Determination of an Optimal Retry Time in Multiple-Module Computing Systems

Chao–Ju Hou and Kang G. Shin

Abstract—The 'optimal' (in some sense) amount of time used for (or the optimal number of times) retrying an instruction upon detection of an error in a computing system is usually determined under the assumption that the system is composed of a single module, within which all fault activities are confined until some module-replacement action is taken. However, a computing system is usually composed of at least three modules, namely, CPU, memory, and I/O, and the execution of an instruction often requires the cooperation of two or more modules. It is thus more realistic to consider the fault activities in multiple-module systems.

In this paper, we first relax the single-module assumption and model the fault activities in a multiple-module system as a Markov process. We apply the randomization method to decompose the continuous-time Markov chain into a discrete-time Markov chain subordinated to a Poisson process. Using this decomposition, we can derive several interesting measures, such as 1) the conditional probability of successful retry given a retry period and the fact that a non-permanent fault has occurred, 2) the mean time-to-system recovery, and 3) the distribution of the time until which all modules in the system enter a fault-free state. All the measures derived are used to determine, along with the parameters characterizing fault activities and costs of recovery techniques, a) whether or not retry should be used as a first-step recovery means upon detection of an error, and b) the best retry period or number of retries that satisfies a given criterion, e.g., a specific probability of successful retry.

Index Terms—Fault-tolerance, error recovery, instruction retry, Markov models, randomization.

1 Introduction

VARIOUS recovery techniques have been proposed to handle different types of fault: *permanent, intermittent,* and *transient*. Permanent faults are solid/hard failures and persist forever, which result mainly from component aging or breakage. Transient faults are caused mainly by temporary changes in environmental, electrical, or mechanical conditions. They may be active for an unpredictable period of time and die out. Intermittent faults are usually the results of manufacturing defects such as loose connections or bonds. They cycle between active and inactive states, also in an unpredictable manner. Since no single recovery technique is known to be effective against all possible faults, we must usually use a combination of several recovery techniques.

Recovery techniques are classified into instruction retry, program rollback, program reload and restart, and module replacement [1], [2], [3]. Whenever an error is detected, instruction retry is applied and the latest unsuccessful instruction is repeated. If this retry is not successful, one can employ program rollback and/or program reload and restart. If all these recovery techniques fail, one has to resort to system diagnosis and reconfiguration, i.e., identify and remove the faulty module and resume the process execution on a new fault-free processor. Since instruction retry requires little additional hardware and software and thus smaller program completion and recovery overheads as compared to the other recovery techniques, it is usually used as a first-step recovery means. However, an instruction retry will be successful only if the following two conditions are satisfied:

C1. The system failed during the execution of the latest uncompleted instruction. This condition can be satisfied if errors are detected upon their occurrence by some signal-level detection mechanism [1], i.e., zero error latency.

C2. The existing fault disappears during the time of retry or *retry period*, i.e., the retry period should be long enough (by perhaps retrying the same instruction more than once)[1] so that the fault dies out within this period.

We assume in this paper that **C1** can be achieved by employing on-line detection mechanisms with high coverage. That is, an error is confined to a module where the fault causing that error had occurred and the affected process can be restored to integrity. One consequence of **C1** is that the damage caused by the fault is recoverable by restoring the process to some prior fault-free state and all data needed to retry the instruction are available. **C2** is impossible in case of permanent faults. Fortunately, only less than 10%, and perhaps as few as 2%, of errors are known to be caused by permanent faults [4], [2], and retry for a nonpermanent fault is likely to succeed if a retry period is selected properly. The retry period should be chosen to maximize the benefit that results from retrying for nonpermanent faults and to alleviate the loss that results from retrying for permanent faults.

The design and analysis of various recovery procedures has been addressed by numerous researchers. They characterize either the process of executing instructions [2], [3] or the process of fault activities [5] on a *single-module* system as a Markov process or a renewal process. The maximum likelihood principle and/or Bayesian decision theory are then applied to determine optimal values of design parameters. The retry period (or, equivalently, the number of times an uncompleted instruction is retried) is either specified a priori in an ad hoc manner [3], or determined by minimizing some average *task-oriented* measure, e.g., mean execution time per instruction [2], mean task-completion time [6], [7], and/or some average objective penalty function [5].

In contrast to the above approaches, we determine the optimal retry period by constructing a continuous-time Markov chain which characterizes fault activities in a multiple-module system. Using the randomization method [8], [9], we then derive:

1) the probability of successful retry, $P_{rs}(t)$, given a retry period t and a fault has occurred;
2) the mean time–to–system recovery, $E(L(t))$, defined as the mean value of the time at which a retry with the maximum retry period of t stops because either all faults became inactive (and thus the retry succeeded) or the retry period expired;
3) the distribution of the time until which all faults, if occurred, become inactive in the system (or equivalently, the time until the corresponding Markov chain to enter a fault-free state), $P(T_{ss} \le t)$.

Based on these quantities and the parameters characterizing fault activities (e.g., failure rate, the probability of a fault being permanent, transient, or intermittent, and the distribution of fault active/benign duration), we can determine:

• *C.-J. Hou is with the Dept. of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706.
E-mail: jhou@ece.wisc.edu.*
• *K. G. Shin is with the Real-Time Computing Laboratory, Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109. E-mail: kgshin@eecs.umich.edu.*

*Manuscript received Aug. 11, 1992; revised Dec. 29, 1994.
A part of this paper appeared in IEEE Second Int'l Symp. on Uncertainty Modeling and Analysis, April 1993, pp. 294-301.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95134.*

1. Since it is easy to convert a retry period to the number of retries, the term "retry period" will be used throughout the paper.

1) whether or not retry should be used before applying a different recovery technique, and
2) the minimum retry period that achieves a given probability of successful retry.

Another point that differentiates our work from others is that we relax the commonly-used assumption that all fault activities are confined to a single module until some module-replacement action is taken. Note that a computing system is composed of at least three modules (i.e., CPU, memory, and I/O), and execution of an instruction usually requires the cooperation of multiple modules. We must therefore consider fault activities in multiple modules. On the other hand, not every instruction being retried use all modules in the system. Using a tagging method described in Section 2.2, we can flexibly tailor the continuous–time Markov chain to accommodate different cases in which the instruction being retried uses only a subset of the modules in the system. To the best of our knowledge, this is the first to relax the single-module assumption in determining the retry period and model fault activities in a multiple-module system.

The rest of the paper is organized as follows. Section 2 describes the fault model used, the assumptions made, the continuous-time Markov chain that characterizes the fault model, and the quantities to be derived. In Section 3, we analytically derive the optimal retry period using the quantities derived in Section 2. We conclude this paper with Section 4.

2 Fault Model and Parameters of Interest

We first describe the fault model of a multiple-module system. Then, we characterize the fault model with an embedded continuous-time Markov chain under the assumption that at most one fault exists in each module at any moment. Finally, we discuss how to extend the model to the (more general but rare) case that multiple faults are possible on a single module. Although all the concepts and expressions are derived for an arbitrary number of modules, we confine our illustrative examples to the case of three modules for the clarity of presentation.

2.1 Fault Model

We assume that faults arrive at the ith module according to a time-invariant Poisson process with rate λ_i. We also assume that transient, intermittent, and permanent faults occur with probability p_{tf}, p_{if}, p_{pf}, respectively, and their occurrences are independent of one another. (Note that $p_{tf} + p_{if} + p_{pf} = 1.0$.) Consequently, transient, intermittent, and permanent faults occur at exponential rate $\lambda_i p_{tf}$, $\lambda_i p_{if}$, and $\lambda_i p_{pf}$, respectively. If a permanent fault occurs, it remains persistent in the system until the component containing the fault is replaced. If a transient fault occurs, it disappears after an active duration, where the active duration is exponentially distributed with rate τ_i. If an intermittent fault occurs, it may become benign after an active duration, and then reappear after a benign duration, where the active and benign time are exponentially distributed with rate μ_i and ν_i, respectively. That is, an intermittent fault cycles between active and benign states.

Because instruction retry is effective only if an error is detected upon its occurrence (otherwise it is impossible to determine which instruction to retry), we assume that errors are detected immediately upon their occurrence by, for example, signal-level detection mechanisms [1]. Also, faults occurred in one module are assumed not to affect other modules, i.e., fault occurrences in different modules are statistically independent. This assumption results from the fact that faults are usually the malfunction of hardware components, and are independent of one another [4], [1], [7].

2.2 Construction of a Continuous-Time Markov Chain

Under the assumptions of fault behavior in Section 2.1 and the assumption that there is at most one fault in each module at any time,[5] we model a multimodule system with a continuous-time Markov chain. The state space S consists of state vectors of the form $(s_1, s_2, \ldots, s_n)$, where n is the number of modules in the system, and $s_i \in \{-2, -1, 0, 1, 2\}$ represents the state of the ith module with the following interpretation:

1) −2 represents the permanent-fault (PF) state, i.e., there exists a permanent fault in the module.
2) −1 represents the transient-fault (TF) state, i.e., there exists a transient fault in the module.
3) 0 represents the no-fault (NF) state, i.e., no fault exists in the module.
4) 1 represents the intermittent-fault (IF) state, i.e., there exists an active intermittent fault in the module.
5) 2 represents the benign-fault (BF) state, i.e., an intermittent fault has become inactive in the module.

For example, the state vector (1, 2, 0) indicates that there exists an active intermittent fault in the first module (CPU), a benign intermittent fault in the second module (memory), and no fault in the third module (I/O).

The Markov model is *flexible* in the sense that it allows for a variety of fault patterns. For example, if only transient and intermittent faults are possible, $s_i \in \{-1, 0, 1, 2\}$, for $1 \le i \le n$, and $|S| = 4^n$. Also, the model allows for a situation where different sets of faults may occur to different modules. For example, if only transient, transient, intermittent faults could occur in the first, second, third module, respectively, in a three-module system, then $s_1 \in \{0, -1\}$, $s_2 \in \{0, -1\}$, $s_3 \in \{0, 1, 2\}$.

Recall that instruction retry will succeed only if all the faults in the set of modules the retried instruction uses have disappeared, or became inactive, during the retry period t. For the clarity of presentation, we assume that every instruction retried uses all the modules in the system.[6] That is, a retry will succeed only if the system has moved to a state vector none of whose components are 1, −1, or −2 during the period t. Based on this observation, we divide S into Failed Set (FS) and Successful Set (SS), where $FS = \{(s_1, s_2, \ldots, s_n): \exists i \text{ such that } s_i = 1, -1, \text{or}, -2\}$, and $SS = \{(s_1, s_2, \ldots, s_n): s_i \neq 1, -1, \text{and}, -2, \forall i\}$. For example, in the case of a three-module system, if both transient and intermittent faults are possible, we have $SS = \{(0, 0, 0), (0, 0, 2), (0, 2, 0), (2, 0, 0), (0, 2, 2), (2, 0, 2), (2, 2, 0), (2, 2, 2)\}$, and all the other 56 states belong to FS. It is straightforward to extend the above discussion to the case when the instruction being retried does not use all modules. If each instruction is tagged (perhaps at compile time) with the modules it will use, then the continuous-time Markov chain describing fault activities can be tailored to remove the coordinate(s) corresponding to the unused module(s). For example, in the three-module example system, if the instruction being retried uses only the first module, the state space for the Markov chain reduces to Fig. 1, because the fault activities on second and third modules are irrelevant to the fact whether or not the retry is successful.

2. This assumption results from the fact that the inter-arrival time of faults is usually much larger than any other fault-related durations. We will discuss in Section 2.3 how to relax the last assumption.

3. As discussed below, this assumption can be relaxed by tagging each instruction with the modules it will use.

Due to the assumption that faults occur independently among modules, state transitions along the same coordinate exhibit the same behavior, and describe fault activities in the associated module. For example, the state transition between (0, 0, 0) and (1, 0, 0) is the same as those between $(0, s_2, s_3)$ and $(1, s_2, s_3)$, $\forall, s_2, s_3 \in \{-2, -1, 0, 1, 2\}$, because they all describe the fault activity from NF to IF in the first module (while the second/third module may be in different fault states). Moreover, state transitions along one coordinate are virtually the same as those along another coordinate except that fault activities/transitions correspond to a different module, and perhaps, have different rates along different coordinates. Consequently, it suffices to characterize the state evolution of the system by a one-dimensional state-transition diagram shown in Fig. 1.

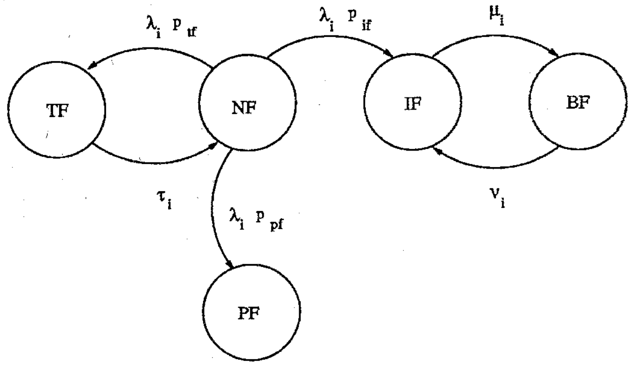

Fig. 1. One-dimensional state transition diagram.

The system state evolution can be described as a Markov process $\{X(t), t \geq 0\}$ on the state space $S = \{(s_1, s_2, ..., s_n): s_i \in \{-2, -1, 0, 1, 2\}, n \in \mathcal{N}$ is the number of modules$\}$. Using the randomization technique summarized in the Appendix, we can decompose $\{X(t), t \geq 0\}$ into a discrete-time Markov chain, $\{Y_n, n = 0, 1, \ldots\}$, embedded in a Poisson process, $\{N(t), t \geq 0\}$ with rate

$$\Lambda = \max_{(s_1, s_2, \ldots, s_n) \in S} \left\{ \left(\sum_{i \text{ s.t. } s_i = 0} \lambda_i + \sum_{i \text{ s.t. } s_i = -1} \tau_i + \sum_{i \text{ s.t. } s_i = 1} \mu_i + \sum_{i \text{ s.t. } s_i = 2} + v_i \right) \right\}, \quad (2.1)$$

where each term in (2.1) is the transition rate of some state $(s_1, s_2, ..., s_n)$.

2.3 Extension to the Multiple-Fault Case

The Markov model described in Section 2.2 can be extended to the more general case in which multiple faults may occur in a single module. The state space S now consists of state vectors $(\underline{s_1}, \underline{s_2}, \ldots, \underline{s_n})$ where n is the number of modules in the system, and $\underline{s_i}$ describes the state of the ith module and is a three-tuple

$$\underline{s_i} \triangleq s_0^i \, s_1^i \, s_2^i \, s_3^i,$$

where s_0^i, s_1^i, s_2^i, and s_3^i are the number of permanent faults, transient faults, active intermittent faults, and benign intermittent faults, in the ith module, respectively. We assume that $0 \leq s_j^i \leq K$, $j = 0, 1, 2, 3$, where K is a sufficiently large number so that the quantities of interest derived from the model that uses K, and those derived from the model that uses $K + 1$ are within a specific error of tolerance.[7]

Similar to the model described in Section 2.2, state transitions along the ith coordinate are associated with the state evolution in the ith module, and can be uncoupled with state transitions along other coordinates under the assumption that faults occur independently among modules. Consequently, the state evolution in the system can be characterized by the state transition diagram that describes fault activities in one module, as shown in Fig. 2 (where only transitions around $\underline{s_i} = s_0^i \, s_1^i \, s_2^i \, s_3^i$ are shown). Note that the number of allowable states for one module (i.e., along one coordinate) is now $(K + 1)^4$ (instead of 5). The transition rates are derived in a straightforward manner as in Fig. 1. For example, the transition from $s_0^i, s_1^i, s_2^i, s_3^i$ to $s_0^i \, s_1^i \left(s_2^i - 1\right) \left(s_3^i + 1\right)$ occurs when an active intermittent fault in the ith module becomes benign and is thus with rate $s_2^i \mu_i$.

The system under consideration can then be described as a Markov process $\{X(t), t \geq 0\}$ on the state space

$$S = \left\{ \left(\underline{s_1}, \underline{s_2}, \ldots, \underline{s_n}\right) : \underline{s_i} = s_0^i \, s_1^i \, s_2^i, s_3^i, \; 0 \leq s_j^i \leq K, \; j = 0, 1, 2, 3 \right\}$$

to which randomization can be applied to obtain a discrete-time Markov chain, $\{Y_n, n = 0, 1, 2, \ldots\}$, and a Poisson process, $\{N(t), t \geq 0\}$, with rate

$$\Lambda = \max_{(\underline{s_1}, \underline{s_2}, \ldots, \underline{s_n}) \in S} \left\{ \sum_{i=1}^{n} \left(\lambda_i + s_1^i \tau_i + s_2^i \mu_i + s_3^i v_i \right) \right\},$$

where $\sum_i \left(\lambda_i + s_1^i \tau_i + s_2^i \mu_i + s_3^i v_i \right)$, as shown in Fig. 2, is the transition rate of state $(s_1, s_2, \ldots, s_n)$.

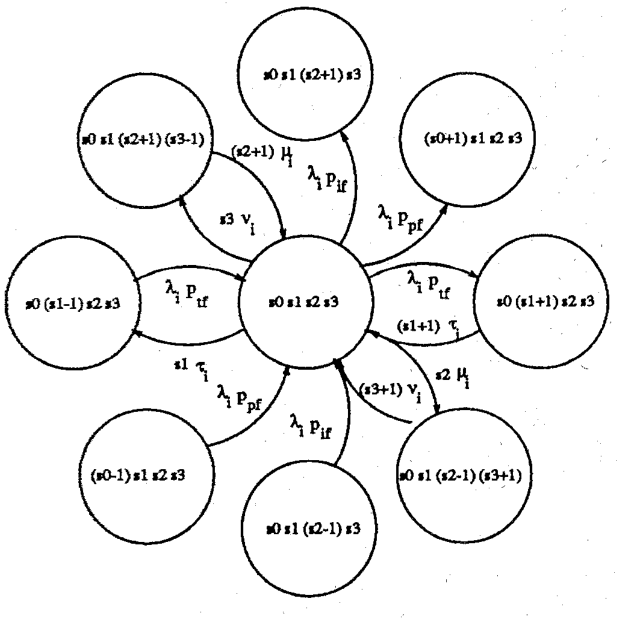

Fig. 2. "One dimensional" state diagram for the case which allows multiple faults on a single module. $0 \leq s_j \leq K, j = 0, 1, 2, 3$. Only transitions around $\underline{s_i} = s_0^i \, s_1^i \, s_2^i \, s_3^i$ are shown. Transitions are applicable only when the corresponding states exist.

4. According to our simulation results, the value of K needs not be large, because the fault occurrence rate, λ_i, is usually several orders smaller than the other rates, and thus, the probability that multiple faults exist is usually negligible.

The set of failed states, FS, and the set of successful states, SS, can be identified as

$$FS = \left\{\left(\underline{s_1}, \underline{s_2}, \ldots, \underline{s_n}\right): \exists i \text{ such that } s_0^i + s_1^i + s_2^i \neq 0\right\},$$

$$\text{and } SS = \left\{\left(\underline{s_1}, \underline{s_2}, \ldots, \underline{s_n}\right): s_0^i + s_1^i + s_2^i = 0, \forall i\right\}.$$

2.4 Parameters to be Derived

Specifically, we want to derive:

$P_{rs}(t)$: the probability that retry succeeds given that the retry period is t and a fault has occurred. Using this information, we can determine the retry period for a specified probability of successful retry.

$E(L(t))$: the mean time–to–system recovery. Specifically, let $r \in (0, \infty)$ be the time for the system to first enter a state that belongs to SS, and let $L(t) = \min(t, r)$ be the time–to–system recovery given a maximum retry period t, i.e., the time at which the retry stops because either all the faults have disappeared/become inactive (i.e., $r < t$, and a successful retry results) or the retry period expired (i.e., $r \geq t$). Obviously, if $p_{pf} > 0$, then $E(L(t)) = t$ (i.e., retry will never succeed). When $t \to \infty$ (i.e., retry an instruction forever), $E(L(t))$ represents the mean time for the system to enter a state $\in$ SS for the first time. Note that $\lim_{t \to \infty} E(L(t)) = \infty$ if $p_{pf} > 0$.

$P(T_{SS} \leq t)$: the distribution of the first SS-passage time. Specifically, let T_{SS} be the first time the system enters a state that belongs to SS, i.e., $T_{SS} = \min\{t\text{: } X(t) \in SS\}$. With this probability distribution, we can compute the mean SS-passage time, $E[T_{SS}]$. The definitions and interpretations of $P_{rs}(t)$, T_{SS}, and $L(t)$ lead to the following relations: 1) $P_{rs}(t) = P(T_{SS} \leq t)$, and 2) $E(T_{SS}) = \lim_{t \to \infty} E(L(t))$, both of which will be used to verify the correctness of our derivation.

3 Derivation

3.1 Probability of Successful Retry

Let $p(n, k)$, $0 \leq k \leq n + 1$, denote the probability that the underlying discrete-time Markov chain (obtained after randomization) visits k fault states (i.e., states in FS) given n state changes. For example, $p(n, n + 1)$ is the probability that the underlying Markov chain always stays in fault states during these n state changes. Consequently, the probability that a retry of period t fails, $1 - P_{rs}(t)$, is the probability that the underlying Markov chain always stays in fault In Section 2, we modeled the fault evolution as a continuous-time Markov chain $\{X(t), t \geq 0\}$ on a finite state space S, where S can be decomposed into two mutually-exclusive subsets, FS and SS. Also, to derive parameters of interest, the constructed Markov chain $\{X(t), t \geq 0\}$ is decomposed by the randomization technique into a discrete-time Markov chain $\{Y_n, n = 0, 1, \ldots\}$ subordinated to a Poisson process $\{N(t), t \geq 0\}$. states regardless of the number of state changes in $[0, t]$, i.e.,

$$1 - P_{rs}(t) = \sum_{n=0}^{\infty} p(n, n+1) \cdot P(n \text{ state changes in time } t)$$

$$= \sum_{n=0}^{\infty} p(n, n+1) \cdot P(N(t) = n) = \sum_{n=0}^{\infty} p(n, n+1) \cdot \frac{e^{-\Lambda t}(\Lambda t)^n}{n!},$$

or,

$$p_{rs}(t) = 1 - \sum_{n=0}^{\infty} p(n, n+1) \cdot \frac{e^{-\Lambda t}(\Lambda t)^n}{n!}, \tag{3.1}$$

where Λ is the rate of the underlying Poisson process obtained after randomization, and is given in (2.1). The error resulting from the truncation of the infinite sum in (3.1) can be easily bounded. Specifically, let R_m denote the error resulting from truncating (3.1) to m steps, then

$$R_m = \sum_{n=m+1}^{\infty} \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} \cdot p(n, n+1) \leq \sum_{n=m+1}^{\infty} \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} = 1 - \sum_{n=0}^{m} \frac{e^{-\Lambda t}(\Lambda t)^n}{n!},$$

m can be evaluated a priori for a given error tolerance.

Now, the remaining task is to calculate $p(n, k)$. Let $p(n, k, a_i)$ be the probability that the underlying Markov chain visits fault states k times out of n steps and let a_i be the state visited after the last transition. Then, we have

$$p(n, k, a_i) = \begin{cases} \sum_{j=1}^{|s|} p(n-1, k-1, a_j) \cdot P_{ji} & \text{if } a_i \in FS, \\ \sum_{j=1}^{|s|} p(n-1, k, a_j) \cdot P_{ji} & \text{if } a_i \in SS, \end{cases}$$

where P_{ji} is the transition probability from state a_j to state a_i in the underlying discrete-time Markov chain $\{Y_n, n = 0, 1, \ldots\}$ (A.1 in the Appendix). The initial conditions are given by

$$p(0, 1, a_i) = \begin{cases} \pi_i(0), & \text{if } a_i \in FS, \\ 0, & \text{otherwise,} \end{cases}$$

where $\pi_i(0) = \phi_i(0)$ is the transient probability of the Markov chain being in state a_i at time 0, and $p(0, 0, a_i) = 0$. Note that whenever an instruction is retried, the system must be in a fault state, i.e., k in $p(n, k, a_i)$ must be ≥ 1, thus $p(0, 0, a_i) = 0$, $\forall a_i$. Finally, by the law of total probabilities, $p(n, k) = \sum_{i=l}^{|S|} p(n, k, a_i)$.

3.2 Mean Time–to–System Recovery

Recall that r is defined in Section 2 as the time the system first enters a state $\in$ SS and $L(t) = \min(t, r)$ is defined as the time–to–system recovery given that the retry period is t. Analytically,

$$E[L(t)] = \int_0^t (1 - P_{rs}(s))\,ds$$

$$= \int_0^t \sum_{n=0}^{\infty} p(n, n+1) \cdot \frac{e^{-\Lambda s}(\Lambda S)^n}{n!}\,ds + t \cdot \left(1 - P_{rs(t)}\right)$$

$$= \frac{1}{\Lambda} \cdot \sum_{n=0}^{\infty} \left(1 - \sum_{j=0}^{n} \frac{e^{-\Lambda t}(\Lambda t)^j}{j!}\right) \cdot p(n, n+1) + t \cdot (1 - P_{rs}(t)) \tag{3.2}$$

As $t \to \infty$, $E[L(t)]$ becomes the mean time for the system to enter a state $\in$ SS, and can be used to indicate whether or not retry should be used for a particular system configuration, i.e.,

$$\lim_{t \to \infty} E[L(t)] = \begin{cases} \frac{1}{\Lambda} \cdot \sum_{n=0}^{\infty} p(n, n+1) & \text{if } p_{pf} = 0; \\ \infty & \text{if } p_{pf} = 0. \end{cases} \tag{3.3}$$

3.3 Distribution of First *SS*–Passage Time

Recall that T_{SS} is defined as the time the system first enters a state $\in$ SS, i.e., $T_{SS} = \min\{t\text{: } X(t) \in SS\}$. Randomization can be used to compute the distribution of T_{SS} as follows: we define an associated process $\{X_{SS}(t), t \geq 0\}$ on the state space $FS \cup \{S_a\}$, where S_a is the absorbing state formed by collapsing all states in SS into it. The corresponding generator matrix Q_{SS} can be expressed as

$$q_{SS;i,j} = q_{i,j} \quad \text{for } i, j \notin SS,$$
$$q_{SS;i,S_a} = \sum\nolimits_{j \in SS} q_{i,j} \quad \text{for } i \notin SS,$$
$$q_{SS;S_a,j} = 0 \quad \text{for all } j.$$

Note that the transition intensities of $\{X_{SS}(t), t \geq 0\}$ are identical to $\{X(t), t \geq 0\}$ except that there does not exist any transition out of S_a (or equivalently, all states in SS). Consequently, we have

$$P(T_{SS} \leq t) = P(X_{SS}(t) = S_a). \tag{3.4}$$

That is, the distribution of the time until the system first enters a state in SS (the left-hand side of (3.4)) is equivalent to computing a transient state probability for the Markov process, $X_{SS}(t)$, where all states in SS are lumped into S_a, and has the generator matrix Q_{SS} (the right-hand side of (3.4)). Using the randomization technique, we have ((A.4) in the Appendix).

$$P(X_{SS}(t) = S_a) \sum_{n=0}^{\infty} \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} \cdot P(X_{SS,n} = S_a) = \sum_{n=0}^{\infty} \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} \cdot \pi_{SS}, S_a(n), \tag{3.5}$$

where $\pi_{SS,S_a}(n)$ can be computed from $\Pi_{SS}(n) = \Pi_{SS}(n-1)$, P_{SS}, and P_{SS} is the transition probability matrix and can be computed from Q_{SS} by the same approach as in (A.1) in the Appendix.

3.4 Determination of Retry Period

The significance of the quantities derived above lies in that they can be used to determine:

1) whether or not to apply instruction retry as a first-step recovery means, and
2) the smallest retry period that achieves a specified probability of successful retry.

Specifically, let $C_1(t)$ and C_2 denote the cost function of instruction retry given the retry period t and the cost function of applying other time-redundancy recovery techniques (e.g., program rollback, program reload and restart),[8] respectively, and let P_{req} denote the required probability of successful retry (which is given as a design parameter). Obviously, $C_1(t)$ is a monotonically nondecreasing function of t.

The first question can be answered as follows. If there does not exist any $t > 0$ such that $C_1(t) + (1 - \cdot P_{rs}(t)) \cdot C_2 \leq C_2$, or, in a simpler form,

$$C_1(t) \leq P_{rs}(t) \cdot C_2, \tag{3.6}$$

then retry should not be applied, where $1 - P_{rs}(t)$ is the probability that retry fails given a retry period t. That is, if for every t the cost of retrying for the period t as the first-step recovery means is greater than the cost of not applying instruction retry, then retry should not be applied at all. The second question can be answered by finding the smallest t that satisfies both (3.6) and

$$P_{rs}(t) \geq P_{req}. \tag{3.7}$$

4 Conclusion

We proposed in this paper a continuous-time Markov model to characterize the fault activities in a multiple-module computing system. The randomization technique is then applied to this model to derive several quantities of interest, i.e., the probability of successful retry given a retry period, the mean end-of-retry time, and the distribution of time for the system to enter a fault-free state for the first time. These quantities can be used to determine whether or not instruction retry should be applied as a first-step recovery means with respect to fault characteristics and recovery costs, and the smallest retry period that achieves a prespecified success probability.

Our analysis is valid as long as errors are detected upon their occurrence. If there is a nonzero latency between error occurrence and detection due to imperfect coverage of detection mechanisms (i.e., not meeting condition **C1** in Section 1), it may be difficult to identify the faulty module because of possible error propagation [10]. Moreover, if the executing task is contaminated before an error is detected, retry cannot succeed in recovering the system even if the fault disappeared (e.g., the data cannot be restored). This problem is worthy of further investigation, and will be reported in a forthcoming paper.

Appendix

Randomization [8], [9], [11] is commonly used as a computational method to compute transient probabilities of Markov processes with finite state spaces. The main idea of this technique is to transform the Markov process into a Markov chain subordinated to a Poisson process as explained below.

Consider a Markov process $\{X(t), t \geq 0\}$ on a finite state space $S = \{0, 1, \ldots, N\}$. The generator matrix, $Q = (q_{ij}, 0 \leq i, j \leq N)$, of the Markov process has the following property: $q_{ii} \triangleq -q_i = -\sum\nolimits_{j \neq i} q_{ij}$, where q_{ij} is the transition rate from state i to state j, and $q_i = \sum\nolimits_{j \neq i} q_{ij}$ is the rate of leaving state i.

Let Λ be any value such that $\Lambda \geq q_i$, $\forall i$. At some instant, if the process is in state i, then it leaves state i at rate q_i, but this is equivalent to assuming that transitions occur at rate Λ, but only the fraction q_i/Λ of them are real (in the sense that the process really leaves state i and enters some other state and hence real transitions occur at rate q_i), and the remaining fraction $1 - \frac{q_i}{\Lambda}$ are fictitious transitions which keep the process in state i. That is, any Markov process can be thought of as being a process which spends an exponential amount of time with rate Λ in state i and then transitions to state j with a probability

$$P_{ij} = \begin{cases} 1 - \frac{q_i}{\Lambda}, & j = i, \\ \frac{q_{ij}}{\Lambda}, & j \neq i, \end{cases} \tag{A.1}$$

or in matrix form, $P = Q/\Lambda + I$. Consequently, there exists two component processes in a Markov process: a discrete-time Markov chain $\{Y_n, n = 0, 1, \ldots\}$ on S with transition matrix $P = Q/\Lambda + I$, and a Poisson process $\{N(t), t \geq 0\}$ with rate Λ. Moreover, $\{Y_n, n = 0, 1, \ldots\}$ and $\{N(t), t \geq 0\}$ are independent of each other, and the process $\{Y_{N(t)}, t \geq 0\}$ has the same finite dimensional distribution as (and is thus probabilistically identical to) the original Markov process.

This decomposition not only gives a physical interpretation of Markov processes, but also facilitates the computation of transient probabilities of a Markov process. Specifically, let the transient probabilities of the Markov process be $\pi_i(t) = P(X(t) = i)$, and $\Pi(t)$

5. C_2 would be dependent on the retry period t if the assumption **C1** does not hold. That is, if errors cannot be detected upon their occurrence, the latest checkpoint used in program rollback might have been contaminated by error propagation, and the program may be forced to roll back to an earlier checkpoint, resulting in a higher cost.

$= (\pi_0(t), \pi_1(t), \pi_2(t), \ldots, \pi_N(t))$. Then, conditioning on the number of occurrences of the Poisson process in [0, t], and using the *law of total probability,* we have

$$\pi_i(t) = P(X(t)=i) = P\left(Y_{N(t)}=i\right) = \sum_{n=0}^{\infty} P\left(Y_{N(t)}=i \middle| N(t)=n\right)\cdot P(N(t)=n) = \sum_{n=0}^{\infty} P(Y_n = i)\cdot\frac{e^{-\Lambda t}(\Lambda t)^n}{n!}, \tag{A.2}$$

where $P(Y_n = i) \equiv \phi_i(n)$ is the probability of the Markov chain being in state i at the nth step. Let $\Phi(n) = (\phi_1(n), \phi_2(n), \ldots)$, then (A.2) can be rewritten as

$$\prod(t) = \sum_{n=0}^{\infty}\Phi(n)\cdot\frac{e^{-\Lambda t}(\Lambda t)^n}{n!}. \tag{A.3}$$

Since $\Pi(0) = \Phi(0)$, and $\Phi(n) = \Phi(0)P^n$, where P is the transition matrix of the discrete-time Markov chain, (A.1), and (A.3) can be rewritten as

$$\prod(t) = \prod(0)\sum_{n=0}^{\infty}P^n\cdot\frac{e^{-\Lambda t}(\Lambda t)^n}{n!}. \tag{A.4}$$

ACKNOWLEDGMENTS

The work reported in this paper was supported in part by the Office of Naval Research under Grant N00014-91-J-1115 and by NASA under Grant NAG-1220. 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] 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.
[2] I. Koren, Z. Koren, and Y.H. Su, "Analysis of a class of recovery procedures," *IEEE Trans. Computers,* vol. 35, no. 8, pp. 703–710, Aug. 1986.
[3] M. Berg and I. Koren, "On switching policies for modular redundancy fault-tolerant computing systems," *IEEE Trans. Computers,* vol. 36, no. 9, pp. 1,052–1,062, Sept. 1987.
[4] D.P. Siewiorek and R.S. Swarz, *The Theory and Practice of Reliable System Design.* Digital Press, 1982.
[5] A.M. Saleh and J.H. Patel, "Transient-fault analysis for retry techniques," *IEEE Trans. Reliability,* vol. 37, no. 3, pp. 323–330, Aug. 1988.
[6] Y.-H. Lee and K.G. Shin, "Optimal design and use of retry in fault-tolerant computer systems," *J. ACM,* vol. 35, no. 1, pp. 45–69, Jan. 1988.
[7] 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.
[8] W.K. Grassman, "Transient solutions in Markovian queueing systems," *Computers and Operations Research,* vol. 4, pp. 47–53, 1977.
[9] D. Gross and D.R. Miller, "The randomization technique as a modeling tool and solution procedure for transient markov processes," *Operations Research,* vol. 32, no. 2, Mar.–Apr. 1984.
[10] K. G. Shin and T.-H. Lin, "Modeling error propagation in a multi-module computing system," *IEEE Trans. Computers,* vol. 37, no. 9, pp. 1,053–1,066, Sept. 1988.
[11] B. Melamed and M. Yadin, "Randomization procedures in the computation of cumulative-time distributions over discrete state Markov processes," *Operations Research.,* vol. 32, no. 4, pp. 926–944, July–Aug. 1984.

Self-Checking Comparator with One Periodic Output

S. Kundu, E.S. Sogomonyan, M. Goessel, and S. Tarnick

Abstract—In this paper we propose a new self-checking comparator with one periodic output. The comparator can be used as a two-rail checker or as an equality checker. Two different input patterns are sufficient to detect all the faults considered.

Index Terms—Two-rail checker, equality checker, self-checking circuit, single periodic output.

1 INTRODUCTION

THE first self-checking comparator (two-rail checker) was proposed in [1]. The detection of all single stuck-at-0/1 faults is possible if all possible correct input code words actually occur as inputs of the comparator. If the comparator is a part of a large circuit this is usually impossible to guarantee. In [2], it is proposed to include additional delay elements into one half of the input lines of the comparator. The additional delay elements allow all possible input code words to actually occur as inputs of the comparator itself with some delay as long as no input line is constant. Aside from the additional hardware costs, a considerable time delay may occur for larger word lengths before an error is detected.

In another approach to solve this problem, an additional BIST structure is recommended in [3]. In a special test mode, additional test inputs are periodically generated by an additional test input generator (TIG). In this approach, the hardware costs are relatively high and some faults may only be detected in test mode after a relatively large time has elapsed.

None of the known comparators can be used as two-rail checkers or equality checkers with a single rail output.

2 NEW COMPARATOR

Fig. 1 shows the general structure of the proposed comparator, which is an improvement of the comparator presented in [4]. One of the inputs of every XOR-gate A_{1i}, $i = 1,\ldots,n$, of the first level is connected to a periodically changing input signal x_0, $x_0 \in \{0,1\}$. As long as the comparator is not faulty and as long we have $x_i' = \bar{x}_i$ $(x_i' = x_i), i = 1,\ldots,n$, for the inputs the outputs of the XOR-gates A_{2i} of the second level are $y_i = \bar{x}_0$ (x_0), and the inputs of the special CMOS-gate G are either 11...1 or 00...0. A design for this gate G at switch level is shown in Fig. 2. For $y_1 = y_2 = \cdots = y_n = \bar{x}_0$ (x_0), we have $z = x_0$ $(\bar{x}_0)$ and $y = \bar{x}_0$ (x_0). Thus, the output y of the comparator is the periodically changing signal $\bar{x}_0$ (x_0). The transistors are not used as bidirectional devices; therefore, the gate G can have a large fan-in. An-

• S. Kundu is with IBM T.J. Watson Research Center, Yorktown Heights, NY 10598,USA. E-mail: kundu@watson.ibm.com.
• E.S. Sogomonyan is with the Institute of Control Sciences, Russian Academy of Sciences, 117806 Moscow, Russia.
• M. Goessel and S. Tarnick are with the Max Planck Society, Fault-Tolerant Computing Group, University of Potsdam, Potsdam 14415 Germany.
E-mail: mgoessel@mpag-inf.uni-potsdam.de.

Manuscript received Jan. 27, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95124.