Proceedings of the 2005 International Conference on Simulation and Modeling
V. Kachitvichyanukul, U. Purintrapiban, P. Utayopas, eds.

HIERARCHICAL AND HYBRID FAULT-TOLERANT SOFTWARE SYSTEM DESIGN

Denvit Methanavyn and Naruemon Wattanapongsakorn

Department of Computer Engineering
Faculty of Engineering
Mongkut's University of Technology Thonburi
91 Suksawasd 48, Bangmod, Tung Kru, Bangkok, 10140, Thailand

ABSTRACT

Reliability enhancement in software system is a crucial and challenging issue. Applying efficient fault-tolerant mechanism can fulfill the system reliability requirement. This paper proposes reliability models for hierarchical and hybrid fault-tolerant software systems considering failure dependencies or related faults in software components/versions. Our system models are based on the classical Recovery Block (RB) and N-Version Programming (NVP) schemes which are the most commonly used software fault-tolerant architectures. Our software system reliability models are generalized for higher degree of fault tolerance when higher number of software versions is provided. We perform reliability evaluation and comparison of these reliability models together with the classical models of RBs and NVPs.

1 INTRODUCTION

Many critical systems, such as power plant control, flight control, transportation and military systems need high reliability. To achieve system reliability and safety requirement, very often fault-tolerant techniques are the only choice to be selected. Several techniques have been proposed for structuring hardware and software systems to provide fault tolerance. Software fault tolerance usually requires design diversity and decision algorithm considering software modules and an adjudicator.

The first developed software fault tolerant technique is Recovery Block (RB), and then is N-Version Programming (NVP). These classical fault-tolerant techniques have some differences in term of judging results (adjudicator) to be final output. For RB, the adjudicator is called an acceptance tester, which acts as a computational module that evaluates results of all software modules, so the tester needs to be specially design for the task. For NVP, the adjudicator is called voter, which acts as a comparator of all software modules and choose the majority results as output, so the voter selects the correct output using majority voting mechanism.

These two classical models can be combined to create hybrid fault-tolerant techniques such as Consensus Recovery Block (CRB), N-Self Checking Programming (NSCP) and Acceptance Voting (Lyu 1996) to provide system fault tolerance in extension to the original RB and NVP schemes.

Faults can be classified into two types: s-independent faults and dependent (or related) faults. S-independent faults usually cause distinct errors and make separated failures. The latter is related faults which cause correlated failures. Examples of related faults are faults in the design specification, common fault in the design and implementation phases. Related (dependent) fault lead to common-mode failure of multiple software modules.

Many researchers have proposed various mathematical reliability models for RB and NVP. In previous work, Randell 1975 developed Recovery Block scheme which is the software fault-tolerant model. N-Version Programming (NVP) was developed in 1985 by Avizienis who directly applied the hardware N-Modular Redundancy to software architecture. In 1988, Arlat *et al*, proposed reliability modeling and evaluation of the Recovery block and N-Version Programming with can tolerate only a single fault. Later in 1993, Dugan *et al* applied this model and proposed a quantitative comparison of RB and NVP schemes considering related faults such as probability of failure between two software modules and among all software modules. In 1996 Wu *et al* proposed hybrid software fault-tolerant models which nested RB with NVP and embedded RB within NVP. They provided system reliability comparison for these architectures without considering related faults. In 1997, Giandomenico *et al* evaluated schemes for handling dependability and efficiency of Self-Configuring Optimal Programming (SCOP) scheme which accepted consistent result (if inconsistent result occurs,

SCOP activates additional module to compare results for consistency), NVP with tie-break (enhancing performance of basic NVP by activating additional module to find an agreement in results) and NVP schemes. Xu *et al*, 1997, proposed new architecture called t/(n-1) variant programming and compare it with RB and NVP architectures considering related faults. In 1999, Berman *et al* developed reliability model for RB considering only s-independent faults. In 2002, Leblance *et al* proposed a simple approach to estimate the reliability of software system that composed of a hierarchical of modules with s-independent of software failure assumption. In summary, there are many literatures on new fault-tolerant software architectures developing as well as software system reliability analysis and optimization. However, none of them provide reliability evaluation of hierarchical or hybrid software systems considering related faults or failure dependencies in the software modules.

In this research, we extend the work of Wu *et al* 1996 considering failure dependencies in software system reliability analysis using sum-of-disjoint products. We consider hierarchical fault-tolerant schemes of multi-level of RBs, multi-level of NVPs and hybrid RB–NVP. We also propose mathematical formulations to find system reliability for these schemes. These system reliabilities are evaluated and compared with those of the traditional RB and NVP models.

Assumptions and Notations used throughout this paper are as follows.

Assumptions:

1. Each software module has 2 states: functional or fail. There is no failure repair for each module or system.
2. Reliability of each software module is known.
3. All software modules have the same reliability, $Pv_i = Pv_j$
4. Related fault between any two software modules have same probability of failure, i.e. $P_{RVij} = P_{RVkl}$
5. Related faults among three or more software modules are assumed to be negligible. However, related faults due to error in the software design specification do exist with probability P_{RALL}.
6. Related fault between adjudicator and software module(s) are not existed.

Notations:

P_V	Probability of failure of each software modules.
Q_V	Reliability of each software modules; $Q_V = 1 - P_V$
P_{RV}	Probability of failure from related fault between two software modules; $Q_{RV} = 1 - P_{RV}$
P_{RALL}	Probability of failure from related fault among all software modules; $Q_{RALL} = 1 - P_{RALL}$
P_D	Probability of failure of an adjudicator (tester or voter); $Q_D = 1 - P_D$
$P_{DEP}(X)$	Probability of failure of system considering related faults (dependent); $Q_{DEP}(X) = 1 - P_{DEP}(X)$
RB_N	Recovery Block with N modules
NVP_N	N-Version Programming

2 RESEARCH BACKGROUND

2.1 Fault Tolerant Techniques

Software Fault Tolerance usually requires design diversity. For design-diversity, two or more software modules are designed to meet a common service specification and provide for redundant computations. The modules are aimed at delivering the same service, but independently implemented. Since at least two modules are involved, an adjudicator is used to determine an error-free result based on the results produced by multiple software modules. Several techniques have been proposed for structuring FT software system. Two common techniques are discussed as follows.

1. Recovery Block (RB) (Randell B., 1975) is the first scheme developed for achieving software fault tolerance. Modules are organized in a manner similar to a standby sparing used in hardware. RB performs a run-time fault detection by augmenting any conventional hardware/software error detection mechanism with an acceptance test applied to the results of execution of a primary software module. If the test fails, an alternate module is invoked after backward error recovery is performed. Figure 1(a) presents the model of RB with three software modules running sequentially on a hardware module. Figure 1(b) presents the functioning sequence of the RB starting at the primary module (M_1) where its result is tested with tester (T) to find an correct result. If the result is not correct, the result from the second module (M_2) will be produced and tested next. The result that passes the test is considered the output of the RB module.

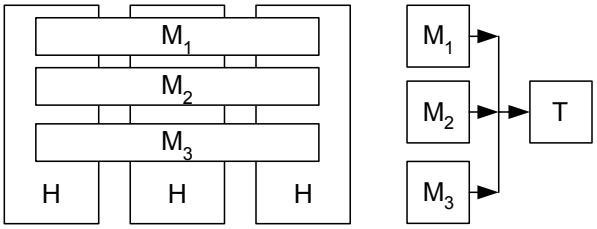

Figure. 1(a) Figure. 1(b)

Figure 1: Recovery Block Scheme: RB_3

2. N-Version Programming (NVP) (Avizienis, 1985), which directly applied the hardware N-Modular Redundancy to software architecture. N versions (modules) of a program are executed in parallel and their results are compared by an adjudicator called a voter. By incorporating a majority vote, the system can eliminate erroneous results and pass on the presumed-correct results. Figure 2(a) presents the model of NVP3 with three software modules running parallel on a hardware module. Figure 2(b) shows the functioning sequence of the NVP_3 where all software modules (M_1, M_2 and M_3) produce their results and find agreement by voter (V).

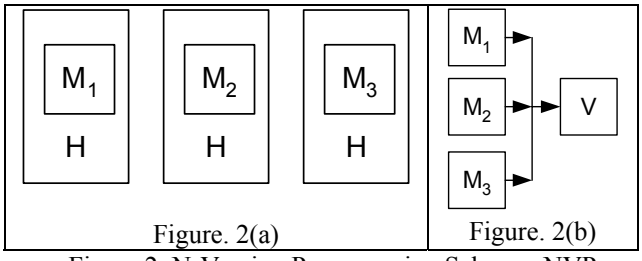

Figure. 2(a) Figure. 2(b)

Figure 2: N-Version Programming Scheme: NVP_3

2.2 Reliability Analysis

In this section, we present reliability analysis models for RB_N and NVP_N schemes when all failures are assumed s-independent, and in another assumption where there exist related faults in the software variants including faults between and among all software variants which are P_{RV} and P_{RALL}, respectively.

2.2.1 Considering S-Independent Faults

The probability of failure of the RB scheme (P_{RB}) with s-independent faults defined by Berman et al 1999 is as follows.

$$P(RB_N) = 1 - R_N ; R_N = P(Y_1) + \sum_{i=2}^{N} \left[\prod_{k=2}^{i-1} P(X_k) \right] P(Y_i)$$

$$P(Y_i) = (1 - P_{Vi})(1 - P_D),$$

$$P(X_i) = P_{V(i-1)}(1 - P_D) + (1 - P_{V(i-1)})P_D$$

(1)

$P(Y_i)$ is probability of tester accepts corrects result while $P(X_i)$ is probability that the tester rejects correct output or tester accepts incorrect result. N is the number of software modules.

The probability of failure of the NVP scheme (P_{NVP}) defined by Wu et al 1996 is as follows.

$$R(N,K) = \sum_{i=K}^{N} C_K^N Q_V^i P_V^{N-i} - P_D \quad where \quad K = (N+1)/2$$

$$P(NVP_N) = 1 - R(N,K)$$

(2)

The reliability of NVP with use majority voting as decision algorithm can find by use summation of reliability of function modules multiply with probability of failure in failure modules. By select K from N software modules then increase K by 1 until K equal N, where K is an odd number.

2.2.2 Considering Related faults

The probability of failures of RB and NVP schemes considering related faults defined by Dugan 1993 are presented in sum-of-disjoint products forms which were prior modified and improved by Veeraraghavan in 1990. The probability of failure from related fault in all software versions (P_{RALL}), the probability of failure of decider (P_D), the probability of failure from related fault between two software versions (P_{RV}) and the probability of failure of a software version (P_V) are considered toward system reliability of RB and NVP schemes.

The probability of failure of the RB scheme with two software modules is as follows.

$$P(RB_2) = P_{RALL} + P_D Q_{RALL} + P_{RV} Q_{RALL} Q_D + P_V^2 Q_{RALL} Q_D Q_{RV} \quad (3)$$

The probability of failure of the NVP scheme with three software modules is as follows.

$$P(NVP_3) = P_{RALL} + P_D Q_{RALL} + P_{RV} Q_{RALL} Q_D (1 + Q_{RV} + Q_{RV}^2) + P_V^2 Q_{RALL} Q_D Q_{RV}^3 (1 + 2Q_V) \quad (4)$$

2.3 Sum of Disjoint Products

A sum-of-disjoint-products (J.B. Dugan 2001) is an alternative method to present a union of sets. By considering sum of the probabilities of individual failure events yields the exact unreliability (i.e. probability of failure) of the system. The system unreliability is $\Pr ob[\cup_{i=1}^{p} C_i]$. A sum-of-disjoint-products method is represented in the following form.

$$\bigcup_{i=1}^{p} C_i = (C_1) \bigcup (\overline{C}_1 C_2) \bigcup (\overline{C}_1 \overline{C}_2 C_3) \bigcup \ldots \bigcup (\overline{C}_1 \overline{C}_2 \overline{C}_3 \ldots \overline{C}_{p-1} C_p)$$
$$= C_1 + \overline{C}_1 C_2 + \overline{C}_1 \overline{C}_2 C_3 + \ldots + \overline{C}_1 \overline{C}_2 \overline{C}_3 \ldots \overline{C}_{p-1} C_p \quad (5)$$

where $\overline{C}_i$ is the part of the universal set that is not in C_i, i.e the negation of C_i. Each term in the right-hand-side of the expression in equation 5 is a disjoint product, where all the terms are mutually exclusive.

3 RELIABILITY MODELS OF SOFTWARE FAULT TOLERANT SYSTEMS WITH FAILURE DEPENDANCIES

In this section, we proposed mathematical models to find system reliability of RB and NVP schemes considering related faults.

3.1 Formulation by Sum of Disjoint Products

With RB_3 scheme, causes of system failure are

1. Failure of all software versions due to fault from the software specification i.e. P_{RALL}.
2. Failure of the adjudicator i.e. P_D.
3. S-independent failure of a software version and related fault in the remaining two software versions. There are three possible distinct events i.e. $P_{V1}P_{RV23}$, $P_{V2}P_{RV13}$ and $P_{V3}P_{RV12}$.
4. S-independent failure of three software versions i.e. $P_{V1} P_{V2} P_{V3} = P_V^3$.

Thus, the probability of failure of the RB scheme with three software modules is as follows.

$$\begin{aligned} P(RB_3) = {} & P_{RALL} + P_D Q_{RALL} + P_V P_{RV} Q_{RALL} Q_D + \\ & P_V P_{RV} Q_{RALL} Q_D (1-P_{RV} P_V) + \\ & P_V P_{RV} Q_{RALL} Q_D (1-P_{RV} P_V)^2 + \\ & P^3{}_V Q_{RALL} Q_D Q^3{}_{RV} \end{aligned} \quad (6)$$

With NVP_3 scheme, causes of system failure are

1. Failure of all software versions due to fault from the software specification i.e. P_{RALL}.
2. Failure of the adjudicator i.e. P_D.
3. Related fault in any two software versions. There are three possible distinct events i.e. P_{RV23}, P_{RV13} and P_{RV12}.
4. S-independent failure of two software versions. There are four possible distinct events i.e. , $P_{V1} P_{V2}$,$P_{V1}P_{V3}$ and $P_{V2}P_{V3}$

Thus, the probability of failure of the NVP scheme with three software modules is mathematically presented.

$$\begin{aligned} P(NVP_3) = {} & P_{RALL} + P_D Q_{RALL} + \\ & P_{RV} Q_{RALL} Q_D + \\ & P_{RV} Q_{RALL} Q_D Q_{RV} + \\ & P_{RV} Q_{RALL} Q_D Q^2{}_{RV} + \\ & P^2{}_V Q_{RALL} Q_D Q^3{}_{RV} + \\ & P^2{}_V Q_{RALL} Q_D Q^3{}_{RV} Q_V + \\ & P^2{}_V Q_{RALL} Q_D Q^3{}_{RV} Q_V \end{aligned} \quad (7)$$

3.2 Generalized Reliability Models

The reliability model formulations represented with sum of disjoint products are quite complex when the number of software modules, N, is large. So we apply a mathematical induction technique to simplify the probability of failure (i.e. 1- reliability) of RB_N module resulted as following.

$$\begin{aligned} P(RB_N) = {} & P_{RALL} + P_D Q_{RALL} + \\ & m P_{RV} P_V{}^{(N-2)} Q_{RALL} Q_D + \\ & P_V^N Q_{RALL} Q_D Q_{RV}^m \end{aligned}$$
$$where \quad m = C_2^N = \frac{N!}{2!(N-2)!}$$
$$and \quad N = 2,3,4,\ldots \quad (8)$$

From equation 8, each term is represented as a disjoint product. There are two common terms P_{RALL} and P_D that always exist at any value of N. The probability of failures of RB_2 and RB_3 can be obtained by substituting N =2 and 3, respectively.

In similar way, with the mathematical induction method we can simplify the probability of failure or unreliability of NVP_N module as following.

$$P(NVP_N) = P_{RALL} + P_D Q_{RALL} + P_{RV} P_V^{(K-2)} Q_{RALL} Q_D S + P_V^K Q_{RALL} Q_D Q_{RV}^M T$$

$$\text{where} \quad S = M \sum_{i=1}^{K} Q_V^{i-1} \{ \sum_{Y_1=1}^{i} \sum_{Y_2=1}^{Y_1} \dots \sum_{Y_{K-4}=1}^{Y_{K-3}} Y_{K-4} \} \quad \text{when} \quad N > 7$$

$$= M \sum_{i=1}^{K} i Q_V^{i-1} \quad \text{when} \quad N = 7$$

$$= M(1 + Q_V + Q_V^2) \quad \text{when} \quad N = 5$$

$$= 1 + Q_{RV} + Q_{RV}^2 \quad \text{when} \quad N = 3$$

$$T = \sum_{i=1}^{K} Q_V^{i-1} \{ \sum_{Y_1=1}^{i} \sum_{Y_2=1}^{Y_1} \dots \sum_{Y_{K-2}=1}^{Y_{K-1}} Y_{K-2} \} \quad \text{when} \quad N > 5$$

$$= \sum_{i=1}^{K} i Q_V^{i-1} \quad \text{when} \quad N = 5$$

$$= 1 + 2Q_V \quad \text{when} \quad N = 3$$

$$M = C_2^N = \frac{N!}{2!(N-2)!} \qquad N = 3,5,7,9...$$

$$K = (N+1)/2 \tag{9}$$

From equation 9, similar to the $P(RB_N)$ model, there are also two common terms P_{RALL} and P_D that always exist at any value of N. The probability of failure of NVP_3 can be obtained by substituting N = 3, respectively.

Detailed information of the derived formulation can be obtained from our special project study (D. Methanavyn et al, 2004).

4 RELIABILITY MODELS OF HIERARCHICAL FAULT -TOLERANT SOFTWARE SYSTEMS

Hierarchical fault-tolerant software system consists of multi-level of fault-tolerant modules. At the lower level, RB or NVP modules are used. Each output from the lower-level modules will be sent to the upper-level module to perform similar process again and then the final output is released.

The probability of failure of the hierarchical fault-tolerant system can be considered in two parts. The first part is from the lower-level modules considering related faults. The latter is from the upper-level module where related faults among the lower-level fault-tolerant modules are assumed negligible. Hence, s-independent assumption is applied at this upper-level. The probability of failure of each lower-level fault-tolerant module is applied as the failure input used in the upper-level reliability analysis.

4.1 Hierarchical Fault-Tolerant Models

4.1.1 RB_iRB_j

$RBiRB_j$ consists of *i* lower-level RB modules each consisting of *j* software modules and a tester, and one upper-level RB module which uses *i* outputs from the lower-level to test for the final output. Example of RB_2RB_3 is shown in Figure 3.

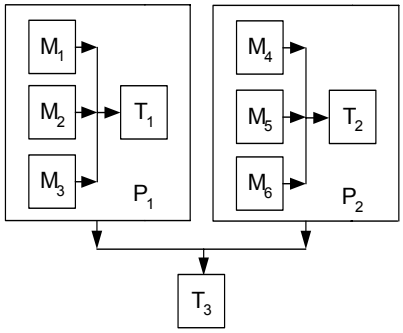

Figure 3: RB_2RB_3

4.1.2 $NVPiRB_j$

NVP_iRB_j consists of *i* lower-level RB modules each consisting of *j* software modules and a tester, and one upper level NVP module which uses i outputs from the lower-level to vote for the final output. Example of NVP_3RB_2 is shown in Figure 4.

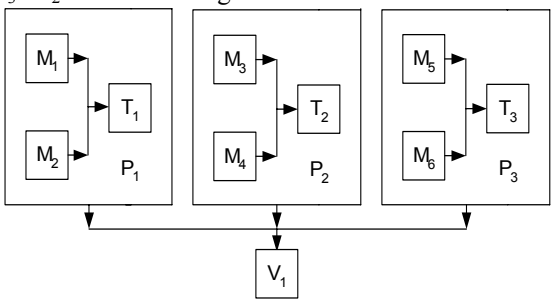

Figure 4: NVP_3RB_2

4.1.3 RB_iNVP_j

RB_iNVP_j consists of *i* lower-level NVP modules each consisting of *j* software modules and a voter, and one upper level RB module which uses i outputs from the lower-level to test for the final output. Example of NVP_3RB_2 is shown in Figure 5.

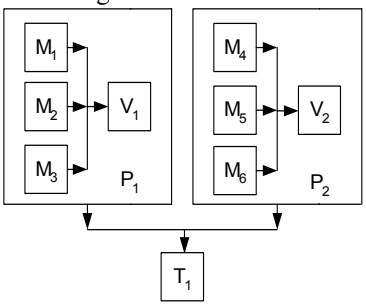

Figure 5: RB_2NVP_3

4.1.4 NVP_iNVP_j

NVP_iNVP_j consists of *i* lower-level NVP modules each consisting of *j* software modules and a voter, and one upper level NVP module which uses i outputs from the lower-level to vote for the final output. Example of NVP_3NVP_3 is shown in Figure 6.

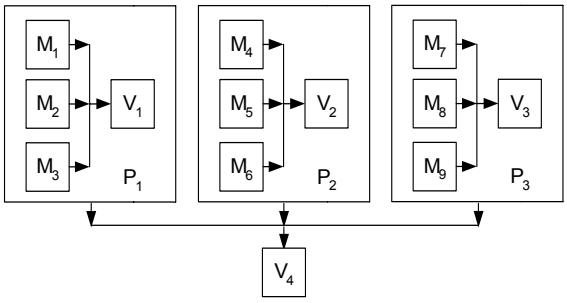

Figure 6: NVP_3NVP_3

4.2 Proposed Reliability Analysis Models

The probability of failure of the hierarchical and hybrid fault-tolerant systems presented in the previous section can be obtained by finding probability of failure of the lower-level modules using equations (8) and (9), where related faults are considered. For the upper-level modules, we use equations (1) and (2), where related faults across the lower-level modules are assumed negligible, to analyze the system reliability.

Therefore, the probability of failures of RB_2RB_3, RB_3RB_2, RB_2NVP_3, NVP_3RB_2, and NVP_3NVP_3 schemes are presented in equations 10-14, respectively.

$$P(RB_2RB_3) = 1 - \{Q_{DEP}(RB_3) \times Q_D + [P_{DEP}(RB_3) \times Q_D + Q_{DEP}(RB_3) \times P_D] \times Q_{DEP}(RB_3) \times (Q_D)\} \quad (10)$$

$$P(RB_3RB_2) = 1 - \{Q_{DEP}(RB_2) \times Q_D + \{[P_{DEP}(RB_2) \times Q_D + Q_{DEP}(RB_2) \times P_D] + [P_{DEP}(RB2) \times Q_D + Q_{DEP}(RB_2) \times P_D]^2\} \times Q_{DEP}(RB_2) \times Q_D\} \quad (11)$$

$$P(RB_2NVP_3) = 1 - \{Q_{DEP}(NVP_3) \times Q_D + [P_{DEP}(NVP_3) \times Q_D + Q_{DEP}(NVP_3) \times P_D] \times Q_{DEP}(NVP_3) \times Q_D\} \quad (12)$$

$$P(NVP_3RB_2) = (P_{DEP}(RB_2))^3 + 3(P_{DEP}(RB_2))^2 \times (1- P_{DEP}(RB_2)+ P_D \quad (13)$$

$$P(NVP_3NVP_3) = (P_{DEP}(NVP_3))^3 + 3(P_{DEP}(NVP_3))^2 \times (1- P_{DEP}(NVP_3)+ P_D \quad (14)$$

5 EXPERIMENTAL RESULTS

The following example illustrates and evaluates our proposed generalized models for RBs, NVPs, hybrid and hierarchical reliability models comparing with the classical reliability models, shown in equations 1 and 2. Table 1 presents an input dataset which is the probability of failures from related faults, and from s-independent fault in a software version, P_V.

Table 1: Input data: probability of failures

P_{RALL}	P_D	P_{RV}	P_V
0.000003	0.0001	0.000374	0.0958

First, we evaluate our generalized reliability models RB_N and NVP_N shown in equations 8 and 9, where failure dependencies or related faults in the software versions are considered, by comparing with the reliability models expressed in equations 1 and 2, where s-independent failure assumption is applied .

Figure 7 presents reliabilities of RB and NVP modules with various number of software versions i.e. 3 up to 9. The terms DEP and IND represents dependent failure, and s-independent failure assumptions, respectively. As expected, when the number of software version increases, or in other words, more faults can be tolerated, the system reliability increases. We can see a great difference in reliability values when N = 3, 5, 7 and 9. In addition, the results agree with those in the literature showing that the RB module offers higher reliability than those of the NVP module when the same number of software versions is provided. When the number of software versions is large, the probability of system failure is getting smaller. In NVP, the probability of two modules producing an agreed result is higher when the number of software modules is larger. At this point, the NVP system is quite comparable with the RB system.

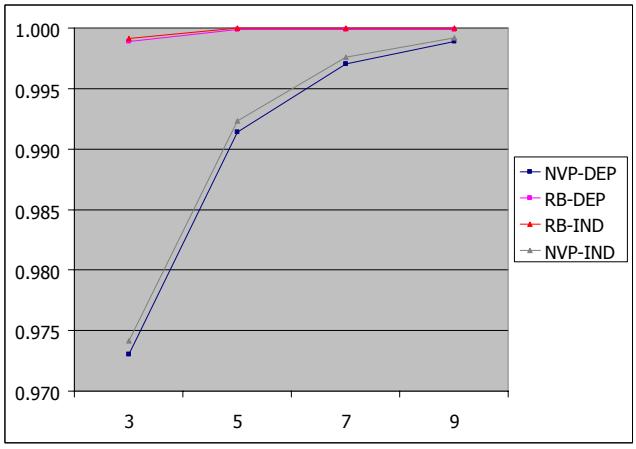

Figure 7: RB and NVP Reliability Trend

In the next step, we consider software systems with various fault-tolerant architectures and a number of software versions. When N, which is the number of software version, is equal to 1, no fault-tolerance can be applied. When N is equal to 3, RB and NVP schemes are considered. Reliability evaluation and comparison when N = 1 and 3 are shown in Table 2. As expected, RB_3 offers higher reliability than the NVP_3's.

Table 2: Reliability Analysis (N = 1 and 3)

#	N	Model	Unreliability	Reliability-Rank
1	1	V	9.5800E-02	-
2	3	RB_3	1.0890E-03	1
3	3	NVP_3	2.6997E-02	2

Then, we consider N = 6, where several fault-tolerant schemes can be applied including classical models (RB6), hierarchical models (RB_2RB_3, RB_3RB_2) and hybrid models (NVP_3RB_2, RB_2NVP_3). The corresponding reliability evaluation and comparison are shown in Table 3.

Table 3: Reliability Analysis (N = 6)

#	Model	Unreliability	Reliability-Rank
1	RB_6	1.0424E-04	3
2	RB_2RB_3	1.5213E-06	1
3	RB_3RB_2	1.9009E-06	2
4	NVP_3RB_2	3.7758E-04	4
5	RB_2NVP_3	7.3512E-04	5

Similarly, we also consider N = 9, where several reliability models are captured. The corresponding reliability evaluation and comparison are shown in Table 4.

Table 4: Reliability Analysis (N = 9)

#	Model	Unreliability	Reliability-Rank
1	RB_9	1.0300E-04	3
2	NVP_9	1.1312E-03	5
3	RB_3RB_3	1.1067E-07	1
4	NVP_3NVP_3	2.2425E-03	6
5	RB_3NVP_3	2.2591E-05	2
6	NVP_3RB_3	1.0355E-04	4

Figures 8 and 9 graphically present reliability comparison of classical, hierarchical and hybrid software fault tolerant systems when the number of software modules equals to six and nine, respectively.

When considering the reliability of software fault-tolerant systems with N = 6, shown in Table 3 and Figure 8, the model with the highest reliability value is RB2RB3 which a hierarchical RB. This result is also the same when N = 9, where RB3RB3 is the best. The next best belongs to hybrid and classical fault-tolerant software models, while NVP hierarchical models offer the lowest reliability values. With hierarchical and hybrid architectures, an important factor to gain high system reliability is the reliability of the low-level modules. With high reliability of the lower-level modules, the over-all system reliability can be enhanced. This can be explained by considering NVP3RB2 and RB2NVP3 models where RB is the lower-level modules for the first model and the NVP is for the later model. With RB, the first model has higher reliability at the lower-level than those of the later model, resulting to higher reliability of the overall NVP3RB2 model. With the same reason, NVP3RB3 model offers higher reliability than those of the RB3NVP3, as shown in Table 4 and Figure 9.

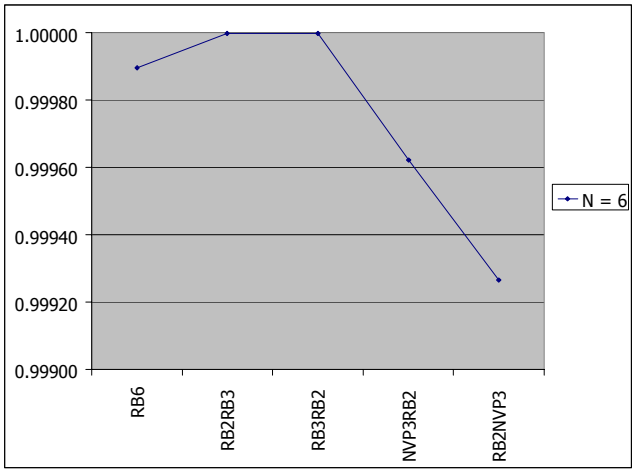

Figure 8: Reliability Comparison When N = 6

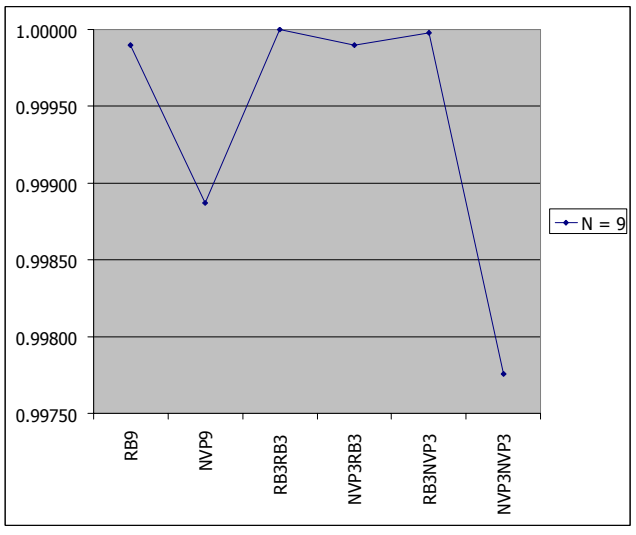

Figure 9: Reliability Comparison When N = 9

Another main factor for high system reliability is the reliability of the higher-level module. For example, let's consider RB3RB3 and NVP3RB3 models which both have RB scheme for the lower-level modules. The first model also has RB scheme for the higher-level module, resulting to higher system reliability than those of the counter part where NVP scheme is applied for its higher-level module.

6 CONCLUSION

In this research, we proposed mathematical models to find probability of failures of RB and NVP schemes considering failure dependencies or related faults in software versions. These models are generalized for any value of N, which is the number of software versions used in the schemes.

In addition, essentially we proposed reliability models for hierarchical and hybrid fault-tolerant software architectures consisting of multi-level RBs, multi-level NVPs, or combinations of RBs and NVPs. We perform reliability evaluation and comparison of these reliability models together with the classical model RB_N and NVP_N.

Our results indicate that hierarchical RB models provide higher reliability than those of the classical models and the hybrid model, while hierarchical NVP models offer lower reliability. These results agree with the literature that RB scheme provides higher reliability than those of the NVP scheme, given the same number of software versions.

REFERENCES

Arlat, J., K. Kanoun, and J.C. Laprie. 1988. Dependability Evaluation of Software Fault-Tolerance. *18th International Symposium on Fault tolerant Computing*: 142-177.

Avizienis, A. 1985. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering Vol.* 12: 1491-1501.

Berman, O., and U.D. Kumar. 1999. Optimization models for recovery block schemes. *European Journal of Operational Research Vol.* 115 : 368-379.

Dugan, J.B. 2001. Fault-Tree Analysis of Computer-Based Systems. In *Proceedings of Annual Reliability and Maintainability Symposium* Tutorial Notes.

Dugan, J.B., and F.A. Patterson-Hine. 1993. Simple Models of Fault Tolerant Software. In *Proceedings Annual Reliability and Maintainability Symposium* : 354-359.

Dugan, J.B., and M.R. Lyu. 1993. System Reliability Analysis of an N-version Programming Application.. *IEEE Transactions on Software Engineering Vol.* SE3: 103-111.

Giandomenico, F.D., A. Bondavalli, J. Xu, and S. Chiaradonna. 1997. Hardware and Software Fault Tolerance: Definition and Evaluation of Adaptive Architectures in A Distributed Computing Environment. In *Proceedings of ESREL 97 Int. Conference on Safety and Reliability* : 341-348.

LeBlance, S.P. and P.A. Roman. 2002. Reliability Estimation of Hierarchical of software system. In *Proceedings of Annual Reliability and Maintainability Symposium.*: 249-253.

Lyu, M.R. 1996. Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGrawHill, Chapter 14.

Methanavyn, D. and N. Wattanapongsakorn. 2004. Hybrid software fault tolerance system design. A special project studies.

Randell, B. 1975. System structure for software fault tolerance. *IEEE Transactions on Software Engineering Vol.* SE-1: 220-231.

Veeraraghavan, M. and K.S. Trivedi. 1989. An Improved Algorithm for the Symbolic Reliability Analysis of Networks. In *Proceedings 9th Symposium on Reliable Distributed Systems*: 34-43.

Wu, J., E.B. Fernandez, and M. Zhang. 1996. Design and Modeling of Hybrid Fault-Tolerant Software With Cost Constraints. *J. System Software Vol.* 35: 141-149.

Xu, J., and B. Randell. 1997. t/(n-1)- variant Programming. *IEEE Transactions on Reliability Vol.* 46: 60-68.

AUTHOR BIOGRAPHIES

DENVIT METHANAVYN is a graduate student in Computer Engineering at King Mongkut's University of Technology Thonburi, Thailand. He received a B. Eng. Degree from Chiang Mai University in 2000. His research interests include fault tolerant system design and reliability engineering. His email address is <denvitbox@yahoo.com>

NARUEMON WATTANAPONGSAKORN is an assistant professor in Computer Engineering at King Mongkut's University of Technology Thonburi. She received the B.S. degree and the M.S. degree, both from The George Washington University, and Ph.D. degree in Electrical Engineering from the University of Pittsburgh, USA. Her research interests include fault tolerant technique, reliability engineering, network design and optimization technique. She is a member of IEEE and IEEE Reliability Society. Her email address is <naruemon@cpe.kmutt.ac.th>