Error Detection Process — Model, Design, and Its Impact on Computer Performance

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

Abstract — **Conventionally, reliability analyses either assume that a fault/error is detected immediately as it occurs, or ignore damage caused by imperfect detection mechanisms and error latency, namely, the time interval between the occurrence of an error and the detection of that error.**

In this paper we consider a remedy for this problem. We first propose a model to describe the entire error detection process and then apply the model to the analysis of the impact of error detection on computer performance under moderate assumptions. Error latency is used to measure the effectiveness of detection mechanisms. Due to the presence of error latency, (i) it is possible to have undetected errors at the end of process execution making the computation result unreliable, and (ii) even if all errors were detected before the completion of process, it is required to apply complicated error recovery resulting in considerable computation loss. We have used the model to (1) predict the *probability of producing an unreliable result,* and (2) estimate the *loss of computation* due to fault and/or error. The former can be used as a measure of *lack of confidence* in the computation results whereas the latter is important to the *timing analysis,* particularly for real-time computations. Various error recovery techniques and their associated overheads are considered for the estimation of the computation loss which can be used for analyzing suitability for time-critical applications.

Finally, a design problem associated with the error detection process is discussed and a feasible design space is outlined.

Index Terms — **Computation loss, diagnostics, error detection, latent errors/faults, recovery methods, unreliable results.**

I. Introduction

During the past decade or so, many reliability related models for fault-tolerant computers have been proposed. Usually, in these models, probability distribution functions are employed to describe the occurrence of component or system failure. Then, various measures such as reliability, computation capacity, performability, etc. are estimated and are then used to properly represent system performance. Such approaches, however, do not consider shortcomings in error detection mechanisms and recovery methods. By contrast, we set out in this paper to present results of a study that incorporates detection mechanisms and recovery methods in system performance and reliability.

Consider the property of a fault. An input signal to a computer may cause the fault to induce some errors, or it may simply be unaffected by this fault and produce a correct output. The fault is said to be *latent* if it does not harm normal operations. The time interval between the moments of fault occurrence and error occurrence is called *fault latency*. For an ultrareliable system, a latent fault is a considerable threat since it may cause a catastrophe in the event that more than one latent fault becomes active at the same time. Bavuso *et al.* investigated the problem of the latent fault and proposed experiments to measure the time interval between the moments of fault injection and error detection [1]. Their study indicated that a significant proportion of faults is not detected even after many iterations of a process.

When an error is generated, it is desired that the error detection mechanisms associated with the system identify it immediately. Nevertheless, some errors may not be captured by error detection mechanisms upon occurrence and then spread as a result of the subsequent flow of information. Thus, the damage caused by an error will propagate until it is detected and handled appropriately. See Fig. 1 for a typical error detection process. The delay between the occurrence of an error and the moment of its detection, called *error latency,* is important to damage assessment, error recovery, and establishing confidence in the computation results. This delay has been defined by Courtois as *detection time* [2], [3] and by Shedletsky as *latency difference* [4]. Courtois also presented results of on-line tests of the M6800 microprocessor that included the distributions of detection time for certain detection mechanisms. Shedletsky proposed a technique to evaluate the error latency based on the "fault set" philosophy and the probability distribution of input signals.

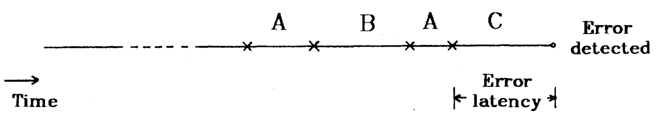
When error latency is significant, there is the possibility of the system putting out incorrect computation results since there may be some undetected errors at the output phase. Also, even if the system detects all errors before the output phase, the computation achieved during the latent period may already have been contaminated, and thus be useless. In practice, error latency is never zero, and in the event of an error the whole system is delayed by the more complicated recovery that is required to remove the contamination that is spread during error latency.

To evaluate these two effects — the probability of producing an unreliable result and the computation loss resulting from error — it is necessary to examine the error detection mechanisms incorporated in computer systems and their respective capabilities. One may then establish a different recovery strategy for the errors captured by each distinct detection mechanism, thus obtaining the most appropriate possible recovery performance and execution cost. To evaluate error handling capability including tradeoffs between various detection mechanisms and recovery methods, it is necessary to consider recovery performance and execution cost, taken as a whole.

In this paper, a model is proposed to describe error detection processes and to estimate their influence on system performance. In the following section, the classification,

Manuscript received July 20, 1983; revised December 10, 1983. This work was supported in part by NASA Grant NAG 1-296.

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



Duration A: A fault exists and active in the system

Duration B: The fault becomes inactive

Duration C: Errors exist in the system

Fig. 1. The error detection process.

properties, and associated recovery methods of error detection mechanisms are discussed. Our model is developed in Section III. Section IV presents the evaluation of the probability of having an unreliable result as well as that of the average computation loss. Also, the design problem of error detection is discussed and a feasible design space is outlined.

It is assumed throughout this paper that faults in hardware components are a potential cause of transition to erroneous states during normal operation. We also assume for the present study that there are no design faults in the system. An *error* is defined to be the erroneous information/data resulting from fault(s).

II. Classification of Error Detection Mechanisms

There are various error detection mechanisms which can be incorporated in a computer system. The basic principle of these mechanisms is the use of redundancy in devices, information, or time. Based on (i) where they are employed, (ii) their respective recovery methods, and (iii) performance measures, error detection mechanisms are divided into the following three categories.

a) Signal level detection mechanisms: Usually, the mechanisms in this category are implemented by built-in self-checking circuits. Whenever an error is generated by a predescribed fault, these circuits detect the malfunction immediately, even if the erroneous signal does not have any logical meaning. Typical methods in this category include error detection codes, duplicated complementary circuits, matchers, etc. The performance of these detection mechanisms is measured by the *coverage,* denoted by c, which is the probability of detecting an error induced by an arbitrary fault. It is difficult to have a perfect coverage because (i) it is prohibitively expensive to design detection mechanisms which cover all types of faults, and (ii) physical dependence between function units and detection mechanisms cannot be completely eliminated.

Since this class of detection mechanisms detects an error immediately upon occurrence, there is no contamination through error propagation. This makes the subsequent recovery operations simple and effective. Two kinds of recovery methods are suitable for this category; one is error masking, in which redundant information is used to retain correctness, the other is retry, in which the previous action is reexecuted.

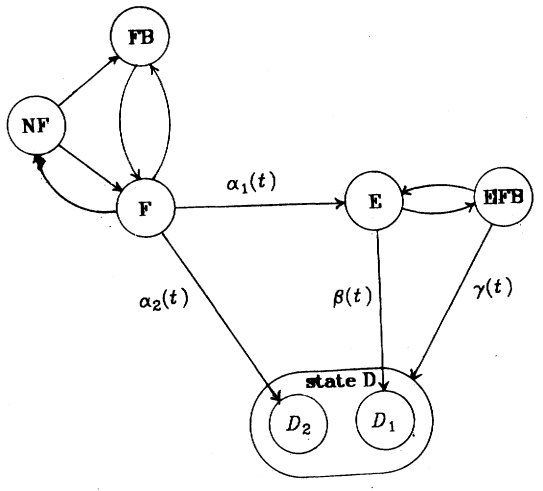
b) Function level detection mechanisms: The detection mechanisms in this category are intended to check out unacceptable activities or information at a higher level than the previous category. Unlike the signal level detection mechanisms, they verify system operations by functional assertions on response time, working area, provable computation results, etc. These detection mechanisms can be regarded as "barriers" or "guardians" around normal operations. After an error is generated by a fault, the resulting abnormality may grow very quickly—the "snowball effect" [3], or "error rate phenomenon" [6]—until it hits the barriers. Several software and hardware techniques such as capability checking, acceptance testing, invalid op-code checking, timeout, and the like can be applied.

The important issues for function level detection mechanisms are error isolation and damage assessment. Both issues depend upon system structure as well as on inherent properties of the executed programs or tasks. When there are clear cleavages between subsystems or subtasks, the effective detection assertions can be easily declared, thus permitting greater error isolation and reducing contamination. Usually, rollback and restart recovery methods are used to rescue failed processes. Rollback requires state restoration such that part of the process can be resumed. The restart method purges the old computation and then reissues the same task to other nonfaulty processors.

c) Periodic diagnostics: This method is usually referred to as off-line testing because the processing unit under test cannot perform any useful task. It is composed of a diagnostic program which supplies imitated inputs such that all existing faults are activated and thus generate errors. Several theoretical approaches have been proposed to determine the probability of finding an error after applying diagnostics for a certain duration (equivalent to the probability of detecting fault as a function of test duration) [7], [8]. Simulation has also been used to study the coverage of self-testing programs [9]. All these results have indicated that the effectiveness of the present category is a monotonically increasing function of testing time. Since the time required for complete testing (i.e., ensuring 100 percent coverage) is in general too long, an appropriate policy of diagnostics is to perform an imperfect test periodically during normal operation and perform a thorough diagnostics when the system is idle.

III. Model of Error Detection Process

For analytical convenience, occurrence of a fault is usually modeled as a Poisson process. Let $MTBF$ be the mean time between two successive fault occurrences. Also, let F_i and p_i $i = 1, 2, 3$ denote the event and the probability that the fault is transient, intermittent, or permanent, respectively. Naturally, $p_1 + p_2 + p_3 = 1$. When the classification of faults into these three types is independent of occurrence of faults, occurrence of event F_i can be modeled as a Poisson process with rate $p_i/MTBF$. Then, the following model can be used for a separate analysis of the effects of each type of faults.



Note: **The transitions between NF, F, FB, and E, EFB are dependent on the type of fault.**

Fig. 2. The model for error detection process.

A. Model Development

Fig. 2 shows our model of the error detection process. The model consists of three parts: the occurrence of a fault, the consequent generation of an error, and the detection of that error. Since the probability of having multiple faults at any time is small, they are excluded from the model.[1] There are six states in the model as follows:

1) **NF** (nonfaulty): In this state no fault exists in the system.

2) **F** (faulty): There is a fault which is active and capable of inducing errors, but there are no errors.

3) **FB** (fault-benign): There is an inactive intermittent fault.

4) **E** (error): There is at least one undetected error in the system and the fault which has caused that error is still present.

5) **EFB** (error-fault-benign): At this state the intermittent fault has become inactive or the transient fault has disappeared after it induced an error.

6) **D** (detection): At this state, the detection mechanisms have identified the error in the system. To determine whether the system has been contaminated or not, two substates, called $\mathbf{D}_1$ and $\mathbf{D}_2$, are included. The system will enter $\mathbf{D}_1$ when the detected error has contaminated at least part of the system. On the other hand, the system enters $\mathbf{D}_2$ when an error is detected before it begins to propagate through the system. Signal level detection and diagnostics cause transitions from **F** to $\mathbf{D}_2$. In fact, these transitions can be divided into two steps: an existing fault induces an error, and the error is detected immediately following its occurrence.

Let λ denote the rate of occurrence of F_i type faults, i.e., $\lambda = p_i/MTBF$ $i = 1, 2, 3$ when transient, intermittent, and permanent faults are separately considered. Since intermittent faults may become inactive, a benign state has to be included in the model. Several models of intermittent faults have been proposed and used for testing and reliability evaluation [10]–[14]. In our model, the transitions between **NF**, **F**, **FB**, and between **E**, **EFB** are used to describe the behavior of intermittent faults. For transient and permanent faults, **FB** does not exist, implying that the transition rates between **F** and **FB**, μ and ν, are zero. Similarly, for intermittent and permanent faults the rate of transition from **F** to **NF**, τ, equals zero.

Consider the process of generating errors by a given fault. With the assumption that the signal patterns of successive inputs are independent, Shedletsky treated the period of fault latency as a random variable with a composite geometric distribution for discrete inputs or cycles [8]. Using the concepts of information theory, Agrawal presented a formula to estimate the probability of inducing error [15]. For tractability we have assumed in our model an exponentially distributed fault latency with rate α when a task is executing. While the diagnostic program is running, the transition duration **F** to $\mathbf{D}_2$ is assumed to be exponentially distributed with parameter ω. If the diagnostic program is executed for period t_p following a normal operation period t_n and a process swapping period t_v as shown in Fig. 3, the coverage of a single diagnostic, denoted by ξ, is equal to $1 - e^{-\omega t_p}$ for each execution of diagnostics.

Once the system enters **E**, the erroneous information starts to spread until function level detection mechanisms identify any unacceptable result. There are two paths to $\mathbf{D}_1$ and they represent transition rates of $\beta(t)$ and $\gamma(t)$, respectively. At state **E**, since the fault still exists, it is possible that the fault is captured by signal level detection mechanisms or diagnostics *prior to* the function level error detection. We exclude this case from the model because the process has already become erroneous, and the subsequent signal level detection has no effect on this error. (Namely, a direct transition from **E** to $\mathbf{D}_2$ is not included.) It is also possible that there are multiple errors induced by the same fault or by an old un-

[1]See Section V for a brief comment on the modeling of multiple faults.

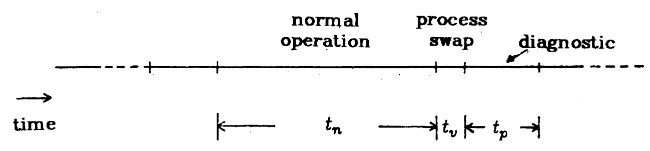


Fig. 3. A cycle of periodic diagnostics.

detected error when the system is in **E** or **EFB**. The function level detection mechanisms will recognize that the system is erroneous regardless of which error is captured. However, the error latency must be measured from the moment that the first error occurs.

B. Mathematical Description of State Transitions

Let a computer system incorporate the three types of error detection mechanisms discussed above. For notational convenience number the states **NF, F, FB, E, EFB, D**$_1$, **D**$_2$ with i for $i = 1, 2, \cdots, 7$. Then one can obtain a transition probability matrix $\mathbf{H}_{7\times7}(t)$ by making use of the model in Fig. 2.

$$H_{7\times7}(t) = \begin{bmatrix} -\lambda & \frac{\lambda\nu}{\mu+\nu} & \frac{\lambda\mu}{\mu+\nu} & 0 & 0 & 0 & 0 \\ 0 & -(\mu+\tau+\alpha_1(t)+\alpha_2(t)) & \mu & \alpha_1(t) & 0 & 0 & \alpha_2(t) \\ 0 & \nu & -\nu & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -(\mu+\tau+\beta(t)) & \mu+\tau & \beta(t) & 0 \\ 0 & 0 & 0 & \nu & -(\nu+\gamma(t)) & \gamma(t) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (1)$$

Since the diagnostic is invoked periodically, transition rates $\alpha_1(t)$, $\alpha_2(t)$, $\beta(t)$, and $\gamma(t)$ are the following functions of time:

$$\alpha_1(t) = \begin{cases} (1-c)\alpha & \text{if } n(t_n+t_p+t_v) < t \le n(t_n+t_p+t_v)+t_n \\ 0 & \text{otherwise} \end{cases} \quad (2a)$$

$$\alpha_2(t) = \begin{cases} c\alpha & \text{if } n(t_n+t_p+t_v) < t \le n(t_n+t_p+t_v)+t_n \\ \omega & \text{otherwise} \end{cases} \quad (2b)$$

$$\beta(t) = \begin{cases} \beta & \text{if } n(t_n+t_p+t_v) < t \le n(t_n+t_p+t_v)+t_n \\ 0 & \text{otherwise} \end{cases} \quad (2c)$$

$$\gamma(t) = \begin{cases} \gamma & \text{if } n(t_n+t_p+t_v) < t \le n(t_n+t_p+t_v)+t_n \\ 0 & \text{otherwise} \end{cases} \quad (2d)$$

where c is the coverage of the signal level detection; α is the transition rate that a fault generates an error; β and γ represent the transition rates that the function level detection captures errors in states **E** and **EFB**, respectively; and n is a positive integer.

Hence, the state probabilities $\pi(t) = [\pi_1(t), \pi_2(t), \cdots, \pi_7(t)]$ can be obtained by solving the following differential equation:

$$\frac{d\pi(t)}{dt} = \pi(t)\mathbf{H}(t); \qquad \pi(0) = \pi_0 \quad (3)$$

where $\pi_i(t)$ is the probability that the system is in state i at time t. Because of the absorbing property of **D**$_1$ and **D**$_2$, one can easily see that $\pi_6(\infty) + \pi_7(\infty) = 1$.

Assume the initial state that the system begins with is **NF**. When a transient or a permanent fault occurs, the system will enter the nonfaulty state again after either the fault disappears or the system is reconfigured to eliminate the source of the fault.

In case of an intermittent fault, it is possible for the system to be in **FB** instead of **NF** even after some recovery procedures are successfully applied. For example, when the fault becomes benign during the retry recovery, the system enters **FB**. Let S_1 (or S_2) be the event that the system is in state **NF** (or **FB**) after recovery from an intermittent fault. This process can be represented by a Markov chain shown in Fig. 4 and the transition probabilities between S_1 and S_2, denoted by δ_1 and δ_2. These transition probabilities are computed using (3) and the corresponding recovery performance will be discussed in the next section. Note that, under S_2, the same intermittent fault will be detected by the signal level detection with probability one if it induces an error again.

A task may start execution when the system is in any one of **NF, F, FB, E, EFB** (but certainly not in **D**$_1$, **D**$_2$,). Using the Markov model in [16], we can calculate (i) the mean number of visits to state i, $i = 1, 2, \cdots, 5$, before the system is absorbed into **D**$_1$ or **D**$_2$ for every F_j $j = 1, 2, 3$, and (ii) the mean time interval, $E[X_i | F_j]$ $j = 1, 2, 3$, during which the system stays in state i before transition to D_1 or D_2 takes place. Then, the probability that a task begins execution when the system is in state i is formulated as follows:

$$\pi_i(0 | F_j) = \begin{cases} E[X_i | F_j] / \sum_{k=1}^{5} E[X_k | F_j] & \text{for } i = 1, 2, \cdots, 5 \\ 0 & \text{for } i = 6, 7. \end{cases} \quad (4)$$

It may be possible that the active duration of an intermittent fault increases every time it becomes active following its first occurrence. This would imply that the transition

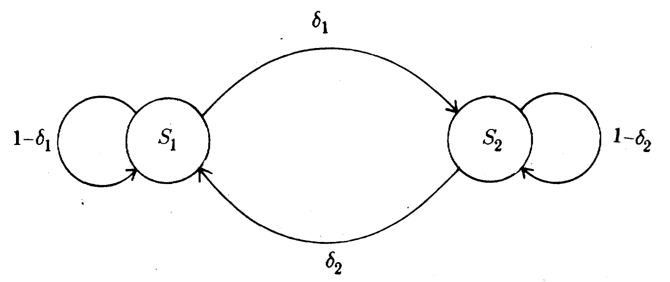


Fig. 4. A Markov chain for the recovery from an intermittent fault.

rates between fault active and fault benign depend on the duration for which an intermittent fault exists. In such a case, the model suggested in [17] can be used in the above system equations.

It cannot be overemphasized that our modeling of the error detection process is intended to evaluate the effects of various detection mechanisms on task execution. This fact is in sharp contrast to most conventional methods in which models have been developed and then used to estimate the system reliability or to determine the coverage of failure. For example, in CARE III [14] the error propagation rate is defined by the user and the model is applied to determine the coverage.[2] Note that a transition to $\mathbf{D}_1$ represents the detection of error by function level detection mechanisms, whereas $\mathbf{D}_2$ is reachable directly from $\mathbf{F}$ by signal level detection mechanisms or diagnostics. The impacts of detection mechanisms on task execution will be reflected through (3) and the state distribution $\pi(t \mid F_j)$.

IV. Analysis and Design of Error Detection Process

In case of imperfect coverage (i.e., $c < 1.0$) in the signal level detection, and nonzero error latency in the function level detection, the system will suffer from the following two undesirable effects: one is the possibility of putting out potentially erroneous results because the system is unaware of the existence of error; the other is the additional recovery overhead resulting from error propagation through the system during error latency. With the model proposed in the previous section and moderate assumptions regarding error recovery, we will in this section analyze these two effects and then use them to specify the requirements for design of error detection.

We used $p_1 = 0.5$, $p_2 = 0.4$, and $p_3 = 0.1$ in simulating the impacts of error detection process on computer performance. This selection is for a numerical purpose only, and thus is arbitrary. The choice of these values does not alter the validity of the method developed here.

A. Estimation of the Probability of Producing an Unreliable Result

The execution of a task consists of parallel and/or serial execution of processes. We can always partition the task into processes in such a way that every process receives all the input data at the beginning of its execution and sends the computation result to its successors at the end of execution. A serious situation, namely the propagation of erroneous information through the system, appears if an error occurs and cannot be discovered before the end of execution. For convenience, let us define an *unreliable result* as follows.

Definition: If there exists at least one error at the moment of process completion and if the system is at that moment still unaware of the presence of that error, the process is said to end with an *unreliable result.*

An unreliable result may even include the cases of producing wrong and/or no outputs. On the other hand, it may yield a correct output despite the presence of error if the computation is not contaminated by the error. However, the result cannot be trusted owing to the presence of an undetected error at the moment of output. (No one would have much confidence in the computation result under this circumstance!) It is therefore important to estimate the probability of producing an unreliable result, denoted by p_e, as a measure of *lack of confidence* in the computation result.

Let T denote the execution time of a process. If T is deterministic, p_e is given by $p_e = \sum_{j=1}^{3} p_j\{\pi_4(T \mid F_j) + \pi_5(T \mid F_j)\}$, which is the probability that the system is in $\mathbf{E}$ or $\mathbf{EFB}$ at the moment of process completion. When T is a random variable with density function $f_T(t)$, then p_e becomes $p_e = \int_0^\infty \{\sum_{j=1}^{3} p_j[\pi_4(t \mid F_j) + \pi_5(t \mid F_j)]\} f_T(t)\, dt$.

When a diagnostic is scheduled periodically for the process, the resulting p_e becomes a function of the time interval between the output moment and the time the previous diagnostic has run. The shorter this time interval, the more reliable the computation result. However, because of the uncertainty of the process execution time, it is difficult to schedule periodic diagnostics so that the system is tested just before the process moves into the output phase. Here, using the proposed model, we can compute the maximum value of p_e, denoted by $\max(p_e)$, which occurs when the time interval between the process completion and the last diagnostic is equal to t_n. Observe that $1 - \max(p_e)$ represents the lower bound of confidence (or *sure confidence*) in the computation results and thus can be used for design specifications. Some simulation results are graphed in Fig. 5 and 6. In Fig. 5, $\max(p_e)$ starts to decrease sharply only when each diagnostic has a higher coverage ($\xi \geq 0.95$). In Fig. 6, we compare three different cases: (i) with periodic diagnostics and $c = 0.6$, (ii) with periodic diagnostics and $c = 0.8$, and (iii) with periodic diagnostics, $c = 0.6$, and doubled function level detection rates. From the model, we can observe that $\max(p_e)$ is linearly related to the coverage of the signal level detection and varies exponentially with respect to the function level detection capability. However, perfect coverage and zero error latency are impossible to attain in practice. Thus, the combination of both the signal level and the function level detection mechanisms have to be used to reduce p_e.

[2]Consequently, even though there are similarities between our model and CARE III, their purposes are quite different.

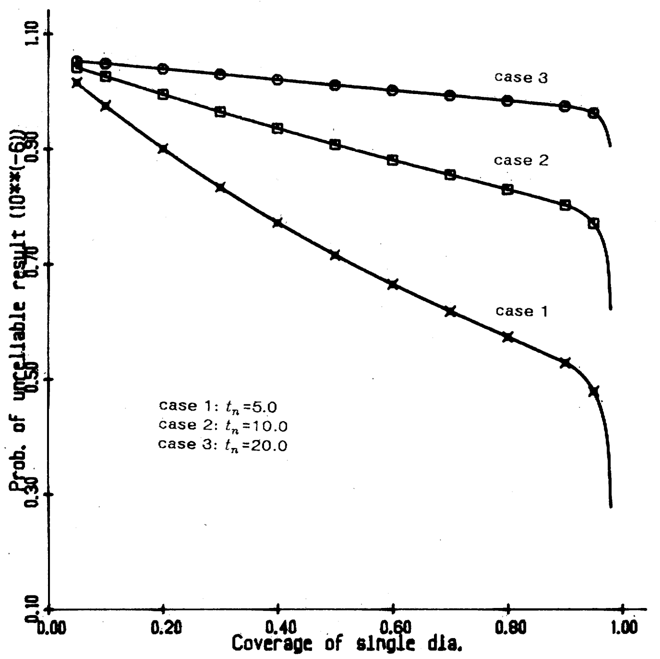


Fig. 5. Max(p_e) versus the coverage of a single diagnostic ξ ($\lambda = 10^{-6}$, $\mu = 0.2, \nu = 0.1, \tau = 0.2, \alpha = 0.2, \beta = 0.5, \gamma = 0.1, \omega = 20.0, c = 0.6, T = 100$).

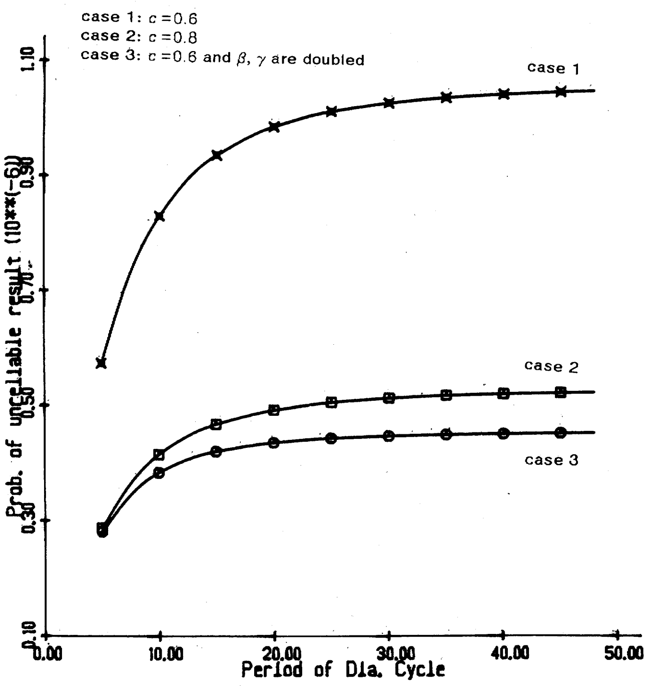


Fig. 6. Max(p_e) versus t_n ($\lambda = 10^{-6}, \mu = 0.2, \nu = 0.1, \tau = 0.2, \alpha = 0.2, \beta = 0.5, \gamma = 0.1, \omega = 20.0, \xi = 0.8, T = 100$).

impact of these detection mechanisms on computer performance: *computation loss* and *execution cost*. Computation loss—a system-oriented view—is represented by the amount of time used for error handling, whereas execution cost—a task-oriented view—shows the effect of error detection and recovery on a particular task in the event that an error is detected during its execution. After the detection of an error, one may use one of several recovery methods to rescue the executing process. Recovery strategies usually

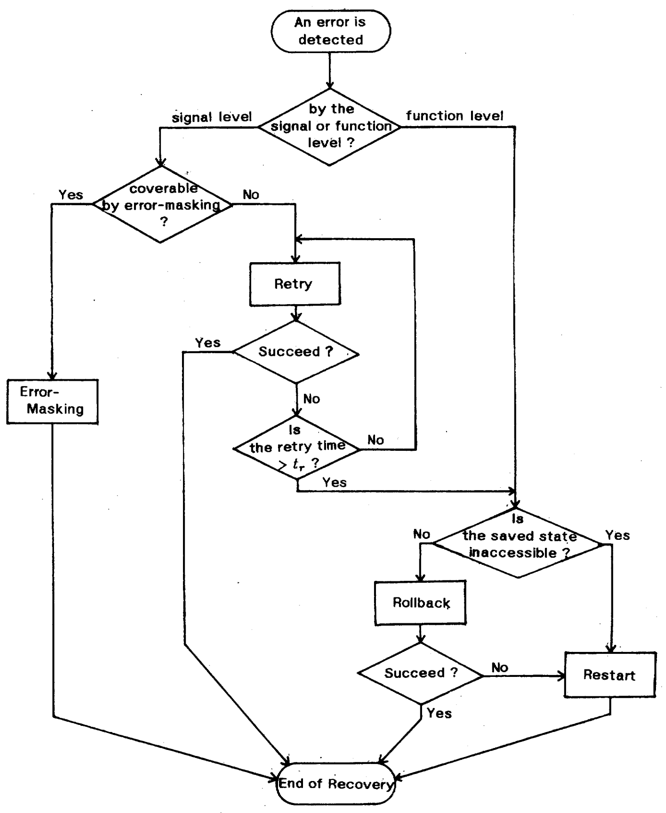


Fig. 7. The flowchart of recovery processes.

depend on the detection mechanisms and the fault/error types.

The overhead and efficiency associated with these recovery methods are briefly discussed in the sequel.[3]

1) Recovery Strategies and Their Respective Overheads: If an error is detected by a detection mechanism, rollback or restart can always be applied to recover the process from the error. It is, however, possible to use masking or retry if the error is captured by signal level detection mechanisms. Fig. 7 illustrates four recovery strategies, their applications, and their application precedence when multiple strategies are used to recover from a single error. In Fig. 8, a probabilistic flow diagram between these recovery methods is presented.

Note that a transient fault may not induce any error before its disappearance. The probability of having an error, given the occurrence of fault, is $P(E) = \alpha p_1/(\alpha + \tau) + p_2 + p_3$. Let $R_{i,j}$ and $\rho_{i,j}$ represent, respectively, the mean overhead and the probability that the ith recovery method is applied to recover from an error which is generated by F_j, where $i = 1, 2, 3, 4$ for masking, retry, rollback, and restart, respectively. We also define $\theta_{i,j}$ as the conditional probability that the process is recovered, given that the ith recovery method is used when F_j occurs. Let p_j' be the probability that F_j has occurred, given that an error is detected. Expressions of p_j' $j = 1, 2, 3$ are listed in the first row of Table I. We can use Fig. 8 to represent the mean total overhead of recovery

[3]This discussion is not intended to present a complete detail of error recovery since it is out of the scope of the present paper. Instead, it is geared toward the analysis of the effects of the error detection process and its associated recovery on computer performance.

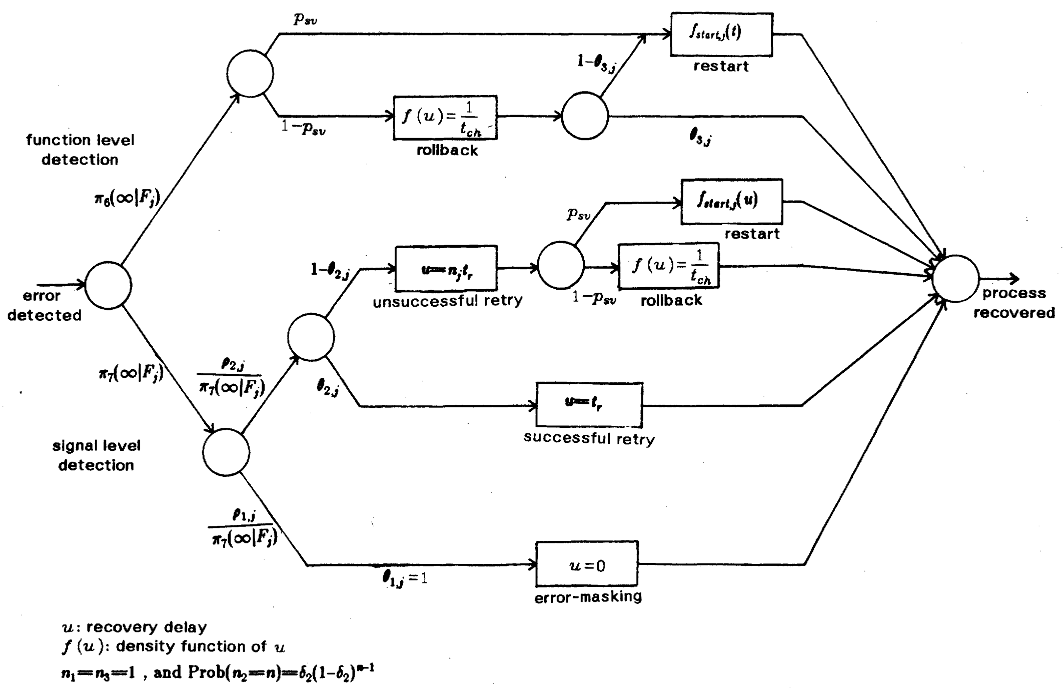


Fig. 8. Probabilistic flow diagram of recovery processes.

TABLE I
MATHEMATICAL EXPRESSIONS OF VARIOUS PARAMETERS RELATED TO THE THREE DIFFERENT TYPES OF FAULTS

	F_1 (transient fault)	F_2 (intermittent fault)	F_3 (permanent fault)
p_j'	$\frac{p_1 PE_1}{P(E)}$	$\frac{p_2}{P(E)}$	$\frac{p_3}{P(E)}$
$\rho_{2,j}$	$(1-\rho_1)\frac{\pi_7(\infty\|F_1)}{PE_1}$	$(1-\rho_1)\pi_7(\infty\|F_2)$	$(1-\rho_1)\pi_7(\infty\|F_3)$
$\theta_{2,j}$	$1-e^{-\tau t_r}$	0	0
$R_{2,j}$	t_r	$\frac{t_r}{\delta_2}$	t_r
$\rho_{3,j}$	$(1-p_{sv})\left\{\frac{\pi_6(\infty\|F_1)}{PE_1}+\rho_{2,1}(1-\theta_{2,1})\right\}$	$(1-p_{sv})\{\pi_6(\infty\|F_2)+\rho_{2,2}(1-\theta_{2,2})\}$	$(1-p_{sv})\{\pi_6(\infty\|F_3)+\rho_{2,3}(1-\theta_{2,3})\}$
$\theta_{3,j}$	$1-\frac{\pi_6(\infty\|F_1)}{PE_1}D_1$	$1-\pi_6(\infty\|F_2)D_2$	$1-\pi_6(\infty\|F_3)D_3$
$R_{3,j}$	$t_b+\frac{t_{ch}}{2}$	$t_b+\frac{t_{ch}}{2}$	$t_b+\frac{t_{ch}}{2}$

Where $D_j = \int_0^{t_{ch}} \frac{1-p_{46}(t|F_j)}{t_{ch}}\,dt$ and $PE_1 = \frac{\alpha}{\alpha+\tau}$.

$RT = \sum_{j=1}^{3} p_j'(\sum_{i=1}^{4} \rho_{i,j}R_{i,j})$ for every error detection.

a) Error masking: Most error masking methods employ error-correcting codes in data transfer, memory, and arithmetic units. Error masking is the most efficient recovery method when it can be applied successfully. In fact, we can regard in this case that the error has never occurred since the system still provides correct results despite the occurrence of error. Thus, one can assume $R_{1,j} = 0$, i.e., zero recovery overhead, and $\theta_{1,j} = 1$. The probability that error masking is used, $\rho_{1,1} = \rho_1\pi_7(\infty|F_1)(\alpha/(\alpha+\tau))$ and $\rho_{1,j} = \rho_1\pi_7(\infty|F_j)$ for all $j = 2, 3$, depends on the conditional probability that error occurs due to the faults in the units with error correcting code and can be corrected by the error correcting code, given that the error occurs. Here, ρ_1 denotes the coverage by error masking mechanisms.

b) Retry recovery: Retry can be attempted at various levels, e.g., at the levels of microinstruction, instruction, or I/O metaoperations. Retry is useful when the error has not propagated yet at the time of detection. Reexecutions of the same operation can produce a correct result only if the related fault is transient or intermittent and disappears during retry. Ideally, the system should apply retry recovery until the fault disappears if it is transient with a short active duration. For intermittent or permanent faults, retry recovery is not helpful. However, after the detection of error by signal level detection mechanisms, it is very difficult, if not impossible, to tell the type of fault. Moreover, if it is transient, it is impossible to predict when the fault will disappear.

Due to the above reasons, assume the system will retry automatically for a fixed duration t_r upon detection of an error by the signal level detection. Then, we can obtain mathematical expressions of $\rho_{2,j}$, $\theta_{2,j}$, and $R_{2,j}$ for $j = 1, 2, 3$ as listed in Table I.

Recall that for an intermittent fault, when S_2 occurs, the same fault will be detected again by the signal level detection and retry recovery will be followed. Thus, there are $1/\delta_2$ retries on the average among which application of the last retry will be unsuccessful. In case of intermittent faults the transition probabilities, δ_1, δ_2, between S_1 and S_2 are expressed as follows:

$$\delta_1 = \pi_7(\infty \mid F_2)(1 - \rho_1)(1 - e^{-\mu t_r}) \tag{5}$$

$$\delta_2 = e^{-\mu t_r}. \tag{6}$$

From (5) and (6), it is easy to see that although it is simple and practical, the above retry method is not intelligent. It may be more desirable to design a retry mechanism which can recognize the intermittent nature of the fault following several consecutive successful retries for the same fault. In such a case, δ_2 gets larger, or would become unity if the retry mechanism is perfect.

c) Rollback recovery: Rollback recovery can be regarded as a type of retry which needs to save process states during normal operation. When an error is detected, the process rolls back to one of the previously saved states. The original idea of rollback recovery is accommodated with acceptance tests for software reliability [18]. Here, for rollback recovery we assume periodic insertion of checkpoints such that the process can be resumed at any one of these checkpoints [19], [20]. Let t_{ov} and t_{ch} be, respectively, the overhead for saving states and the interval between two adjacent checkpoints. Then, the percentage of the overhead for establishing checkpoints is $t_{ov}/(t_{ov} + t_{ch})$. Note that rollback recovery fails if the states saved are destroyed by a fault, or if the states are contaminated by error (e.g., due to the presence of error during the state saving).

The time lost in rollback recovery is the sum of the computation undone and the setup time[4] for rollback t_b. When we consider the reoccurrence of error during recovery, it is extremely difficult to determine this time loss. However, when the fault occurrence rate is very small (typically 10^{-6} per second for the IC's manufactured today), we can assume no error occurrence during rollback. We also assume that only the most recently saved state is kept in order to minimize the storage requirements for checkpoints. Then, the time loss in computation simply becomes the interval between the moment of the last state saving and that of the error occurrence which cannot be recovered by error masking or retry.

[4]The setup times for both rollback and restart recoveries are needed for hardware reconfiguration and software initialization. The hardware reconfiguration is to eliminate the source of error [i.e., fault(s)] for the resident process in the faulty module.

Since the *MTBF* is in general much greater than the inter-checkpoint interval t_{ch}, one can assume that the occurrence of rollback recovery is uniformly distributed within the inter-checkpoint interval, given that it is applied. Let p_{sv} be the probability that the saved state becomes inaccessible or unusable and $p_{46}(t \mid F_j)$ be the probability distribution function of error latency for fault type F_j, i.e., the probability that the system is in $\mathbf{D}_1$ at time t when the system starts from $\mathbf{E}$. $p_{46}(t \mid F_j)$ is equal to $\pi_6(t \mid F_j)$ in (3) when

$$\pi(0) = [0, 0, 0, 1, 0, 0, 0].$$

Then, we obtain $\rho_{3,j}$, $\theta_{3,j}$ and $R_{3,j}$ as listed in Table I.

d) Restart recovery: When restart recovery is applied, the whole process is reexecuted from the beginning to recover from an error. Since the system can be reconfigured to replace the faulty component, restart recovery will eventually succeed as long as there are enough resources to replace faulty components. Hence, we can represent that $\theta_{4,j} = 1$ and $\rho_{4,j} = 1 - \rho_{1,j} - \rho_{2,j}\theta_{2,j} - \rho_{3,j}\theta_{3,j}$. The time wasted in each restart is the sum of the setup time for reconfiguration and reinitialization, and the time of error detection T_d measured from the beginning of process execution. For simplicity, we assume that the moment of restart recovery is uniformly distributed within the task execution period. Thus, the density function of the overhead involved in restart, $f_{\text{start},j}(t)$ is equal to $1/T$ for $t_s \le t \le T + t_s$, and $R_{4,j} = t_s + T/2$ where t_s is the setup time for restart. Details of the effects on task execution time by successive restarts can be found in [21], [22].

2) Calculation of Computation Loss and Execution Cost: Now with the preceding overhead analyses, consider the computer time that is used for actual computation instead of error handling. The average computation loss due to a single error detected, denoted by *CL*, has to include the overheads due to periodic diagnostics, periodic insertion of checkpoints, and recovery in the event of error. Define η as the percentage of the average computation loss for each error detection, which is expressed by

$$\eta \approx \frac{CL}{1/(\lambda P(E))} = \sigma + \lambda P(E) \sum_{j=1}^{3} p_j' \left(\sum_{i=1}^{4} \rho_{i,j} R_{i,j} \right) \tag{7}$$

where $1/(\lambda P(E))$ is an approximate mean time between two successive error detections, and σ is the percentage loss due to periodic diagnostics and insertion of checkpoints and is given by

$$\sigma = \frac{t_p + t_v}{t_n + t_p + t_v} + \frac{t_{ov}}{t_{ov} + t_{ch}}.$$

The above equation indicates that the time wasted for executing periodic diagnostics and checkpointing is a dominating factor in the total computation loss when the system is highly reliable (i.e., the system has a small λ). In Fig. 9, plotted are the simulation results for the percentage of the total computation loss η, and the mean loss in recovery *RT*. The reduction in recovery loss by periodic diagnostics is small because (i) the diagnostic is useful only if it can capture faults before they induce errors, and (ii) the diagnostic is incapable of detecting an intermittent fault when the fault is

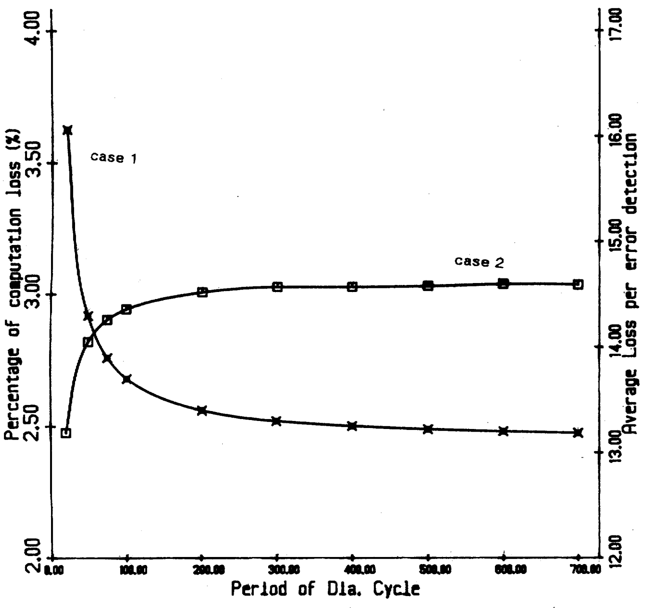


Fig. 9. The effects of periodis diagnostics on percentage of total loss (case 1) η, and total recovery loss (case 2) RT ($\lambda = 10^{-6}$, $\mu = 0.2$, $\nu = 0.1$, $\tau = 0.2$, $\alpha = 0.2$, $\beta = 0.5$, $\gamma = 0.1$, $\omega = 20.0$, $\xi = 0.8$, $c = 0.6$, $T = 100$).

inactive; (iii) even if the diagnostic identifies a fault, the system still has to reconfigure or retry to eliminate this fault. Detection mechanisms other than on-line diagnostics are more advantageous due to their favorable effects and overheads on the computer performance.

Observe that this time loss is related to the system, not to tasks to be executed on the system. One can therefore regard this as the *system overhead*. On the other hand, tasks executing on the system may suffer from delays in execution due to error detection and recovery overhead which are *task specific*. When a task is time critical, the delay in its execution may cause a catastrophe (e.g., loss of human lives, economic and social disaster, etc.) if the execution is not completed within a specified time limit called *hard deadline*, denoted by t_{dead}. This was termed *dynamic failure* in [23], [24]. Also, the running cost—the cost for use of computer as well as controlling an actual system which uses the computed results—will certainly go up with the increase of the execution delay. In case of error, based on Fig. 8, we can write the probability density function of the execution delay due to the recovery from an F_j type fault, $f_r(t \mid F_j, T)$, where T is the needed time for task completion under a fault-free condition. These density functions are listed in Appendix A. Note that for intermittent faults the task may be completed with successful retries. In the expression for $f_r(t \mid F_2, T)$ given in Appendix A, for simplicity we used the upper bound of error handling delay; that is, whenever an error occurs, the task completion is achieved with rollback or restart recovery.

Since the overhead associated with checkpointing and diagnostics has to be included, the time needed for task execution under the fault-free condition becomes $\hat{T} = (1 + \sigma)T$. For any computation process, the delay in execution may induce an extra cost. For example, in real-time applications this cost may be the additional energy or fuel used for the controlled system, the consequence of longer response time, etc. Given a cost function for the execution time t, $C(t)$, which is a monotonic nondecreasing function (see [23], [24] for an example of its detailed derivation), we can obtain the total execution cost $COST$ and the probability of dynamic failure p_{dyn} as below.

$$COST = \sum_{j=1}^{3} p_j \int_{\hat{T}}^{\infty} C(t) f_r(t \mid F_j, \hat{T})\, dt \tag{8}$$

$$p_{\text{dyn}} = \sum_{j=1}^{3} p_j \int_{t_{\text{dead}}}^{\infty} f_r(t \mid F_j, \hat{T})\, dt. \tag{9}$$

C. Design Considerations for Detection Mechanisms

Consider the performance and reliability measures p_e, p_{dyn}, and $COST$. These measures quantitatively represent the consequences of imperfect detection mechanisms and then reflect the effects of detection mechanisms on the system performance. In this section, these measures are used to address problems in the design of detection mechanisms.

Suppose that the specifications of performance requirements and application tasks are now given. To provide the required fault tolerance in the design, we have to answer the following two questions: (i) what kinds of detection mechanisms should be incorporated in the computer system to be designed, and (ii) what are their properties in meeting the specifications? In other words, we need to know the coverage by signal level mechanisms, the error latency in function-level mechanisms, and the period of diagnostics. Suppose, for instance, that the real-time operations and time-critical processes are now our major design concern. The specifications must include the limit for the probability of failure as well as the maximum allowable extra cost caused by shortcomings of detection mechanisms.

According to our simulation results in Fig. 9, the avoidance of error by diagnostics appears useful only if the cycle time of diagnostics is not much greater than the fault's active period, which is usually small for transient and intermittent faults. This implies that a frequent application of diagnostics is needed. However, in such a case, the computation time wasted for executing diagnostics as well as the total execution cost increases prohibitively, making the periodic use of diagnostics during normal operation less useful. It also indicates that the probability of capturing intermittent faults and the improvement of loss in recovery by diagnostics are small. Consequently, on-line diagnostics are not useful for time-critical applications.

As a conservative measure, the probability of failure due to imperfect detection mechanisms, denoted by p_f, can be represented by the sum of p_e and p_{dyn}. From the model, one can see that p_e is dependent exponentially on error latency but linearly on coverage c. That is, the decreasing of error latency has a greater impact on p_e than does the increasing of the coverage. However, an improvement in the coverage will decrease the probability of error propagation, and thus reduce the recovery overhead. In Fig. 10, curves with constant p_f and constant $COST$ are plotted, where $C(t)$ is assumed to be $(t - \hat{T})^2$ for $t \geq \hat{T}$. Note that for simplicity we used a quadratic incremental cost in the above plotting. It shows the

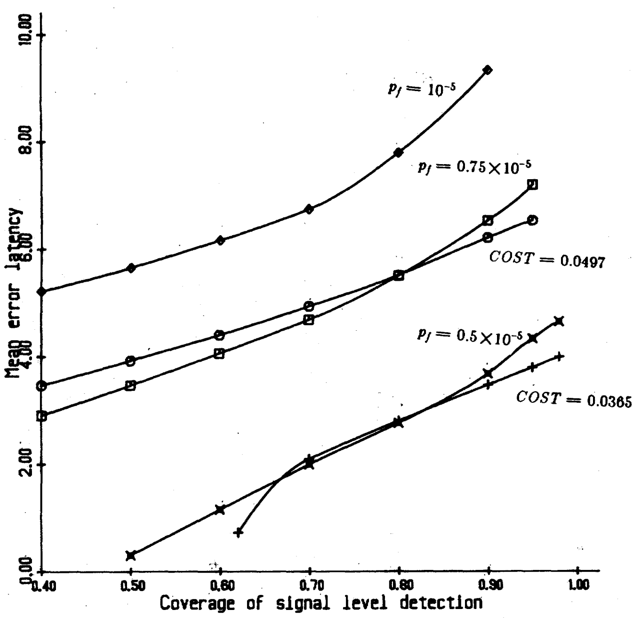


Fig. 10. Design space for coverage and mean error latency subject to constraints of p_f and $COST$ (with the same system parameters as in Fig. 9 and $t_{\text{dead}} = 150$, $p_{sv} = 0.2$).

combination of the coverage and the mean error latency required to attain p_f and $COST$, below the specified values. The area under both the constant p_f and constant $COST$ lines indicates the design space for selecting the coverage and the mean error latency. It is clear that perfect signal level detection is within the design space, though it is impractical. By contrast, the combination of small error latency and zero signal level detection may not satisfy the specifications. This can be seen easily from the fact that with a zero signal level detection, every recovery must require rollbacks and/or restarts. The use of rollbacks and/or restarts for recovery is more time consuming than error masking and retry which are available only to signal level detection mechanisms. Hence, signal level detection mechanisms must be included in the design. The curves with constant $COST$ show that the average execution cost is insensitive with respect to the coverage of the signal level detection mechanism. This is due to the fact that all errors induced by intermittent or permanent faults have to be recovered by rollback or restart irrespective of the nature of the error detection process, and that because of the overheads imposed on saving states, recovery points have to be placed relatively far apart. It is important to recognize that any sophisticated recovery method will cause a severe delay in task execution.

The feasible design space indicated in Fig. 10 will provide the requirements in detection mechanisms for certain system performance specifications. However, it is very difficult to objectively determine an optimal combination of signal level and function level detection mechanisms. The main reasons for this are that (i) the coverage has to be related to actual hardware costs, (ii) error latency and performance of function level detection mechanisms are application dependent, and (iii) the cost of function level detection mechanisms, especially software checking, is neither well structured nor well understood at present.

V. Concluding Remarks

In this paper, we have presented first a general model for the error detection process and then a method for estimating two important performance-related parameters of fault-tolerant computers. These two are not usually included in the traditional reliability models. The first parameter, the probability of having an unreliable result, indicates the degree of lack of confidence in computation results. Suspicion in the computation results is wholly due to the imperfect nature of error detection. Unfortunately, such imperfection cannot be eliminated completely from any practical error detection mechanism. For the second parameter, we take a more detailed account of the computation loss and execution cost resulting from the occurrence of error, its detection, and its subsequent recovery. Since most reliable systems either include error recovery mechanisms with unknown overheads or may suffer from an erroneous output, any reliability analysis has to quantify the above overheads and uncertainty and also has to provide a good method for estimating these quantities.

Though there are several assumptions to be justified by experiment, the model developed in this paper is general enough to include all aspects from fault occurrence to error detection with various detection mechanisms. Note that in the estimation of the above two parameters, the model provides solutions only when there is one or zero error detection at a given time during task execution. The higher order effects — more than one error detected at a time during task execution due to multiple faults — are negligible since the probability of such an event is quite small.

Finally, we have outlined a feasible design space in which a proper combination of different imperfect detection mechanisms needed to meet the specifications is indicated. Since the determination of a feasible design space of detection mechanisms must integrate the recovery methods used in the system, we also briefly presented the performance of various recovery methods. Unfortunately, we cannot determine an optimal tradeoff between various detection mechanisms because of the insufficient understanding of the function level detection and the lack of relations between hardware costs and the signal level detection capability. Further research is needed along these directions, especially experiments of program behavior under erroneous conditions and the design of function level detection mechanisms.

Also of interest would be an analysis that allows the treatment of simultaneously extant multiple faults. Since most faults in the system are likely to be transient or intermittent, there is the possibility that the fault latency is large. Note that the retry recovery is applied as a temporary remedy when an intermittent fault becomes benign shortly after its occurrence. This intermittent fault may still exist but is inactive. These would cause faults to accumulate in **F** and/or **FB**, thus making the entire system vulnerable to any environmental or other events that might activate them. The difficulty with any such model is likely to be a considerable expansion in the number of states, thus increasing the model complexity. It is likely that in any realistic analysis, some means must be sought to reduce the state-space size by approximating suitably. The approach used in CARE III [14], where states are aggregated and the state transition rates are separately determined, may be an appropriate attempt although the model is forced to be nonhomogeneous. The nature of such approximations is a matter for further research.

Appendix A
Density Functions of Task Execution Time

The density functions of task execution time with error occurrence due to three different types of faults (i.e., transient, intermittent, and permanent) are expressed as follows:

$$\begin{aligned}
f_r(t|F_1,T) &= \{1 - \pi_6(T|F_1) - \pi_7(T|F_1)(1-\rho_1)\}\delta_T(t) \\
&\quad + \pi_6(T|F_1) f_{rbs,1}(t,0) \\
&\quad + \pi_7(T|F_1)(1-\rho_1)[1 - e^{-\pi_r}\delta_T(t-t_r) \\
&\quad + (e^{-\pi_r}) f_{rbs,1}(t,1)] \\
f_r(t|F_2,T) &= \{1 - \pi_6(T|F_2) - \pi_7(T|F_2)(1-\rho_1)\}\delta_T(t) \\
&\quad + \pi_6(T|F_2) f_{rbs,2}(t,0) + \pi_7(T|F_2)(1-\rho_1) \\
&\quad \cdot \left[\sum_{n=1}^{\infty} (1-\delta_2)^{n-1}\delta_2 f_{rbs,2}(t,n)\right] \\
f_r(t|F_3,T) &= \{1 - \pi_6(T|F_3) - \pi_7(T|F_3)(1-\rho_1)\}\delta_T(t) \\
&\quad + \pi_6(T|F_3) f_{rbs,2}(t,0) \\
&\quad + \pi_7(T|F_3)(1-\rho_1) f_{rbs,2}(t,1)
\end{aligned}$$

where $f_{rbs,j}(t,n)$ is the density function of the time loss in recovery from an error induced by F_j after n unsuccessful retries, which is given as follows:

$$\begin{aligned}
f_{rbs,j}(t,n) &= (1-p_{sv})\,p_{46}(t - nt_r - t_b|F_j)\frac{1}{t_{ch}}\{u_T(t-nt_r) \\
&\quad - u_T(t - nt_r - t_{ch})\} + \Bigg[p_{sv} + (1-p_{sv}) \\
&\quad \cdot \left\{1 - \int_0^{t_{ch}} \frac{p_{46}(t|F_j)}{t_{ch}}\,dt\right\}\Bigg] f^T_{\text{start},j}(t - nt_r)
\end{aligned}$$

where $\delta_T = \delta(t-T), u_T = u(t - T - t_b), f^T_{\text{start},j}(t) = f_{\text{start},j}(t-T)$, and $\delta(t)$ and $u(t)$ are impulse and step functions, respectively.

Appendix B
Notations

The following notations are defined and used in the paper to represent various measures:

CL:	Average computation loss due to diagnostics, checkpoints, and error recovery for each error detection.
$COST$:	Average total execution cost for the execution of a single task.
$C(t)$:	Task execution cost when it is completed at time t.
F_j:	Event that the fault is transient, intermittent, or permanent for $j = 1, 2, 3$, respectively.
$P(E)$:	Probability of having an error given that a fault occurs.
$R_{i,j}$:	Average time loss when error masking, retry, rollback, or restart is applied to recover from an error induced by F_j for $i = 1, 2, 3, 4$, respectively.
RL:	Average time loss used to recover from an error.
$S_1(S_2)$:	Event that rollback or restart (retry) is used to recover from an intermittent fault.
T:	Time needed to complete the task execution under a fault-free condition.
$\hat{T}$:	Time needed to complete the task execution when periodic diagnostics and checkpointing are inserted.
c:	Coverage of signal level detection mechanisms.
$f_r(t\|F_j,T)$:	Density function of time delay in task execution given that the fault type is F_j and the fault-free task execution time is T.
$f_{\text{start},j}(t)$:	Density function of time loss when restart is used to recover the failed task.
$p_{46}(t\|F_j)$:	Distribution function of error latency for the error induced by F_j, which is calculated from (3) and with initial condition $\pi = [0, 0, 0, 1, 0, 0, 0]$.
p_{dyn}:	Probability of dynamic failure in which the execution of task has missed the specified deadline.
p_e:	Probability of having an unreliable result at the completion of task execution.
p_f:	Probability of system failure which is defined as $p_{\text{dyn}} + p_e$.
p_j:	Probability of event F_j.
p_j':	Probability of event F_j given an error is detected.
p_{sv}:	Probability that the saved state becomes inaccessible after the occurrence of fault.
t_b:	Time needed to set up the task for rollback recovery.
t_{ch}:	Time interval between two successive checkpoints.
t_{dead}:	Hard deadline associated with a task.
t_n:	Time interval between two successive diagnostics.
t_{ov}:	Time needed to establish a checkpoint.
t_p:	Time needed to swap the executing task for diagnostics.
t_r:	Time used for a single retry recovery.
t_s:	Setup time for restarting a task.
α:	Error generation rate by a fault.
β:	Error detection rate by the function level when the system is in **E**.
γ:	Error detection rate by the function level when the system is in **EFB**.
λ:	Fault occurrence rate.
τ:	Transition rate that an existing transient fault disappears.

μ:	Transition rate that an active intermittent fault becomes benign.
ν:	Transition rate that a benign intermittent fault becomes active.
ω:	Detection rate when the diagnostic program is running.
ξ:	Probability of detecting faults in a single diagnostic given that faults exist.
$\delta_1(\delta_2)$:	Transition probability from event S_1 to S_2 (S_2 to S_1).
$\rho_{i,j}$:	Probability that error masking, retry, rollback, or restart is applied to recover from an error due to F_j for $i = 1, 2, 3, 4$, respectively.
$\theta_{i,j}$:	Probability that the recovery succeeds from an error due to F_j when error masking, retry, rollback, or restart is applied for $i = 1, 2, 3, 4$, respectively.
σ:	Percentage of time loss due to periodic diagnostics and insertions of checkpoints.
η:	Percentage of average computation loss for each error detection.
$\pi_i(t \mid F_j)$:	Probability that the system is in state i at time t, given that the type of fault is F_j.

Acknowledgment

The authors are grateful to R. Butler and M. Holt of the NASA Langley Research Center, C. M. Krishna of the University of Michigan, and anonymous referees for assistance in improving the technical content of this paper.

References

[1] S. J. Bavuso *et al.*, "Latent fault modeling and measurement methodology for application to digital flight control," in *Proc. Advanced Flight Control Symp.*, USAF Acad., 1981.

[2] B. Courtois, "Some results about the efficiency of simple mechanisms for the detection of microcomputer malfunction," in *Proc. 9th Annu. Int. Symp. on Fault-Tolerant Computing*, 1979, pp. 71–74.

[3] ——, "A methodology for on-line testing on microprocessors," in *Proc. 11th Annu. Int. Symp. on Fault-Tolerant Computing*, 1981, pp. 272–274.

[4] J. J. Shedletsky, "A rollback interval for networks with an imperfect self-checking property," *IEEE Trans. Comput.*, vol. C-27, pp. 500–508, June 1978.

[5] H. Ball and F. Hardie, "Effects and detection of intermittent failures in digital systems," in *Proc. AFIPS Conf.*, Fall 1969, pp. 229–235.

[6] S. Osden, "The DC-9-80 digital flight guidance system's monitoring techniques," in *Proc. AIAA Guidance and Control Conf.*, 1979, pp. 64–79.

[7] N. L. Gunther and W. C. Carter, "Remarks on the probability of detecting faults," in *Proc. 10th Annu. Int. Symp. on Fault-Tolerant Computing*, 1980, pp. 213–215.

[8] J. J. Shedletsky, "Random testing: Practicality vs. verified effectiveness," in *Proc. 7th Annu. Int. Symp. on Fault-Tolerant Computing*, 1977, pp. 175–179.

[9] V. Tasar, "Analysis of fault-detection coverage of a self-test software program," in *Proc. 8th Annu. Int. Symp. on Fault-Tolerant Computing*, 1978, pp. 65–74.

[10] M. T. Breuer, "Testing for intermittent faults in digital circuits," *IEEE Trans. Comput.*, vol. C-22, pp. 241–246, Mar. 1973.

[11] Y. K. Malaiya and S. Y. H. Su, "Reliability measure of hardware redundancy fault-tolerant digital systems with intermittent faults," *IEEE Trans. Comput.*, vol. C-30, pp. 600–604, Aug. 1981.

[12] Y. W. Ng and A. A. Avizienis, "A unified reliability model for fault-tolerant computers," *IEEE Trans. Comput.*, vol. C-29, pp. 1002–1011, Nov. 1980.

[13] K. S. Trivedi and R. M. Geist, "A tutorial on the CARE III approach to reliability modeling," NASA Rep. 3488, 1981.

[14] J. J. Stiffler and L. A. Bryant, "CARE III phase report—Mathematical description," NASA Rep. 3566, Nov. 1982.

[15] V. D. Agrawal, "An information theoretic approach to digital fault testing," *IEEE Trans. Comput.*, vol. C-30, pp. 582–587, Aug. 1981.

[16] E. Cinlar, *Introduction to Stochastic Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[17] Y. K. Malaiya and S. Y. H. Su, "Analysis of an important class of non-Markov system," *IEEE Trans. Rel.*, vol. R-31, pp. 64–67, Apr. 1982.

[18] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 220–232, June 1975.

[19] K. M. Chandy, J. C. Browne, C. W. Dissly, and W. R. Uhrig, "Analytic models for rollback and recovery strategies in data base systems," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 100–110, Mar. 1977.

[20] E. Gelenbe, "On the optimum checkpoint interval," *J. Ass. Comput. Mach.*, vol. 26, no. 2, pp. 259–270, Apr. 1979.

[21] X. Castillo and D. P. Siewiorek, "A performance-reliability model for computing system," in *Proc. 10th Int. Symp. on Fault-Tolerant Computing*, 1980, pp. 187–192.

[22] Y. H. Lee and K. G. Shin, "Rollback propagation detection and performance evaluation of FTM^2P—A fault-tolerant multiprocessor," in *Proc. 9th Annu. Symp. on Comput. Arch.*, 1982, pp. 171–180.

[23] C. M. Krishna and K. G. Shin, "Performance measures for multiprocessor controllers," in *Performance '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam: North-Holland, pp. 229–250.

[24] K. G. Shin, C. M. Krishna, and Y. H. Lee, "The application to the aircraft landing problem of a unified method for characterizing real-time systems," in *Proc. Real Time Syst. Symp.*, Arlington, VA, Dec. 1983.

Kang G. Shin (S'75–M'78–SM'83), for a photograph and biography, see p. 124 of the February 1984 issue of this Transactions.

Yann-Hang Lee (S'81), for a photograph and biography, see p. 124 of the February 1984 issue of this Transactions.