IEEE TRANSACTIONS ON COMPUTERS, VOL. 43, NO. 9, SEPTEMBER 1994

An Optimal Retry Policy Based on Fault Classification

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

Abstract— **An optimal (in some sense) retry policy in a computer system is usually derived under an unrealistic assumption that fault characteristics are known *a priori* and remain unchanged throughout the mission lifetime. In such a case, the optimal retry period depends only upon the system's status at the time of fault detection. We propose to remedy this deficiency by formulating the optimal retry problem as a Bayesian decision problem where not only the time of fault detection but also the results of earlier retries are used to estimate the current fault characteristics. Previous knowledge about fault characteristics is represented by the prior distributions of fault-related parameters which are updated whenever new samples are obtained from retry and detection mechanisms.**

A new fault classification scheme is proposed to assign a temporal fault type (i.e., permanent or intermittent or transient) to each detected fault so that the corresponding fault parameters can be estimated. The estimated fault parameters are then used to derive the optimal retry period that minimizes the mean task completion time. Efficient algorithms are developed to determine the optimal retry period on-line upon detection of each fault. To evaluate the goodness of the proposed retry policy, it is compared with, and is always found to outperform, a number of fixed-retry-period policies.

Index Terms— **Retry, fault classification, prior and posterior distributions, parameter estimation, error recovery, Bayesian decision theory.**

I. Introduction

RETRY means re-execution of the latest instruction which could not be completed because of a fault until either the fault disappears or a different recovery scheme such as rollback and restart is called in. We will use the definition of fault in [1] that fault is the source of an error which represents a logically incorrect state of the system. Based on their temporal effects, faults are classified into *permanent, intermittent*, and *transient* [2], [3]. The time period during which a fault has adverse effects (inducing errors) is called the *active period*, and the time period during which the fault is not active is called the *benign period*. Thus, a permanent fault is viewed to have an infinite active period and a transient fault is viewed to have a finite active period followed by an infinite benign period. An intermittent fault cycles between finite active and benign periods.

If the fault in question is permanent, retry will never succeed and the time used for retry is simply wasted; but if the fault is nonpermanent, the usefulness of retry depends greatly on how the retry period or the number of retries is chosen. (Note that it is easy to convert a retry period to a number of retries.) For a transient fault, the retry period should be longer than the active period of the fault. It may not be desirable to retry for an intermittent fault, since the same intermittent fault may recur an infinite number of times in future. But if an intermittent fault occurs near the end of the execution of a task, then one may choose retry over restart or rollback to recover from the intermittent fault, because the fault can recur only a limited number of times before the task completion. In that case the retry period should be longer than the active periods of all recurring faults. The major problem of determining the optimal retry period stems from the fact that it is impossible to know the type of a fault upon its detection. So, we have to consider the probability of a detected fault being permanent or nonpermanent in deriving an "optimal" (in the sense to be defined later) retry policy. A common policy is to retry for a pre-determined period and if it is not successful, then initiate other recovery schemes, such as rollback and restart. This policy works well if the retry period is chosen properly, because the majority of faults are known to be nonpermanent [4], [5].

The retry period is usually chosen empirically in the various forms of retry used in earlier mainframe computers as a means of recovery [6]–[9]. The optimal design of retry has recently been proposed by several researchers [10]–[12]. In [11], the problem of combined retry and rollback recovery is studied under the assumption that faults are classified as 1) permanent faults, 2) intermittent faults recoverable by instruction retry, and 3) intermittent faults not recoverable by instruction retry. In their model, all detected faults are handled by the same procedure, i.e., retry for a fixed period and then roll back a fixed number of times. The optimal retry period and optimal inter-checkpoint intervals are derived for the three recovery procedures by minimizing the average time spent per instruction. Berg and Koren [10] refined the above model and derived an optimal policy on whether or not to use retry upon detection of a fault, depending on the remaining mission time. They considered computer systems in which each module has several standby spares ready to be switched in upon detection of a fault. In their model, a module's fault occurrence rate is assumed to increase with the number of faults occurred in that module. Upon detection of a fault, their policy is to choose one

Manuscript received January 27, 1992; revised April 5, 1993. This work was supported in part by the NASA under Grant NAG-1-1220, the ONR under Grant N00014-91-J-1115, and the NSF under Grants MIP-9010948 and MIP-9012549.

T.-H. Lin is with the Department of Electrical and Computer Engineering, State University of New York, Buffalo, NY 14260 USA; e-mail: thlin@eng.buffalo.edu.

K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122 USA.

IEEE Log Number 9401633.

of the following two actions, whichever has a lower average cost: 1) replacing the faulty module immediately with a good spare and 2) retrying the faulty module and replacing it only if the retry is unsuccessful. However, they did not discuss how to determine the retry period. Taking a task-oriented view, Lee and Shin [12] derived two types of the optimal retry period by minimizing the mean remaining task execution time: one for the case of faults with unknown types, and the other for the case of intermittent faults.

The most influential parameters in determining the retry period are those related to the characteristics of faults, such as the fault occurrence rate, the mean active period of a transient or intermittent fault, and the probability of a detected fault being permanent, transient, or intermittent. These parameters are commonly assumed to be known *a priori*. Hence, the results obtained under this assumption are useful only if the fault parameters can be estimated accurately. No previous work except for [12] has addressed the problem of deriving retry policies in the absence of *a priori* knowledge about the fault parameters. Lee and Shin [12] proposed a Bayesian decision approach where the active and benign periods of an intermittent fault are estimated on-line and the estimated parameters are then used to derive the optimal retry periods for future recurrences of the intermittent fault. However, they did not address how to estimate other fault parameters, and used an ad hoc criterion in recognizing an intermittent fault. That is, if two consecutive fault detections are made within a *short* period and their symptoms are similar, they are assumed to be two manifestations of the same intermittent fault.

Based on our earlier results on on-line fault parameter estimation and fault classification [13], we will formulate the optimal retry problem as a Bayesian decision problem [14], [15]. The advantage of using the Bayesian approach is that parameter estimation can be performed *progressively* from sample data. The sample data of the fault parameters are obtained from monitoring the fault behavior through retry and detection mechanisms, as well as through the fault classification scheme [13] which assigns an appropriate fault type to each detected fault, thus allowing for (probabilistically) correct interpretation of the corresponding sample data.

Throughout this paper, faults are assumed to be detected immediately upon their occurrence by, for example, built-in testers or error detecting codes. If faults cannot be detected upon their occurrence, instruction retry is no longer effective, because it is impossible to determine which instruction to retry.

In the following section, the Bayesian formulation of the optimal retry problem is presented with a brief description of parameter estimation and fault classification. The optimal retry periods which minimize the mean remaining task execution time in two distinguishable situations are derived in Section III and IV, respectively. We will also compare our optimal retry policy with a number of fixed-retry-period policies via simulation in Section V, and conclude this paper with Section VI.

II. Optimal Retry and Fault Classification

Determination of an optimal retry policy can be formulated as a Bayesian decision problem. The essential elements of this formulation include the definitions of an action space $\mathcal{A}$, a parameter space $\mathcal{P}$, and a loss function L defined on $\mathcal{A}$ and $\mathcal{P}$. The idea is to choose an action from $\mathcal{A}$ which minimizes the loss function L. $\mathcal{A}$ is defined as the set of all possible retry periods. The retry period, denoted by r, must be an integral multiple of the time required for a basic retry operation such as the execution of a microinstruction. However, in order to exploit the continuous optimization techniques, r is assumed to be any nonnegative real number, i.e., $\mathcal{A} = R^+$. A practical solution can be obtained by evaluating the two integer solutions nearest to the continuous solution and choosing the better of the two. The action of no retry is represented by $r = 0$.

$\mathcal{P}$ contains all fault parameters which may affect the retry policy under consideration. Our principal concern is whether a detected fault is permanent or intermittent or transient, and hence, the fault parameters are related to the occurrences of different types of faults. Arrivals of permanent, intermittent, and transient faults are assumed to be independent Poisson processes with rate λ_p, λ_i, and λ_t, respectively. Other parameters include the active period of a transient fault, and the active and benign periods of an intermittent fault, which are assumed to be distributed exponentially with rates λ_{ta}, λ_{ia}, and λ_{ib}, respectively. The parameter space is thus defined as

$$\mathcal{P} = \{\mathbf{p} : \mathbf{p} = (\lambda_p, \lambda_t, \lambda_i, \lambda_{ta}, \lambda_{ia}, \lambda_{ib})\}.$$

In the Bayesian analysis, the uncertainty about a parameter is quantified by assigning a distribution to the parameter. Let $\pi(\mathbf{p})$ represent the joint density function of $\mathbf{p}$ and the loss function $L(\mathbf{p}, r)$ be defined as the penalty incurred by one retry of period r for a given $\mathbf{p}$, then the *Bayesian expected loss* is defined as

$$\int_{-\infty}^{\infty} L(\mathbf{p}, r)\pi(\mathbf{p})d\mathbf{p}.$$

The *Bayesian decision*, r_E^*, is the (optimal) retry period which minimizes the Bayesian expected loss. Unfortunately, no closed-form solution for r_E^* can be derived due to the complexity of the above integral. In this paper, we will derive an alternative solution which minimizes the *maximum likelihood loss* $L(\hat{\mathbf{p}}, r)$, where $\hat{\mathbf{p}}$ denotes the mode of $\pi(\mathbf{p})$.[1] This solution will henceforth be called the *maximum likelihood decision*, denoted by r_M^*. In general, r_M^* is different from r_E^*, nevertheless, if $\mathbf{p}$ is distributed densely over a narrow range around the mode, r_M^* will be very close to r_E^*.

A. Parameter Estimation

Each fault parameter is independent of others, and hence, can be estimated separately. Furthermore, the estimation procedures for all fault parameters are identical since each of them represents the rate of an exponential distribution. This distribution is updated whenever a new sample is obtained. Let x_i be the ith sample. According to the Bayesian theorem,

$$\pi^i(\lambda) = \frac{\pi^{i-1}(\lambda)f(x_i|\lambda)}{\int \pi^{i-1}(\lambda)f(x_i|\lambda)d\lambda} \tag{2.1}$$

[1] For ease of parameter estimation, the distributions of all parameters are assumed to be Gamma, each with only one mode.

where $f(x_i|\lambda)$ is the sample probability given the parameter λ. In (2.1), $\pi^{i-1}(\lambda)$ ($\pi^i(\lambda)$) will be called the *prior* (*posterior*) distribution of λ with respect to x_i. The posterior distribution for the current sample becomes the prior distribution for the next sample. The first prior distribution (i.e., $\pi^0(\lambda)$) is determined from experiences,[2] but all the other distributions will be calculated from the samples collected. To simplify the calculation of these distributions from sample to sample, it is desirable that both the prior and the posterior distributions belong to the same distribution family. In such a case, (2.1) can be reduced to a transformation on the key parameters of the distribution. For a class of sample density functions, any class of prior density functions which have the above desirable property is called a *conjugate family* [14]. The conjugate prior distribution family for the class of exponential sample distributions turns out to be the Gamma distribution. The mode, mean, and variance of a Gamma distribution have closed-form expressions. Moreover, as the number of samples increases, both the mode and the mean become less dependent on the initial prior distribution while the variance gets smaller. This implies that the difference between the maximum likelihood decision and the Bayesian decision become insignificant as the number of samples increases.

The parameters in $\mathcal{P}$ are to be estimated and updated continuously as more information is obtained through fault detection and retry mechanisms. The detection mechanism reveals the existence of a fault, whereas the retry mechanism indicates the active period of the fault. In addition to the knowledge of the existence and the active period of a fault, the type (permanent or intermittent or transient) of the fault must be known. In an earlier paper [13], we have proposed a systematic fault classification scheme which uses the collected fault samples for on-line parameter estimation. For completeness, we will briefly summarize this scheme below.

B. Fault Classification

By "fault classification" we mean unambiguously assigning a fault type to every detected fault. The assigned fault type may not always be correct because, for example, a transient fault with a long active period is indistinguishable from a permanent fault, or two consecutive transient faults with the same symptom are indistinguishable from two recurrences of an intermittent fault. However, the classification error or the probability of incorrect classification can be minimized using the maximum likelihood principle. The information used in fault classification includes the time of each fault detection and the detection symptom which refers to the observable and incorrect system state identified by the detection mechanism. Different faults may produce the same detection symptom, but different detection symptoms must come from different faults.

Faults could be classified *before* or *after* a retry. In the before-retry classification, we want to determine if the newly-detected fault is the recurrence of an earlier unclassified fault. Because faults do not occur frequently in most systems, we assume that only the most recent two faults can remain unclassified in this scheme. If the likelihood of the current fault being the recurrence of one of the two previous faults outweighs other possible scenarios, we will notify the system by raising a flag that an intermittent fault is detected. Otherwise, the newly-detected fault is left unclassified and no further classification will be done until after the retry. If the retry for an unclassified fault is successful and the number of unclassified faults exceeds two, we will classify the earliest unclassified fault as a transient fault. If the retry for the current fault is unsuccessful, its fault type is determined by the diagnosis following a reconfiguration while all the other unclassified faults are classified as transient faults. Note that a fault can only remain unclassified if it is successfully recovered by a retry. Therefore, as far as the retry policy is concerned, the fault classification scheme provides valuable information on whether the newly-detected fault is intermittent or not.

See [13] for a detailed account of this fault classification scheme.

C. Retry Policy

The notable difference between the proposed retry policy and the other retry policies lies in the treatment of intermittent faults. There are two conflicting options associated with the treatment of intermittent faults. On one hand, it is necessary to remove/repair the module with intermittent faults as soon as possible, so that the damage by the recurrences of each intermittent fault may not accrue. On the other hand, insofar as the completion of the tasks affected by intermittent faults is concerned, retry may sometimes be more beneficial than other time-consuming recovery methods such as rollback and restart. We propose to make a compromise between these two options as follows. Upon detection of a fault, if there are reasons to believe that the fault is the recurrence of an intermittent fault, one of the above two options is chosen based on the cost (to be defined in the next section) associated with the task executing on the faulty module. If retry is chosen, a single optimal retry period is determined which will be used for all future recurrences of the intermittent fault until the completion of the current task. The faulty module will be taken off from the system for repair at the conclusion of the current task. The optimal retry period for intermittent faults should be longer than the mean active period of intermittent faults, so that the current task can almost always be completed without rollback or restart as a result of unsuccessful retries. The purpose of repairing the faulty module at the end of the current task is to prevent the same intermittent fault from causing any further damages to the system. Using the same retry period for all recurrences of an intermittent fault simplifies our analysis, since results of future retries are predictable in such a case. Moreover, this same-retry-period policy also eliminates the need to derive an optimal retry period for every future recurrence of an intermittent fault.

Since most performance indices for computer systems are related to the time overhead, and since the solution minimizing the time overhead also minimizes any monotonically increasing function of the time overhead, we shall define "loss" as the average task execution time including the overhead of retry

[2] If no information is available, the noninformative prior distribution will be used for $\pi^0(\lambda)$.

and/or rollback/restart. Let the status of a task be represented by a two-tuple (R, S) where R (S) is the task's remaining execution time if retry succeeds (fails), and let t_s denote the setup time for rollback/restart. The loss L is a function of R, S, and t_s.

III. Retry for Intermittent Faults

Consider the case when the current fault is classified as the recurrence of an intermittent fault. An optimal retry period is determined and used for all future recurrences of this intermittent fault until the completion of the current task. Before attempting to derive the optimal retry period, we need to check if retry is more beneficial (in the sense of completing the current task) than restart/rollback. Assume that the optimal retry period is longer than the active periods of almost all future recurrences of the intermittent fault as well as those of future transient faults. The average number of recurrences of the intermittent fault during the remaining task execution time is $\lambda_{ib}R$ and each recurrence requires a mean retry period of $1/\lambda_{ia}$. The average number of transient faults to occur during the remaining task execution time is $\lambda_t R$ and each occurrence requires a mean retry period of $1/\lambda_{ta}$. So, the mean time to complete the current task with retry recovery is

$$K_R = R\left(1 + \frac{\lambda_{ib}}{\lambda_{ia}} + \frac{\lambda_t}{\lambda_{ta}}\right). \tag{3.1}$$

In case of restart/rollback, a setup time t_s needs to be added. The remaining task execution time becomes S, but in this case, there will be no overhead from the existing intermittent fault. Thus, the mean time to complete the current task with restart/rollback recovery is

$$K_S = S\left(1 + \frac{\lambda_t}{\lambda_{ta}}\right) + t_s. \tag{3.2}$$

In (3.1) and (3.2), the overhead that might occur due to a second permanent or intermittent fault is ignored to simplify the otherwise required computation. This can be justified by the fact that λ_p and λ_i are usually much smaller than λ_t and λ_{ib}. Retry will be chosen over restart/rollback if $K_R < K_S$, or if $R < R_0$, where

$$R_0 = \frac{(1 + \lambda_t/\lambda_{ta})S + t_s}{1 + \lambda_t/\lambda_{ta} + \lambda_{ib}/\lambda_{ia}}. \tag{3.3}$$

To determine the optimal retry period, a more accurate estimation of the loss is needed. Suppose r is the chosen retry period for the current intermittent fault. The probability of an unsuccessful retry for any future recurrence of the intermittent fault is $e^{-\lambda_{ia}r}$. Similarly, the probability of an unsuccessful retry for any occurrence of a transient fault is $e^{-\lambda_{ta}r}$. If retry for the current fault is successful, three independent Poisson processes could lead to an unsuccessful retry in future before the current task is completed. They are: 1) occurrence of permanent faults with rate λ_p, 2) occurrence of transient faults causing unsuccessful retries with rate $\lambda_t e^{-\lambda_{ta}r}$, and 3) recurrence of intermittent faults causing unsuccessful retries with rate $\lambda_{ib}e^{-\lambda_{ia}r}$. The sum of these three Poisson processes is also a Poisson process with rate

$$\lambda_u = \lambda_p + \lambda_t e^{-\lambda_{ta}r} + \lambda_{ib}e^{-\lambda_{ia}r}.$$

The probability that all retries during R are successful, denoted by p_{sr}, is equivalent to the probability that no unsuccessful retry occurs during R. So,

$$p_{sr} = e^{-\lambda_u R} = e^{-(\lambda_p + \lambda_t e^{-\lambda_{ta}r} + \lambda_i b e^{-\lambda_{ia}r})R}.$$

The loss is computed as

$$\begin{aligned} L(r) &= (1 - e^{-\lambda_{ia}r})\left[p_{sr}K_R + (1 - p_{sr})\left(r + K_S + \frac{K_R}{2}\right)\right] \\ &\quad + e^{-\lambda_{ia}r}(r + K_S) \\ &= (1 - e^{-\lambda_{ia}r})\left[e^{-\lambda_u R}K_R + (1 - e^{-\lambda_u R})\right. \\ &\quad \times \left.\left(r + K_S + \frac{K_R}{2}\right)\right] + e^{-\lambda_{ia}r}(r + K_S) \\ &= (r + K_S)\left[1 - (1 - e^{-\lambda_{ia}r})e^{-\lambda_u R}\right] + \frac{K_R}{2} \\ &\quad \times (1 - e^{-\lambda_{ia}r})(1 + e^{-\lambda_u R}). \end{aligned} \tag{3.4}$$

This equation is derived by enumerating all possible events. The current retry may fail with the probability $e^{-\lambda_{ia}r}$ and the corresponding mean task completion time is $r + K_S$. Even if the current retry is successful, a future retry may become unsuccessful during the remaining task execution time R with probability $1 - p_{sr}$. In such a case, the mean task completion time is $r + K_S + K_R/2$ where $K_R/2$ is the mean time until the first future unsuccessful retry, since if only one unsuccessful retry does occur during R, its occurrence time will be uniformly distributed over R. The event leading to the mean task completion time K_R is when the current retry and the future retries during R are all successful.

Using the chain rule, the derivative of p_{sr} becomes $Je^{-\lambda_u R}$ where

$$J = \left(\lambda_t \lambda_{ta} e^{-\lambda_{ta}r} + \lambda_{ib}\lambda_{ia}e^{-\lambda_{ia}r}\right)R. \tag{3.5}$$

The derivative of $L(r)$ is

$$\begin{aligned} L'(r) &= 1 - e^{-\lambda_u R}(1 - e^{-\lambda_{ia}r}) + \frac{K_R}{2} \\ &\quad \cdot \left[\lambda_{ia}e^{-\lambda_{ia}r}(1 + e^{-\lambda_u R}) + Je^{-\lambda_u R}(1 - e^{-\lambda_{ia}r})\right] \\ &\quad - (r + K_S)\left[\lambda_{ia}e^{-\lambda_{ia}r}e^{-\lambda_u R} + Je^{-\lambda_u R}(1 - e^{-\lambda_{ia}r})\right]. \end{aligned} \tag{3.6}$$

Necessary conditions for the optimal solution r_M^* are $L'(r_M^*) = 0$ and $L''(r_M^*) > 0$. The possibility that $r_M^* = 0$ is ruled out because the decision to retry has already been made (see (3.3)). Furthermore, it is easy to see that r_M^* should be much larger than the active period of the intermittent fault, since the decision to retry is based on the assumption that retry will be successful for almost all recurrences of the intermittent fault. Hence, we are interested in finding r_M^* in the region where the probability of an unsuccessful retry due to a long active period is much less than one, i.e., $e^{-\lambda_{ia}r} \ll 1$. Another (realistic) assumption is that the recurrence rate of an intermittent fault is much higher than a transient fault's occurrence rate. The term J defined in (3.5) can thus be approximated by

$$J \approx \lambda_{ib}R\lambda_{ia}e^{-\lambda_{ia}r}.$$

Under these assumptions, the following theorem states and proves that in the region where $e^{-\lambda_{ia}r} \ll 1$, $L(r)$ has exactly one local minimizer which is therefore the global minimizer. (Proofs of all theorems can be found in the Appendix.)

Theorem 1: In the region where $e^{-\lambda_{ia}r} \ll 1$, $L(r)$ has exactly one local minimizer r^* if $J \approx \lambda_{ib}R\lambda_{ia}e^{-\lambda_{ia}r}$, and

$$\frac{2}{\lambda_{ia}} < r^* + K_S - \left(1 + \frac{e^{\lambda_u R}}{1+\lambda_{ib}R}\right)\frac{K_R}{2}.$$

Note that $2/\lambda_{ia}$ is twice the mean active period of an intermittent fault and $e^{\lambda_u R} \ll 1$ since λ_u is very small. Thus,

$$K_S - \left(1 + \frac{e^{\lambda_u R}}{1+\lambda_{ib}R}\right)\frac{K_R}{2} \approx K_S - \frac{K_R}{2} > \frac{K_S}{2}$$

because $K_R < K_S$ in this case. The last assumption of the above theorem is easily satisfied since $K_S/2$ is almost always greater than $2/\lambda_{ia}$.

A. Efficient Implementation

Since instruction retry is a low level mechanism, there is obviously not much time available after a fault detection to derive the optimal retry period. The retry period must be determined *a priori*, perhaps before a task is scheduled to run. If the task length is known, then R_0 can be computed easily by (3.3) with S replacing the task length. However, in most cases, the task length is unknown. This difficulty can be handled by either replacing S with the expected task length or revising (3.3) such that retry will be attempted only if the expected remaining task execution time is less than

$$\left(\frac{\lambda_{ia}}{\lambda_{ib}}\right)\left(1+\frac{\lambda_t}{\lambda_{ta}}\right)(F+t_s) \tag{3.7}$$

where F is the accumulative task time upon detection of a fault. The above value can be precomputed and stored in a table.

The optimal retry period would then have to be precomputed for many combinations of R and S depending on the storage and computation overhead constraints. These values need only be updated when one or more fault parameters get updated by the fault classification scheme. To solve for r_M^*, a recursive formula based on (A.1) in the Appendix is derived as the equation found at the bottom of the page where $k = 0, 1, 2, \cdots$. Initially, r_0 can be set to any large value such that $e^{-\lambda_{ia}r_0} \ll 1$. Usually, a couple of steps of this formula are enough to yield an accurate estimate of r_M^*. A more accurate solution for r_M^* can be obtained from (3.6). After this, r_M^* can be updated quickly since the current value should not be too far off from the new value.

Fig. 1 shows r_2, r_3, and r_M^* versus different R values, where R_0 is approximately 137 units of time. It is clear that r_3 is a very close estimate of r_M^* although r_3 is always greater than r_M^*. r_2 is always less than r_M^*, and less accurate than r_3.

IV. Retry for Unclassified Faults

In this case, the type of the current fault (ω_i) is unknown. Let U be the expected loss if retry for ω_i fails, and let V (W) be the expected loss if retry for a transient (intermittent) fault ω_i is successful. L is then calculated as

$$L(r) = p_p \cdot U(r) + p_t \cdot \left[\int_0^r V(u)\lambda_{ta}e^{-\lambda_{ta}u}du + e^{-\lambda_{ta}r}U(r)\right] + p_i \cdot \left[\int_0^r W(u)\,\lambda_{ia}e^{-\lambda_{ia}u}du + e^{-\lambda_{ia}r}U(r)\right] \tag{4.1}$$

where p_p, p_t, and p_i are the probabilities of the detected fault being permanent, transient, and intermittent, respectively, and are computed as

$$p_p = \frac{\lambda_p}{\lambda_p+\lambda_t+\lambda_i}, \qquad p_t = \frac{\lambda_t}{\lambda_p+\lambda_t+\lambda_i},$$
$$p_i = \frac{\lambda_i}{\lambda_p+\lambda_t+\lambda_i}.$$

The expected losses U and V are defined as

$$U(r) = r + t_s + S,$$
$$V(r) = r + R.$$

The overhead that might occur due to any further occurrence of faults during R is not included in U and V. This greatly simplifies the computation without losing much of accuracy, since the probability of multiple faults occurring during the execution of a task is small and the difference between the overhead in U and that in V is even smaller.

For the same reason as above, the expected loss W is defined without including the overhead from any other faults. However, the overhead from future recurrences of the current intermittent fault must be considered since they are likely to occur many more times during R. If retry for ω_i is successful, the intermittent flag will be set when the first recurrence of ω_i is detected, and rollback/restart will be chosen at that time if $R > R_0$; otherwise, retry will be chosen. Since the overheads from other faults are ignored, R_0 in (3.3) is approximated as

$$R_0\left(1+\frac{\lambda_{ib}}{\lambda_{ia}}\right) = S + t_s.$$

Let x be the task execution time to be accrued from now till the first recurrence of ω_i, N_i be the number of recurrences of ω_i in the remaining task execution time, and O_i be the overhead

$$r_{k+1} = \frac{1}{\lambda_{ia}}\ln\left(\frac{\lambda_{ia}(1+\lambda_{ib}R)\left[r_k + K_S - \left(1 + \frac{e^{(\lambda_p+\lambda_t e^{-\lambda_{ta}r_k}+\lambda_{ib}e^{-\lambda_{ia}r_k})R}}{1+\lambda_{ib}R}\right)\frac{K_R}{2}\right]}{1 - e^{(\lambda_p+\lambda_t e^{-\lambda_{ta}r_k}+\lambda_{ib}e^{-\lambda_{ia}r_k})R}}\right);$$

of retry for all the recurrences of ω_i. If $R \le R_0$, then

$$W(r) = r + R\left(1 + \frac{\lambda_{ib}}{\lambda_{ia}}\right).$$

If $R > R_0$, then

$$\begin{aligned}
W(r) &= r + \text{Prob}[0 \le x < R - R_0] \\
&\qquad \times (S + t_s + E[x|0 \le x < R - R_0]) \\
&\quad + \text{Prob}[R - R_0 \le x \le R] \\
&\qquad \times (R + E[O_i|R - R_0 \le x \le R]) \\
&\quad + \text{Prob}[x > R]R \\
&= r + \left(1 - e^{-\lambda^{ib}(R-R_0)}\right)\left[S + t_s + \frac{R - R_0}{2}\right] \\
&\quad + \left(e^{-\lambda_{ib}(R-R_0)} - e^{-\lambda_{ib}R}\right) \\
&\qquad \times \left[R + \frac{R_0\lambda_{ib}}{\lambda_{ia}(1 - e^{-\lambda_{ib}R_0})}\right] \\
&\quad + e^{-\lambda_{ib}R}R \\
&= r + \left(1 - e^{-\lambda_{ib}(R-R_0)}\right)\left[S + t_s + \frac{1}{2}(R - R_0)\right] \\
&\quad + e^{-\lambda_{ib}(R-R_0)}\left[R - Re^{-\lambda_{ib}R_0} + \frac{R_0\lambda_{ib}}{\lambda_{ia}}\right] \\
&\quad + e^{-\lambda_{ib}R}R \\
&= r + S + t_s + \frac{1}{2}(R - R_0)\left(1 + e^{-\lambda_{ib}(R-R_0)}\right)
\end{aligned}$$

since

$$\begin{aligned}
E[O_i|R - R_0 \le x \le R] &= \frac{E[N_i]/\lambda_{ia}}{\text{Prob}[N_i \ge 1]} \\
&= \frac{\lambda_{ib}R_0}{\lambda_{ia}(1 - \text{Prob}[N_i = 0])} = \frac{\lambda_{ib}R_0}{\lambda_{ia}(1 - e^{-\lambda_{ib}R_0})}
\end{aligned}$$

and

$$R + \frac{R_0\lambda_{ib}}{\lambda_{ia}} = (R - R_0) + R_0\left(1 + \frac{\lambda_{ib}}{\lambda_{ia}}\right) = (R - R_0) + S + t_s.$$

Then,

$$\begin{aligned}
\frac{dL}{dr} &\equiv g(r) \\
&= p_p\frac{dU}{dr} + p_t\left[Vf_{A_t}(r) + (1 - F_{A_t}(r))\frac{dU}{dr} - Uf_{A_t}(r)\right] \\
&\quad + p_i\left[Wf_{A_i}(r) + (1 - F_{A_i}(r))\frac{dppU}{dr} - Uf_{A_i}(r)\right] \\
&= p_p - p_t[\lambda_{ta}(U - V) - 1]e^{-\lambda_{ta}r} - p_i \\
&\quad \times [\lambda_{ia}(U - W) - 1]e^{-\lambda_{ia}r}
\end{aligned}$$

since $\frac{dU}{dr} = \frac{dV}{dr} = \frac{dW}{dr} = 1$ from the definitions of U, V, and W. Furthermore,

$$\begin{aligned}
g'(r) = p_t\lambda_{ta}[\lambda_{ta}(U - V) - 1]e^{-\lambda_{ta}r} \\
+ p_i\lambda_{ia}[\lambda_{ia}(U - W) - 1]e^{-\lambda_{ia}r}.
\end{aligned}$$

It is easy to see that $\lim_{r\to\infty} g(r) = p_p > 0$, meaning that $\lim_{r\to\infty} L(r) = \infty$. Hence, the optimal retry period r_M^* is either 0 or a positive local minimizer of $L(r)$.

Theorem 2: $L(r)$ in (4.1) has at most one positive local minimizer r^* satisfying $g(r^*) = 0$ and $g'(r^*) > 0$.

In proving the above theorem, we found that $g'(r) = 0$ has exactly one solution at r_x which will be called the *inflection point* of $L(r)$.

To solve for r_M^*, we first check if $L(r)$ has a positive local minimizer, r_m. If such an r_m does not exist, then $r_M^* = 0$. If such an r_m exists and $g(r) < 0$, $\forall r < r_m$, then $r_M^* = r_m$, since $L(r)$ is monotonically decreasing in $[0, r_m]$; otherwise, r_M^* is either 0 or r_m depending on whether $L(0) \le L(r_m)$ or $L(0) > L(r_m)$.

Assuming that $\lambda_{ta}(U - V) - 1 > 0$, the problem can be divided into eight mutually exclusive cases, depending on the values of $g(0)$, $g'(0)$, $g(r_x)$, $\lambda_{ta} - \lambda_{ia}$, and $\lambda_{ia}(U - W) - 1$. The trivial case when $\lambda_{ta} = \lambda_{ia}$ will not be discussed.

Retry should not be attempted (i.e., $r_M^* = 0$) in the following four cases for various reasons:

Case 1: $\lambda_{ia}(U - W) - 1 \ge 0$ and $g(0) \ge 0$.

Since no inflection point exists and $\lim_{r\to\infty} g(r) = p_p$, $g(r)$ is monotonic and ranges from $g(0)$ to p_p. Because both $g(0)$ and p_p are positive, $g(r)$ is positive $\forall r \ge 0$. Thus, $r_M^* = 0$ because $L(r)$ is monotonically increasing.

Case 2: $\lambda_{ia}(U - W) - 1 < 0$, $g'(0)(\lambda_{ta} - \lambda_{ia}) \le 0$, and $g(0) \ge 0$.

Here, an inflection point does exist but is negative. So $g(r)$ is monotonic and positive on $[0, \infty]$ as in Case 1.

Case 3: $\lambda_{ia}(U - W) - 1 < 0$, $g'(0) > 0$, $(\lambda_{ta} - \lambda_{ia}) > 0$, and $g(0) \ge 0$.

A positive inflection point r_x exists, and $g(r_x) > g(0) \ge 0$ since $g'(0) > 0$. From the fact that $g(\infty) = p_p > 0$, it can be concluded that $g(r)$ is again positive $\forall r > 0$.

Case 4: $\lambda_{ia}(U - W) - 1 < 0$, $g'(0) < 0$, and $(\lambda_{ta} - \lambda_{ia}) < 0$ and $g(r_x) \ge 0$.

In this case, $g(r)$ is decreasing in $[0, r_x]$ but increasing in $[r_x, \infty]$. So, $g(r) \ge 0$ $\forall r > 0$ since $g(r) \ge g(r_x) \ge 0$.

Generally, if $g(0) \ge 0$, it is very likely that $r_M^* = 0$ except in Case 4 where we have to examine the value of $g(r_x)$.

In the remaining four cases, a positive local minimizer r_m exists—which can be derived rather quickly by any equal-interval search algorithm—if we can find two points r^- and r^+ such that $r^- < r_m < r^+$, $g(r^-) < 0 < g(r^+)$, and $g(r)$ is monotonically increasing in $[r^-, r^+]$. Since the larger or the smaller of λ_{ta} and λ_{ia} will be mentioned frequently in the discussion to follow, let $\lambda_{\max} = \max\{\lambda_{ta}, \lambda_{ia}\}$ and $\lambda_{\min} = \min\{\lambda_{ta}, \lambda_{ia}\}$.

Case 5: $\lambda_{ia}(U - W) - 1 \ge 0$ and $g(0) < 0$.

This is similar to Case 1 except that the range of $g(x)$, $[g(0), p_p]$, includes zero. And $g(r)$ is monotonically increasing because $g(0) < 0 < p_p$. Hence, r_m exists and $r_M^* = r_m$ because $g(r) < 0$, $\forall r < r_m$. To solve for r_m, let $r^- = 0$ and

$$r^+ = \frac{1}{\lambda_{\min}}\ln\left(\frac{p_p - g(0)}{p_p}\right).$$

because

$$\begin{aligned} g(r^+) &= p_p - p_t[\lambda_{ta}(U-V)-1]e^{-\lambda_{ta}r^+} \\ &\quad - p_i[\lambda_{ia}(U-W)-1]e^{-\lambda_{ia}r^+} \\ &> p_p - p_t[\lambda_{ta}(U-V)-1]e^{-\lambda_{\min}r^+} \\ &\quad - p_i[\lambda_{ia}(U-W)-1]e^{-\lambda_{\min}r^+} \\ &= 0. \end{aligned}$$

Case 6: $\lambda_{ia}(U-W)-1<0$, $g'(0)(\lambda_{ta}-\lambda_{ia})\leq 0$, and $g(0)<0$.

Since the inflection point r_x is negative, this case is similar to Case 5. Therefore, $r_M^* = r_m$ can be solved by letting $r^- = 0$ and

$$r^+ = \frac{1}{\lambda_{ta}}\ln\left(\frac{p_t[\lambda_{ta}(U-V)-1]}{p_p}\right),$$

because $\lambda_{ia}(U-W)-1<0$ and

$$\begin{aligned} g(r^+) &= p_p - p_i[\lambda_{ia}(U-V)-1]e^{-\lambda_{ta}r^+} \\ &\quad - p_i[\lambda_{ia}(U-W)-1]e^{-\lambda_{ia}r^+} \\ &> p_p - p_i[\lambda_{ia}(U-V)-1]e^{-\lambda_{ta}r^+} \\ &= 0. \end{aligned}$$

Case 7: $\lambda_{ia}(U-W)-1<0$, $g'(0)>0$, $(\lambda_{ta}-\lambda_{ia})>0$, and $g(0)<0$.

In this case a positive inflection point r_x exists, and $g(r)$ is increasing in $[0, r_x]$ but decreasing in $[r_x, \infty]$. So, $g(r_x) > g(\infty) = p_p > 0$. It is then easy to see that $0 < r_m = r_M^* < r_x$ and r_m can be obtained by letting $r^- = 0$ and $r^+ = r_x$.

Case 8: $\lambda_{ia}(U-W)-1<0$, $g'(0)<0$, and $(\lambda_{ta}-\lambda_{ia})<0$ and $g(r_x)<0$.

In this case, $g(r)$ is decreasing in $[0, r_x]$ but increasing in $[r_x, \infty]$. So, r_m should lie in $[r_x, \infty]$, and can be determined by letting $r^- = r_x$ and

$$r^+ = \frac{1}{\lambda_{ta}}\ln\left(\frac{p_t[\lambda_{ta}(U-V)-1]}{p_p}\right),$$

as in Case 6. But $r_M^* = r_m$ only if $g(0)\leq 0$, since $g(r)<0$, $\forall r < r_m$ in this situation. If $g(0)>0$, $L(r)$ will rise first, then fall to $L(r_m)$ before rising again. It is possible that $L(0)$ may be lower than $L(r_m)$. According to (4.1), $L(0) = U(0) = S + t_s$ and

$$\begin{aligned} L(r_m) &= p_pU(r_m) + p_t\Bigg[V(r_m)F_{A_t}(r_m) \\ &\quad - \int_0^{r_m} F_{A_t}(u)du + (1-F_{A_t}(r_m))U(r_m)\Bigg] \\ &\quad + p_i\Bigg[W(r_m)F_{A_i}(r_m) - \int_0^{r_m} F_{A_i}(u)du \\ &\quad + (1-F_{A_i}(r_m))U(r_m)\Bigg] \\ &= U(r_m) - p_t\Bigg[\left(U - V - \frac{1}{\lambda_{ta}}\right) \\ &\quad \times (1-e^{-\lambda_{ta}r_m}) + r_m\Bigg] \\ &\quad - p_i\Bigg[\left(U - W - \frac{1}{\lambda_{ia}}\right)(1-e^{-\lambda_{ia}r_m}) + r_m\Bigg] \\ &= S + t_s + p_pr_m - p_t(1-e^{-\lambda_{ta}r_m}) \\ &\quad \times \left(U - V - \frac{1}{\lambda_{ta}}\right) \\ &\quad - p_i(1-e^{-\lambda_{ia}r_m})\left(U - W - \frac{1}{\lambda_{ia}}\right). \end{aligned}$$

Therefore, in case of $g(0)>0$, if

$$\begin{aligned} &p_pr_m - p_t(1-e^{-\lambda_{ta}r_m})\left(U - V - \frac{1}{\lambda_{ta}}\right) \\ &\quad - p_i(1-e^{-\lambda_{ia}r_m})\left(U - W - \frac{1}{\lambda_{ia}}\right) < 0, \end{aligned}$$

then $r_M^* = r_m$ else $r_M^* = 0$.

A. Efficient Implementation

According to our extensive simulation results, Case 5 occurs most frequently, and hence, we need an efficient way to derive r_m in Case 5, which will greatly enhance the overall performance. In fact, in Case 5, if λ_{ta} and λ_{ia} do not differ by much, r_m can be derived by the recursive formula found at the bottom of the page with the convergence ratio of $(\lambda_{\max} - \lambda_{\min})/\lambda_{\min}$.

To prove the above convergence ratio, let $\alpha = \lambda_{\min}$ and $\beta = \lambda_{\max} - \lambda_{\min}$. Then, it can be shown that $g(r) = 0$ is equivalent to

$$p_pe^{\alpha r} = Ae^{-\beta r} + B \tag{4.2}$$

where both A and B are positive constants. Starting with any r_k, we get

$$\begin{aligned} p_pe^{\alpha r_{k+1}} &= Ae^{-\beta r_k} + B, \\ p_pe^{\alpha r_{k+2}} &= Ae^{-\beta r_{k+1}} + B. \end{aligned}$$

$$r_{k+1} = \begin{cases} \frac{1}{\lambda_{ia}}\ln\left(\frac{p_i[\lambda_{ia}(U-W)-1]+p_t[\lambda_{ta}(U-V)-1]e^{-(\lambda_{ta}-\lambda_{ia})r_k}}{p_p}\right), & \text{if } \lambda_{ta}\geq\lambda_{ia} \\ \dfrac{1}{\lambda_{ta}}\ln\left(\dfrac{p_t[\lambda_{ta}(U-V)-1]+p_i[\lambda_{ia}(U-W)-1]e^{-(\lambda_{ia}-\lambda_{ta})r_k}}{p_p}\right), & \text{if } \lambda_{ta}<\lambda_{ia} \end{cases}$$

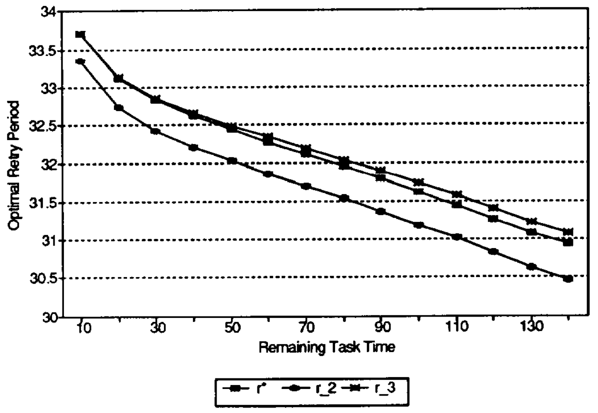


Fig. 1. r^*_M vs. R when intermittent, $t_s = 40$, $S = 200$, $1/\lambda_p = 2500$, $1/\lambda_t = 500$, $1/\lambda_i = 3500$, $1/\lambda_{ta} = 2$, $1/\lambda_{ia} = 3$, and $1/\lambda_{ib} = 4$.

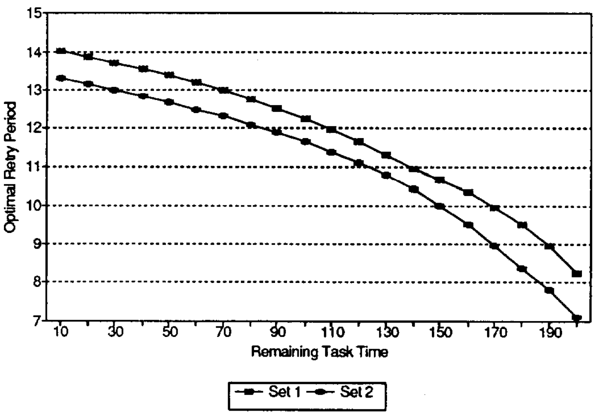


Fig. 2. r^*_M vs. R when unclassified, $t_s = 40$, and $S = 200$.

r_{k+1} will be either the largest or the smallest among r_k, r_{k+1}, and r_{k+2}, since one side of (4.2) is monotonically increasing and the other side is monotonically decreasing. If r_{k+1} is the largest, then

$$e^{\alpha(r_{k+1}-r_{k+2})} = \left(\frac{Ae^{-\beta r_k} + B}{Ae^{-\beta r_{k+1}} + B}\right) < \left(\frac{Ae^{-\beta r_k} + Be^{\beta(r_{k+1}-r_k)}}{Ae^{-\beta r_{k+1}} + B}\right) = e^{\beta(r_{k+1}-r_k)}.$$

Similarly, if r_{k+1} is the smallest, then

$$e^{\alpha(r_{k+2}-r_{k+1})} < e^{\beta(r_k - r_{k+1})}.$$

Thus, one can conclude that

$$|r_{k+1} - r_{k+2}| < \left(\frac{\beta}{\alpha}\right)|r_k - r_{k+1}|.$$

If $\beta/\alpha \leq 1/2$, this algorithm converges faster than any equal-interval search.

Figure 2 shows the r^*_M versus R plot for the following two sets of fault parameters:

	$1/\lambda_p$	$1/\lambda_t$	$1/\lambda_i$	$1/\lambda_{ta}$	$1/\lambda_{ia}$	$1/\lambda_{ib}$
Set 1	2500	500	3500	2	3	4
Set 2	2500	1000	3500	2	3	8

The value of R_0 is 137 in Set 1, and 175 in Set 2. The difference between Sets 1 and 2 is due mainly to the difference in λ_t. The more frequently transient faults occur, the greater the optimal retry period becomes. Comparing Fig. 2 with Fig. 1 it can be concluded that the r^*_M (the range of variation) for unclassified faults is much smaller (greater) than the r^*_M (the range of variation) for intermittent faults.

V. Performance Evaluation and Discussions

In the proposed retry policy, the retry period for a fault is determined on-line based on the current system's state. This retry policy will thus be called an *adaptive retry policy* (ARP). Though the ARP is optimal with respect to the Bayesian loss, it is interesting to see how it performs in general, as compared to other retry policies. Most commercial systems adopt a retry policy where the retry period is chosen *a priori* and fixed for all detected faults. This kind of retry policy will be called a *fixed retry policy* (FRP). In what follows, we will compare the ARP with FRPs with different retry periods via simulation.

A system with two identical processing modules is simulated. Tasks are executed on one module while the other module is used as a spare; the former is referred to as the *running module* and the latter as the *spare module* . When a fault is detected in the running module, a predetermined retry period is used in a FRP while in the ARP, the retry period is determined on-line by the methods described in Sections III and IV. If this retry fails, the current task will be restarted on the spare module which then becomes the running module. It is assumed that the faulty module can be repaired quickly and returns as the spare module, or a new module can be "hired" in as the spare module, before the next fault occurs. This essentially means that in our analysis, if a running module fails then a spare is always available.

There are two possibilities that cause a retry to fail. First, the retry period is shorter than the fault's active period. Second, a second fault occurs during the retry, which usually happens when an intermittent fault is resident in the running module for a sustained period.

To simplify the description of system workload, we assume that the running module is always busy executing tasks with exponentially-distributed execution times. The performance of a retry policy will be judged by the mean-time-between-repair (MTBR) and the mean *overhead ratio* which is defined as the percentage of time spent on retry and restart between repairs. A good retry policy should have a low overhead ratio but respectable MTBR, since unnessary repairs due to transient faults should be minimized while handling permanent and intermittent faults effieciently.

Each time a fault occurs, the S value of the current task is randomly generated based on an exponential distribution and the R value of the current task is distributed uniformly between 0 and S, unless the task affected by an earlier fault has not yet been completed; in such a case R is calculated based on the previous detection time.

The simulation is run under a basic fault generation process with the following fault parameters:

$1/\lambda_p$	$1/\lambda_t$	$1/\lambda_i$	$1/\lambda_{ta}$	$1/\lambda_{ia}$	$1/\lambda_{ib}$
5000	500	10000	2	5	10

The inverses of rate parameters are listed above because they are easy to compare with other time-related variables. For example, $1/\lambda_p$ is the mean time between two permanent faults, $1/\lambda_{ta}$ is the mean active period of a transient fault, and so on. The maximum amount of time for diagnosing a faulty module is assumed to be 100, i.e., any fault with an active period greater than 100 will be classified as a permanent fault by our fault classification procedure. The mean execution time for each task is assumed to be 200 which, in our simulation, is the mean value assigned to S since restart is used as the only alternative when retry fails. The setup time for restart recovery, t_s, is assumed to be 20. All these variables have the same time unit and thus are listed without explicitly specifying their units. It is further assumed that there are 10 different detection symptoms and their occurrences are uniformly distributed.

The following retry policies are simulated and compared with the ARP.

Policy F0: The policy that initiates restart/rollback upon every fault detection.

Policy Fn: The policy that employs a fixed retry period of n.

The mean task execution time does not have any notable effect on the performance of ARP, since the optimal retry period is partially dependent on the value of S. However, the task execution time may have great effects on the performance of certain FRPs. Generally, the FRP with a longer retry period performs better for those tasks with longer execution times, as can be seen in Fig. 3. In what follows, we will compare these retry policies while changing only one parameter at a time.

The first comparison is made while varying t_s from 20 to 100 and the results are plotted in Fig. 4. The effect of increasing t_s is similar to the effect of increasing S. Generally, the larger the t_s, the larger the overhead ratio. The ARP has the lowest overhead ratio and among FRP's, F10 always has lower overhead than any other FRP's within the range of t_s mentioned above. The t_s does not affect MTBR too much in all policies. It should be pointed out that policies with a smaller retry period will have a smaller MTBR, but a very short retry period (e.g., F5) and a very long retry period (e.g., F30) usually produce a large overhead ratio. While the ARP's MTBR is not the largest, its value does not differ from the largest value by too much. The task execution time will have similar effects (as t_s) on FRPs since the greater the task execution time, the larger the restart overhead may result.

The second comparison is made while varying $1/\lambda_t$ from 300 to 1500, and the results are plotted in Fig. 5. The overhead ratio tends to decrease when the mean time between transient faults increases. However, the magnitude of decrease is almost indistinguishable in F20 and this trend is even reversed in F30,

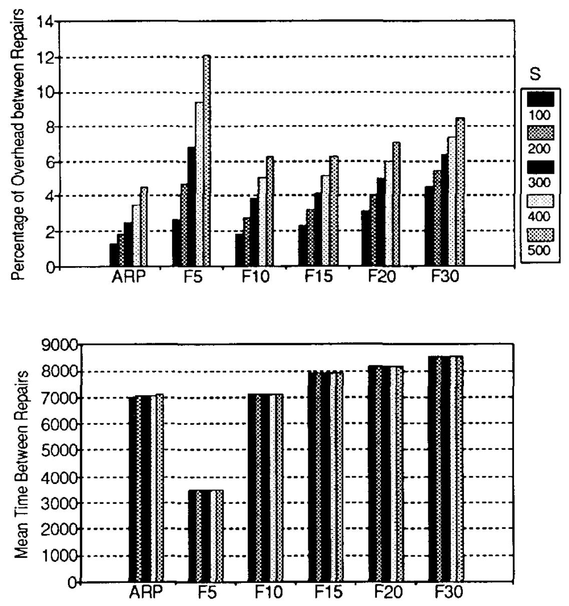


Fig. 3. The impact of S in different retry policies.

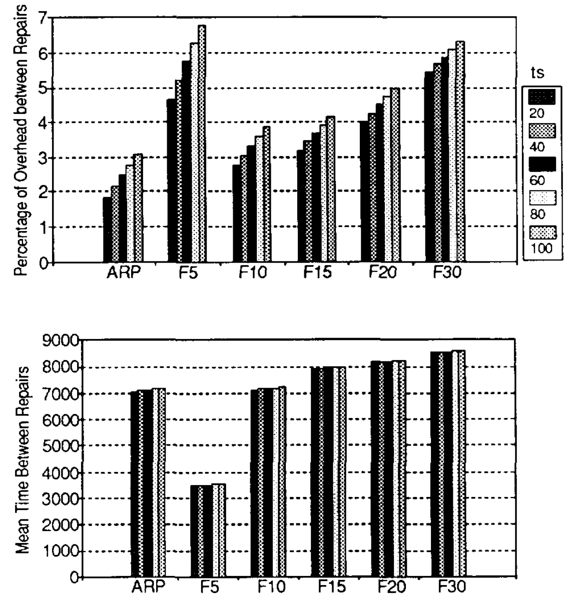


Fig. 4. The impact of t_s in different retry policies.

because their retry period is usually larger than the transient fault's active period, which is small and is the primary source of the overhead. In policies with a shorter retry period such as F5, the difference is quite significant. The change of λ_t has great impact on the MTBR.

The third comparison is made while varying $1/\lambda_{ta}$ from 1 to 5, and the results are plotted in Fig. 6. It is evident that for FRPs the overhead is very sensitive to the transient fault's active period unless the retry period is much larger than the active period, which is the case in F20 and F30. For

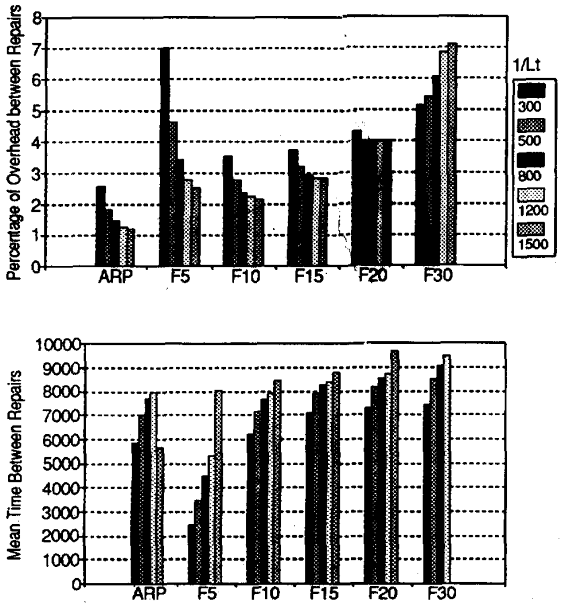


Fig. 5. The impact of λ_t in different retry policies.

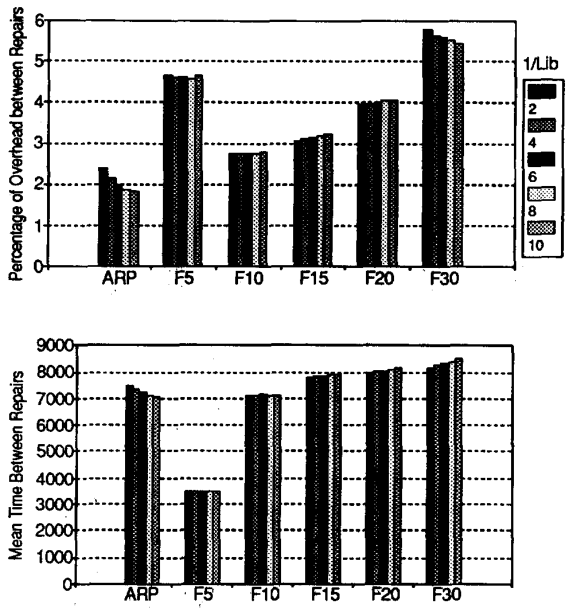


Fig. 7. The impact of λ_{ib} in different retry policies.

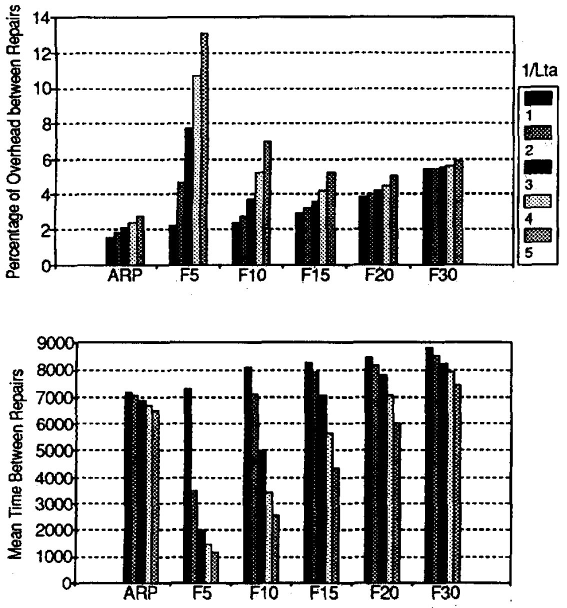


Fig. 6. The impact of λ_{ta} in different retry policies.

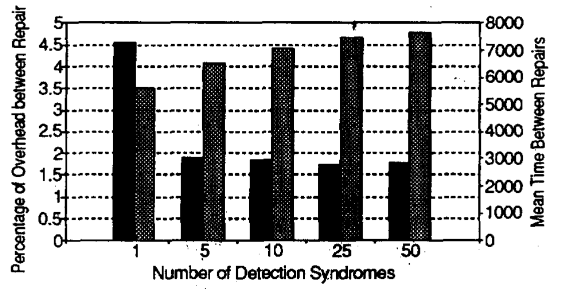


Fig. 8. The impact of different number of detection syndromes for ARP.

the ARP, the overhead ratio is consistently low for all active periods.

The fourth comparison is made while varying $1/\lambda_{ib}$ from 2 to 10, and the results are plotted in Fig. 7. The overhead ratio appears to be insensitive to the benign period of intermittent faults in all retry policies, because intermittent faults occur so infrequently that the overhead induced by intermittent faults constitutes only a small portion of the total overhead. In the ARP and F30, the overhead decreases as the benign period gets larger since retries for intermittent faults are consistently successful, unlike other FRPs with a shorter retry period. However, F30 has a much higher overhead ratio because it cannot identify the occurrence of intermittent faults and make timely repair.

It is obvious from the above comparisons that the overhead ratio in the ARP is always lower than those in FRPs. In many cases, however, the performance of F10 is quite close to that of ARP. This is due to the set of parameters used in the simulation; since from Fig. 2, the optimal retry period is in the range from 7 to 13, and the retry period in F10 is closest to the optimal value among FRPs. However, if the fault paramters change, F10 may not remain to be the best FRP, while the ARP will still be the optimal regardless of the change of parameters because it estimates the parameters on-line based on the detected faults.

To see the impact of the fault classification scheme on the ARP, we carried out simulations while varying the number of detection symptoms from 1 to 100. In Fig. 8, both the overhead ratio and the MTBR are plotted. The overhead ratio improves with the number of detection symptoms, because the classification gets more accurate with more detection symptoms. The performance enhancement is very significant when the number of detection symptoms is increased from 1 to 5, but the effect of any further increase shows only a marginal imrpovement. Thus, it is not necessary to design a complicated fault detection scheme with a large number of detection symptoms solely for the sake of the ARP.

VI. Conclusion

In this paper, we have developed an adaptive retry policy which does not require any accurate prior knowledge of the fault parameters and adapts itself to the changes of parameters during operation. A distinct feature of this policy is the treatment of intermittent faults. Whenever an intermittent fault is judged to have occurred, an optimal retry period is determined and then applied to all future recurrences of the same intermittent fault until the current task is completed, and at that time, the module containing the intermittent fault will be switched out—even if the fault is not active—to prevent any further damages (by this fault) to the system.

The main reason that our adaptive retry policy performs so well under a changing environment is that parameters are estimated on-line using the most up-to-date information. The on-line parameter estimation is made possible by using the fault classification scheme we developed earlier [13]. Based on this fault classification scheme, every detected fault will eventually be classified as one of the four fault types: permanent fault, transient fault, first occurrence and recurrence of an intermittent fault. Extensive efforts were made on the derivation of the optimal retry period since this must be done on-line. Numerical algorithms were also developed so that an approximate optimal retry period can be obtained quickly.

Our adaptive retry policy is effective as long as faults are detected immediately upon their occurrence. If there is a latency between fault occurrence and detection, the parameter estimation has to be modified to account for this latency. Furthermore, if the task is contaminated before a fault is detected, the instruction retry defined in this paper cannot succeed in recovering the system even if the fault disappears. These problems are interesting, difficult, and worthy of further investigation.

Appendix
Proofs of Theorems 1 and 2

Proof of Theorem 1: Under the first assumption, we get

$$\frac{dJ}{dr} \approx -\lambda_{ib} R \lambda_{ia}^2 e^{-\lambda_{ia} r}.$$

The first and second derivatives of $L(r)$ are thus approximated by

$$L'(r) \approx 1 - e^{-\lambda_u R} - \lambda_{ia}(1+\lambda_{ib}R)e^{-\lambda_{ia} r}e^{-\lambda_u R}$$
$$\left[r + K_S - \left(1 + \frac{e^{\lambda_u R}}{1+\lambda_{ib}R}\right)\frac{K_R}{2}\right],$$
$$L''(r) \approx \lambda_{ia}^2(\lambda_{ib}R+1)e^{-\lambda_{ia} r}e^{-\lambda_u R}$$
$$\left[r + K_S - \left(1 + \frac{e^{\lambda_u R}}{\lambda_{ib}R+1}\right)\frac{K_R}{2} - \frac{2}{\lambda_{ia}}\right].$$

$e^{\lambda_u R}$ is approximated by $\lambda_u R + 1$ because the rate of unsuccessful retries during R is very small. λ_u is approximated by λ_p because $e^{-\lambda_{ia} r} \ll 1$.

$$\frac{e^{\lambda_u R}}{1+\lambda_{ib}R} \approx \frac{1+\lambda_p R}{1+\lambda_{ib}R} < 1,$$

because λ_{ib} is usually much larger than λ_p. So,

$$K_S - \left(1 + \frac{e^{\lambda_u R}}{1+\lambda_{ib}R}\right)\frac{K_R}{2} > K_S - K_R > 0$$

and the equation $L'(r) = 0$ can be approximated by

$$e^{-\lambda_{ia} r} = \frac{\lambda_p R}{\lambda_{ia}(1+\lambda_{ib}R)\left[r + K_S - \left(\frac{1+\lambda_p R}{1+\lambda_{ib}R}\right)\frac{K_R}{2}\right]}. \tag{A.1}$$

The left-hand side (LHS) of (A.1) is a positive, decreasing function of r, so is the right-hand side (RHS). When r is small, LHS is larger than RHS. But, when r becomes larger, LHS will eventually be smaller than RHS, since LHS has a much higher decreasing rate. Therefore, LHS = RHS will hold at one and only one point, r^*. From (A.1), $L''(r^*) > 0$ if the third condition of the theorem is satisfied. Hence, r^* is the only local minimizer of L. □

Proof of Theorem 2: It is assumed that

$$\lambda_{ta}(U-V) - 1 = \lambda_{ta}\left(t_s + (S-R) - \frac{1}{\lambda_{ta}}\right) > 0$$

even when $R = S$, since t_s is usually much longer than the mean active period of a transient fault, $1/\lambda_{ta}$. Consider the following cases. If $\lambda_{ta} = \lambda_{ia}$, then either $g(r) \equiv p_p$ or $g(r)$ is a monotonic function. If $\lambda_{ia}(U-W) - 1 \geq 0$, then $g'(r) > 0$, and thus, $g(r)$ is a monotonically increasing function of r. If $\lambda_{ia}(U-W) - 1 < 0$, then $g'(r) = 0$ has exactly one solution at

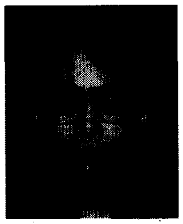
$$r_x = \frac{1}{(\lambda_{ta} - \lambda_{ia})}\ln\left(\frac{-p_t\lambda_{ta}[\lambda_{ta}(U-V)-1]}{p_i\lambda_{ia}[\lambda_{ia}(U-W)-1]}\right),$$

which will be called the *inflection point* of $L(r)$. If $r_x > 0$, $g(r)$ is monotonically increasing in either $[0, r_x]$ or $[r_x, \infty]$; if $r_x < 0$, $g(r)$ is a monotonic function. It can be concluded from the above cases that $L(r)$ has at most one positive local minimizer. □

References

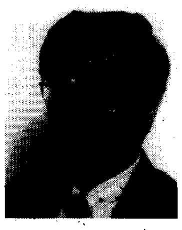
[1] K. G. Shin and Y.-H. Lee, "Error detection process—model, design, and its impact on computer performance," *IEEE Trans. Comput.*, vol. C-33, pp. 529–540, June 1984.
[2] T. Anderson and P. A. Lee, *Fault Tolerance Principles and Practice*. London: Prentice-Hall International, 1981.
[3] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*. Bedford, MA: Digital Equipment Corporation, 1982.
[4] D. P. Siewiorek, V. Kini, H. Mashburn, S. R. McConnel, and M. M. Tsao, "A case study of c.mmp, cm*, and c.vmp: Part i—Experiences with fault tolerance in multiprocessor systems," *Proc. IEEE*, vol. PROC-66, pp. 1178–1199, Oct. 1978.
[5] O. Tasar and V. Tasar, "A study of intermittent fault in digital computers," in *Proc. Nat. Comput. Conf.*, June 1977, pp. 807–811.
[6] L. A. Boone, H. L. Liebergot, and R. M. Sedmak, "Availaiblity, reliability, and maintainability aspects of the sperry univac 1100/60," in *Dig. of Papers, FTCS-10*, June 1980, pp. 3–9.
[7] W. C. Carter, "A short survey of some aspects of hardware design techniques for fault tolerance," IBM Res. Rep. RC-10811, IBM, Yorktown Heights, NY, 1984.

[8] D. L. Droulette, "Recovery through programming system/360-system/370," in *Proc. 1971 AFIPS Conf.*, vol. 38, Spring 1971, pp. 467–476.
[9] G. H. Maestri, "The retryable processor," in *Proc. 1972 AFIPS Conf.*, vol. 41, Fall 1972, pp. 273–277.
[10] M. Berg and I. Koren, "On switching policies for modular redundancy fault-tolerant computing systems," *IEEE Trans. Comput.*, vol. C-36, pp. 1052–1062, Sept. 1987.
[11] I. Koren, Z. Koren, and S. Y. H. Su, "Analysis of a class of recovery procedures," *IEEE Trans. Comput.*, vol. C-35, pp. 703–712, Aug. 1986.
[12] Y.-H. Lee and K. G. Shin, "Optimal design and use of retry in fault-tolerant computer systems," *J. ACM*, vol. 35, pp. 45–69, Jan. 1988.
[13] T.-H. Lin and K. G. Shin, "A bayesian approach to fault classification," *Performance Evaluation Review*, vol. 18, no. 1, pp. 58–66, 1990.
[14] J. O. Berger, *Statistical Decision Theory, Foundations, Concepts, Methods*, 2nd ed. New York: Springer-Verlag, 1985.
[15] M. H. DeGroot, *Optimal Statistical Decisions*. New York: McGraw-Hill, 1970.



Tein-Hsiang Lin (S'83–M'88) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Republic of China, in 1980, the M.S. degree in electrical engineering from the Iowa State University, Ames, in 1984, and the Ph.D. degree in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1988.

He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at the State University of New York at Buffalo, where he has served on the faculty since 1988. His research interests include fault-tolerant computing, real-time systems, computer architecture, and medical imaging. He has published many articles on checkpointing, fault diagnosis, retry and reconfiguration, real-time task scheduling, X-ray cone-beam 3-D reconstruction, and confocal microscopy.



Kang G. Shin (S'75–M'78–SM'83–F'92) received the B.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D. degrees in Electrical Engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively.

He is Professor and Associate Chair of Electrical Engineering and Computer Science for the Computer Science and Engineering Division, The University of Michigan, Ann Arbor, Michigan. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA. He has also been applying the basic research results of real-time computing to manufacturing-related applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. Recently, he has initiated research on the open-architecture Information Base for machine tool controllers.

Dr. Shin has authored/coauthored over 240 technical papers (about 110 of these in archival journals) and several book chapters in the area of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called HARTS, to validate various architectures and analytic results in the area of distributed real-time computing. He was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems, a Program Co-Chair for the 1992 *International Conference on Parallel Processing,* and served numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991–93, is a Distinguished Visitor of the Computer Society of the IEEE, an Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED COMPUTING, and an Area Editor on *International Journal of Time-Critical Computing Systems.*