Performability and Reliability Modeling of N Version Fault Tolerant Software in Real Time Systems

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

The paper presents a hierarchical modeling approach of the N version programming in a real – time environment. The model is constructed in three layers. At the first layer we distinguish the NVP structure from its operational environment. The NVP structure submodel considers both failures of functionality and failures of performance. The operational environment submodel is based on the concept of the operational profile. The second layer consists of a per run reliability and performance submodels. The first considers per run failure probabilities, while the second is responsible for modeling the series of successive runs over a mission. The information contributed by the second layer constitutes third layer models which support the evaluation of a performability and reliability over mission. The work presented here generalizes our previous work as it considers general distributions of the versions time to failure and execution time. Also, in addition to the performability model, the third layer includes a model aimed at reliability assessment over a mission period.

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 [1], [21]. A consistent set of inputs is supplied to all N versions that are executed in parallel. A decision mechanism must gather the available results from the 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 will 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 experimental studies of NVP have introduced diversity in the form of different specifications [1], [26], [21], different programming languages [26], and for different distributions of test values over the input space [26]. All versions were developed independently, by different teams, in some studies even by geographically distinct participants [26], [7], [17], [24]. These experiments reveal several characteristics of NVP. First, they show that the assumption of independence of failures between independently developed programs does not hold. Next, coincident failures were observed in every experiment conducted thus far. Also, the failure behavior is very sensitive to the distribution of test values over the input space [26].

Dependability models of software fault tolerance may conveniently grouped into two classes. On one side, the major goal for the first class is the modeling of the dependability measures of the particular fault tolerant structure. Methods of specifying the system structure include combinational [3], [28], discrete time Markov chain [15], continuous time Markov process [10], [12], [11], fault trees and Markov reward models [16], extended stochastic Petri net and simulation [27], and generalized stochastic Petri nets [22] model types. On the other side, the major goal of second class models that are based on the ground breaking work of Eckhardt and Lee's [8], is the precise meaning of the independence referred to the failure behavior of the diverse program versions. The key idea is that the intensity of coincident errors $\theta(x)$ will generally take different values for different inputs x. The situation when θ vary not only from one input to another, but from one development methodology to another is presented in [5]. Other contributions in this class examine several methods for determining the intensity distribution [29], [23].

In order to place certain constraints on how properties affecting performance interact with those affecting dependability, a unified measure called performability has been introduced [20]. Until the recently proposed performability models of the NVP in [13] and [4] model based evaluation of software fault tolerance techniques has been focused on either separate evaluation of performance and dependability

[3],[25], or strict measures of dependability.

2. NVP model

With the benefit of this background, we have developed a hierarchical modeling approach aimed at performability and reliability assessment of the NVP. The model presented in this paper generalizes our previous work reported in [13] and [14] by considering general distributions of the time to failure and execution time, by introducing two different performability measures and by developing a new model aimed at reliability assessment over a mission.

The fault tolerant software system investigated here is for the real – time mission – critical applications. We consider the systems for which it is not possible to perform a repair during their mission. These systems are characterized by high reliability requirements and stringent deadlines. The acceptable probability of failure is very small, typically in the range of 10^{-5} to 10^{-10} per hour. The real – time performance requirements are also very demanding. For example, advanced variable – cycle jet engines can blow up if correct control outcomes are not applied every 20 – 50 milliseconds. The success of such a system depends not only on its logical correctness, but also on its timing correctness, that is it must make correct responses to environmental changes within specified time intervals or deadlines.

In the real –time mission critical systems the software periodically gets the inputs from the environment, updates its internal states based on those inputs, and generates control output. It means that a mission is composed of a series of runs of fault tolerant software. If the run lasts beyond a pre – set maximum duration, it is aborted by a watchdog timer. The control stimulii are generated either by completing the execution or by timers, that is the software periodically checks to see if an event of the required type has occurred, instead of passively waiting to be triggered by an event.

Our approach keeps the solution efficient by using hierarchical decomposition wherein submodels represent different parts of an object system and time scale distinctions. The model is constructed in three layers as shown in Fig. 1. The arrows indicate the interaction between submodels and the flow of information.

The first layer consists of NVP structure submodel and operational environment submodel. Since the effects of design faults are typically quite sensitive to just how a system is utilized, it is helpful to distinguish an object system (NVP) from its environment (other systems, physical or human which interact with the object system during its use).

The per run reliability submodel and performance submodel, that constitute the second layer, integrate the first layer submodels. At this layer we made time and behavior distinctions. The per run reliability submodel represents failure and execution behavior of NVP and the time is treated

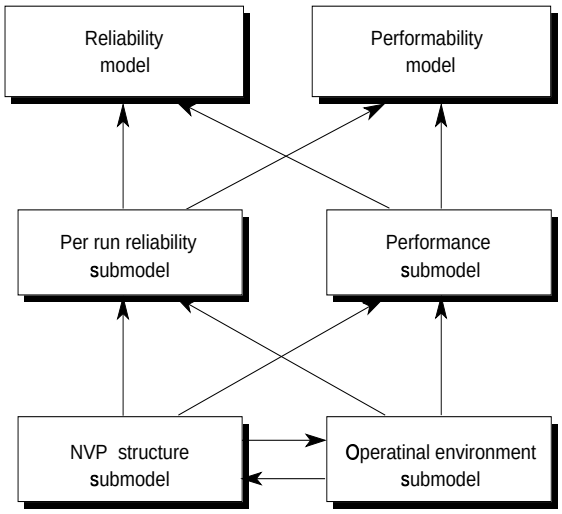

Figure 1. Hierarchical model

locally measured from the beginning of a run. The performance submodel This submodel is responsible for modeling the series of runs over a mission duration. It considers only the execution behavior of NVP and treats time globally.

The information contributed by the second layer constitutes the third layer which supports the assessment of a performability and reliability over a mission. The performability model considers the collective effect of reliability and performance attributes on the ability of NVP to complete a certain amount of useful work over a mission. The reliability model accounts for performance requirements which is particularly relevant for real – time applications because failing to meet deadline have an adverse effect on system reliability.

2.1. NVP structure submodel

The NVP structure 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 next. *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 [19]. 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. To the best of our knowledge our approach is the only one allowing analytical modeling of reliability that depends not only on the conventional concept of reliability, based on failures of functionality, but also reliability based on failures of performance. So far, this is-

sue has been addressed only in the simulation based method presented in [27].

For this study we do not distinguish between detected and undetected coincident failures, that is we do not investigate the error detection capabilities of NVP[1].

In the NVP structure submodel the time is treated locally measured from the beginning of a run. This submodel is based on the following assumptions:

- The versions behavior are conditionally independent given a particular input state.
- Times to failure of program versions for given input state are identically distributed random variables with pdf $f_F(t;\upsilon)$ depending on d dimensional parameter vector $\upsilon = (\upsilon_1, \ldots, \upsilon_d)$.
- Execution times of program versions for given input state are identically distributed random variables with pdf $f_E(t;\psi)$ depending on b dimensional parameter vector $\psi = (\psi_1, \ldots, \psi_b)$.
- Execution time of the voting algorithm is negligible compared to the execution time of each version.
- Due to a real – time constraint the system must make correct responses within a time interval $\tau > 0$.

Due to the first assumption the first step is to model single version behavior. If the version produces output within specified time τ with its contents corrupted then a functional failure occurs. If the output (either correct or incorrect) is produced later than specified deadline τ then the version is said to have suffered a performance (timing) failure. In other words, a potential failure of functionality could be masked by a performance failure.

The distribution function $F_F(t;\upsilon)$ gives the probability that a single version will fail before t and it considers the failures of functionality. Since the duration of the execution time t is a random variable with density $f_E(t;\psi)$ we use the method that in [9] is described probabilistically as randomized time. It follows that the probability that single version produces functionally incorrect result before τ becomes

$$P_f(\tau;\upsilon,\psi) = \int_0^\tau F_F(t;\upsilon) f_E(t;\psi)dt \qquad (1)$$

and the probability that a single version produces correct result before τ is given by

$$P_e(\tau;\upsilon,\psi) = \int_0^\tau [1 - F_F(t;\upsilon)] f_E(t;\psi)dt. \qquad (2)$$

[1]The distinction between detected and undetected failures has already been considered in dependability models of software fault tolerant techniques [12] and [11] that consider only the failures of functionality.

The failure of performance (timing failure) occurs if a single version do not complete the execution until deadline τ

$$P_{ne}(\tau;\psi) = 1 - \int_0^\tau f_E(t;\psi)dt = 1 - F_E(\tau;\psi) \qquad (3)$$

Next, consider the system with n versions. Since there are three distinct outcomes and due to the first assumption, the probability that n_1 versions produce functionally correct result before τ, n_2 versions produce functionally incorrect result before τ, and $n_3 = n - n_1 - n_2$ versions do not complete the execution until τ is given by

$$\frac{n!}{n_1!\, n_2!\, n_3!}\, P_e^{n_1}(\tau;\upsilon,\psi)\, P_f^{n_2}(\tau;\upsilon,\psi)\, P_{ne}^{n_3}(\tau;\psi) \qquad (4)$$

which is multinomial pmf. Using the marginal pmf's of the distribution (4) we obtain the probabilities which characterize NVP failure and execution behavior for given input state. Thus, we define *timing failure* of N version system ($n = 2m - 1$) for given input state to be the event that majority of versions do not produce output in time $\leq \tau$

$$P_{tf}(\tau;\psi) = \sum_{n_3=m}^{n} \binom{n}{n_3} P_{ne}^{n_3}(\tau;\psi) \left[1 - P_{ne}(\tau;\psi)\right]^{n-n_3} \qquad (5)$$

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

$$P_{ff}(\tau;\upsilon,\psi) = \sum_{n_2=m}^{n} \binom{n}{n_2} P_f^{n_2}(\tau;\upsilon,\psi) \left[1 - P_f(\tau;\upsilon,\psi)\right]^{n-n_2} \qquad (6)$$

a majority of correct results (*success*)

$$P_{ok}(\tau;\upsilon,\psi) = \sum_{n_1=m}^{n} \binom{n}{n_1} P_e^{n_1}(\tau;\upsilon,\psi) \left[1 - P_e(\tau;\upsilon,\psi)\right]^{n-n_1} \qquad (7)$$

or there is no majority of either correct or incorrect results

$$P_{nm}(\tau;\upsilon,\psi) = 1 - P_{tf}(\tau;\psi) - P_{ok}(\tau;\upsilon,\psi) - P_{ff}(\tau;\upsilon,\psi).$$

2.2. Operational environment submodel

The concept of the operational environment is reviewed next. We use the same notion as Musa did in [19] and [18]. 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 run types selection have changed, then the operational profile has changed and that will affect the NVP behavior.

Developing this submodel we make the following additional assumptions:

- The environment is homogeneous (time invariant).
- 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.

Since the failure and execution behavior are quite sensitive to just how a system is utilized we need to take into consideration the change of the parameter vectors υ and ψ for different run types (input states). Therefore, the parameter vectors υ and ψ appear as random vectors Υ and Ψ respectively. The pair of random vectors $(\Upsilon\Psi) = (\Upsilon_1, \ldots, \Upsilon_d, \Psi_1, \ldots, \Psi_b)$ may be thought of as an event defined on a sample space $\mathcal{R}^{d+b}$ with $d+b$ dimensional distribution function $H_{\Upsilon\Psi}(\upsilon, \psi)$, while b variate distribution $H_\Psi(\psi)$ can be viewed as being marginal distribution of $d+b$ variate distribution $H_{\Upsilon\Psi}(\upsilon, \psi)$.

In order to obtain numerical results it is possible to make assumptions and to use some theoretical distribution functions. Instead, we choose to develop *the user – oriented model of operational environment*. Therefore, we partition the input space Ω by grouping run types that exhibit as nearly as possible homogeneous failure and execution behavior into *run categories*[2]. Suppose that input space Ω is partitioned into run categories A_{kj}, $1 \le k \le r_1, 1 \le j \le r_2$ such that $\sum_k \sum_j A_{kj} = \Omega$. Each run category A_{kj} is defined by parameter vectors $\Upsilon = \upsilon^k$ and $\Psi = \psi^j$. In the case the operational profile Q gives the probabilities $P\{\Upsilon = \upsilon^k, \Psi = \psi^j\} = p_{kj} = Q(A_{kj})$ that successive input states are chosen at random in run category A_{kj}.

2.3. Per run reliability submodel

The per run reliability submodel treats the time locally measured from the beginning of a run and represents failure and execution behavior of NVP. This submodel integrates the NVP structure submodel and operational environment submodel using the randomization approach called stratification [9]. Therefore, we model parameter vectors υ and ψ as random vectors Υ and Ψ. The unconditional per run probabilities of success, timing failure and functional failure for randomly chosen input state become:

$$P_{ok}(\tau) = \int_{\mathcal{R}^{d+b}} P_{ok}(\tau|\Upsilon = \upsilon, \Psi = \psi)\, dH_{\Upsilon\Psi}(\upsilon, \psi) \qquad (8)$$

$$P_{tf}(\tau) = \int_{\mathcal{R}^b} P_{tf}(\tau|\Psi = \psi)\, dH_\Psi(\psi) \qquad (9)$$

$$P_{ff}(\tau) = \int_{\mathcal{R}^{d+b}} P_{ff}(\tau|\Upsilon = \upsilon, \Psi = \psi)\, dH_{\Upsilon\Psi}(\upsilon, \psi) \qquad (10)$$

where $P_{ok}(\tau|\Upsilon = \upsilon, \Psi = \psi)$, $P_{tf}(\tau|\Psi = \psi)$, $P_{ff}(\tau|\Upsilon = \upsilon, \Psi = \psi)$ are given in (7), (5), (6). Note that, the Lebesgue – Stieltjes integral covers both discrete and continuous distribution functions for $H_{\Upsilon\Psi}(\upsilon, \psi)$ and $H_\Psi(\psi)$. Distribution functions of form (8), (9), (10) are called mixtures or compound distributions, while the distribution function $H_{\Upsilon\Psi}(\upsilon, \psi)$ is called mixing distribution [6], [9].

In the case of the user – oriented model of the operational environment Υ and Ψ take finite number of values, that is $H_{\Upsilon\Psi}(\upsilon, \psi)$ and $H_\Psi(\psi)$ are the discrete distribution function and the relations (8), (9), and (10) signify the following

$$P_{ok}(\tau) = \sum_k \sum_j P_{ok}(\tau|\Upsilon = \upsilon^k, \Psi = \psi^j)\, p_{kj} \qquad (11)$$

$$P_{tf}(\tau) = \sum_j P_{tf}(\tau|\Psi = \psi^j)\, p_j \qquad (12)$$

$$P_{ff}(\tau) = \sum_k \sum_j P_{ff}(\tau|\Upsilon = \upsilon^k, \Psi = \psi^j)\, p_{kj} \qquad (13)$$

where $p_j = P\{\Psi = \psi^j\} = \sum_k p_{kj}$ is a marginal distribution of a distribution $p_{kj} = P\{\Upsilon = \upsilon^k, \Psi = \psi^j\}$.

Methods that have been derived and used for estimating the mixing proportions p_{kj}, parameter vectors (υ^k, ψ^j) and the number of components of the finite mixture $r = r_1 r_2$ could be found in [6].

The reader is referred to our previous papers [13] and [14] for detail analysis of the correlation between versions. It is shown that there is a correlation between versions behavior for a single input whenever Υ and Ψ vary for different run categories. Our approach is more general and much more realistic then previous ones since the arguments given in [8] and [5] for existence of correlation between failures of independently developed versions apply to versions execution times, as well. Even more, the execution times of versions are not likely to be independent of their producing erroneous or correct result. Our approach considers implicitly this type of dependence since it is possible Υ and Ψ to be correlated random vectors, thus reflecting the influence of failure behavior on execution behavior.

2.4. 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. This submodel treats time globally and represents the iterative nature of software's execution, that is the series of runs during the mission duration. At each run, the software

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

accepts an input and produces an output that is a function only of the most recently accepted input[3].

For the performance submodel it 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 time elapsed from $(i-1)$st run until the occurrence of ith run. We derive the distribution of the time between successive renewals T_i integrating the information contributed by the NVP structure submodel and operational environment submodel.

First, we derive the conditional distribution for a given input state using the order statistics. Let $X_1, X_2, \ldots, X_n$ be the random variables that represent the execution time of each version given a particular input state, each having a distribution function $F_E(t;\psi)$, as assumed in the NVP structure submodel. The probability that at least m of the X_n's lie in the interval $(0,t]$ is given by

$$F(t;\psi) = \sum_{l=m}^{n} \binom{n}{l} F_E^l(\tau;\psi)[1-F_E(\tau;\psi)]^{n-l} \qquad (14)$$

Next, we obtain the unconditional probability distribution function of the time between successive renewals for random input state using the randomization procedure called stratification. As in the reliability submodel we treat the parameter vector ψ as random vector $\mathbf{\Psi}$, so (14) signifies the conditional probability distribution $F(t|\mathbf{\Psi})$. It follows that

$$F(t) = \int_{\mathcal{R}^b} F(t|\mathbf{\Psi}=\psi)\, dH_{\mathbf{\Psi}}(\psi). \qquad (15)$$

For the user oriented model of the operational environment (15) becomes

$$F(t) = \sum_j F(t|\mathbf{\Psi}=\psi^j)\, p_j. \qquad (16)$$

Finally, due to the real – time constraint, T_i is the time upon the completion of the NVP execution or upon reaching τ, whichever occurs first. It means that the probability distribution of T_i is

$$F_\tau(t) = \begin{cases} F(t), & \text{for } t < \tau \\ 1, & \text{for } t \geq \tau \end{cases} \qquad (17)$$

with mean recurrence time $\quad m_\tau = \int_0^\tau [1-F(t)]dt \quad (18)$

and variance $\quad \sigma_\tau^2 = 2\int_0^\tau t[1-F(t)]dt - m_\tau^2. \qquad (19)$

The performance submodel is also responsible for supplying the expected number of runs during the mission:

$$E[N(t)] = M(t) = \sum_{k=1}^{\infty} F_\tau^{k*}(t) \qquad (20)$$

[3]Although this is a usual assumption in most software models and software testing experiments, it is a simplification of a 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.

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

We emphasize that the versions execution times are correlated for a single input whenever the parameter vector $\mathbf{\Psi}$ is not identical for all run categories. Considering the correlation between execution times much more realistic than the assumption of independence made in [4] and [25].

2.5. Reliability model

The first of the third layer models supports the assessment of the NVP reliability over a mission, that is the distribution of the time to failure as a function of global time. This is done by using the decomposition of a renewal process. Consider a renewal process with distribution $F_\tau(t)$ defined by performance submodel (17), and suppose that each event is erased with probability $p = P_{ok}(\tau)$ computed by the per run reliability submodel (11). The resulting sequence of events constitutes a renewal process $\{\hat{N}(t), t \geq 0\}$ that registers only the successive occurrences of NVP failures. Its interval distribution is

$$\hat{F}(t) = \sum_{k=1}^{\infty} p^{k-1}\,(1-p)\, F_\tau^{k*}(t). \qquad (21)$$

If the Laplace transform of $F_\tau(t)$ is $B(s)$ then the Laplace transform of the distribution of the time to failure $\hat{F}(t)$ is

$$\hat{B}(s) = \sum_{k=1}^{\infty} p^{k-1}\,(1-p)\, B^k(s) = \frac{(1-p)\,B(s)}{1-pB(s)}. \qquad (22)$$

The expression $\hat{B}(s)$ in s – domain (22) can be inverted numerically to obtain the solution in time domain for $\hat{F}(t)$, which leads to non trivial conservative estimates.

If only a value of the mean time to failure MTTF (expected amount of time that software operates before failure) is a desired solution, procedure is typically less complex due to the well known property of Laplace transform:

$$MTTF = E[\hat{F}(t)] = -\left.\frac{d\hat{B}(s)}{ds}\right|_{s=0} = \frac{m_\tau}{1-p}. \qquad (23)$$

2.6. Performability model

The performability model is the second model at the third level and it combines the information contributed by the per run reliability and performance submodels. It is possible to consider a variety of performability measures depending on particular application. Similarly to the experimental study [24] which examined two levels of granularity in counting errors (so – called errors by time and errors by case) we examine two performability measures:

1. number of successful runs over a mission duration t
2. number of successful runs provided there are no failures over a mission duration t

In the case of the first measure the failures are counted only at the run in which they manifested themselves, which means that all successful runs during the mission period are beneficial. As for the second performability measure, if NVP fails (either by value or by time) at any run in a mission it is considered failed for the whole mission. In this case, no run either prior or subsequent to such failure is beneficial.

For the first performability measure we associate with the k-th renewal interval an indicator random variable Z_k which may be interpreted as reward rate

$$Z_k = \begin{cases} 1 & \text{if the k-th run is successful} \\ 0 & \text{otherwise.} \end{cases} \quad (24)$$

with expected value $E[Z_k] = P_{ok}(\tau) = p$ defined by (11).

The accumulated reward up to time t is defined by the cumulative process

$$W(t) = \sum_{k=1}^{N(t)+1} Z_k. \quad (25)$$

Conditioning on the time $T_1 = x$ until the first renewal, and examining the two possibilities $x > t$ and $x \leq t$, we secure for $A(t) = E[W(t)]$ the renewal equation

$$A(t) = E[Z_1] + \int_0^t A(t-x)\, dF_\tau(x) \quad (26)$$

which resuls in

$$E[W(t)] = E[Z_1]\,[1 + M(t)] = P_{ok}(\tau)\,[1 + M(t)]. \quad (27)$$

It is obvious that performability measure represents the collective effect of system attributes computed from the per run reliability submodel $P_{ok}(\tau)$ (11) and performance submodel $M(t)$ (20).

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 [9] leads to

$$\lim_{t\to\infty} A(t) = P_{ok}(\tau)\,\left[1 + \frac{t}{m_\tau}\right] \quad (28)$$

where m_τ

The second performability measure is formulated in a similar fashion to the "effectiveness" in [4], but it is even more restrictive as it counts as one unit the successful runs conditioned by the event that no run at all fails during the period $(0, t]$. To aid formulation of $U(t)$ we define the following intermediate lavel random variables:

SR_t – number of successful runs during $(0, t]$

FF_t – number of runs during $(0, t]$ which provide incorrect outcome on time (functional failure)

TF_t – number of runs during $(0, t]$ which doesn't provide outcome on time (timing failure).

$U(t)$ can be formulated in terms of SR_t, FF_t, and TF_t as

$$U(t) = \begin{cases} SR_t & \text{if } FF_t + TF_t = 0 \\ 0 & \text{otherwise.} \end{cases} \quad (29)$$

The moment generation function of the variable $U(t)$ is

$$E[e^{sU(t)}] = E[1 - p^{N(t)} + p^{N(t)}\, e^{sN(t)}] \quad (30)$$

and its expectation can be expressed as

$$E[U(t)] = \left.\frac{dE[e^{sU(t)}]}{ds}\right|_{s=0} = E[N(t)\, p^{N(t)}]. \quad (31)$$

The asymptotic behavior is determined by the fact that for large t the number $N(t)$ of renewal intervals is approximately normally distributed with mean t/m_τ and variance $t\sigma_\tau^2/m_\tau^3$ [9]. If further, we let $\hat{m} = t/m_\tau$, $\hat{\sigma^2} = t\sigma_\tau^2/m_\tau^3$ and $\alpha = \ln p$ we have

$$E[U(t)] = e^{\alpha\hat{m}} \int_{-\hat{m}/\hat{\sigma}}^{\infty} (\hat{\sigma}x + \hat{m})\, e^{\alpha\hat{\sigma}x}\, d\Phi(x) \quad (32)$$

where $\Phi(x) = 1/\sqrt{2\pi}\, \int_{-\infty}^{x} e^{-\frac{u^2}{2}}\, du$ is the normal integral.

3. Numerical example

The presented model allows the use of general distributions of the time to failure and execution time and numerical solutions could be obtained using the evaluation tools. Nevertheless, using exponential distributions (which are the usual assumptions in most software reliability models) enable us to derive close form solutions. The numerical example is based on assumptions that versions pdf of the time to failure is $f_F(t;\upsilon) = \lambda e^{-\lambda t}$ and pdf of the execution time is $f_E(t;\psi) = \mu e^{-\mu t}$. It means that the parameter vectors consists of one component $\Upsilon = (\lambda)$ and $\Psi = (\mu)$. This leads to the operational profile which is defined by two variate pmf $p_{kj} = P\{\Upsilon = \lambda_k, \Psi = \mu_j\}$ that gives the probability that a successive input states are chosen at random in the run category defined by failure rate λ_k and execution rate μ_j. The values assigned to model parameters are shown in Table 1.

Fig. 2 plots the MTTF for varying number of versions and four different operational profiles. The operational profile $P1$ encounters inputs that result in the small versions execution period compared to the time to failure ($\mu \gg \lambda$) and versions are uncorrelated since there is no variation of the failure and execution rates over the input space. It is evident that NVP substantially increases the MTTF. The operational profile $P2$ also encounters inputs that result in the small versions execution period compared to the time to failure, but in this case the varsions failures and execution times are correlated. It is obvious that increasing the number of versions still substantially increases the MTTF. However, seven versions would be required to achieve the same level of MTTF as in the case of 3 version system under the assumptions of independence. The operational

$\lambda = [10^{-10} \quad 10^{-8} \quad 10^{-6} \quad 10^{-4}] \qquad \mu = [0.5 \quad 0.2 \quad 0.125] \qquad \tau = 30 \qquad t = 36 \cdot 10^6$

$$P1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P2 = \begin{bmatrix} 0.999 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P3 = \begin{bmatrix} 0.999 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.001 \\ 0 & 0 & 0 \end{bmatrix} \quad P4 = \begin{bmatrix} 0.999 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}$$

Table 1. Parameter values

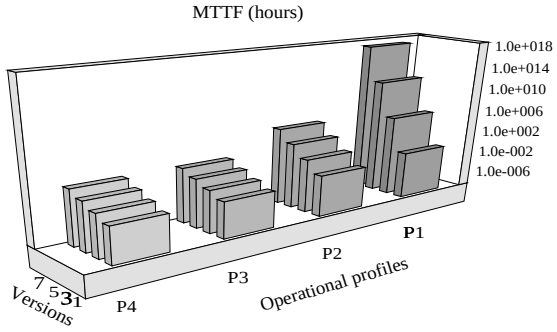

Figure 2. Mean time to failure

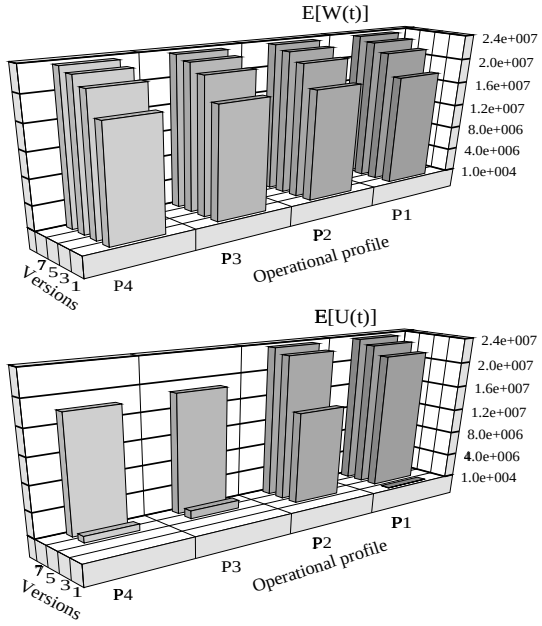

Figure 3. Performability measures

profile $P3$ and $P4$ assign the same probability 0.001 as $P2$ to the worse program characteristics (higher failure rate and smaller execution rate). As a result, there is a decrease in estimated MTTF over that exhibited for operational profile $P2$. Note that, this choice of parameters implies a mean recurrence time m_τ of the same order of magnitude, so the variations of the MTTF due mostly to the variations of the total per run failure probability $1 - p = 1 - P_{ok}(\tau)$. That is, the increased failure probability results in greater chance to encounter unsuccessful run over a mission.

As indicated on Fig. 3 the performability related measures are also degraded by the operational profiles that encounter inputs associated with worse program characteristics. However, increasing the number of versions for given operational profile results in performability gain. The first performability measure $E[W(t)]$ is less sensitive to the variations in the operational profile because the failures are counted only at the run in which they manifested themselves. Comparing the two performability measures we observe that in the case of the operational profiles that encounter inputs associated with moderate to high per run failure probability ($P3$ and $P4$) the reduction of $U(t)$ is more dramatic. This is due to the fact that in the case of the second performability measure if NVP fails at any run in a mission it is considered failed for the whole mission, which in turn results in no performability gain.

Certain operational profiles, although higly unlikely, can select inputs associated with even worse program characteristics ($\mu \sim \lambda$) which result in highly unreliable system. Accordingly, there will be more significant reduction of $E[W(t)]$. Evenmore, MTTF and $E[U(t)]$, that are far more sensitive to the variation of program characteristics over the input space than $E[W(t)]$, become exceedingly low and can not be improved by any degree of fault tolerance.

4. Conclusions and future work

This paper presents the modeling based study of the N version fault tolerant software in a real - time environment. Our modeling approach is systematic as opposed to the ad hoc methods used in related works. Moreover, it is more general and much more realistic than earlier models since it

- considers general distributions of the time to failure and execution time
- integrates the NVP structure and system usage in its operational environment
- considers the correlation between versions failure and execution behavior for a single input

- results in the mission reliability that accounts for performance requirements
- considers two different performability measures that reflect the collective effect of reliability and performance attributes on the ability of NVP to complete a certain amount of useful work over a mission.

The hierarchical decomposition keeps the solution efficient and permits modifications to be flexibly made at a specific level or in a specific submodel. There are several possible extensions that could be easily handled. First of all, the numerical evaluation for different time to failure and execution time distributions could be made using evaluation tools or simulation. Next, the correlation of successive input states might be considered at the second layer submodels in order to model the phenomena of failure clustering. Finally, the performance submodel could account for distribution functions between renewals that also depend on the outcome result, that is on the versions per run reliability.

References

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

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

[3] A.Grnarov, J.Arlat, and A.Avižienis. On the performance of software fault tolerance strategies. In *Proc. of the IEEE Int. Symp. Fault Tolerant Computing*, pages 251–253, 1980.

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

[5] B.Littlewood and D.R.Miller. A conceptual model of multiversion software. In *Proc. of the IEEE Int. Symp. Fault Tolerant Computing*, pages 150–155, 1987.

[6] B.S.Everitt and D.J.Hand. *Finite Mixture Distributions*. Chapman and Hall, 1981.

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

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

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

[10] K. Goševa-Popstojanova and A.Grnarov. A new markov model of n version programming. In *Proc. of the IEEE Int. Symp. Software Reliability Engineering*, pages 210–215, 1991.

[11] K. Goševa-Popstojanova and A.Grnarov. Dependability modeling and evaluation of recovery block systems. In *Proc. of the IEEE Int. Symp. Software Reliability Engineering*, pages 112–120, 1993.

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

[13] K. Goševa-Popstojanova and A.Grnarov. Performability modeling of n version programming technique. In *Proc. of the IEEE Int. Symp. Software Reliability Engineering*, pages 209–218, 1995.

[14] K. Goševa-Popstojanova and A.Grnarov. N version programming: An unified modeling approach. In *Proc. of the EUROMICRO-22*, pages 363–370, 1996.

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

[16] J.B.Dugan and M.R.Lyu. System reliability analysis of n version programming application. In *Proc. of the IEEE Int. Symp. Software Reliability Engineering*, pages 103–111, 1993.

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

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

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

[20] J.F.Meyer. On evaluation the performability of degradable computing systems. *IEEE Trans. on Computers*, 29(8):720–731, August 1980.

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

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

[23] L.A.Tomek, J.K.Muppala, and K.S.Trivedi. Modeling correlation in software recovery blocks. *IEEE Trans. on Software Engineering*, 19(11):1071–1086, November 1993.

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

[25] M.Vouk, A.M.Paradkar, and D.F.McAllister. Modeling execution time of multi – stage n version fault tolerant software. In *Proc. of the IEEE Computer Software and Applications Conference*, pages 505–511, 1990.

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

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

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

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