Performability Modeling of N Version Programming Technique

Katerina Goševa – Popstojanova
Department of Computer Science
Faculty of Electrical Engineering
Skopje, Macedonia 91000

Aksenti Grnarov
Department of Computer Science
Faculty of Electrical Engineering
Skopje, Macedonia 91000

Abstract

This paper presents a detailed, but efficiently solvable model of the N version programming for evaluating the reliability and performability over a mission period. Employing a hierarchical decomposition we reduce the model complexity and provide a modeling framework for evaluating the NVP failure and execution time behavior and the operational environment, as well. The failure and execution rates are treated as random variables and the operational profile is analyzed on microstructure level, looking at probabilities of occurrence, failure and execution rates for each partition of input space. The reliability submodel, that represents per run behavior of NVP, includes both functional failures and timing failures thus resulting in system reliability which accounts for performance requirements. The successive runs are modeled by the performance submodel, that represents the iterative nature of software's execution. Combining the results of both submodels, we assess the performability over a mission period that represents the collective effect of multiple system attributes on the NVP effectiveness.

1 Introduction

In this paper we analyze the software fault tolerance technique based on N version programming, first proposed in [2]. It relies on the application of design diversity: program versions are independently designed to meet the same system requirements [3], [16]. A consistent set of inputs is supplied to all versions and all N versions are executed in parallel. A decision mechanism must gather the available results from the N versions and determine the result to be delivered to the user. If a decision mechanism requires all N versions to produce a result, a slow or fail stop version would delay this process indefinitely. In a real –time environment, such a delay is unacceptable, so a timing constraint is used to ensure that results are delivered in a timely manner.

In order to investigate the effectiveness of software fault tolerance in improving reliability the analysis has either been performed by empirical studies or by modeling techniques. We provide an overview of the current state of the art of the software fault tolerance analysis and, through this evaluation, lay the groundwork for future research directions.

The several experimental studies investigated the key assumption that design diversity will result in software versions that have sufficiently different failure characteristics. Diversity has been introduced in the form of different specifications [3], [4], [10], [16], different programming languages [4], [10], and for different distributions of test values over the input space [4]. All versions were developed independently by different teams, in some studies even by geographical distinct participants [4], [7], [17], [20].

Examining the results obtained by these experiments reveals several characteristics of NVP:

- The assumption of independence of failures between independently developed programs does not hold. For example, the experiments [7], [17] or [20] observed identical or similar faults involving up to 5, 4 or 2 versions, respectively.
- The coincident failures were observed in every experiment conducted thus far. Coincident failures do not necessarily result from related design faults. Independent faults causing coincident failures were also observed.
- The failure behavior is very sensitive to the distribution of test values over the input space.
- The faults that were tolerated were not the same as the faults that were detected by fault elimination techniques and the faults that reduce the effectiveness of the NVP are among the most difficult to detect [28].

In general, using NVP in these experimental studies show more or less gain in reliability. The success of the NVP approach clearly depends on the degree to which we can achieve the diversity of the versions' failure behavior.

1071-9458/95 $4.00 ©1995 IEEE

A number of papers devoted to the dependability modeling of software fault tolerant techniques have appeared in the literature. It is obvious that there are two disjoin modeling approaches. On one side, the major goal for the first approach is the modeling and evaluation of the dependability measures of the particular fault tolerant structure. Methods of specifying the system structure include combinational [14], [27], discrete time Markov chain [1], [18], [31], continuous time Markov process [11], [12], [13], fault trees and Markov reward models [5], extended stochastic Petri net and simulation [9], and generalized stochastic Petri nets [15] model types.

On the other side, the major goal of models that belong to the second class, based on the ground – breaking work of Eckhardt and Lee's [6], is the precise meaning of the independence refered to the failure behavior of the diverse program versions. The intensity of coincident errors $\theta(x)$ has a central role in this analysis. It describes the notion that component versions may fail together and gives the probability that a component, when chosen at random, fails on a particular input x. The key idea is that $\theta(x)$ will generally take different values for different inputs x. The objective of [19] is to study the situation when there are available several software development methodologies. The idea is that θ will vary not only from one x to another, but from one methodology to another. Other contributions in this class include modeling the intensity of coincident errors by a random variable with a Beta density function in [25] and the recovery block modeling considered in [30] which examines several methods for determining the intensity distribution: the use of the beta binomial distribution suggested in [25], and two new methods based on the pairwise correlation of modules.

Safety critical real – time computer applications are characterized by stringent deadlines and high reliability requirements. In these terms, deadlines on the time that certain processes should be completed is performance property, while the fault tolerance is a reliability related property. However, one must place certain constraints on how properties affecting performance interact with those affecting dependability. For this purpose, a unified measure, called performability has been introduced [21]. In spite of the vast number of papers that consider performability of hardware systems [22], model based evaluation of software fault tolerance techniques has been focused on either separate evaluation of performance and dependability, or strict measures of dependability until the recently proposed performability modeling framework of the NVP [29]. This hierarchical model is constructed in two layers: a dependability submodel (discrete time Markov chain suggested in [1]) and a performance submodel (a renewal process where each program iteration is represented by a renewal cycle). The information contributed by the lower layer constitutes the upper layer model thus combining performance and dependability measures, via the concept of performability.

2 The NVP model

With the benefit of this background, we develop an unified approach aimed at modeling both the N version system behavior (including time aspects) and the operational environment. The fault tolerant software system investigated here is for the real – time, mission critical applications. The success of such a system depends not only on its logical correctness, but also on its timing correctness. It must make correct responses to environmental changes within specified time intervals or deadlines. The program periodically gets the inputs from the environment and generates the output that is a function only of the most recently accepted input.

In order to reduce the model complexity we use the hierarchical decomposition. The reliability submodel considers both functional and timing failures and accounts for the correlation between versions for single input due to the influence of the operational environment. On the other hand, the performance submodel considers the execution behavior of the NVP including the correlation between versions' execution times. The performability model combines information contributed by these submodels by associating the reward value obtained by reliability submodel to each iteration cycle of the performance submodel.

2.1 Reliability submodel

The reliability submodel incorporates the basic concepts of software reliability theory, so a precise though informal definitions for a set of terms relating to software reliability are given. *A fault* is the defect in the program that executed under particular conditions causes failure. Thus, *a software failure* is the event which occurs when the software is subjected to an input condition such that, due to the presence of one or more faults in code, the resultant output will be different (in time or value) from the required output according to design specifications [23]. Such a general definition of failures enables us to combine *functional failures*, where the output is incorrect, and *timing failures*, where the output is not produced on time. It is very important for real – time applications because failing to meet deadline may have an adverse effect on system reliability. This issue has been addressed only in simulation based method presented in [9].

In accounting for dependence we use the idea that the influence of the operational environment on ver-

sions failures and execution times induces correlation. Since the effects of faults are typically quite sensitive to just how a system is utilized, it is helpful to distinguish the NVP system from its environment. The concept of the operational environment (already used for software testing [24]) is reviewed next. The operation of a software is broken down into series of *runs* and each run performs mapping between a set of input variables and a set of output variables and consumes a certain amount of execution time. Usually a run is a quantity of work initiated by some input. Runs that are identical repetitions of each other are said to form *a run type*. Because the probabilities of occurrence of input states are the natural way of representing the program usage in its operational environment *the operational profile* is defined as a set of relative frequencies of occurrence of the run types. If the relative frequencies of selection have changed, then the operational profile has changed and that will affect the reliability.

For this study we do not distinguish between detected and undetected failures, that is we do not investigate the error detection capabilities of NVP. The view point taken here is that if there is a requirement for fault tolerance, then there is also a requirement for the system to provide a continuation of service.

The reliability submodel, which represents per run behavior of NVP, is constructed in two steps. First, we develop the continuous time Markov model which considers both the failure and execution behavior of NVP, given a particular input state. Then, the user – oriented model of the operational environment is developed, and the probability distributions are defined in execution time – input space domain, thus accounting for the correlation between versions due to the common input.

2.1.1 Model of NVP behavior. We assume that:

- Corresponding to the different development processes, the version failures and execution times are conditionally independent, given a particular input state.
- For each input state, times to failure of program versions are identically distributed random variables, as well as the execution times. Further, we take both of them to be exponentially distributed. The particular values of the failure rate and execution rate, given an input state, are λ and μ respectively.

These assumptions guarantee that, given a particular input state, a finite state continuous time Markov process can be used for the reliability modeling of the N version system. A state is defined as a vector (i, j, k), $0 \leq i, j, k \leq n, \quad i + j + k \leq n$, where

i is the number of versions that have encountered functional failure during the execution and have not completed the execution

j is the number of versions that have completed the execution producing functionally incorrect result

k is the number of versions that have completed the execution producing correct result.

The Markov model for the particular case of three version system and the transitions and associated rates for the case of N versions are presented in figure 1.

Due to the real – time constraint, we define the deterministic parameter characterized as a fixed bound $\tau > 0$ on the time to complete a run. Note that all probabilities in this section are conditional on particular values of failure rate λ and execution rate μ which correspond to the given input state. To simplify notation, we write $P\{t \leq \tau | \lambda, \mu\}$ as $P\{\tau\}$.

Performing derivations of the Markov process yields:

$$P_{ijk}(\tau) = \frac{n!}{i!\,j!\,k!\,(n-i-j-k)!} P_{fex}^{i}(\tau)\, P_{fend}^{j}(\tau) \cdot P_{end}^{k}(\tau)\, P_{ex}^{n-i-j-k}(\tau) \quad (1)$$

where

$$P_{fex}(\tau) = e^{-\mu\tau}\left(1 - e^{-\lambda\tau}\right) \quad (2)$$

$$P_{fend}(\tau) = \frac{\lambda}{\lambda+\mu} - e^{-\mu\tau} + \frac{\mu}{\lambda+\mu} e^{-(\mu+\lambda)\tau} \quad (3)$$

$$P_{end}(\tau) = \frac{\mu}{\lambda+\mu}\left[1 - e^{-(\lambda+\mu)\tau}\right] \quad (4)$$

$$P_{ex}(\tau) = e^{-(\lambda+\mu)\tau}. \quad (5)$$

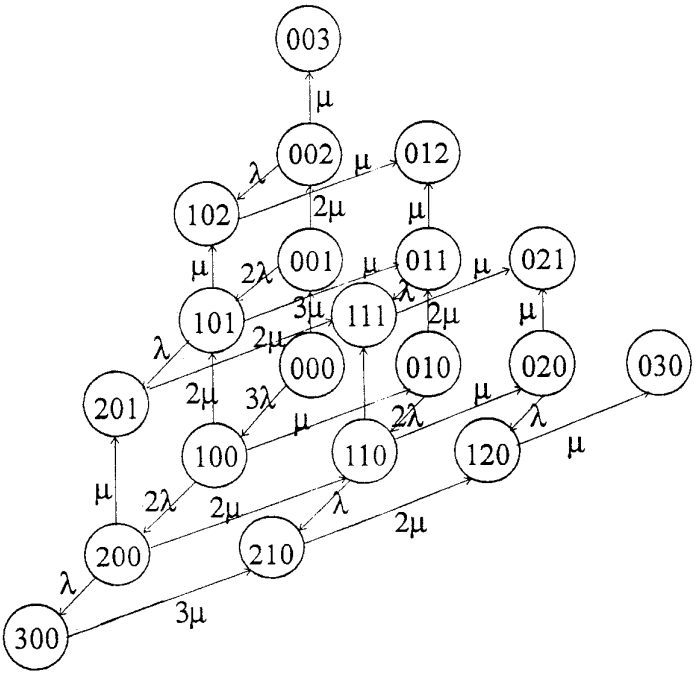
Since the decision algorithm does not distinguish the versions that have not completed the execution (with or without functional failure) we can rewrite the expression (1) as:

$$P_{jk}(\tau) = \frac{n!}{j!\,k!\,(n-d)!} P_{fend}^{j}(\tau)\, P_{end}^{k}(\tau)\, P_{noend}^{n-d}(\tau) \quad (6)$$

where $d = j + k$ is the number of versions that have completed the execution until τ and

$$P_{noend}(\tau) = P_{fex}(\tau) + P_{ex}(\tau) = e^{-\mu\tau}. \quad (7)$$

It is obvious that the above joint pmf is the multinomial pmf and the marginal pmf's are binomial. They are characterized in terms of the probabilities P_{end} (individual version produces correct result before τ),



Transitions	Transition rates	
$(i,j,k) \longrightarrow (i+1,j,k)$	$[n-(i+j+k)]\lambda$,	if $j+k<n$
$(i,j,k) \longrightarrow (i-1,j+1,k)$	$i\mu$,	if $i>0$
$(i,j,k) \longrightarrow (i,j,k+1)$	$[n-(i+j+k)]\mu$,	if $i+j+k<n$.

Figure 1: Markov reliability model

P_{fend} (individual version produces functionally incorrect result before τ) and P_{noend} (individual version has not completed the execution until τ). Using the marginal pmf's of the distribution (6) we obtain the probabilities which characterize NVP failure and execution behavior. Thus, we define *timing failure* of N version system ($n = 2m-1$), on given input state, to be the event that majority of versions do not produce output in time $\leq \tau$

$$P_{tf}(\tau) = \sum_{l=m}^{n} \binom{n}{l} P_{noend}^{l}(\tau)\ [1 - P_{noend}(\tau)]^{n-l}. \quad (8)$$

If the majority of versions have completed the execution before τ it is possible that there is a majority of correct results (*success*)

$$P_{ok}(\tau) = \sum_{k=m}^{n} \binom{n}{k} P_{end}^{k}(\tau)\ [1 - P_{end}(\tau)]^{n-k}, \quad (9)$$

a majority of incorrect results (*functional failure*)

$$P_{ff}(\tau) = \sum_{j=m}^{n} \binom{n}{j} P_{fend}^{j}(\tau)\ [1 - P_{fend}(\tau)]^{n-j}, \quad (10)$$

or there is no majority of either correct or incorrect results

$$P_{nm}(\tau) = 1 - P_{tf}(\tau) - P_{ok}(\tau) - P_{ff}(\tau). \quad (11)$$

The analysis outlines some relations between failure probabilities $1-P_{ok}(\tau)$, $P_{ff}(\tau)$ and $P_{tf}(\tau)$, given a particular input state. Total failure probability $1-P_{ok}(\tau)$, likewise the functional failure probability $P_{ff}(\tau)$, increases for higher failure rate λ, as well as for longer average execution time (smaller execution rate μ). However $P_{ff}(\tau)$ is significantly more sensitive to the variation of the failure rate than the total failure probability. Note that the total failure probability is approximately equal to the timing failure probability $P_{tf}(\tau)$ if $\mu \gg \lambda$. If $\mu \sim \lambda$, or even $\mu < \lambda$ the major contribution to

the total failure probability comes from the functional failure probability.

Influence of the parameters λ, μ, τ. We desire a condition on model parameters λ, μ and τ such that the NVP processing a given input improves the probability of success, and reduces the probabilities of functional and timing failure. Note that the probabilities of success, functional failure and timing failure of NVP are functions of a form

$$h(y,n) = \sum_{i=m}^{n} \binom{n}{i} y^i (1-y)^{n-i}, \qquad 0 \le y \le 1$$

where y is either $P_{end}(\tau)$, $P_{fend}(\tau)$ or $P_{noend}(\tau)$. Rather than examine the series of parameters it is desirable to examine the function $\Phi(y,n) = h(y,n) - y$. A sufficient condition under which NVP improves the probability of success is $P_{end}(\tau) > 0.5$. Considering model parameters it means that NVP has greater probability of producing the correct result on time than do single version if

$$\mu > \lambda \qquad \text{and} \qquad \tau > \frac{1}{\mu+\lambda} \ln \frac{2\mu}{\mu-\lambda}. \qquad (12)$$

NVP reduces the timing failure probability if $P_{noend}(\tau) < 0.5$, that is whenever

$$\mu \neq 0 \qquad \text{and} \qquad \tau > \frac{\ln 2}{\mu}. \qquad (13)$$

NVP reduces the probability of functional failure if $P_{fend}(\tau) < 0.5$, that is

$$\mu \ge \lambda \qquad \text{and} \qquad \tau > 0, \text{ or} \qquad (14)$$

$$\mu < \lambda \qquad \text{and} \qquad \tau < \tau_1 \qquad (15)$$

where τ_1 is the numerical value obtained by solving $P_{fend}(\tau_1) = 0.5$. The condition (15) is highly undesirable because such values of the model parameters certainly degrade the probability of success. It follows that the NVP effectiveness on particular input is improved if the condition (12) is satisfied.

Generalization of the assumptions. Markov model is based on assumptions that the time to failure and the time to end of execution are both exponentially distributed. Next, we introduce a method that can be used to relax these assumptions and thus to account for quite general distributions. We define the following random variables: X time to the occurrence of functional failure and Y time to the end of execution of individual versions. Let denote by $\mathcal{A}$ the event that $Y < X$, by $\mathcal{B}$ the event that $X \le Y$, and by $\mathcal{C}$ the event that $Y \le \tau$. Using the conditional distribution functions and the low of total probability we obtain

$$P_{end}(\tau) = P\{\mathcal{CA}\} = \int_0^\tau P\{Y < X | Y = t\} f_Y(t)\, dt \qquad (16)$$

$$P_{fend}(\tau) = P\{\mathcal{CB}\} = \int_0^\tau P\{X \le Y | Y = t\} f_Y(t) dt \qquad (17)$$

$$P_{noend}(\tau) = 1 - P\{\mathcal{C}\} = 1 - (P\{\mathcal{CA}\} + P\{\mathcal{CB}\}) \qquad (18)$$

Equations (16), (17) and (18) enable the different distributions to be used for the time to the occurrence of functional failure X and the time to complete the execution Y. Provided that both X and Y are exponentially distributed leads to the equivalent expressions to ones derived earlier (4), (3) and (7).

2.1.2 Operational environment model. In order to obtain the probabilities of program failing on randomly chosen input state the particular input state must be unconditioned from the above probabilities. We make the following additional assumptions:

- The environment is homogeneous or time invariant and the operational period is sufficiently long so the input state selection probabilities can be characterized by a steady state.
- The input states occur randomly and independently according to the operational profile.

The last assumption, although usual in most software models [6], [19], [25] and software testing experiments, is a simplification of real life, as it does not explicitly model the phenomena of failure clustering which typically occurs when successive input states are related to one another.

Since the failure and execution behavior are quite sensitive to just how a system is utilized we need to account the change of the failure rate λ and execution rate μ. Therefore, we model the failure and execution rates as random variables, Λ and M respectively. Given a pair (Λ, M) of random variables their joint distribution function is $H(\lambda,\mu) = H_{\Lambda M}(\lambda,\mu) = P\{\Lambda \le \lambda, M \le \mu\}$ and the unconditional probabilities of success, timing failure and functional failure for randomly chosen input state are:

$$P_{ok}(\tau) = \int_0^\infty \int_0^\infty P_{ok}(\tau | \Lambda = \lambda, M = \mu)\, dH(\lambda,\mu) \qquad (19)$$

$$P_{tf}(\tau) = \int_0^\infty \int_0^\infty P_{tf}(\tau | \Lambda = \lambda, M = \mu)\, dH(\lambda,\mu) \qquad (20)$$

$$P_{ff}(\tau) = \int_0^\infty \int_0^\infty P_{ff}(\tau | \Lambda = \lambda, M = \mu)\, dH(\lambda,\mu) \qquad (21)$$

where $P_{ok}(\tau|\Lambda = \lambda, M = \mu)$, $P_{tf}(\tau|\Lambda = \lambda, M = \mu)$, $P_{ff}(\tau|\Lambda = \lambda, M = \mu)$ are given in (9), (8), (10). Note that, we use the Lebesgue – Stieltjes integral, thus covering both the discrete and continuous distribution functions $H(\lambda,\mu)$.

It is possible to make assumptions about the independence of random variables Λ and M and to use some theoretical distribution functions for $H(\lambda, \mu)$ in order to obtain numerical results. Instead, we choose *the user – oriented model of operational environment.* Therefore, we partition the input space by grouping run types that exhibit (as nearly as possible) homogeneous failure and execution behavior. The operational profile is analyzed on a microstructure level, looking at probabilities of occurrence, failure rate and execution time rate for each partition of the input space. Suppose $\Lambda = \lambda_i$ and $M = \mu_j$ for each input state $x \in A_k$, where $A_1, A_2, \ldots, A_r$ is a partition of input space Ω. The operational profile Q gives the probabilities $P\{\Lambda = \lambda_i, M = \mu_j\} = p_{ij} = p_k = Q(A_k)$ that successive input states are chosen at random in subset A_k of the input space Ω. Since the joint pmf p_{ij} is determined, there is no need to make assumptions about the independence of random variables Λ and M. In general, it is possible Λ and M to be negatively, as well as positively correlated random variables, thus reflecting the influence of the failure behavior on performance behavior. In our model Λ and M take a finite number of values over subsets of Ω, and the relations (19), (20) and (21) signify the following:

$$P_{ok}(\tau) = \sum_i \sum_j P_{ok}(\tau|\Lambda = \lambda_i, M = \mu_j)\, p_{ij} \tag{22}$$

$$P_{if}(\tau) = \sum_i \sum_j P_{if}(\tau|\Lambda = \lambda_i, M = \mu_j)\, p_{ij} \tag{23}$$

$$P_{ff}(\tau) = \sum_i \sum_j P_{ff}(\tau|\Lambda = \lambda_i, M = \mu_j)\, p_{ij} \tag{24}$$

It should be emphasized that these results concern averages over input space. This curious result acts as a warning that actual behavior can differ from the average behavior [19]. Typically possible, but rarely occurring input conditions would not significantly contribute to over – all failure probability even through it may be crucial that software works correctly under this conditions. This reality can be handled by introducing categories of criticality and generating operational profiles for each category, as suggested in [24].

Condition of independence. Consider for the moment only two versions processing a randomly selected input. Let denote the probability that the single version fails on a given input $x \in A_k$ as $\rho_k = 1 - P_{end}(\tau|\Lambda = \lambda_i, M = \mu_j)$ and define the indicator random variable I_k^s taking value 0 if the sth version produces correct output on time for given input, and value 1 otherwise. Its expectation $E[I_k^s] = \rho_k$ is the probability that version fails on given input $x \in A_k$. The probability that it fails on the randomly selected input is $E[\sum_{k=1}^r I_k^s\, p_k] = \sum_{k=1}^r \rho_k\, p_k$. The probability that both versions fail on randomly selected input is $E[\sum_{k=1}^r I_k^1 I_k^2\, p_k] = \sum_{k=1}^r \rho_k^2\, p_k$. The condition that versions fail independently on randomly selected input can be formulated as

$$\sum_{k=1}^r \rho_k^2 p_k - \left(\sum_{k=1}^r \rho_k p_k\right)^2 = \sum_S p_{i(1)}\, p_{i(2)} \left[\rho_{i(1)} - \rho_{i(2)}\right]^2 = 0 \tag{25}$$

where the sum is over the set S of all distinct subsets $\{i(1), i(2)\}$ chosen without replacement from $\{1, 2, \ldots, r\}$. It follows that a necessary and sufficient condition for uncorrelated failures and execution times of the component versions is $\rho_{i(1)} = \rho_{i(2)}$, that is Λ and M to be identical for all subsets A_k for which $p_k = Q(A_k) \neq 0$. This result shows that version failures and execution times are correlated whenever Λ and M vary for different input states. Since the expression on the left in relation (25) is the covariance and $Cov \geq 0$ there is a tendency all versions to perform relatively well or relatively poorly on a randomly chosen input. Note that this approach is pessimistic as it enables us to incorporate only positive correlation.

Informally, only the total failure probability along the interval $0.5 \leq \rho_k \leq 1$ limits the benefit that can be obtained with fault tolerance. We have already shown that the total failure probability ρ_k on given input is bounded by 0.5 if the condition (12) is satisfied. However, it is possible, although perhaps highly unlikely that this condition is violated for some subset of the input space. Even if this is the case, N version system still have smaller probability of failure than do single version when the operation profile assigns greater mass to intervals of the type $(0.5 - b, 0.5 - a]$, $0 \leq a < b$ than to their symmetrically located counterparts $[0.5 + a, 0.5 + b)$, as shown in [6].

2.2 Performance submodel

The performance submodel considers only the execution behavior of the NVP and the events are distinguished only by their occurrences in time, independent of outcome result. Due to the iterative nature of software's execution the performance submodel suffices to consider a renewal process $\{N(t), t \geq 0\}$ where each run is represented by a renewal cycle. Let the time between successive renewals be such that T_i is elapsed time from $(i-1)$st run until the occurrence of ith run. In order to derive the distribution of T_i we first lump all states with the same value of $j+k$ of the Markov reliability model in equivalent state $l = j+k, (0 \leq l \leq n)$ and obtain death process with a linear rate $(n-l)\mu$ for all states l. If $Y_1, Y_2 \ldots Y_n$ are interpreted as the length of the execution times of n versions, given a particular input state, then the event that at least m among n

variables Y_j are $\leq t$ has the conditional distribution function

$$F(t|M=\mu) = \sum_{l=m}^{n} \binom{n}{l} e^{-(n-l)\mu t} \left(1 - e^{-\mu t}\right)^l \quad (26)$$

Next, we obtain the unconditional probability distribution function of the time between successive renewals

$$F(t) = \int_0^\infty F(t|M=\mu)\, dH_M(\mu) \quad (27)$$

where $H_M(\mu) = P\{M \leq \mu\}$ is the marginal probability distribution of $H_{\Lambda M}(\lambda, \mu)$. For our model of operational environment (27) becomes

$$F(t) = \sum_j F(t|M=\mu_j)\, p_j \quad (28)$$

where $p_j = P\{M = \mu_j\} = \sum_i p_{ij}$.

Note that the versions' execution times are correlated whenever the execution rates are not identical for all subsets of input states A_k for which $p_k = Q(A_k) \neq 0$. In other words, we consider the correlation between execution times which is much more realistic than the assumption of independence made in [29].

Due to the real – time constraint, T_i is the time upon the end of NVP execution or upon reaching τ, whichever occurs first. It follows that the probability distribution of T_i is

$$F_\tau(t) = \begin{cases} F(t), & \text{for } t < \tau \\ 1, & \text{for } t \geq \tau \end{cases} \quad (29)$$

with mean recurrence time m_τ and variance σ_τ^2.

The performance submodel is also responsible for supplying the expected number of renewals for the time duration $(0, t]$, called the renewal function:

$$E[N(t)] = M(t) = \sum_{k=1}^{\infty} F_\tau^{k*}(t) \quad (30)$$

where F_τ^{k*} denotes the n-fold convolution of F_τ.

Since the consequences of failing to meet deadline could be catastrophic it is important to compute the distribution of the time to timing failure. Therefore, we consider a renewal process with distribution function $F(t)$ and look for the first occurrence of a time interval of duration τ free of renewal epochs. Accordingly, the time to timing failure X is the time when interval duration from the preceding renewal event first exceeds the real – time constraint $\tau > 0$. The distribution $V(t) = P\{X \leq t\}$ can be obtained by deriving the renewal equation

$$V(t) = \begin{cases} 1 - F(\tau) + \int_0^\tau V(t-y)\, dF(y), & t \geq \tau \\ 0, & t < \tau \end{cases} \quad (31)$$

which reduces to the standard renewal equation with the defective distribution L defined by

$$L(t) = \begin{cases} F(t), & \text{for } t < \tau \\ F(\tau), & \text{for } t \geq \tau \end{cases} \quad (32)$$

It follows that the distribution of the time to timing failure is

$$V(t) = P\{X \leq t\} = [1 - L(\tau)]\,[1 + M_L(t)] \quad (33)$$

where $M_L(t) = \sum_{n=1}^{\infty} L^{n*}(t)$ equals the expected number of renewal epochs within $(0, t]$. According to [8] if there exists a number k such that $\int_0^\infty e^{ky}\, dL(y) = 1$ then the asymptotic estimate for the distribution $V(t)$ can be expressed as

$$1 - V(t) \sim \frac{1 - F(\tau)}{k m^\#}\, e^{-k(t-\tau)} \quad (34)$$

where $m^\# = \int_0^\infty e^{ky}\, y\, dL(y) \neq 0$.

2.3 Performability model

The performability model combines the information contributed by the reliability and performance submodels. This is done by associating a reward rate obtained by the reliability submodel with each renewal cycle of the performance submodel. The performance variable is then taken to be the reward accumulated over some utilization period and could be interpreted as the number of successful runs (correct and timely outputs) during the bounded time interval $(0, t]$

$$W(t) = \sum_{k=1}^{N(t)+1} Z_k, \quad (35)$$

where Z_k is a reward value associated with the kth renewal interval T_k. The indicator random variable Z_k takes value 1 if the NVP system provides correct and timely output in the kth renewal interval T_k, and value 0 otherwise. Its expected value $E[Z_k] = P_{ok}(\tau) = p$ is defined by (22).

The renewal equation for *the mean number of successful runs* $A(t) = E[W(t)]$ *over the misson's time* t can be derived

$$A(t) = E[Z_1] + \int_0^t A(t-x)\, dF_\tau(x) \quad (36)$$

resulting in

$$E[W(t)] = E[Z_1]\,[1 + M(t)] = P_{ok}(\tau)\,[1 + M(t)] \quad (37)$$

It represents the collective effect of multiple system attributes computed from the reliability submodel $P_{ok}(\tau)$ (22) and performance submodel $M(t)$ (30).

Since the mission duration t is much greater then the renewal interval times T_k the asymptotic expansion of the renewal function for large t [8] leads to

$$\lim_{t\to\infty} A(t) = P_{ok}(\tau) \left[1 + \frac{t}{m_\tau} + \frac{\sigma_\tau^2 - m_\tau^2}{2m_\tau^2}\right] \quad (38)$$

$$\lambda = [10^{-8}, 10^{-4}, 10^{-2}, 0.1, 0.5] \qquad \mu = [0.5, 0.2, 0.1] \qquad \tau = 30 \qquad t = 36 \cdot 10^6$$

$$P1 = \begin{bmatrix} 0.999 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P2 = \begin{bmatrix} 0.98 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P3 = \begin{bmatrix} 0.98 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.02 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P4 = \begin{bmatrix} 0.88 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.02 \end{bmatrix}$$

Table 1: Parameter values

3 Numerical results

The objective of this section is to demonstrate the impact of the variation of program characteristics (failure and execution rate) and the operational environment (input state selection probabilities) on the reliability and performability related measures of the NVP. Suppose that each partition of the input space is defined by failure rate $\Lambda = \lambda_i$ and execution rate $M = \mu_j$ and the operational profile gives the probabilities $p_{ij} = P\{\Lambda = \lambda_i, M = \mu_j\}$ that the successive input states are chosen at random in that partition. In applications of software redundancy it is reasonable to expect that $P = [p_{ij}]$ assigns high probabilities of encountering inputs that result in small duration of versions execution period compared to the time to failure ($\mu \gg \lambda$). The values assigned to model parameters are shown in Table 1.

Figure 2 and 3 plot the mean number of successful runs and the failure probabilities for varying number of versions. The operational profile $P1$ encounters inputs that result in the small version's execution period compared to the time to failure ($\mu \gg \lambda$). It is evident that the major contribution to the total failure probability comes from the timing failure probability. We also notice that increasing N does substantially reduce the failure probability. For example, $N = 3$ version system will reduce the total failure probability by approximately two orders of magnitude relative to that of a single version. Also evident is that a slight modification to the operational profile for the same program characteristics ($P2$) leads to the significant increase of failure probabilities and smaller mean number of successful runs. For example, $1 - P_{ok}(\tau) < 10^{-7}$ is achieved by 3 version system when $P1$ is assumed, rather then the five versions when $P2$ is considered. The operational profile $P3$ assigns the same probability 0.02 as $P2$ to the worse program characteristics (higher failure rate and smaller execution rate). As a result, 21 versions would be required to achieve the same level of the total failure probability 10^{-7}. Moreover, the benefit of increasing the number of versions is less significant compared to $P1$ and $P2$. The operational profile $P4$ selects (with higher probability) inputs associated with even worse program characteristics ($\mu \sim \lambda$) which results in highly unreliable system. The reliability can not be improved by any degree of fault tolerance since the benefit gained from increasing N is almost negligible. We also notice that the total failure probability is approximately equal to the functional failure probability. The mean number of successful runs is also degraded under this operational profile. However, it is less sensitive to the variations in the operational profile because the values of the execution rate and real – time constraint as defined above do not result in significant performance reduction.

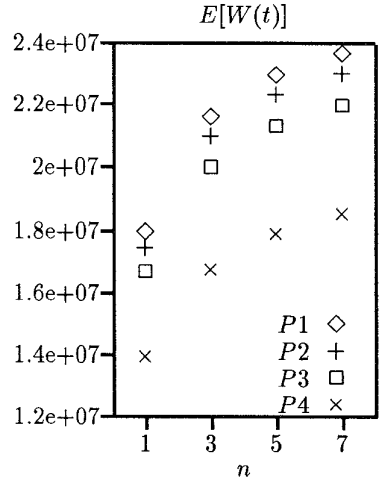


Figure 2: Mean number of successful runs

Certain operational profiles, although highly unlikely, can result in N version system being more prone to failures than a single software component (corresponding diagrams are omitted due to space limitations). However, it is clear that redundancy alone does not guarantee fault tolerance, and that the degree of improvement depends both on program characteristics and the operational profile.

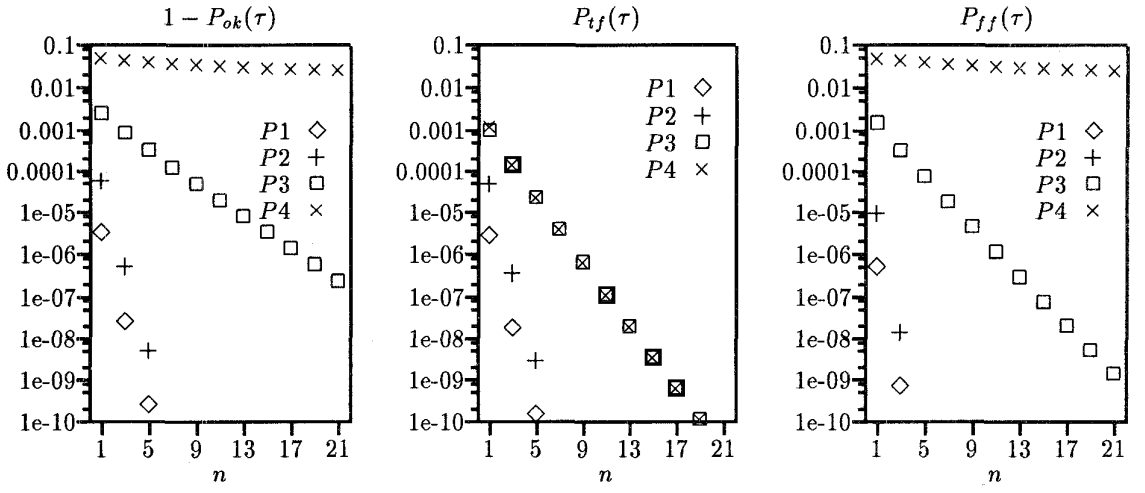


Figure 3: Failure probabilities

4 Conclusions and future work

This paper presents the modeling based study of both N version system and its operational environment. The base model supporting the solution is constructed in two submodels: a reliability submodel and a performance submodel. The reliability submodel, that represents per run behavior of NVP, is constructed in two steps. First, we develop the continuous time Markov model which considers both the failure and execution time behavior of NVP, given a particular input state. The synchronization structure is represented in terms of the execution time distributions because failing to meet deadline has an adverse effect on the reliability. Then, we develop the user – oriented model of operational environment, thus accounting for the correlation between versions due to the common input. Therefore, the input space is partitioned by grouping run types that exhibit (as nearly as possible) homogeneous failure and execution time behavior and the operational profile is analyzed on a microstructure level, looking at probabilities of occurrence, failure and execution rates for each group of run types. The parameters (failure and execution rate) are treated as random variables, and the new probability distribution is defined in execution time – input space domain. We have also derived the conditions under which N version system improves the probability of producing correct result on time, and reduces the probabilities of functional and timing failure.

The performance submodel considers only the execution behavior of the NVP and represents the iterative nature of software executions. The successive runs are modeled by renewal process where each run is represented by a renewal cycle. As with the reliability submodel, we consider the correlation between versions' execution times for a single input. The performance submodel permits us to determine the expected value and the variance of the duration of each run, the expected number of runs for the mission duration time and the distribution of the time to timing failure.

The performability model combines the information contributed by the reliability and performance submodels thus representing the collective effect of multiple system attributes on the NVP effectiveness. Informally, this is done by associating a reward rate (obtained by the reliability submodel) with each renewal cycle of the performance submodel. The performance variable is taken to be the reward accumulated over mission period and we choose to settle for the mean number of successful runs (correct and timely outputs) during the bounded interval $(0, t]$.

Anticipating the future work, it is useful to consider several possible extensions or modifications of our model:

- The alternative method presented in the paper could be used to relax the assumptions that the time to failure, as well as the execution time are exponentially distributed random variables and thus to account for quite general distributions.
- The correlation of subsequent input states might be considered as suggested in [26] or [30].
- Rarely occurring input conditions that would not significantly contribute to over – all failure probability (even through it may be crucial that software works correctly under this conditions) could be handled by introducing categories of criticality into analysis.

References

[1] J.Arlat, K.Kanoun, J.Laprie, "Dependability Modeling and Evaluation of Software Fault Tolerant Systems", *IEEE Trans. on Computers*, Vol.39, No.4, Apr 1990, pp. 504 – 513.

[2] A.Avižienis, L.Chen, "On the Implementation of N Version Programming for Software Fault Tolerance during Program Execution", *Proc. COMPSAC 77*, 1977, pp. 149 – 155.

[3] A.Avižienis, J.Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments", *IEEE Computers*, Aug 1984, pp. 67 – 80.

[4] P.Bishop et al. "Project on Diverse Software - An Experiment in Software Reliability", *Proc. 4th IFAC Workshop SAFECOMP*, 1985, pp. 153 – 158.

[5] J.B.Dugan, M.R.Lyu, "System Reliability Analysis of N Version Programming Application", *Proc. 4th IEEE Int'l Symp. Software Reliability Engineering*, Nov 1993, pp. 103 – 111.

[6] D.E.Eckhardt, L.D.Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors", *IEEE Trans. on Software Engineering*, Vol.SE-11, No.12. Dec 1985, pp. 1511 – 1517.

[7] D.E.Eckhardt et al, "An Experimental Evaluation of Software Redundancy as a Strategy For Improving Reliability", *IEEE Trans. on Software Engineering*, Vol.17, No.7, Jul 1991, pp. 692 – 702.

[8] W. Feller *"An Introduction to Probability and Its Applications"* Volume II, John Wiley & Sons, 1971.

[9] R.Geist, A.J.Offult, F.C. Harris Jr, "Estimation and Enhancement of Real Time Software Reliability through Mutation Analysis" *IEEE Trans. on Computers*, Vol. 41, No. 5, May 1992, pp. 550 – 558.

[10] A.L.Goel, S.N.Sahoo, "Formal Specification and Reliability: An Experimental Study", *Proc. 2nd IEEE Int'l Symp. Software Reliability Engineering*, 1991, pp. 139 – 142.

[11] K.Goševa – Popstojanova, A.Grnarov, "A New Markov Model of N Version Programming", *Proc. 2nd IEEE Int'l Symp. Software Reliability Engineering*, May 1991, pp. 210 – 215.

[12] K.Goševa – Popstojanova, A.Grnarov, "N Version Programming with Majority Voting Decision: Dependability Modeling and Evaluation" *Proc. Euromicro 93*, Sep 1993, pp. 811 – 818.

[13] K.Goševa – Popstojanova, A.Grnarov, "Dependability Modeling and Evaluation of Recovery Block Systems", *Proc. 4th IEEE Int'l Symp. Software Reliability Engineering*, Nov 1993, pp. 112 – 120.

[14] A.Grnarov, J.Arlat, A.Avižienis, "On the performance of Software Fault Tolerance Strategies", *Proc. 10th IEEE Int'l Symp. Fault Tolerant Computing*, Oct 1980, pp. 251 – 253.

[15] K.Kanoun et al. "Reliability Growth of Fault Tolerant Software" *IEEE Trans. on Reliability*, Vol. 42, No. 2, Jun 1993, pp. 205 –219.

[16] J. Kelly, T. McVittie, W. Yamamoto, "Implementing Design Diversity to Achieve Fault Tolerance", *IEEE Software*, Jul 1991, pp. 61 – 71.

[17] J.C.Knight, N.G.Leveson, "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming", *IEEE Trans. on Software Engineering*, Vol.SE-12, No.1, Jan 1986, pp. 96 – 109.

[18] J.Laprie, "Dependability Evaluation of Software Systems in Operation", *IEEE Trans. on Software Engineering*, Vol.SE-10, No. 6, Nov 1984, pp. 701 –714.

[19] B.Littlewood, D.R.Miller, "A Conceptual Model of Multiversion Software", *Proc. 17th IEEE Int'l Symp. Fault Tolerant Computing*, Jul 1987, pp. 150 – 155.

[20] M.R.Lyu, Yu – Tao He, "Improving the N Version Programming Process through the Evaluation of a Design Paradigm", *IEEE Trans. on Reliability*, Vol. 42, No. 2, Jun 1993, pp. 179 –189.

[21] J.F.Meyer, "On Evaluation the Performability of Degradable Computing Systems", *IEEE Trans. on Computers*, Vol. 29, No. 8, Aug 1980, pp. 720 – 731.

[22] J.F.Meyer, "Performability: A Retrospective and Some Pointers to the Future", *Performance Evaluation*, Vol.14, 1992, pp. 139 – 156.

[23] J.D.Musa, A.Iannino, K.Okumoto, *Software Reliability: Measurement, Prediction, Application*, Mc Grow-Hill, 1987.

[24] J.D.Musa, "Operational Profiles in Software Reliability Engineering", *IEEE Software*, Mar 1993, pp. 14 – 32.

[25] V.F.Nicola, A.Goyal, "Modeling of Correlated Failures and Community Error Recovery in Multiversion Software", *IEEE Trans. on Software Engineering*, Vol.16, No.3, Mar 1990, pp. 350 – 359.

[26] C.Ramamoorthy, F.B. Bastani, "Software Reliability - Status and Perspectives", *IEEE Trans. on Software Engineering*, Vol.SE-8, No.4, Jul 1982, pp. 354 – 371.

[27] R.Scott, J.Gault, D.McAllister, "Fault Tolerant Software Reliability Modeling", *IEEE Trans. on Software Engineering*, Vol.13, No.5, May 1987, pp. 582 – 592.

[28] T.J.Shimeall, N.G. Leveson, "An Empirical Comparison of Software Fault Tolerance and Fault Elimination", *IEEE Trans. on Software Engineering*, Vol.17, No.2, Feb 1991, pp. 173 – 182.

[29] A.T.Tai, J.F.Meyer, A.Avižienis, "Performability Enhancement of Fault Tolerant Software", *IEEE Trans. on Reliability*, Vol. 42, No. 2, Jun 1993, pp. 227 –237.

[30] L.A.Tomek, J.K.Muppala, K.S.Trivedi, "Modeling Correlation in Software Recovery Blocks" *IEEE Trans. on Software Engineering*, Vol. 19, No. 11, Nov 1993, pp. 1071 – 1086.

[31] K.Tso, A.Avižienis, J.Kelly, "Error Recovery in Multiversion Software", *Proc. 5th IFAC Workshop SAFECOMP*, 1986, pp. 35 – 41.