A New Markov Model of N Version Programming Systems

Katerina D. Goševa-Popstojanova
Aksenti L. Grnarov

Faculty of Electrical Engineering, University of Skopje
P.O.Box 574, Karpos bb, Skopje 91000, Yugoslavia

Abstract

The paper presents reliability performance modeling of N Version Programming. The study is based on continuous time Markov model for the general case of N versions. Derived mathematical relations between reliability performances (as a function of version execution time) and modeling parameters enable us to gain a great deal of quantitative results. The obtained results can be used to guide a design of actual systems.

Topic:	Reliability modeling
Domain Area:	Software fault - tolerant, NVP
Language/Tool:	IBM PC Turbo Pascal
Status:	Research
Effort:	1 person - year
Impact:	The assessment of the NVP reliability

1 Introduction

Computing systems are used in increasingly complex situations, hence system complexity itself becomes one of the major barriers to achieve required high level of reliability. A great part of this complexity is in the software so it is a critical part of any high reliable computing system. Despite of fault prevention techniques (precise specifications, design methodologies, structured programming techniques, proving, testing, etc.) which may have been used, a complex software system will always contain design faults when it is put into operation. Therefore, the importance of software fault – tolerance techniques can only increase.

N Version Programming [2], [3] has been proposed as a method of providing fault - tolerance in software. The approach (so called design diversity) requires independent design of multiple ($N > 2$) versions of a piece of software for the same application which satisfy a common specification. These versions are executed in parallel in the same application environment: each of them receives identical inputs and each produces its version of the required outputs. The outputs are collected by a voter and results of the majority are assumed to be the correct output used by the system. The "bad" versions are recovered by updating them with data provided by a majority vote and processing of all N versions can be resumed.

The use of design diversity introduces "similarity" considerations about results and errors. A decision algorithm has to determine the decision result from a set of similar, but not identical, results. Similar results are defined as two or more results (correct or erroneous) that are within the range of variation (inexact voting). If there is an agreement of the majority correct outputs, the system is entered in the state of successful execution. Similar, but not identical, errors can cause the decision algorithm to produce an erroneous decision result. This means that invalid results are passed to the next segment without warning or attempt for global recovery. In that case, the system is entered in the state of undetected failure which may have catastrophic consequences. In the detected failure state a high level (global) recovery is initiated (if there is one) since no acceptable result has been produced because of the majority of distinct erroneous results. We assume that the N Version Programming is not used in situations that can have multiple distinct correct outputs (because the NVP does not recognize an output as correct if it occurs more than once). A flowchart of N Version Programming execution is presented in Fig.1.

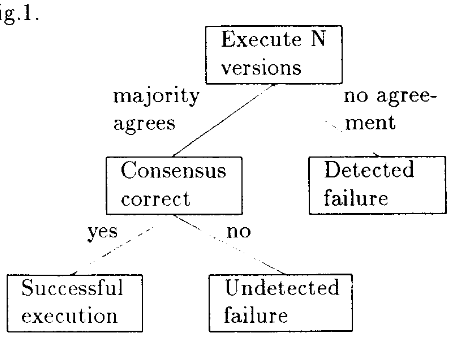


Fig.1. Flowchart of NVP execution

2 N Version Programming Reliability Model

According to widely accepted definition, reliability is the probability of fault–free operation of a system for a specified time in a specified environment. Mathematical modeling is the major approach to the assessment of the reliability in fault – tolerant computer systems. There have been a number of reliability models for N Version Programming system. In [6] a queueing model for the assessment of the average segment processing time of software fault tolerant systems is presented. The reliability model of the N Version Programming system is based on correlated errors and the time is not incorporated in the expression of the reliability. Also, a N Version Programming reliability model based on probability axioms is proposed in [5]. In the case of the independent model, it is the well known reliability of a 2–out–of–n system. The proposed dependent N Version Programming reliability model can be used to predict reliability only if the joint probabilities could be found. These models do not incorporate a time parameter. In [11] the reliability of the NVP system (three versions and a decider) is modeled asymptotically by a homogeneous Poisson process. An equivalent failure rate is derived using the departure rate and the probability of failure obtained from the embedded discrete Markov chain.

In the reliability modeling of fault – tolerant computer systems, we are interested in whether the system is in an operational or in a failure state. In our model we distinguish two failure states: undetected and detected failure. The N Version Programming failure probabilities and the reliability are modeled by means of mathematical relations on well–defined parameters (measured or derived) which characterize a configuration of the system, its various failure mechanisms and its ability to recover from failures.

We consider the execution of the N Version Programming system because software faults can manifest only when it is executed. During the execution no error compensation may take place. We made an assumption that the number of failures, in a given time period, follows a Poisson distribution (i.e. the failure intervals follow an exponential distribution) with a failure rate λ as a parameter.

This assumption guarantees that a finite state continuous time Markov chain can be used for reliability modeling of the N version system.

During the operational phase the failure rate λ is constant and it can be determinated using the maximum likelihood estimation applied to failure data [7]. Experiments in multiversion programming [3], [4] indicate a surprisingly high number of coincident failures (on the same input data) in the set of independently developed programs. In many cases, similar errors are caused by related faults, but independent faults often produce similar errors [3]. However, from the operational viewpoint, it does not matter why programs fail on the same input, it merely matters that they do. Therefore, we define c as the fraction of version failures that are similar $(0 \leq c \leq 1)$. Hence, each version failure rate λ is the sum of the distinct $(1-c)\lambda$ and the similar failure rate $c\lambda$.

Fig. 2 presents the proposed Markov reliability model for the N Version Programming system. A state is defined as a vector (i, j) where:

i is the number of versions with similar failures

j is the number of versions with distinct failures

$(0 \leq i, j \leq n)$.

The states of this Markov chain are either transient $(i+j<n)$ or absorbing $(i+j=n)$ and the total number of states is $1+n(n+1)/2$.

The Chapman–Kolmogorov equations are not easy to handle for the general case of n versions. Using theory of Lumpable Markov chains we lump equivalent states and obtain a pure death process presented in Fig. 3, whose state probabilities $P_k(t)$ are given by:

$$P_k(t) = \binom{n}{k} e^{-k\lambda t}\,(1-e^{-\lambda t})^{n-k}, \quad 0 \leq k \leq n$$

The state probabilities $P_{ij}(t)$ of the chain presented in Fig. 2 are now easily evaluated as:

$$P_{ij}(t) = \binom{i+j-1}{j} c^{i-1}(1-c)^j\, P_{n-(i+j)}(t), \quad \text{for } 1 < i \leq n,\ 0 \leq j \leq n,$$

$$P_{0j}(t) = (1-c)^{j-1}\, P_{n-j}(t), \quad \text{for } 0 < j \leq n,$$

$$P_{00}(t) = P_n(t).$$

The operational states of the N version system (according to Fig. 1) are characterized by a collection of states, each corresponding to the following operational configurations:

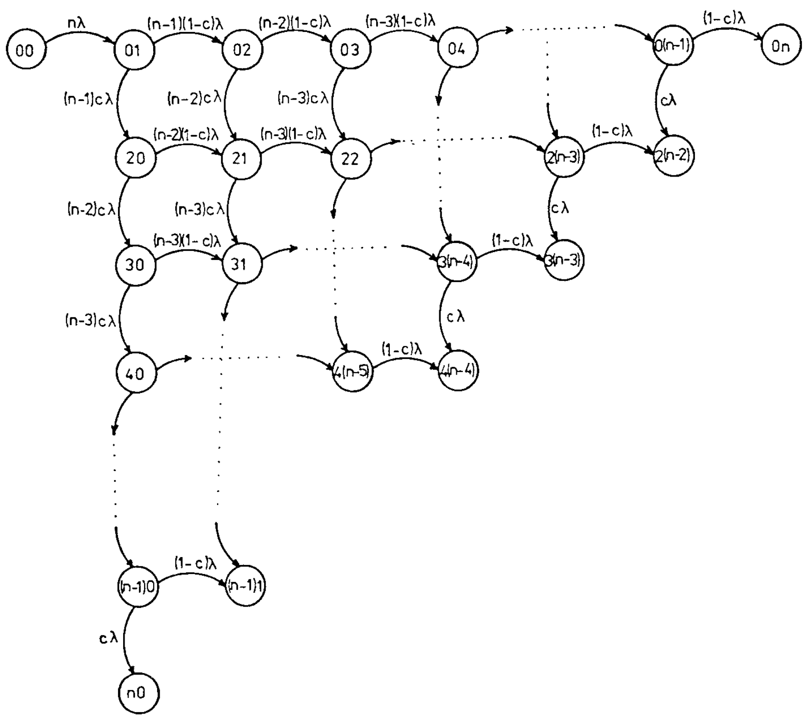


Fig.2. Continuous time Markov model of NVP System

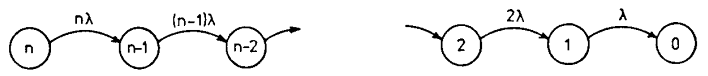


Fig.3. Equivalent Markov chain

successful execution	at least m correct outputs agree
undetected failure	at least m erroneous outputs agree
detected failure	at least m erroneous outputs disagree

where majority m is defined as $m = \lceil (n+1)/2 \rceil$.

We also define the following associated probabilities to these states:

$P_u(t)$ = Prob (successful execution state is entered)
$P_f(t)$ = Prob (undetected failure state is entered)
$P_r(t)$ = Prob (detected failure state is entered)

where t is the execution time.

Product form solutions of these probabilities are:

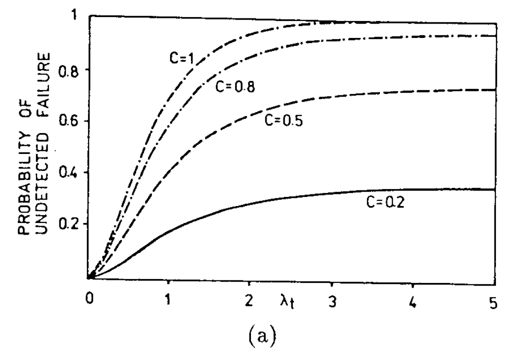
$$P_u(t) = \sum_{i=m}^{n} P_i(t)$$
$$= \sum_{i=m}^{n} (-1)^{i-m} \binom{n}{i} \binom{i-1}{m-1} e^{-i\lambda t}$$

$$P_f(t) = \sum_{i=m}^{n} \sum_{j=0}^{n-i} P_{ij}(t)$$

$$P_r(t) = 1 - P_u(t) - P_f(t)$$

If we denote the probability of successful global recovery as r, than the system reliability is given by:

$$R(t) = 1 - [\, P_f(t) + (1-r)P_r(t) \,]$$



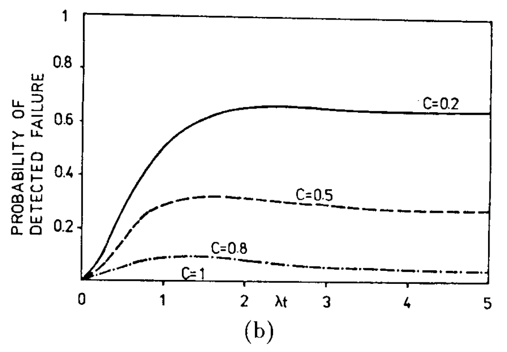


Fig. 4. Undetected (a) and detected (b) failure probability of 3VP

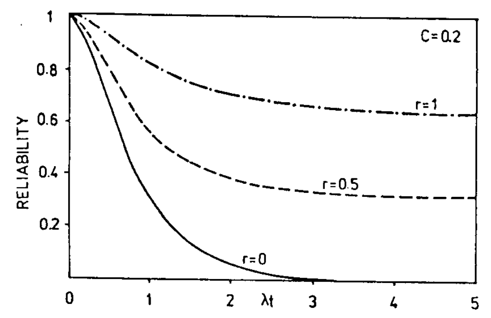
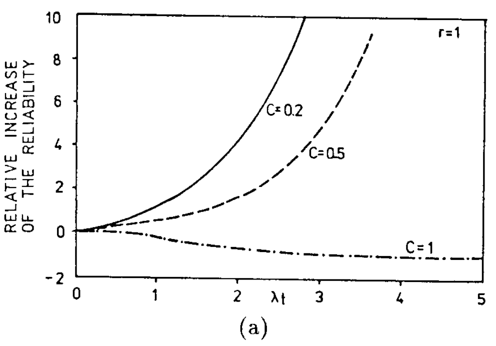


Fig. 5. Reliability of 3VP



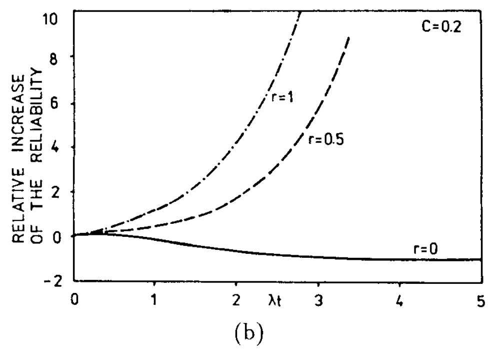


Fig. 6. Relative increase of the reliability

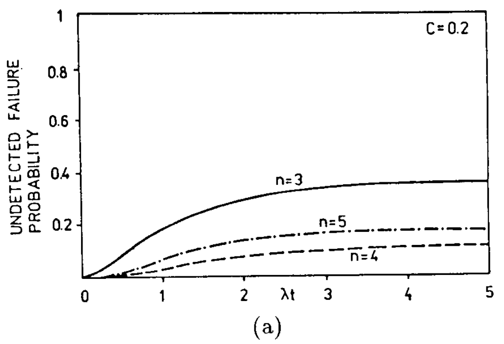
3 Reliability Evaluation

Derived time dependent failure probabilities enable us to gain a great deal of quantitative results about the influence of the execution time and various parameters on the system performances. As expected, failure probabilities increase with version failure rate λ and the execution time t. This confirms a general system reliability result i.e. the use of more reliable components leads to more reliable fault - tolerant system.

Fig. 4 (a) and 4 (b) present the probabilities of undetected and detected failure of the 3 Version Programming as a function of λt (for different values of the fraction of similar failures c). In both cases, the undetected and the detected failure probabilities are extremely sensitive to the variation of c. When c increases, the undetected failure probability increases and the detected failure probability decreases.

A comparison of the 3VP system reliability for different values of the probability of successful global recovery r is presente in Fig. 5. It can be seen that the reliability of N Version Programming increases with the parameter r. In the case of $r = 0$, the reliability is minimal and it does not depend on c. The best reliability is obtained in the case of an ideal global recovery ($r = 1$).

As indicated in Fig. 6 (a and b), the relative increase of the reliability that results from the use of the 3VP (compared to a nonfault - tolerant system) strongly depends on c and r. In the cases when $r = 0$ (for



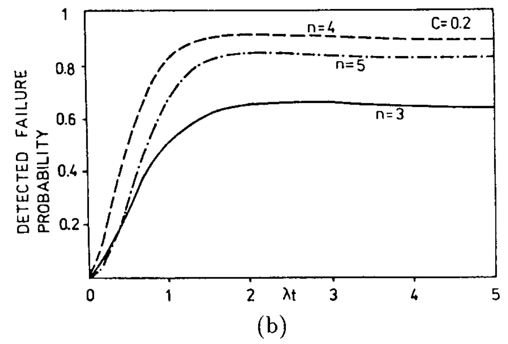
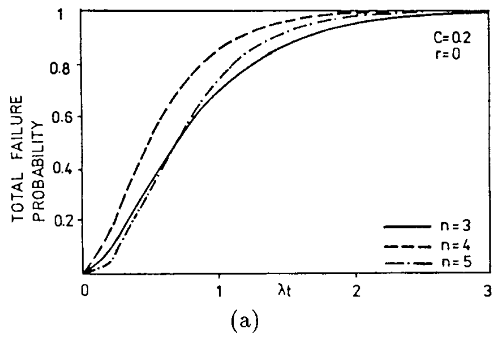


Fig. 7. Undetected (a) and detected (b) failure probability for different number of versions



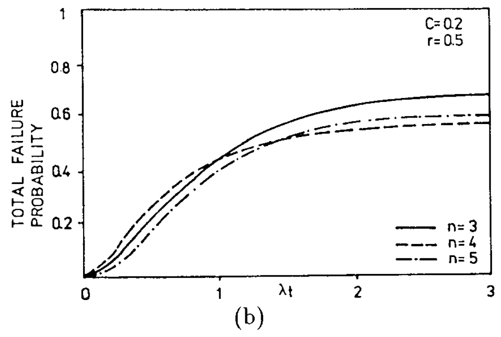


Fig. 8. Total failure probability for different n

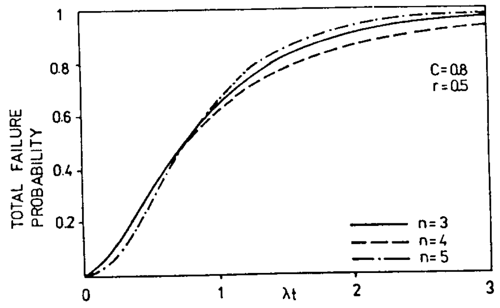


Fig. 9. Total failure probability for different number of versions

any c) and $c = 1$ (for any r), the reliability of the 3VP is equal to the probability of successful execution $P_u(t)$ and provides better performance than a single version only for $\lambda t < 0.7$. It means that if similar failures dominate ($c \approx 1$) no improvement can be achieved and this confirms the results obtained in [3], [4], [11].

Fig. 7 presents the probability of undetected (a) and detected failure (b) for different number of versions n, as a function of λt. It can be seen that the probability of undetected failure is smaller and the probability of detected failure is greater for $2k$ than $(2k+1)$ Version Programming (because of the same majority). Hence, if we are interested only in the undetected failure probability it is better to use an even number of versions. The same conclusion is obtained in [11] for the special case of two and three versions in spite of all differences of model assumptions.

Fig. 8 compares total failure probability for different number of versions n and $c = 0.2$, $r = 0$ (a) i.e. $c = 0.2$, $r = 0.5$ (b). It can be seen that if we are interested in total failure probability it is better to use odd number of versions. The reliability improvement for greater r is pointed out in Fig. 5. One can notice that for $r = 1$ (ideal global recovery) the total failure probability is equal to the undetected failure probability (analyzed above).

If similar failures dominate significantly over distinct failures ($c > 0.8$) no improvement is obtained using higher number of versions (Fig. 9).

4 Conclusion

The paper presents a new Markov model of the N Version Programming systems. The main goal in our work was to develop reliability modeling as a practical easy–to–use analysis tool for software fault – tolerant designer.

Experiments in multiversion programming have shown that, in spite of the independent version design, there are coincident failures (on the same input data) in two or more versions. Therefore, in our model, we distinguish two types of failures: similar and distinct. Consequently, there are three operational states: successful execution, detected and undetected failure.

We use continuous time Markov chain for reliability modeling of N Version Programming for the general case of n versions. The power and simplicity of the used Markov approach led to a more usable model than existing reliability models on N version systems and it allows incorporation of version execution time into the analysis. The failure probabilities i.e. the reliability are dependent not only on the system parameters but also on the period of version execution time. The established mathematical relations enable us to make dynamic analysis and to gain a great deal of quantitative results from the model. The presented model of the N Version Programming could be used both to gain a deeper understanding of system behavior and to guide a design of actual systems.

Finally, it is worth noting that we focus our future work on the consideration of the faults in the decider and the specific fault treatment mechanisms in our model.

References

[1] T.Anderson, P.A.Lee, "FAULT TOLERANCE – Principles and Practice", *Prentice/Hall International*, 1981.

[2] Liming Chen, "Improving Software Reliability by N Version Programming", Ph.D. Thesis, *Department of Computer Science, UCLA*, Los Angeles, 1978.

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

[4] J.Knight, N.Leveson, "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming", *IEEE Transactions of Software Engineering*, Vol.SE-12, No.1, pp. 96 - 109, 1986.

[5] R.Scott, J.Gault, D.McAllister, "Fault Tolerant Software Reliability Modeling", *IEEE Transactions of Software Engineering*, Vol.SE-13, No.5, pp. 582 - 592, May 1987.

[6] A.Grnarov, J.Arlat, A.Avizienis, "On the Performance of Software Fault – Tolerance Strategies", in *Proc. FTCS-10*, Kyoto, Japan, Oct. 1980, pp. 251 - 253.

[7] J.D.Musa, A.Iannino, K.Okumoto, "Software Reliability, Measurement, Prediction, Application" *Mc Graw-Hill*, 1987.

[8] M.Trachtenberg, "The Linear Software Reliability Model and Uniform Testing", *IEEE Transactions on Reliability*, Vol.R-34, No.1, pp. 8 - 16, April 1985.

[9] B. Gnedenko, The Theory of Probability, *Nauka*, Moscow, 1988.

[10] K. Goševa – Popstojanova, "Software Fault – Tolerant Performance Modeling", M.Sc. Thesis, *Faculty of Electrical Engineering, University of Skopje*, Yugoslavia, April 1990.

[11] J.Arlat, K.Kanoun, J.C.Laprie, "Dependability Modeling and Evaluation of Software Fault-Tolerant Systems", *IEEE Transaction on Computers*, Vol. 39, No. 4, pp. 504 - 513, April 1990.