N Version Programming: An Unified Modeling Approach

Katerina Goševa – Popstojanova, Aksenti Grnarov
Faculty of Electrical Engineering, Department of Computer Science
P.O.Box 574, 91000 Skopje, Macedonia
E-mail: kate@cerera.etf.ukim.edu.mk

Abstract

This paper presents an unified approach aimed at modeling the joint behavior of the N version system and its operational environment. Our objective is to develop reliability model that considers both functional and performance requirements which is particularly important for real – time applications. The model is constructed in two steps. First, the Markov model of N version failure and execution behavior is developed. Next, we develop the user – oriented model of the operational environment. In accounting for dependence we use the idea that the influence of the operational environment on versions failures and execution times induces correlation. The model addresses a number of basic issues and yet yields closed – form solutions that provide considerable insight into how reliability is affected by both versions characteristics and the operational environment.

1. Introduction

Software fault tolerance relies on the application of design diversity: program versions are independently designed to meet the same system requirements [3],[14]. In this paper we analyze the software fault tolerance technique based on N version programming (NVP), first proposed in [2]. 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, form a decision vector from the results and determine the result to be delivered to the user. In some instances, the versions may deliver their results to the decision mechanism at markedly different times due to external events, internal events or differences in the normal execution time of the versions themselves. If a decision mechanism required 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. The timing constraint defines a time interval within which the result corresponding to a decision must arrive.

The main reason for the NVP modeling and evaluation is to investigate how effective this particular approach is in improving reliability. The analysis of software fault tolerance has been performed either by empirical studies of multiple versions of software modules or by modeling techniques. We present 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 investigate the key assumption that design diversity will result in software versions that have sufficiently different failure characteristics such that fault tolerant system can provide continued service in the presence of failures of the component versions. Diversity has been introduced in the form of different specifications [3], [4], [14], different programming languages [4], and for different input space distributions [4]. All versions were developed independently by different teams, in some studies even by geographical distinct participants [4], [6], [15], [17].

Examining the results obtained by the previous experiments reveals several characteristics of NVP. First, they show that the assumption of independence of failures between independently developed programs does not hold. Next, the coincident failures (failures of the two or more versions on the same input), observed in every experiment conducted thus far, reduce the effectiveness of NVP in dealing with faults. Related design faults are only part of the problem that must be solved because coincident failures do not necessarily result from related design faults. Independent faults causing coincident failures were also observed. Also, the failure behavior is very sensitive to the distribution of test values over input domain. Input domain related faults may prove to be much more difficult to prevent since there is not logical relationship between these faults. Moreover, it was indicated [21] that the faults that were tolerated were not the same as the faults that were detected by fault elimination techniques and that the faults that reduce the effectiveness of the NVP are among the most difficult to detect.

So far, modeling work of the software fault tolerance

techniques has been concentrated mainly on the dependability. 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 [12], [20], discrete time Markov chain [1], continuous time Markov process [9], [10], [11], fault trees and Markov reward models [5], extended stochastic Petri net and simulation [8], and generalized stochastic Petri nets [13] model types. On the other side, the second approach based on the ground – breaking work of Eckhardt and Lee [7], considers the precise meaning of the independence referred to the failure behavior of the diverse program versions [16], [19].

Recently in [11], [22] have been proposed NVP models that combine performance and dependability measures via the concept of performability.

2. Modeling approach: basic concepts

Our approach to model NVP reliability is more general than previous ones as it combines information on the software structure and on the operational environment, taking into account the correlation among versions failures and execution times. Furthermore, the model parameters have clear physical interpretation and can be related to information about characteristics of the software and its operational environment.

The N version fault tolerance structure, as it operates in its use environment will be referred to as total system $S = (NVS, OE)$, where informally, N version system NVS and operational environment OE can be described as follows.

NVS consists of n independently designed program versions which are executed in parallel on a common given input. Since software faults can manifest only when it is executed, the execution time is the basic dimension of reliability measurement. We define software failure as 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 from the required output (in time or value) according to design specifications. Such a general definition enable us to consider both *functional* and *timing failures*. It is very important for real – time applications since failing to meet hard deadline has an adverse effect on system reliability. This issue has been addressed only in simulation based method presented in [8].

The detailed view of the operational environment OE is reviewed next. The operation of a software is broken down into series of *runs*. 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 or a set of tasks initiated by some intervention or input. Runs that are identical repetitions of each other are said to form a *run type*. Note that run type represents a transformation between an input state and an output state. Thus, the input state uniquely determines the particular instructions that will be executed and the value of their operands. A run type should ordinarily be associated with the accomplishment of a user function. Therefore, the variations in the environment can usually be characterized by variations in the relative probabilities of demand for different user functions. Since the probabilities of occurrence of input states are the natural way of representing the program usage the *operational profile* is defined as a set of relative frequencies of occurrence of the run types. If the relative frequencies of the run types occurrence have changed, then the operational profile has changed which affects the reliability.

3. N version system model

This model represents the NVS failure and execution behavior per run, that is on given input state. For this study we do not distinguish between similar and distinct coincident failures of the versions. 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 in the presence of software failures.

The model of the N version system is based on the following assumptions:

1. Corresponding to the different development processes, the version failures and execution times are conditionally independent, given a particular input state.

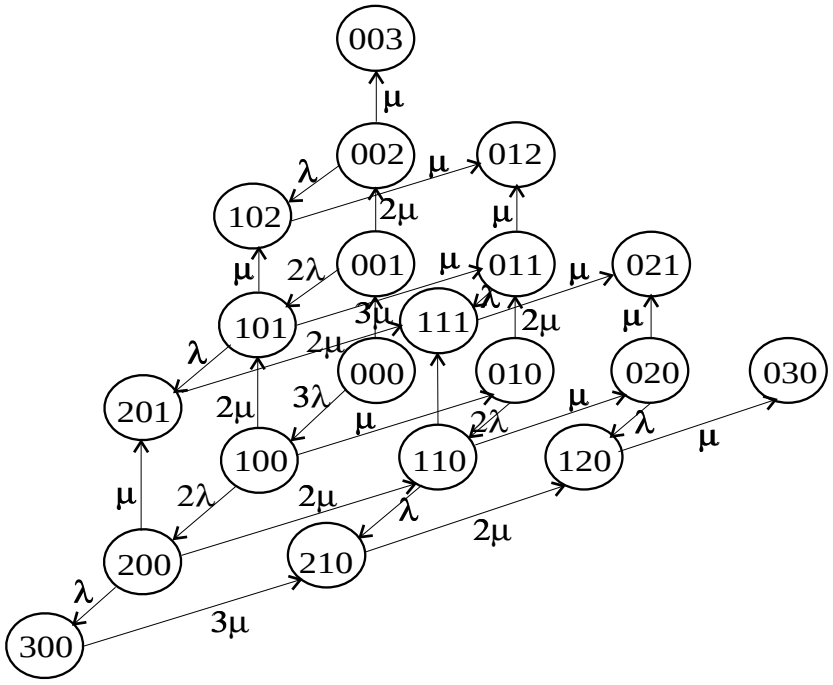
2. For each input state, times to failure of program versions are identically distributed random variables, as well as the execution times. Further, we take both of them to be exponentially distributed. The particular values of the failure rate and execution rate, given an input state, are λ and μ respectively.

The basic assumptions guarantee that a finite state continuous time Markov process can be used for the reliability modeling of the NVS. A state is defined as a vector (i, j, k), $0 \leq i, j, k \leq n$, $i + j + k \leq n$, where

i is the number of versions that have not completed the execution, and functional failure has been occurred during 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.



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 model of the NVS

The Markov model for the particular case of three version system, as well as the transitions and associated rates for the case of n versions are presented in Fig. 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 time rate μ which correspond to the given input state. To simplify notation, we write $P\{t \leq \tau|\lambda,\mu\}$ as $P\{\tau\}$.

Analytic processing of the Markov process leads to:

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

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 ended execution (with or without occurrence of functional failure) we can rewrite the expression (1) as:

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

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

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

The marginal pmf's of (6) are binomial and they are characterized in terms of the probabilities $P_{end}(\tau)$ (individual version produces correct result before τ), $P_{fend}(\tau)$ (individual version produces functionally incorrect result before τ), and $P_{noend}(\tau)$ (individual version has not completed the

execution until τ). Using the marginal pmf's we obtain the probabilities that characterize the NVS 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 $\leq \tau$

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

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

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

a majority of incorrect results (*functional failure*)

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

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 reader is referred to [11] for the method that could be used to relax the assumptions that time to failure and the execution time are exponentially distributed, which is omitted here due to space limitations.

3.1. Influence of NVS parameters

We desire a condition such that the NVS 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 are functions of a form

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

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 NVS has a greater probability of producing the correct result on time than do single version is

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

NVS reduces the timing failure probability whenever

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

NVS reduces the probability of functional failure if

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

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

where τ_1 is the numerical value obtained by solving $P_{fend}(\tau_1) = 0.5$.

The condition (15) is highly undesirable since such values of the model parameters certainly degrade the probability of success. It follows that the NVS effectiveness on particular input is improved with increasing n if the condition (12) is satisfied.

The following numerical examples illustrate the obtained analytical results. The value assigned to the timing constraint is $\tau = 30$ *msec.*

First, Fig. 2 shows the timing failure probability. The NVS has smaller timing failure probability then single version since the condition (13) is satisfied. However, if μ decreases (the average execution time of component versions increases) it takes significantly more versions to produce the same level of timing failure probability.

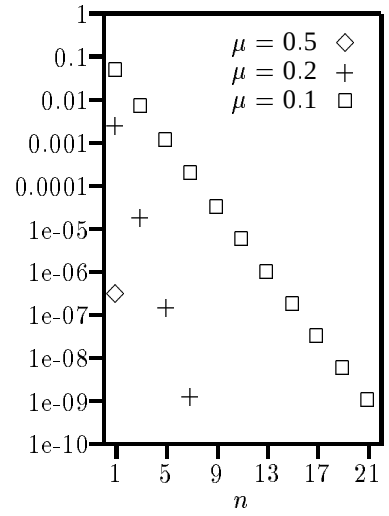
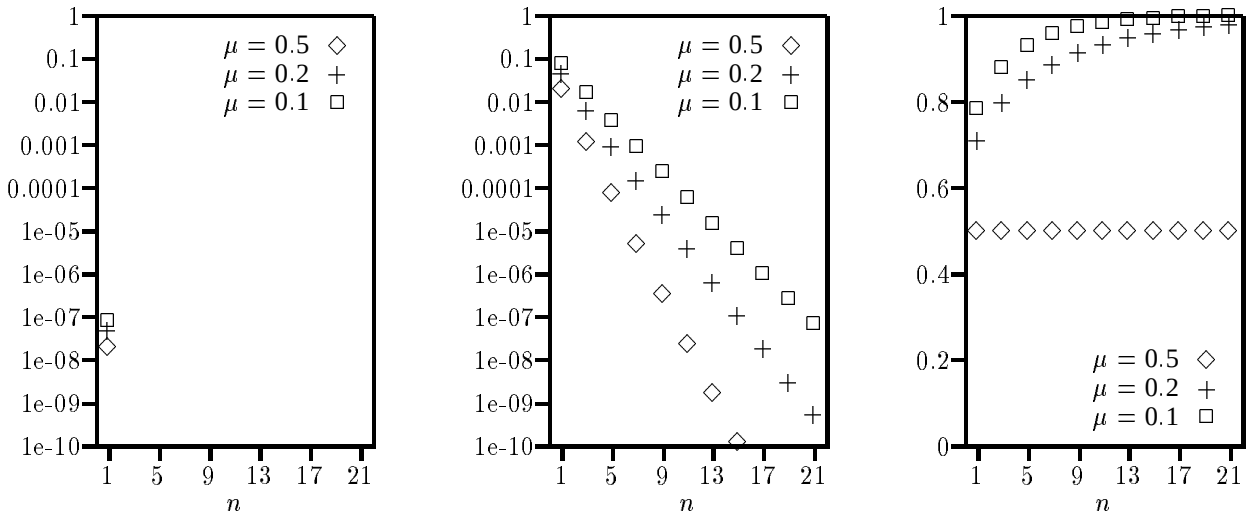


Figure 2. Timing failure probability on given input

In Fig. 3a and 3b we consider the functional failure probability when the condition (14) is satisfied. If $\mu \gg \lambda$ then only the functional failure probability of single version is greater then 10^{-10}, as plotted in Fig. 3a. Functional failure probability increases for higher failure rate λ, as well as for the smaller execution rate μ. The increase of failure rate results in substantial increase of the functional failure probability. In Fig. 3c we consider the functional failure probability for $\mu = \lambda$, or even $\mu < \lambda$. Since the condition (15) is not satisfied the NVS actually increases the functional failure probability.



(a) $\lambda = 10^{-8}$ (b) $\lambda = 0.01$ (c) $\lambda = 0.5$

Figure 3. Functional failure probability on given input

At last, Fig. 4 shows the total failure probability. It is clear that an increase in average execution time of component versions (decrease in μ) results in a substantial increase in total failure probability. Also, increasing failure rate λ results in an increase of total failure probability, although the effect is extremely small when $\mu \gg \lambda$. Note that the total failure probability is approximately equal to the timing failure probability if $\mu \gg \lambda$. The total failure probability when the condition (12) is satisfied is considered in Fig. 4a and 4b. Notice that the benefit of increasing the number of versions on the total failure probability is less significant in the case when λ is smaller by approximately several orders of magnitude than μ. As shown in Fig. 4c NVS is more prone to failures than a single version when $\mu \leq \lambda$ and increasing n degrades the probability of success. In this case the major contribution to the total failure probability comes from the functional failure probability.

4. Operational environment model

As indicated above, the objective of this model is to highlight the influence of the operational environment on the NVP reliability. We make the additional assumptions:

1. The environment is homogeneous or time invariant.

2. The operational period is sufficiently long so the input state selection probabilities can be characterized by a steady state.

3. Input states occur randomly and independently according to the operational profile.[1]

In order to obtain the probabilities of program failing on randomly chosen input the particular input defined by failure rate $\Lambda = \lambda$ and execution rate $M = \mu$ must be unconditioned from the probabilities obtained in previous section. Therefore, we model the failure rate and the execution rate as random variables, Λ and M respectively. Given a pair (Λ, M) of random variables their joint distribution function is given by

$$H(\lambda, \mu) = H_{\Lambda M}(\lambda, \mu) = P\{\Lambda \leq \lambda, M \leq \mu\}. \quad (16)$$

The probability of success, timing failure and functional failure for randomly chosen input state are:

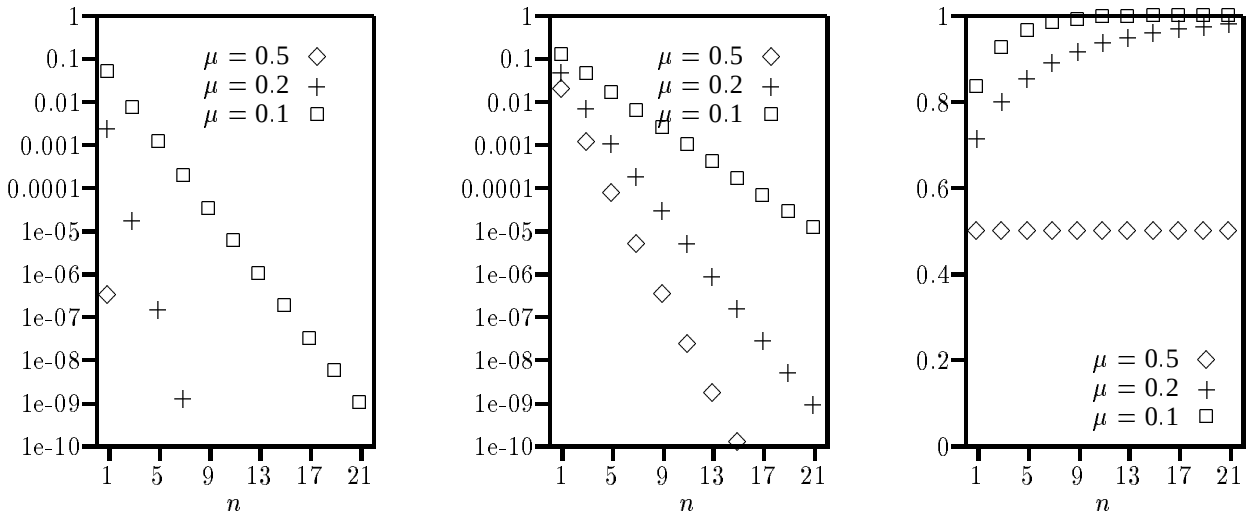
$$P_{ok}(\tau) = \int_0^\infty \int_0^\infty P_{ok}(\tau|\Lambda = \lambda, M = \mu)\, dH(\lambda, \mu) \quad (17)$$

$$P_{tf}(\tau) = \int_0^\infty \int_0^\infty P_{tf}(\tau|\Lambda = \lambda, M = \mu)\, dH(\lambda, \mu) \quad (18)$$

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

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). In general, we use the Lebesgue – Stieltjes integral, thus covering both the discrete and continuous distribution functions $H(\lambda, \mu)$.

[1] The last assumption, although usual in most software models [7], [16], [19] and software testing experiments is a simplification of real life, as it does not explicitly model the phenomena of failure clustering, that is the correlation of successive input states.



(a) $\lambda = 10^{-8}$ (b) $\lambda = 0.01$ (c) $\lambda = 0.5$

Figure 4. Total failure probability on given input

Instead of making assumptions about the independence of random variables Λ and M and using some theoretical distribution functions we develop *the user – oriented model of operational environment*. First, we partition the input space by grouping run types that exhibit (as nearly as possible) homogeneous failure and execution behavior into *run category*.[2] Suppose that input space Ω is partitioned in run categories $A_1, A_2, \ldots, A_r$. Each run category A_k is defined by failure rate $\Lambda = \lambda_i$ and execution rate $M = \mu_j$. 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. In this case Λ and M are discrete random variables and the relations (17),(18) and (19) signify the following:

$$P_{ok}(\tau) = \sum_{i=1}\sum_{j=1} P_{ok}(\tau|\Lambda = \lambda_i, M = \mu_j)\, p_{ij} \tag{20}$$

$$P_{tf}(\tau) = \sum_{i=1}\sum_{j=1} P_{tf}(\tau|\Lambda = \lambda_i, M = \mu_j)\, p_{ij} \tag{21}$$

$$P_{ff}(\tau) = \sum_{i=1}\sum_{j=1} P_{ff}(\tau|\Lambda = \lambda_i, M = \mu_j)\, p_{ij} \tag{22}$$

4.1. Analysis of the correlation

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 Z_k^s taking value 0 if the sth version produces correct output on time for given input, and value 1 otherwise. Its expectation $E[Z_k^s] = \rho_k$ is the probability that version fails on given input $x \in A_k$. The probability that version fails on the randomly selected input is $E[\sum_{k=1}^{r} Z_k^s\, p_k] = \sum_{k=1}^{r} \rho_k\, p_k$ while the probability that both versions fail on randomly selected input is $E[\sum_{k=1}^{r} Z_k^1 Z_k^2\, p_k] = \sum_{k=1}^{r} \rho_k^2\, p_k$. It follows that a necessary and sufficient condition for uncorrelated failures and execution times of the component versions is

$$\sum_{k=1}^{r} \rho_k^2 p_k - \left(\sum_{k=1}^{r} \rho_k p_k\right)^2 =$$

$$\sum_{S} p_{i(1)}\, p_{i(2)} \left[\rho_{i(1)} - \rho_{i(2)}\right]^2 = 0 \tag{23}$$

where the sum is over the set S of all distinct subsets $\{i(1), i(2)\}$ chosen without replacement from $\{1, 2, \ldots, r\}$.

This result shows that version failures and execution times are correlated whenever $\rho_{i(1)} \neq \rho_{i(2)}$, that is when Λ and M vary for different run categories A_k for which $p_k = Q(A_k) \neq 0$. The expression on the left in relation (23) is the covariance and *Cov* ≥ 0. It follows that this approach is pessimistic as it enables us to incorporate only positive correlation.

4.2. Influence of the operational environment

It is obvious that the degree of reliability improvement depends both on the variation of program characteristics (failure and execution rate) and the operational environment (input state selection probabilities). Informally, if $\rho_k >$

[2] It is reasonable to expect that failure behavior and execution time relate to the function implemented, the development methodology employed, or the capability of designer.

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

$$P1 = \begin{bmatrix} 0.999 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P2 = \begin{bmatrix} 0.999 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.001 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P3 = \begin{bmatrix} 0.999 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.001 & 0 \end{bmatrix}$$

Table 1. Parameter values

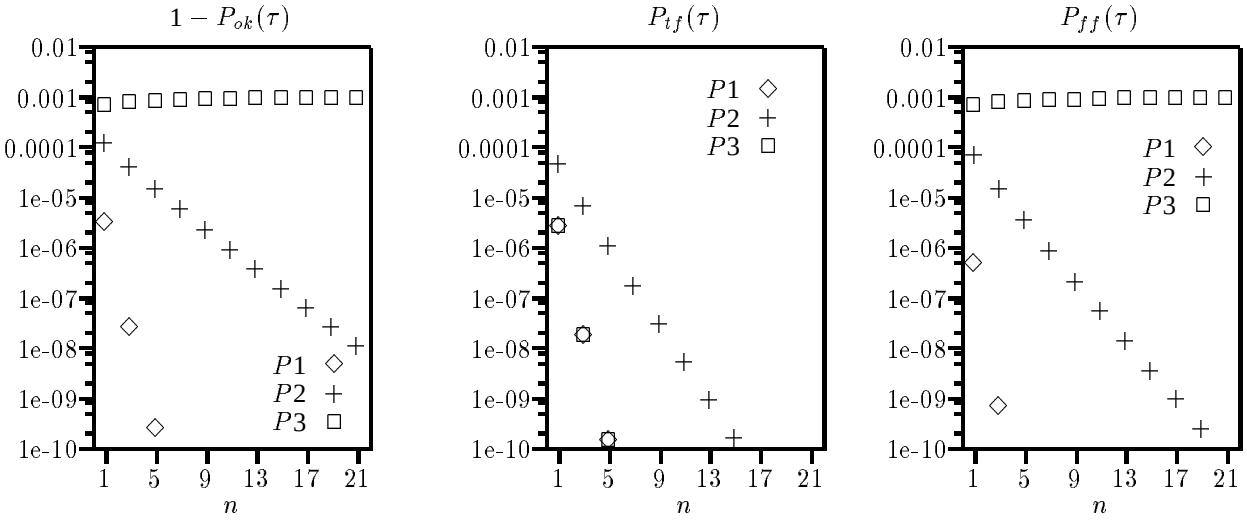


Figure 5. Failure probabilities on random input

0.5 for some run category A_k of the input space Ω for which $Q(A_k) > 0$ it becomes increasingly more difficult with increasing n to realize a majority of versions having correct output on time. 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. 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, NVP still have smaller probability of failure than do single version when the operation profile assigns greater mass to intervals of the input space $(0.5 - b, 0.5 - a]$, $0 \leq a < b$ than to their symmetrically located counterparts $[0.5 + a, 0.5 + b)$, as shown in [7].

The objective of the numerical results shown in Fig. 5 is to demonstrate the impact of the variation of version characteristics (failure and execution rate) and the operational environment (input state selection probabilities) on the failure probabilities. The values assigned to model parameters are shown in Table 1.

In the case of the operational profile $P1$ which encounters inputs that result in the small version's execution period compared to the time to failure ($\mu \gg \lambda$) increasing n does substantially reduce the failure probabilities. For example, 3 version system will reduce the total failure probability by approximately two orders of magnitude relative to that of a single version. Also, it is evident that the major contribution to the total failure probability comes from the timing failure probability.

A slight modification to the operational profile ($P2$) leads to the significant increase of failure probabilities. For example, $1 - P_{ok}(\tau) < 10^{-8}$ is achieved by 5 version system when $P1$ is assumed, rather then 21 versions when $P2$ is considered. Operational profile $P2$ also increases the timing failure probability since it encounters inputs that result in smaller execution rate μ.

The operational profile $P3$ assigns the same probability 0.001 as $P1$ to the worse program characteristics ($\lambda > \mu$). As a result NVP is more prone to failures than a single software component. Furthermore, both total and functional failure probabilities increase with increasing the number of versions. Note that the timing failure probability in the case of the operational profile $P3$ is the same as in the case of $P1$, and that the major contribution to the total failure probability comes from the functional failure probability.

In general, if the operational profile $P = [p_{ij}]$ encounters inputs that result in small duration of versions execution period compared to the time to failure ($\mu \gg \lambda$) the NVP leads to the significant reliability improvement. 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.

5. Conclusion

The reliability model of NVP as its operates in its operational environment presented in this paper provides a sound foundation for study of software fault tolerance. Representing the synchronization structure of the versions in terms of the execution time distribution is very important for real – time applications which are characterized by stringent deadlines and high reliability requirements. To the best of our knowledge, our approach is the only one allowing to consider both functional failures and timing failures. It is a useful contribution since failing to meet deadline has an adverse effect on system reliability. Moreover, our modeling approach permits the formulation of the user – oriented model of the operational environment, thus accounting the effects of the environment on the correlation of versions failures and execution times. The model addresses and resolves a number of basic issues, and yet yields closed – form solutions which reveal how model parameters influence the reliability measures. Furthermore, model based evaluation provides considerable insight into the conditions under which NVP improves the probability of producing correct result on time.

Anticipated future work include the correlation of successive input states (that is failure clustering), as well as the failure severity classes into analysis.

References

[1] J. Arlat, K. Kanoun, and J. Laprie. Dependability modeling and evaluation of software fault tolerant systems. *IEEE Transactions on Computers*, 39(4):504–513, April 1990.

[2] A. Avižienis and L. Chen. On the implementation of N version programming for software fault tolerance during program execution. In *Proceedings of the COMPSAC 77*, pages 149–155, 1977.

[3] A. Avižienis and J. Kelly. Fault tolerance by design diversity: Concepts and experiments. *IEEE Computers*, pages 67–80, August 1984.

[4] P. Bishop et al. Project on diverse software - an experiment in software reliability. In *Proceedings of the 4th IFAC Workshop SAFECOMP*, pages 153–158, 1985.

[5] J. Dugan and M. Lyu. System reliability analysis of N version programming application. In *Proceedings of the 4th IEEE International Symposium on Software Reliability Engineering*, pages 103–111, November 1993.

[6] D. Eckhardt et al. An experimental evaluation of software redundancy as a strategy for improving reliability. *IEEE Transactions on Software Engineering*, 17(7):692–702, July 1991.

[7] D. Eckhardt and L. Lee. A theoretical basis for the analysis of multiversion software subject to coincident errors. *IEEE Transaction on Software Engineering*, 11(12):1511–1517, December 1985.

[8] R. Geist, A. Offult, and F. Harris Jr. Estimation and enhancement of real time software reliability through mutation analysis. *IEEE Transactions on Computers*, 41(5):550–558, May 1992.

[9] K. Goševa – Popstojanova and A. Grnarov. A new Markov model of N version programming. In *Proceedings of the 2nd IEEE International Symposium on Software Reliability Engineering*, pages 210–215, May 1991.

[10] K. Goševa – Popstojanova and A. Grnarov. N version programming with majority voting decision: Dependability modeling and evaluation. In *Proceedings of the Euromicro 93*, pages 811–818, September 1993.

[11] K. Goševa – Popstojanova and A. Grnarov. Performability modeling of N version programming technique. In *Proceedings of the 6th IEEE International Symposium on Software Reliability Engineering*, pages 209–218, October 1995.

[12] A. Grnarov, J. Arlat, and A. Avižienis. On the performance of software fault tolerance strategies. In *Proceedings of the 10th Fault Tolerant Computing Symposium*, pages 252–253, October 1980.

[13] K. Kanoun et al. Reliability growth of fault tolerant software. *IEEE Transaction on Reliability*, 42(2):205–219, June 1993.

[14] J. Kelly, T. McVittie, and W. Yamamoto. Implementing design diversity to achieve fault tolerance. *IEEE Software*, pages 61–71, July 1991.

[15] J. C. Knight and N. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, 12(1):96–109, January 1986.

[16] B. Littlewood and D. Miller. A conceptual model of multiversion software. In *Proceedings of the 17th Fault Tolerant Computing Symposium*, pages 150–155, July 1987.

[17] M. Lyu and Y. T. He. Improving the N version programming process through the evaluation of a design paradigm. *IEEE Transaction on Reliability*, 42(2):179–189, June 1993.

[18] J. Musa. Operational profiles in software reliability engineering. *IEEE Software*, pages 14–32, March 1993.

[19] V. Nicola and A. Goyal. Modeling of correlated failures and community error recovery in multiversion software. *IEEE Transaction on Software Engineering*, 16(3):350–359, March 1990.

[20] R. Scott, J. Gault, and D. McAllister. Fault tolerant software reliability modeling. *IEEE Transactions on Software Engineering*, 13(5):582–592, May 1987.

[21] T. Shimeall and N. Leveson. An empirical comparison of software fault tolerance and fault elimination. *IEEE Transactions on Software Engineering*, 17(2):173–182, February 1991.

[22] A. Tai, J. Meyer, and A. Avižienis. Performability enhancement of fault tolerant software. *IEEE Transaction on Reliability*, 42(2):227–237, June 1993.