites and VRTX real-time kernal. Prior to joining Microtec, Dr. Lin was an assistant professor in the Department of Electrical and Computer Engineering at the State University of New York at Buffalo. He specializes in task scheduling algorithms for hard real-time systems and the design of real-time kernels and cross debuggers for multitasking operating systems. His research interests include real-time computing, fault-tolerant computing, and distributed computing. Dr. Lin is a member of the IEEE Computer Society.

Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. He is a professor and director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor.

From 1978 to 1982, Dr. Shin was a member of the faculty at Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, the Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California at Berkeley, International Computer Science Institute, Berkeley, California, IBM T.J. Watson Research Center, and the Software Engineering Institute at Carnegie Mellon University. He also chaired the Computer Science and Engineering Division, EECS Department at the University of Michigan for three years beginning in January 1991.

Dr. Shin has authored/coauthored more than 400 technical papers (about 160 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing an control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He is the coauthor (with C.M. Krishna) of the textbook *Real-Time Systems* (McGraw-Hill, 1997). In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he received the Research Excellence Award from the University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

Dr. Shin has also been applying the basic research results of real-time computing to multimedia systems, intelligent transport systems, embedded systems, and manufacturing applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. (The latter is being pursued as a key thrust area of the newly established U.S. National Science Foundation Engineering Research Center on Reconfigurable Machining Systems.)

Dr. Shin is an IEEE fellow, was the program chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the general chairman of the 1987 RTSS, the guest editor of the August 1987 special issue on real-time systems of the *IEEE Transactions on Computers*, a program co-chair for the 1992 International Conference on Parallel Processing, and has served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993, was a distinguished visitor of the IEEE Computer Society, an editor of the *IEEE Transactions on Parallel and Distributed Systems*, and an area editor of the *International Journal of Time-Critical Computing Systems*.