NASA TM-86369

NASA Technical Memorandum 86369

A Theoretical Basis for the Analysis of Redundant Software Subject to Coincident Errors

Dave E. Eckhardt, Jr. and Larry D. Lee

JANUARY 1985

NASA
National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665

SUMMARY

Fundamental to the development of redundant software techniques (known as fault-tolerant software) is an understanding of the impact of multiple joint occurrences of errors, referred to here as coincident errors. A theoretical basis for the study of redundant software is developed which (1) provides a probabilistic framework for empirically evaluating the effectiveness of the general (N-Version) strategy when component versions are subject to coincident errors, and (2) permits an analytical study of the effects of these errors. The basic assumptions of the model are: (i) independently designed software components are chosen in a random sample and (ii) in the user environment, the system is required to execute on a stationary input series. An intensity function, called the intensity of coincident errors, has a central role in the model. This function describes the propensity of a population of programmers to introduce design faults in such a way that software components fail together when executing in the user environment. The model is used to give conditions under which an N-Version system is a better strategy for reducing system failure probability than relying on a single version of software. In addition, a condition which limits the effectiveness of a fault-tolerant strategy is studied, and we ask whether system failure probability varies monotonically with increasing N or whether an optimal choice of N exists.

1.0 INTRODUCTION

The use of independently designed, redundant software is an intuitively appealing approach to increasing software reliability. The redundancy principle, after all, has long been accepted as an effective means for improving the reliability of hardware devices. The basic premise in both cases is that components (either software or hardware) will have independent failure characteristics so that the probability of failures occurring simultaneously is small (ideally the product of the individual component failure probabilities). Fault-tolerant software is the methodology for structuring software components to cope with residual software design faults. The most widely known, N-Version programming [1] and recovery block [2], are analogous to the hardware techniques of N-Modular redundancy and stand-by sparing, respectively.

Although redundancy has been successfully applied to fault-tolerant computer systems (e.g., [3], [4]), its application to software has been slow to develop. One reason for this may be that little empirical data is available that demonstrates an increase in reliability sufficient to justify the increased cost of the software development, although it has been suggested that fault-tolerant software is cost effective [5].

More importantly, however, is the reliability degradation of fault-tolerant software structures caused by either: (1) multiple faults which produce dissimilar outputs but are manifested by the same input conditions, or (2) related software design faults causing identical incorrect outputs. The general notion of related software design faults is often referred to as "correlated" faults. This term, however, appears to have different meanings to different authors and it is sometimes not clear what combinations of the above fault types and the degree of the attribute, "related", is being discussed. We will refer, collectively, to errors manifested by both of the above fault types as coincident errors. What will distinguish correlated errors from those that occur simultaneously by chance, we presume, will be the intensity of coincident errors as discussed in more detail later. In an extreme case one might imagine that all residual design faults are common to all versions of a redundant software structure and thus there is no reliability gain over randomly selecting a single version of the software. More typical might be a situation where a majority of identical faulty modules, in a voting scenario, outvote the correct versions which are in the minority.

Although it is true that detected failures are potentially less serious than undetected failures since control, in the case of detected failures, can be passed to a higher authority, both are, in fact, failures of the fault-tolerant structure. For applications in which fault-tolerant software is performing some critical function, we take the conservative position that any higher authority could not adequately cope with this loss of critical function and that there is no safe-down state to repair the software (more likely reset to some initial state). Thus we are concerned with both types of errors, which are described by coincident errors.

Given that coincident errors are potentially devastating to redundant software systems, it is fundamental to understand and assess the effects of these errors, both analytically and empirically, on the general strategy of software redundancy. Hardware designers, to date, have not been concerned with this issue. The assumption is that hardware components do not share common design faults but rather it is their independent degradation processes which mainly contribute to unreliability. Independence, then justifies the use of combinatorial methods for estimating hardware reliability. In the independence case, conditions for which redundancy is a better strategy for reducing failure probability than the use of a single component are well known [6].

In the case of redundant software, it is suggested in [7] that the independence model when applied to software components leads to poor predictions of reliability. Further, the analysis given in this paper shows that for cases of coincident errors which appear reasonable to expect in applications, the independence model gives estimates which fail to be conservative.

Upon recognizing that statistically independent failures among software components is a questionable assumption, the model suggested in [8] includes a "correlation" factor. However, it too assumes a form of higher order independence by representing the probabilities of joint occurrence of identical, incorrect output in terms of the probability of pairwise occurrence of such events. Furthermore, since the probability of identical, incorrect output among component versions will likely vary with the input, the idea that all of this complexity can be captured in a single scalar correlation coefficient is questionable. For this reason, we employ an intensity function defined on the input space (similar in several ways to a parameter vector) which permits variation in the probability that software components fail together. We shall not attempt to evaluate one fault-tolerant technique over another but rather we shall examine the principle of redundant software as represented by multiple (i.e., N) versions which are independently developed to a common set of requirements and then operationally subjected to a perfect majority voter.

We submit that there are a number of questions which must be answered in order to provide a basic understanding of the effects of coincident errors on redundant software. The framework discussed in the present paper does not require unnecessary assumptions concerning independent failure of software components; rather a model is derived from assumptions concerning the process of selecting independently designed software components and testing them on an input series chosen to emulate the user environment. In other words, we believe the model has sufficient generality to warrant conclusions concerning questions of the following type:

(1) Is an N-Version software structure always more effective at reducing failure probability than a single version of software? If not, what are the conditions which cause this?

(2) What are the effects of different intensities of coincident errors on a general N-Version system?

(3) What are the effects of increasing N? Does the failure probability always increase or decrease with increasing N, as for the independence model used for hardware, or might there exist an optimal choice of N other than N=1 or N=∞? Is there a limit on the effectiveness of fault-tolerance at reducing the probability of failure?

(4) Does the independence model give a valid estimate of the failure probability of a redundant N-Version system?

(5) Under what conditions does the assumption of independence hold?

In order to give a framework for evaluating the effectiveness of a fault-tolerant strategy and, in particular, to answer the above questions, we propose a model based on formalizing the notion of coincident errors. The basic assumptions of this model are: (i) that independently designed software components are chosen in a random sample and (ii) each component and each system is required to execute on a stationary, independent input series. We derive the failure probability of a redundant N-Version system and establish general conditions giving answers to (1), (3), and (5) above. The main quantities describing the model are: an intensity function defined on the input space which models the occurrence of coincident errors and a usage distribution which gives the probabilities of inputs occurring in various subsets. Also important to our description is an intensity distribution derived from the intensity function and the usage distribution. The intensity distribution completely specifies the failure probability of a redundant system; that is, if the intensity distribution is known or can be estimated, answers to questions of type (1) - (5) can be given. Since empirical information concerning the intensity distribution is unavailable, we study the effects of coincident errors by varying the choice of intensity distributions.

Notation

We follow the usual convention in which random variables are denoted by capital letters and their realizations are denoted by the corresponding lower case. We also use the following:

Ω input set for software components designed to a common specification;

x a variable representing elements of Ω;

Q the usage distribution, a probability measure defined on (measurable) subsets of Ω;

$v(x)$ the score function, a binary function distinguishing the occurrence of correct and incorrect output when a software component executes on $x \epsilon \Omega$;

$\theta(x)$ intensity of coincident errors;

$E(\cdot)$ $(P(\cdot))$ mathematical expectation (probability) derived from a product probability space as specified by the two-stage process of selecting software components at random and testing them on inputs chosen at random from Ω;

p_N average probability of failure of an N-Version system

p average probability of failure of a single software component;

N number of software components in a multiple version program;

n number of software components chosen in a random sample;

$G(y)$ intensity distribution induced by the mapping $x \rightarrow \theta(x)$ from Ω into $[0, 1]$;

$G\_(y)$ left continuous version of $G(y)$;

$g(\theta)$ probability mass function for a discrete intensity distribution;

$h(y;N)$ $\sum_{\ell=m}^{N} \binom{N}{\ell} y^{\ell}(1-y)^{N-\ell}$, $0 \le y \le 1$ where $m = (N+1)/2$;

σ^2 variance of the intensity distribution;

$\phi(y;N)$ $h(y;N) - y$.

2.0 THE MODEL

Suppose we are told that a particular software component, having input set Ω, gives incorrect output when executing on inputs in some subset F of Ω and gives correct output when executing on inputs in the complementary set F'. If all inputs arriving in the user environment belong to F, then the component is totally unreliable whereas if all inputs arrive in F', then the component is perfectly reliable. It is clear that some structure is required of the input process in order to evaluate reliability; for example, the inputs could alternate between F and F' or they may occur randomly in Ω.

We assume that an input series X_1, X_2,..., is stationary and independent; that is, successive inputs occur or are chosen at random in a series of independent trials according to a common distribution. Some software reliability models [9] and software testing experiments [10], [11] implicitly assume or suggest this structure. The common distribution, say Q, is the usage distribution which gives the probabilities Q(A) that successive inputs are chosen at random in subsets A of Ω.

At this stage of the discussion, other than the usage distribution itself, the full probabilistic structure of an input series is not needed. Our concern is mainly with the probability that a software component, and a redundant structure developed from a set of components, fails on successive trials.

Let $v(x)$, $x \epsilon \Omega$ denote the score function for a particular component: $v(x) = 1$ ($v(x) = 0$) if the component gives incorrect (correct) output when executing on $x \epsilon \Omega$. Note that the subset F of Ω for which the component gives incorrect output is $\{x: v(x) = 1\}$. The probability $Q(F)$ that the particular component fails on successive trials is

$$Q(F) = \int v(x) dQ. \qquad (1)$$

Now consider either a physically existing population of programmers who would design software to a given specification, or a conceptual population based on what would happen in a large number of repetitions of an experiment such as one which is designed to study the long term effectiveness of a fault-tolerant strategy. Let $\theta(x)$ describe the proportion of this population giving errors in the output when executing on $x \epsilon \Omega$. This intensity function can be interpreted a number of ways: for example, it models the occurrence of coincident errors; it gives the probability that a software component, when chosen at random, fails on a particular input; and it describes a propensity for software components to fail together when executing on a single input.

If a component is chosen at random, then for fixed $x \epsilon \Omega$, its score function $V(x)$ is a binary random variable taking values zero and one with probabilities $1 - \theta(x)$ and $\theta(x)$ and, therefore, its mathematical expectation is $E[V(x)] = \theta(x)$ for each $x \epsilon \Omega$.

As previously stated, (1) gives the probability that a particular component gives an error in its output. This probability, however, is a random variable which varies over repeated selections of software components. The mean of its distribution is

$$E[\int V(x)dQ] = \int \theta(x)dQ. \qquad (2)$$

The conceptual distinction between (1) and (2) is analogous to the process for estimating the reliability of hardware devices. That is, they capture, respectively, the difference between the reliability of a particular hardware device and the reliability of a population of devices of its type. While reliability predictions are actually desired for the device on hand, they are usually made on the basis of empirical results reported from testing a subset of the population.

Neither the score function nor its expected value has introduced any assumptions to our model. However, when describing the reliability of a redundant structure, we need to state what is meant by independently designed versions of software components. We shall mean a set of n components which is chosen at random from a population so that: (a) $\{V_1(x);\ x\varepsilon\Omega\}$, $\{V_2(x);\ x\varepsilon\Omega\}$, ..., $\{V_n(x);\ x\varepsilon\Omega\}$ are independent collections of random variables and (b) for each $x\varepsilon\Omega$, $V_1(x)$, $V_2(x)$, ..., $V_n(x)$ are identically distributed random variables. This assumption describes the usual conditions required of a random sample. The condition that $V_1(x)$, $V_2(x)$, ..., $V_n(x)$ are identically distributed implies that the probabilities $\int V_i(x)dQ$, $i = 1, 2, \ldots, n$ of incorrect outputs, which are themselves random variables, vary according to a common distribution and the mean of this distribution is $\int\theta(x)dQ$, as given earlier. Note that condition (a) is similar to the condition defining independence of a collection of stochastic processes indexed by a time parameter. It is also similar to the process of recording independent vector measurements for a sample of individuals taken from a human population. We emphasize that statistical independence in the current context refers only to the selection process and does not imply statistically independent failures among software components. This point is discussed further in Section 3.0.

Although empirical studies of fault-tolerant software are not likely to often be conducted in the strict sense defining a random sample, repetitions of the version selection process does involve uncertainty concerning the subsets of Ω on which the component versions fail. The probabilistic structure implied by the conditions defining a random sample gives a meaningful way to interpret experimental results when the main interest lies in the long term effectiveness of a fault-tolerant strategy rather than the study of a particular instance of its application.

Now consider an N-Version (N = 1, 3, 5, ...) structure consisting of N software components, each designed to a common specification and required to execute on a single input series in the user environment. The outputs given after each execution are compared and, in case of disagreement, a consensus result is obtained by majority vote. An N-Version structure fails when executing on some subset F of Ω and, as before, this subset is conveniently described by a score function $v(x)$, $x \epsilon \Omega$, which is

$$v(x) = \sum_{\ell=m}^{N} \Sigma\, v_{i(1)}(x) \ldots v_{i(\ell)}(x)[1 - v_{i(\ell+1)}(x)] \ldots [1-v_{i(N)}(x)] \qquad (3)$$

where $v_{i(1)}(x), v_{i(2)}(x), \ldots, V_{i(N)}(x)$ is a permutation of the score functions for the component versions. The second sum in (3) is over all distinct subsets of $\{1, 2, \ldots, N\}$ and $m = (N+1)/2$ corresponds to the case of a redundant system that fails when at least a majority of its components fail.

We now state the main result of this Section:

Theorem 1. Under the condition that the component versions are the result of a random sample and each is required to execute on common inputs chosen at random, the expected probability of system failure is

$$p_N = \int \sum_{\ell=m}^{N} \binom{N}{\ell} [\theta(x)]^{\ell} [1 - \theta(x)]^{N-\ell} dQ. \qquad (4)$$

Proof. Upon conditioning on $V_1(\cdot), V_2(\cdot), \ldots, V_N(\cdot)$, the probability of failure is $\int v(x)dQ$ where $v(\cdot)$ is given by (3). Now taking the expectation inside the integral and using the independence of $V_1(\cdot), \ldots, V_N(\cdot)$ due to sampling, together with the condition $E[V_i(x)] = \theta(x)$, $i = 1, 2, \ldots, N$, gives the desired result.

Although the main interest may often lie in the probability, $\int v(x)dQ$ (where $v(x)$ is given by (3)), of failure of a particular N-Version system rather than the population average, p_N, as given by (4), the quantity, $\int v(x)dQ$, will vary from one application to another and, unless we replace $v(x)$ by its expected value as done in (4), there is no basis for further simplification. This same point was mentioned earlier when comparing (1) and (2) and, as before, is analogous to the difference between the reliability of a particular hardware device and the average reliability for a population of devices of its type.

While $\theta(x)$, $x \epsilon \Omega$ together with N and the usage distribution completely specify p_N, little empirical evidence is available from which to estimate $\theta(x)$, $x \epsilon \Omega$; thus reasonable choices of the intensity to expect in applications is unclear. For this reason, we reparameterize p_N in terms of the following:

$$h(y;N) = \sum_{\ell=m}^{N} \binom{N}{\ell} y^{\ell}[1 - y]^{N-\ell}, \quad 0 \leq y \leq 1 \tag{5}$$

and

$$G(y) = \int_{\{x:\theta(x) \leq y\}} dQ, \quad -\infty < y < \infty \tag{6}$$

We shall refer to $G(y)$ as the intensity distribution which is induced by the mapping $x \rightarrow \theta(x)$ from Ω into $[0, 1]$.

Before proceeding to give a reparameterization of p_N, consider the interpretation of $G(y)$ in the discrete case which arises when $\theta(x)$ takes a finite number of values over subsets of Ω. Suppose $\theta(x) = \theta_i$ for $x \varepsilon A_i$ where $A_1, A_2, \ldots, A_r$ is a partition of Ω and suppose the sets giving a common value under the mapping have been combined and indexed so that $0 \leq \theta_1 < \theta_2 < \ldots < \theta_r \leq 1$. Then, in this case,

$$G(y) = \sum_{\{i:\ \theta_i \leq y\}} q_i, \quad -\infty < y < \infty \tag{7}$$

where $q_i = Q(A_i)$, $i = 1, 2, \ldots, r$ is the probability mass given by the usage distribution. Since $G(y)$ is right continuous, $G(b) - G(a)$ gives the probability that inputs are chosen so that the proportion of a population of components that fail is in the range $(a, b]$, $a < b$ (the upper limit, b, is included in the interval $(a,b]$ but the lower limit, a, is not included).

For later reference, we restate our earlier result in reparameterized form:

Corollary. Under the conditions stated in the previous theorem,

$$p_N = \int h(y;\ N)\,dG(y) \tag{8}$$

where $h(y;\ N)$ is given in (5) and $G(y)$ is given by (6).

The result follows by substitution (e.g., see [12], p. 43).

3.0 INDEPENDENT ERRORS

The assumption that failures occur independently (in a statistical sense) in hardware components is a widely used and often successful model for predicting the reliability of hardware devices. Thus, it is tempting to assume that software components also fail independently and, on this basis, estimate the failure probability of a redundant N-Version system from

$$\sum_{\ell=m}^{N} \binom{N}{\ell} p^{\ell}(1-p)^{N-\ell}. \tag{9}$$

This gives a computationally convenient formula for which the only information required is the average failure probability p of the software components. However, it clearly differs from the representation of p_N given earlier in (4). In this Section we ask whether independence implies a condition on the intensity distribution which is reasonable to expect in applications. Also, we ask whether it is correct to interpret a low intensity as implying statistical independence and a high intensity as implying statistical dependence in the context of coincident errors.

Consider for the moment only two versions. Suppose, as before, they are chosen in a random sample and each is required to execute on common inputs chosen at random from Ω. The two versions fail independently if

$$P(F_1 \cap F_2) - P(F_1) \cdot P(F_2) = 0. \tag{10}$$

We have

$$P(F_i) = \int\theta(x)dQ, \ i = 1, 2 \tag{11}$$

and

$$P(F_1 \cap F_2) = E[\int V_1(x) \ V_2(x)dQ] \tag{12}$$

where $V_1(\cdot)$ and $V_2(\cdot)$ are the score functions for the individual versions. Upon taking the expectation inside the integral in (12) and using the assumption that $V_1(\cdot)$ and $V_2(\cdot)$ are the result of a random sample, we have

$$P(F_1 \cap F_2) = \int\theta^2(x)dQ. \tag{13}$$

Now the condition for independence, as stated in (10), is that

$$\int\theta^2(x)dQ - \int\theta(x)dQ \cdot \int\theta(x)dQ = 0. \tag{14}$$

However, the term on the left is the variance,

$$\sigma^2 = \int y^2dG(y) - \int ydG(y) \cdot \int ydG(y), \tag{15}$$

of the intensity distribution and

$$\int ydG = \int\theta(x)dQ \tag{16}$$

is its mean.

The variance of a distribution can equal zero only if the mass of the distribution is concentrated at a single point. Therefore, we state the following:

Theorem 2. Under the conditions stated in the previous theorem, a necessary and sufficient condition for (unconditional) independent failure of the component versions is that $\theta(x)$ be constant except on a subset A of Ω for which $Q(A) = 0$.

Proof. In the general case, independence holds if

$$P(\bigcap_{1}^{n} F_i) = \prod_{1}^{n} P(F_i),$$

or if,

$$\int \theta^n(x)dQ = [\int \theta(x)dQ]^n.$$

By substitution, a constant intensity implies that $F_1, F_2, \ldots, F_n$ are independent events. Conversely, independence of $F_1, F_2, \ldots, F_n$ implies pairwise independence which in turn implies a constant intensity as shown for the case n=2.

A few words of explanation are in order to illustrate the difference between unconditional probabilities which are used in Theorem 2 and conditional probabilities that are appropriate when the discussion is limited to particular versions. This difference was discussed earlier following the statement of Theorem 1 and also when comparing (1) and (2). Suppose that two particular independently designed versions fail on inputs chosen from the sets $F_i = \{x:v_i(x) = 1\}$, i = 1,2. The conditional probability (given the particular versions) that both versions fail on inputs chosen from Ω is

$$Q(F_1 \cap F_2) = \int v_1(x)\ v_2(x)dQ$$

and the individual conditional probabilities are

$$Q(F_i) = \int v_i(x)dQ, \ i = 1,2.$$

If F_1 and F_2 are disjoint sets and if $Q(F_i) > 0$, i = 1,2, then

$$Q(F_1 \cap F_2) < Q(F_1)\, Q(F_2).$$

Thus the two particular versions represent a case of negative (conditional) dependence. Further these two versions may have been chosen from a population having constant intensity. This does not invalidate the statement of Theorem 2 for the same reason that a coin cannot be declared biased on the basis of observing two heads in two tosses. Repetitions of the process of selecting independently designed versions would typically result in conditional probabilities which vary over repeated selections and it is the average of these conditional probabilities to which we refer in Theorem 2.

A constant intensity is probably unreasonable to expect in most applications. For example, if for some population, none of the component versions fail on most inputs while a small percentage fail on a small portion of the inputs, then independence cannot hold.

Now consider whether it is physically plausible that a constant intensity should imply the independent occurrence of errors in component versions. This same question can arise in the context of a coin tossing experiment. Suppose that if two similar coins (software components designed to a common specification) are tossed (execute) under one condition (on input x_1) then the probabilities of each giving tails is .4, but if each is tossed under another condition (input x_2), the probability of each giving tails is .6. Now if the condition (input) is chosen at random and the pair of coins is tossed, the probability of both giving tails is $.5(.4)^2 + .5(.6)^2 = .26$ while the probabilities that they individually give tails is $.5(.6) + .5(.4) = .5$. Independence fails to hold $(.26 \neq (.5)^2)$ since the probability of tails varies with the input conditions. Independence in the software context is, therefore, no less plausible than for other experiments in which the results are given by a two-stage process.

Even though the notion of a constant intensity might seem unacceptable at first, we assert that users of the independence model implicity make this assumption. Given that information concerning the intensity is unavailable, the most logical choice would be the average intensity $\int\theta(x)dQ$, which is also the mean component failure probability. Substituting the average intensity for $\theta(x)$ in (4) gives the independence model.

Our results show it is incorrect to interpret a low intensity as implying statistical independence and a high intensity as implying statistical dependence. Rather the variance σ^2 of the intensity distribution gives a measure of departure from the independence model. However, a more useful approach may be to compare directly computations given by (8) and (9). This difference describes the effect of assuming independence when predicting the failure probability of an N-Version system. We examine this difference in a later section.

4.0 A SUFFICIENT CONDITION FOR REDUNDANCY TO IMPROVE RELIABILITY

Whereas estimates of p_N, N = 1, 3, 5, ... can be given directly on the basis of a random sample of independently designed versions, such estimates would provide little insight concerning the effect of coincident errors. Moreover, in terms of efficiency, rather than examine a series of parameters to decide whether redundancy improves reliability, it is desirable to give a global condition which permits examining the intensity distribution. The difference in failure probabilities for the N-Version and single version cases is

$$p_N - p = \int[h(y;N)-y]dG(y) \qquad (17)$$

where G(y) is the intensity distribution and h(y;N) is given in (5). We desire a condition on G(y) which insures that (17) would be negative. Here and in later discussion of this problem we refer only to the case m = (N+1)/2.

Insight into the type of condition required is gained by examining the integrand $\phi(y;N) = h(y;N) - y$ appearing in (17). As shown in the Appendix, $\phi(y;N)$ is an antisymmetric function (a class of functions studied in [13]), with center of antisymmetry at .5; that is,

$$\phi(.5 + y;N) = - \phi(.5 - y;N),\ 0 \leq y \leq .5. \qquad (18)$$

In addition, $\phi(y;N)$ is convex over the range $0 \leq y \leq .5$, concave over $.5 \leq y \leq 1$, $\phi(0;N) = \phi(.5;N) = \phi(1;N) = 0$, and $\phi(y;N)$ lies below (above) the horizontal axis for $0 < y < .5$ ($.5 < y < 1$). The antisymmetry of $\phi(y;N)$

suggests that a sufficient condition for (17) to be negative is when the intensity distribution assigns greater mass to intervals of the type $(.5 - b, .5 - a]$, $0 \le a < b$, than to their symmetrically located counterparts $[.5 + a, .5 + b)$.

To describe this condition, we require that

$$G(.5-a) - G(.5-b) \ge G_{-}(.5+b) - G_{-}(.5+a) \qquad (19)$$

for all $0 \le a < b$ where $G_{-}(y)$ is given by the left continuous version of $G(y)$; namely, by

$$G_{-}(y) = \int_{\{x:\ \theta(x) < y\}} dQ \qquad (20)$$

Note that if equality holds in (19) for all $0 \le a < b$, then $G(y)$ is a symmetric distribution with center of symmetry at .5. Thus condition (19) describes an asymmetry of the intensity distribution relative to the center point of [0, 1].

The asymmetry condition (19) can also be described by either of the following conditions:

$$G(.5 - y) + G_{-}(.5 + y) \text{ is nonincreasing in } y \ge 0 \qquad (21)$$

or

$$G(y) + G_{-}(1 - y) \text{ is nondecreasing in } y \le .5. \qquad (22)$$

A sufficient condition under which redundant N-Version ($N = 1, 3, 5, \ldots$ and $m = (N+1)/2$) structures "on the average" have smaller probability of failure than do single versions is as stated in the following:

Theorem 3. If the intensity distribution satisfies the asymmetry condition (19), then $\int \phi(y;N)dG \le 0$. Equality holds when $G(y)$ is a symmetric distribution.

Proof.

Since $\phi(.5;N) = 0$,

$$\int \phi(y;N)dG = \int_{-\infty}^{.5} \phi(y;N)\, dG + \int_{.5}^{\infty} \phi(y;N)dG\_$$

and by substitution, the expression on the right becomes

$$-\int_{0}^{\infty} \phi(.5 - y;N)dG(.5-y) + \int_{0}^{\infty} \phi(.5 + y;N)dG\_(.5+y).$$

Now using the antisymmetry of $\phi(y;N)$ gives

$$\int \phi(y;N)dG = \int_{0}^{\infty} \phi(.5 + y;N)d[G(.5 - y) + G\_(.5 + y)].$$

If $G(y)$ is symmetric then $G(.5 - y) + G\_(.5 + y)$ is constant in $y \geqq 0$ so that $\int\phi(y;N)dG = 0$. On the other hand, if condition (19) holds then (21) implies that $G(.5 - y) + G\_(.5 + y)$ assigns a negative measure to each interval and implies the desired result.

Although asymmetry of the intensity distribution is not a necessary condition, it does describe a wide class of cases for which an N-Version structure is better than a single version. In particular note that if 1 -

$G(.5) = 0$, then the sufficient condition is met; that is, if $\theta(x) \leq .5$ for $x\varepsilon\Omega$ except on a set A for which $Q(A) = 0$, then an N-Version structure gives a smaller probability of failure than does a strategy based on a single version.

Whereas for hardware devices the independence model and the average component failure probability, p, can be used to give a condition under which redundancy improves reliability, this is not true, in general, for redundant software subject to coincident errors. In particular, the average component failure probability being less than .5 does not imply that redundancy decreases system failure probability as is demonstrated in the next section.

5.0 EFFECTS OF COINCIDENT ERRORS

In this section we examine the effects of coincident errors on the failure probability, p_N, ($N = 1, 3, 5, \ldots$) of an N-Version software structure. Since coincidence, in the current context, refers to an intensity function $\theta(x)$, $x\varepsilon\Omega$, we are confronted with the problem of having to hypothesize a probability mass function (pmf), $g(\theta)$, of the type suggested earlier in (7). We will assume a highly skewed distribution as in Table 1a to represent a form we believe is reasonable to expect in applications of software redundancy.

The interpretation of $g(\theta)$ is the probability of encountering an $x\varepsilon\Omega$ whose coincidence intensity is the proportion θ. Thus ideally, we have high probabilities of encountering inputs that result in low values of θ and significantly less probability of encountering the higher intensity coefficients at the tail of the distribution. For the given pmf, we would expect all (i.e. $\theta=0$) of the programs of our population to provide correct outputs on 98.98% of the input cases. The average failure probability for a single version (which is the same as the mean of the intensity distribution) is $p = \Sigma\theta g(\theta) = 2 \times 10^{-4}$.

θ	$g(\theta)$
0	.98977
.01	.00512
.02	.00256
.03	.00128
.04	.00064
.05	.00032
.06	.00016
.07	.00008
.08	.00004
.09	.00002
.10	.00001

(a)

θ	$g_1(\theta)$	$g_2(\theta)$	$g_3(\theta)$
0	.99999	.99997	.99993
.05	.00001	.00002	.00004
.10	0	.00001	.00002
.15	0	0	.00001

(b)

θ	$g(\theta)$
0	.99899
.10	.00100
.50	.00001

(c)

θ	$g(\theta)$
0	.99998
.05	.00100
.60	.00001

(d)

θ	$g(\theta)$
0	.99999
.60	.00001

(e)

θ	$g(\theta)$
0	.99998
.10	.00001
.60	.00001

(f)

Table 1. - Probability mass functions for figures 1-6.

Effect of Independence Assumption

The expected system failure probability on the basis of the pmf of Table 1a is shown in Figure 1. Also shown is the result of assuming independent errors. It is evident that increasing N does substantially reduce the probability of incorrect output for an N-Version system. A N=5 version system, for example, will reduce this failure probability by approximately two orders of magnitude relative to that of a single version. However, also evident is the fact that the assumption of independent errors leads to predictions of improvement of more than five orders of magnitude. This underestimation can be seen another way: it would take seventeen versions from a population whose average failure probability is 2 x 10^{-4} to produce a system with $p_N < 10^{-9}$ rather than the five versions when independence is assumed.

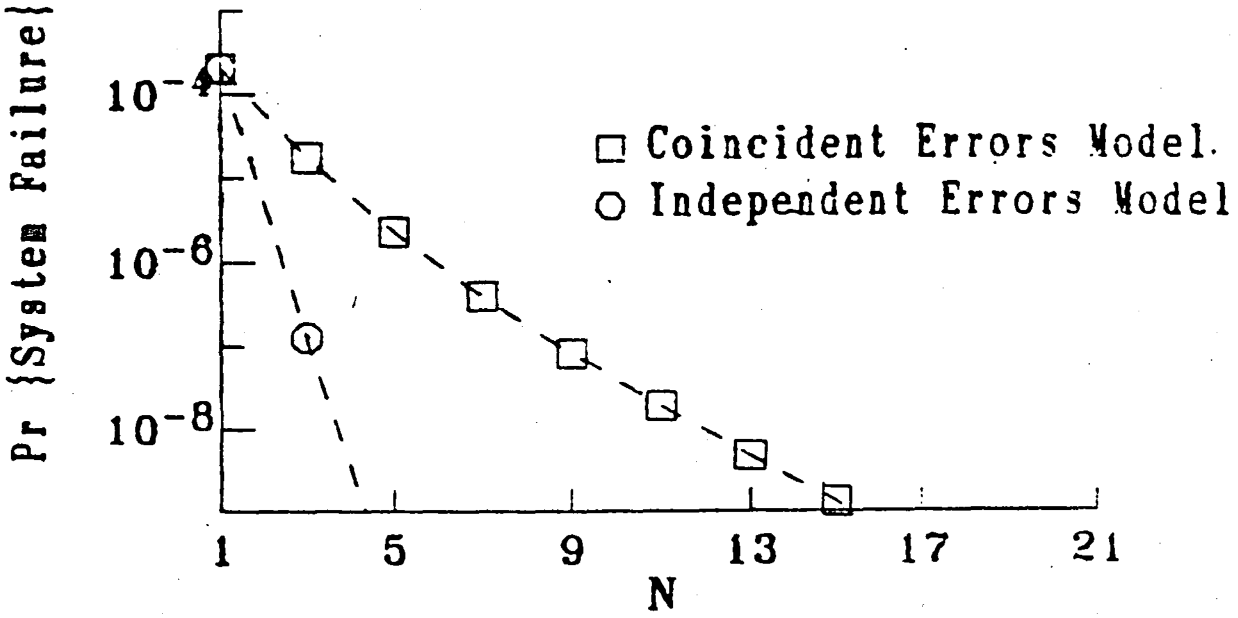


Figure 1. - Effect of independent errors assumption.

Effect of Shifted Intensity Distribution

Figure 2 shows the effect of shifting the mass points of the intensity distribution to the right, thereby, increasing the intensity of coincident errors. The coincident errors increase from a maximum of 5.0 percent for $g_1(\theta)$ to 15.0 percent for $g_3(\theta)$ as shown in Table 1b. This shift has degraded average component failure probability, p, from 5.0×10^{-7} to 5.5×10^{-6}. If these components were used in a critical application requiring $p_N < 10^{-9}$ then twenty-one components would be required from the population with $g_3(\theta)$ compared to nine components corresponding to $g_1(\theta)$.

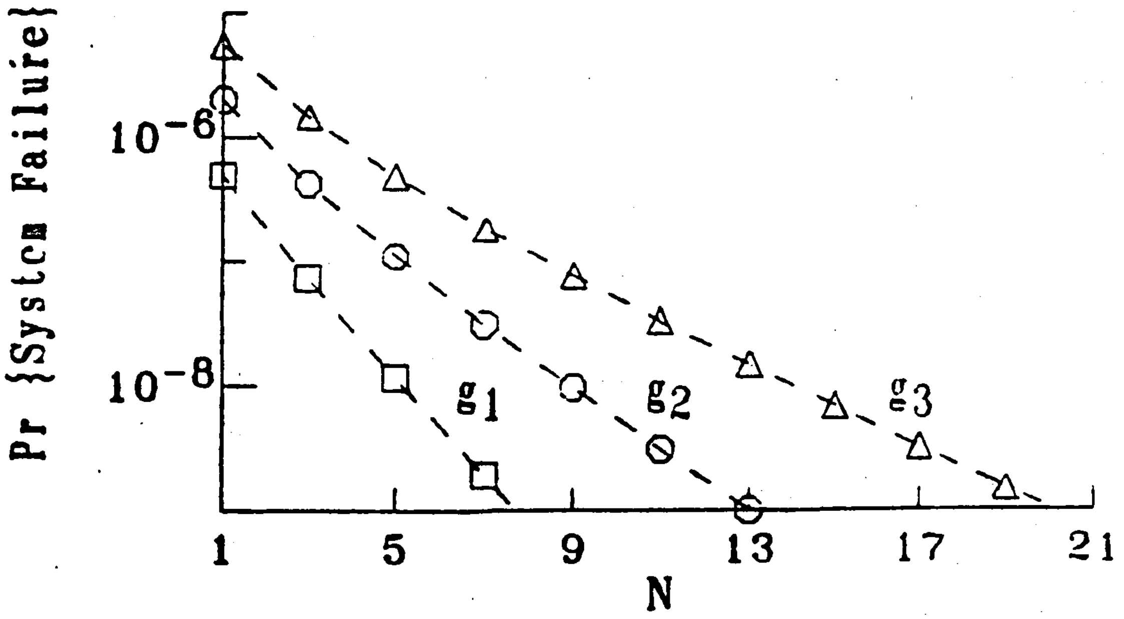


Figure 2. - Effect of a shifted intensity distribution.

The Limiting Probability of System Failure

Here we examine the limiting value of p_N as N increases. Using property (ii) of the Appendix it is easily shown that this limiting value is

$$\ell im\ p_N = .5[G(.5+)-G(.5-)] + \int_{.5+}^{1} dG(\theta). \tag{23}$$

This effect is illustrated in Figure 3 using the pmf of Table 1c.

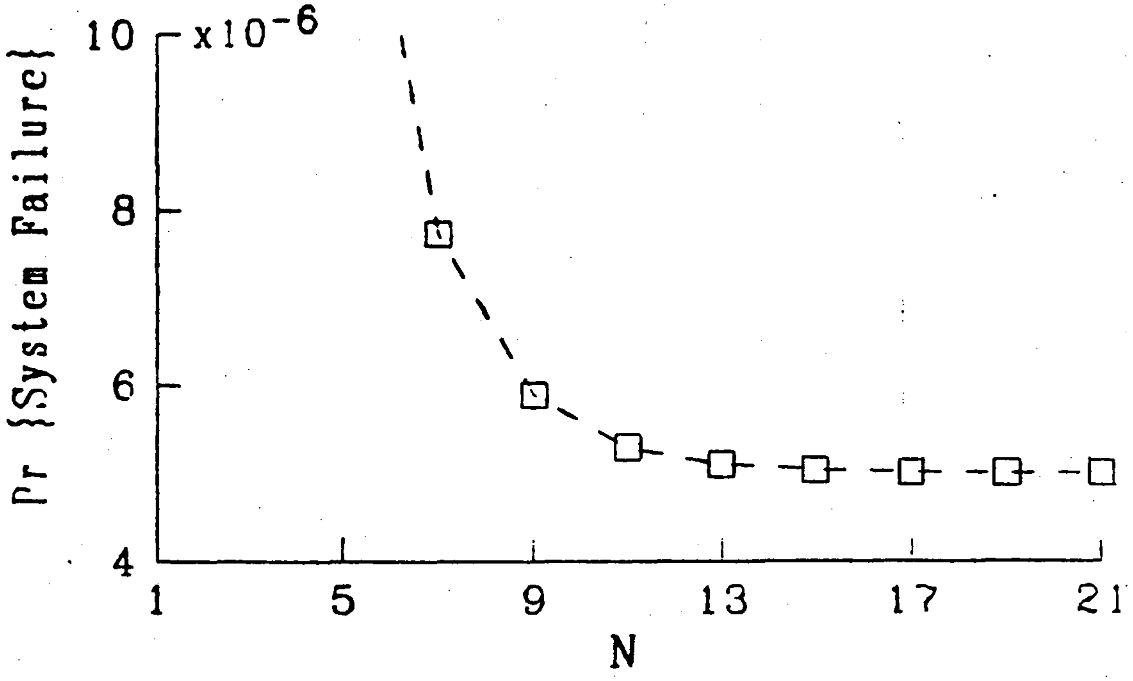


Figure 3. - Limit on Pr {System Failure}.

Although it is true for this example that a fault-tolerant approach is better than a single version of software, the coincidence mass points distributed along the interval $.5 \leq \theta \leq 1$ limits the reliability that can be obtained with fault tolerance. For this example p_N can never fall below 5×10^{-6} with any degree of fault tolerance.

A Condition For System Degradation In The Limit.

Consider the pmf of Table 1d and the corresponding p_N shown in Figure 4. Here we have a case where the value of N corresponding to the minimum failure probability is not the limiting case (N → ∞) but rather an intermediate value, N = 7. Increasing N beyond this point actually degrades the system. What has been the condition that has brought about this degradation with increasing N?

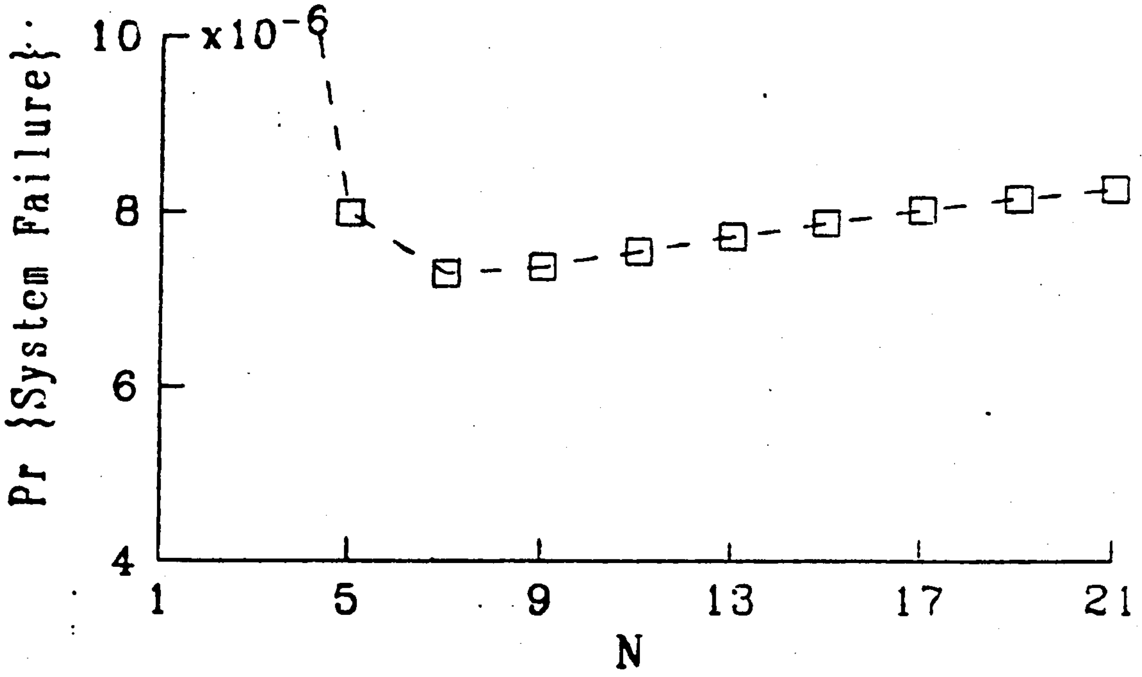


Figure 4. - Existence of optimal N.

This condition will exist when the failure probability for some N-Version system is less than the limiting failure probability, i.e., when for some N,

$$p_N < .5[G(.5+) - G(.5-)] + \int_{.5+}^{\infty} dG(\theta) \tag{24}$$

Using (8) for p_N this can be written as

$$\int_{-\infty}^{.5-} h(\theta;N)dG(\theta) + \int_{.5+}^{\infty} [h(\theta;N) - 1]dG\_(\theta) < 0. \qquad (25)$$

Using the symmetry, $h(\theta; N) = 1 - h(1 - \theta; N)$, we have

$$\int_{-\infty}^{.5-} h(\theta; N)d[G(\theta) + G\_(1 - \theta)] < 0. \qquad (26)$$

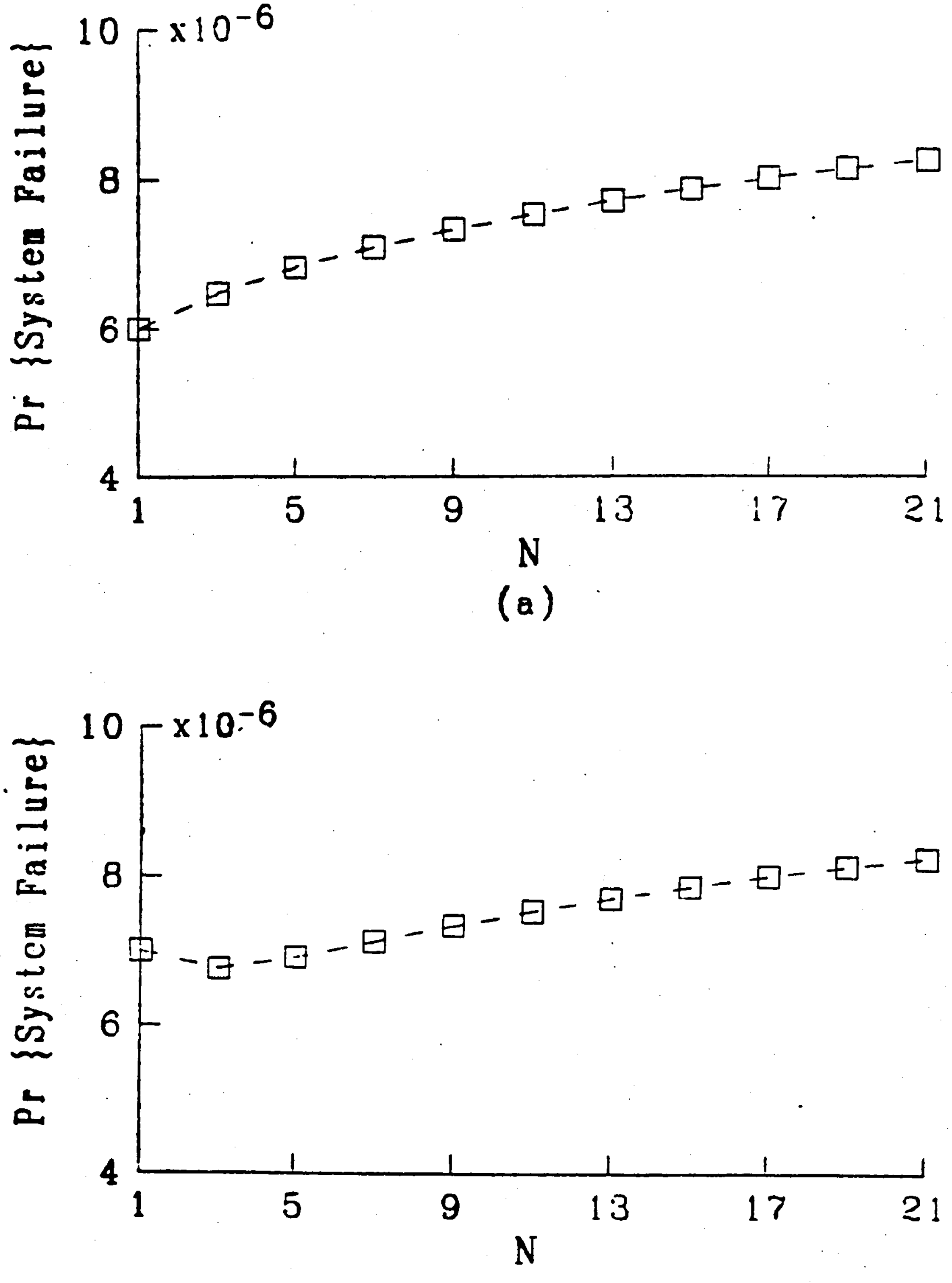
The sufficiency condition of Theorem (2) implies that $G(\theta) + G\_(1 - \theta)$ is increasing for $\theta \leq .5$ which is inconsistent with inequality (26) above. Therefore, a necessary condition for a system to degrade in the limit is a violation of the sufficiency condition of Theorem (2).

This example illustrates the possibility of coincident errors causing an increase in system failure probability with increasing N. However, the end result is still better than a single version system. Also note that the sufficiency condition given in Theorem 2 is not a necessary condition for $p_N < p$.

Effect of Highly Coincident Errors

As we have shown earlier, certain intensity functions can result in an N-Version system being more prone to failure than a single software component. An example of this, although perhaps highly unlikely, is shown in Figure 5a (corresponding to the pmf of Table 1e). Here all programs produce correct output except for a subset A of the input space for which $\theta(x) = \theta = .6$,

$x \in A$. Thus for this subset, 60 percent of the population would produce an error. In this case it is clear why increasing N degrades system reliability. In the case of the independence model, if the average component failure probability, p, exceeds .5, it becomes increasingly more difficult with increasing N to realize a majority of components having correct output. Similarly, for the coincident error model, if $\theta(x) > .5$ for x in some subset A for which $Q(A) > 0$, it also becomes increasingly more difficult with increasing N to realize a majority of components having correct output. Moreover, conditions could exist when one must specify a value of N in order to assess whether N-Version is better than a single version. This is illustrated in Figure 5b (corresponding to the pmf of Table 1f). Increasing N initially decreases system failure probability but eventually heads for its limiting value which is worse than for a single component.



(a)

(b)

Figure 5. - Effect of highly coincident errors.

6.0 CONCLUSIONS

The application of redundancy to hardware components has long been established as an effective methodology for increasing reliability. Its application to software is a relatively new and untested technology largely motivated by the need for high reliability in life-critical applications such as flight control. Thus, at least in the initial stage of studying fault-tolerant software, much interest is likely to lie in evaluating the long term effectiveness of a fault-tolerant strategy rather than in examining only a single instance in which, for example, a particular system has smaller failure probability than its component versions.

In this paper a theoretical basis for the analysis of redundant software has been developed which directly links certain basic quantities with the experimental process of testing independently designed software components. We used this model to study in some detail the case of N-Version redundancy in which the system fails if at least a majority of its components fail. Our main conclusion in this case is that if the intensity distribution is asymmetric in a certain way (see Section 4), then we can ensure that an N-Version strategy is better than one based on using a single software component.

This condition differs sharply from what is required on the basis of the independence model commonly used to estimate the reliability of hardware devices. In the latter case, a necessary and sufficient condition (assuming an N-modular redundant system which fails if a majority of its components fail) for redundancy to improve reliability over that of a single component is that the component failure probability be less than .5 and, further, system reliability would then increase as the number of components is increased. The same thing cannot be said of redundant software systems which are subject to coincident errors (see Section 5).

This only points out one major difference between the type of model needed for redundant software and the independence model used for hardware devices. Our model also gives some insight concerning the validity of assuming that software components fail independently in a statistical sense. A low coincidence of errors does not describe independence. Rather a constant intensity characterizes the case of independence and the variance of the intensity distribution measures departure from the independence model. We believe a constant intensity is a condition unlikely to hold in most applications. Therefore, the combinatorial method, based on independence and requiring only information concerning the failure probability of component versions, is unlikely to give accurate estimates when applied to redundant software systems.

We have illustrated the effects of coincident errors on the failure probability of redundant software systems. It is clear that redundancy under certain conditions can improve reliability. However, the effects of coincident errors, as a minimum, required an increase in the number of software components greater than would be predicted by calculations using the combinatorial method which assumes independence. Further, the effects of a high intensity of coincident errors can be much more serious to the extent of making a fault tolerant approach, on average, worse than using a single version. Here again we must reassert that the assumption we are making is that we equate the process of developing a single version with that of randomly selecting a program from a population of programs which have been independently developed.

For purposes of illustration we have postulated in some cases a rather high intensity of coincident errors. It is clear we need empirical data to truly assess the effects of these errors on highly reliable software systems. Additionally, efforts to identify the sources of coincident errors and to develop methods to reduce their intensity (hopefully that will come with an understanding of the common source of the errors) will not only benefit the development of fault-tolerant software but also software engineering in general.

APPENDIX

Here we summarize some properties of $h(y;N)$. A real valued function $f(y)$, say, is antisymmetric [13] on [0, 1] with center at .5 if

$$f(.5-y) + f(y+.5) = 2f(.5),\ 0 \leq y \leq .5. \tag{A.1}$$

The function $h(y;N)$ given by (5) for $N = 1, 3, 5, \ldots$ and $m = (N+1)/2$ can be written

$$h(y;N) = N!(k!)^{-2} \int_{o}^{y} u^k(1-u)^k du,\ 0 \leq y \leq 1, \tag{A.2}$$

where $k = (N-1)/2$; this is a well-known formula [14] for a sum of binomial terms.

The main properties of interest concerning $h(y;N)$ and $\phi(y;N) = h(y;N)-y$ are:

(i) $h(0;N) = 0$, $h(1;N) = 1$ and $h(.5;N) = .5$ for $N = 1, 2, 3, \ldots$;

(ii) as $N\to\infty$, $\lim h(y;N) = 0, .5, 1$ whenever $y<.5$, $y=.5$, and $y>.5$, respectively;

(iii) $h(y;N)$ is antisymmetrical with center at .5;

(iv) $h(y;N)$ is convex on [0,.5] and is concave on [.5,1];

(v) $\phi(y;n)$ is antisymmetrical with center at .5 and $\phi(0;N)=\phi(.5;N)=\phi(1;N)=0$ for $N = 1, 3, 5, \ldots$;

(vi) $\phi(y;N)$ is convex on [0,.5] and is concave on [.5,1];

(vii) $h(y;N)$ is nonincreasing in $N = 1, 3, 5, \ldots$ for $y<.5$;

(viii) $h(y;N)$ is nondecreasing in $N = 1, 3, 5, \ldots$ for $y>.5$.

Proof. The result (i) follows by substitution and by symmetry of the binomial distribution when y=.5; (ii) follows from the weak law of large numbers applied to the binomial distribution; (iv) and (vi) can be seen directly by examining the second derivatives of $h(y;N)$ and $\phi(y;N)$.

To prove (iii), note that symmetry of the integrand in (A.2) gives

$$N!(k!)^{-2}\int_{o}^{.5+y} u^{k}(1-u)^{k}du = N!(k!)^{-2} \int_{.5-y}^{1} u^{k}(1-u)^{k}du$$

where the term on the left is $h(.5+y;N)$ and the term on the right is $1-h(.5-y;N)$. Therefore, $h(.5+y;N) + h(.5-y;N) = 1 = 2h(.5;N)$. Now (v) also follows by using the antisymmetry of $h(y;N)$ established in (iii).

To prove (vii), let $f(y) = h(y;N+2)/h(y;N)$ and use (A.2) to get

$$f(y) = c \int_{o}^{y} u^{k+1}(1-u)^{k+1}du/ \int_{o}^{y} u^{k}(1-u)^{k}du$$

where $c = (N+2)(N+1)(k+1)^{-2}$, k=0, 1, 2, ... The derivative $\partial/\partial y\{f(y)\}$ is nonnegative when $y < .5$ providing

$$y(1-y) \int_{o}^{y} u^{k}(1-u)^{k}du - \int_{o}^{y} u^{k+1}(1-u)^{k+1}du \geq 0.$$

But $u(1-u)$ when $0 \leq u \leq y < .5$ takes the maximum value $y(1-y)$ so that

$$\int_{o}^{y} u(1-u)u^{k}(1-u)^{k}du \leq y(1-y) \int_{o}^{y} u^{k}(1-u)^{k}du$$

which proves that $f(y)$ is nondecreasing for $0 \leq y \leq .5$. This proves (vii) since $f(.5) = 1$. Since $h(.5+y,N) = 1 - h(.5-y;N)$ and $h(.5-y;N)$ is nonincreasing in $N = 1, 3, 5, \ldots$, we have also proved (viii).

REFERENCES

1. Avizienis, A., "Fault Tolerance and Fault Intolerance: Complementary Approaches to Reliable Computing," Proc. 1975 Int. Conf. Reliable Software, pp. 458-464.

2. Randell, B., "System Structure for Software Fault Tolerance," IEEE Trans. Software Eng., June 1975, pp. 220-232.

3. Weinstock, C. B., "SIFT Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," Proc. of IEEE, Vol. 66, No. 10, 1978, pp. 1240-1255.

4. Hopkins, A. L., et al., "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," Proc. of IEEE, Vol. 66, No. 10, 1978, pp. 1221-1239.

5. Migneault, G. E., "The Cost of Software Fault Tolerance Techniques," NASA Technical Memorandum 84546, Sept. 1982.

6. Barlow, R. E., and Proschan, F., "Statistical Theory of Reliability and Life Testing," Holt, Rinehart, and Winston, Inc. 1975.

7. Scott, R. K., Gault, J. W., McAllister, D. F., and Wiggs, J., "Experimental Validation of Six Fault-Tolerant Software Reliability Models," IEEE Conf. on Fault-Tolerant Computing, 1984, pp. 102-107.

8. Grnavou, A., Arlat, J., and Avizinienis, A., "Modeling of Software Fault Tolerance Strategies," Proc. 1980 Pittsburgh Modeling and Simulation Conf., Pittsburgh, Pennsylvania, May 1980.

9. Littlewood, B., "Theories of Software Reliability: How Good Are They and How Can They Be Improved," IEEE Trans. on Software Eng., Vol. SE-6, No. 5, 1980, pp. 489-500.

10. Nagel, P. M., and Skrivan, J. A., "Software Reliability: Repetitive Run Experimentation and Modeling," NASA CR-165036, 1982.

11. Nagel, P. M., Scholz, F. W., and Skrivan, J. A., "Software Reliability: Additional Investigations into Modeling with Replicated Experiments," NASA CR-172378, 1984.

12. Chung, Kai Lai, "A Course in Probability Theory," New York: Harcourt, Brace and World Inc., 1968.

13. Van Zwet, W. R., "Convex Transformations of Random Variables," Armsterdam: Mathematisch Centrum, 1964.

14. Abramowitz, M., and Stegun, I. A., ed., "Handbook of Mathematical Functions," New York: Dover Publications, Inc., 1965

1. Report No. NASA TM-86369	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle

A Theoretical Basis For The Analysis Of Redundant Software Subject To Coincident Errors

5. Report Date

January 1985

6. Performing Organization Code

505-34-13-35

7. Author(s)

Dave E. Eckhardt, Jr.
Larry D. Lee

8. Performing Organization Report No.

9. Performing Organization Name and Address

NASA Langley Research Center
Hampton, Virginia 23665

10. Work Unit No.

11. Contract or Grant No.

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, DC 20546

13. Type of Report and Period Covered

Technical Memorandum

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Fundamental to the development of redundant software techniques (known as fault-tolerant software) is an understanding of the impact of multiple joint occurrences of errors, referred to here as coincident errors. A theoretical basis for the study of redundant software is developed which (1) provides a probabilistic framework for empirically evaluating the effectiveness of the general (N-Version) strategy when component versions are subject to coincident errors, and (2) permits an analytical study of the effects of these errors. The basic assumptions of the model are: (i) independently designed software components are chosen in a random sample and (ii) in the user environment, the system is required to execute on a stationary input series. An intensity function, called the intensity of coincident errors, has a central role in the model. This function describes the propensity of a population of programmers to introduce design faults in such a way that software components fail together when executing in the user environment. The model is used to give conditions under which an N-Version system is a better strategy for reducing system failure probability than relying on a single version of software. In addition, a condition which limits the effectiveness of a fault-tolerant strategy is studied, and we ask whether system failure probability varies monotonically with increasing N or whether an optimal choice of N exists.

17. Key Words (Suggested by Author(s)) Fault-Tolerant Software Redundant Software Reliability Coincident Errors Intensity Distribution	18. Distribution Statement Unclassified - Unlimited Subject Category - 61 & 65

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 39	22. Price A03

For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document