

**DESIGN AND MODELING OF
HYBRID SOFTWARE FAULT-TOLERANT SYSTEMS**

by
Man-Xia Zhang

A Thesis Submitted to the Faculty of the
College of Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Florida Atlantic University

Boca Raton, Florida

April 1992

DESIGN AND MODELING OF HYBRID SOFTWARE FAULT-TOLERANT SYSTEMS

by

Man-xia (Maria) Zhang

This thesis was prepared under the direction of the candidate's thesis advisor Dr. Jie Wu, Department of Computer Science and Engineering and has been approved by the members of her supervisory committee. It was submitted to the faculty of the College of Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering.

SUPERVISORY COMMITTEE:



Thesis Advisor
Dr. Jie Wu



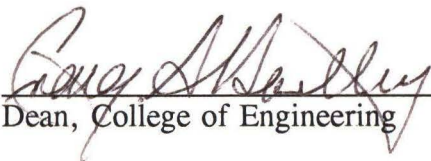
Dr. Eduardo B. Fernandez



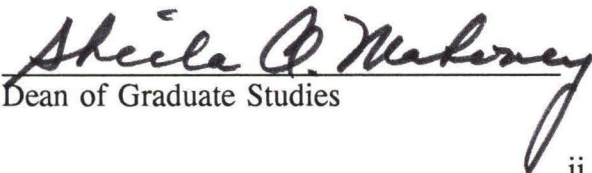
Dr. Imadeldin O. Mahgoub



Chairperson, Department of Computer
Science and Engineering



Dean, College of Engineering



Dean of Graduate Studies

23 January, 1992

Date

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr. Jie Wu, for his constant support and guidance through this research. I am also thankful to my supervisory committee members Professors Eduardo B. Fernandez and Imadeldin O. Mahgoub who have contributed to my graduate study and who spent countless hours reviewing and correcting the thesis.

I also like to say thanks to my parents Professor Bao-An Zhang and Professor Dai Jin for the love and support they have always given to me.

Finally, I want to thank my dear husband Ching-Ping Han and my dear son George Han, for their support, understanding and love.

ABSTRACT

Author: Man-xia (Maria) Zhang
Title: Design and Modeing of Hybrid Software Fault-Tolerant Systems
Institution: Florida Atlantic University
Thesis Advisor: Jie Wu
Degree: Master of Science in Computer Engineering
Year: 1992

Fault tolerant programming methods improve software reliability using the principles of design diversity and redundancy. Design diversity and redundancy, on the other hand, escalate the cost of the software design and development. In this thesis, we study the reliability of hybrid fault tolerant systems. Probability models based on fault trees are developed for the recovery block (RB), N-version programming (NVP) and hybrid schemes which are the combinations of RB and NVP. Two heuristic methods are developed to construct hybrid fault tolerant systems with total cost constraints. The algorithms provide a systematic approach to the design of hybrid fault tolerant systems.

TABLE OF CONTENTS

	<u>PAGE</u>
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
LIST OF TABLES	ix
LIST OF FIGURES	x
I. INTRODUCTION	1
1.1 Motivation	1
1.2 Basic Concepts	2
1.2.1 Software Fault Tolerance	2
1.2.2 Recovery Blocks	2
1.2.3 N-Version Programming	3
1.2.4 Hybrid Fault-Tolerant System and Cost Constraints ...	3
1.2.5 Hierarchical N-Version Programming	4
1.3 Contributions of the Thesis	4
1.4 Thesis Overview	5
II. BACKGROUND AND REVIEW	7
2.1 Fault-Tolerant Software	7
2.1.1 Software Failure Behavior	7

2.1.2	Software Fault-Tolerance	11
2.2	Recovery Blocks and N-Version Programming	11
2.2.1	Recovery Blocks	12
2.2.2	N-Version Programming	15
2.2.3	Methods	17
2.3	Fault Tree for Analyzing Fault-Tolerant Systems	18
2.4	Cost Analysis of Fault-Tolerant Systems	20
2.5	Summary	20
III.	RELIABILITY ANALYSIS OF HYBRID FAULT-TOLERANT SYSTEMS	21
3.1	Recovery Block and N-Version Programming Models	21
3.2	Hybrid RB and NVP Fault-Tolerant Systems and Their Mathematical Models	28
3.3	Static Modeling of the Hybrid Systems	31
3.3.1	Recovery Block Structure	31
3.3.2	N-Version Programming Structure	33
3.3.3	Hybrid Structure	33
3.3.4	Simulation Models and Modeling Results	41
3.3.5	Discussion	49
3.4	Summary	56

IV. HYBRID FAULT-TOLERANT SYSTEM DESIGN WITH COST CONSTRAINTS 61

4.1 Hybrid Fault-Tolerant System 61

4.2 Fault-Tolerant System Design with Equal Program Reliability and Costs 64

4.2.1 Method I: Symmetrical Balanced Fault-Tolerant System Design 65

4.2.2 Illustration of Method I 67

4.3 Hybrid Fault-Tolerant System Design with Cost Constraints: General Model 71

4.3.1 Fault-Tolerant System Design with Different Program Reliability and Costs 71

4.3.2 Illustration of Method II 76

4.4 Discussion 78

4.5 Summary 84

V. MODELING OF HIERARCHICAL N-VERSION SOFTWARE FAULT-TOLERANT SYSTEMS 85

5.1 Hierarchical N-Version Programming 85

5.2 ARIES-82 Reliability Modeling Software 86

5.3 Simulation Models 87

5.4 Analysis of Results 89

5.5 Summary 89

VI. CONCLUSIONS AND FUTURE STUDY	95
6.1 Conclusions	95
6.2 Limitations and Future Study	96
REFERENCES	98

LIST OF TABLES

4-1	System Reliability of Different Hybrid Fault-Tolerant Systems	69
4-2	Input Data for Illustration II	77
4-3	Feasible Solutions of Illustration II	83

LIST OF FIGURES

2-1	Three Stages of Software Development Cycle	8
2-2	Conceptual Diagram of Software Failure	10
2-3	General Structure of the Recovery Block	13
2-4	Conceptual Diagram of the Recovery Block	14
2-5	Conceptual Structure of the N-Version Programming	16
2-6	Fault Tree for TMR System	19
3-1	Generalized RB Structure	22
3-2	Fault Tree for Generalized RB	24
3-3	Generalized NVP Structure	26
3-4	Fault Tree for Generalized NVP	27
3-5	Structure of Hybrid RB & NVP Fault-Tolerant System	29
3-6	Scheme (1) 6^1 RB	32
3-7	Scheme (2) 6^1 NVP	34
3-8	Scheme (3) 2^1 RB 3^1 NVP	35
3-9	Scheme (4) 2^1 NVP 3^1 RB	36
3-10	Scheme (5) 3^1 RB 2^1 NVP	37
3-11	Scheme (6) 3^1 NVP 2^1 RB	38
3-12	Failure Rates for Scheme (1)	42

3-13	Failure Rates for Scheme (2)	43
3-14	Failure Rates for Scheme (3)	44
3-15	Failure Rates for Scheme (4)	45
3-16	Failure Rates for Scheme (5)	46
3-17	Failure Rates for Scheme (6)	47
3-18	Comparison of System Failure Rates ($e=2\%$, $d=0.1\%$ to 0.6%)	50
3-19	Comparison of System Failure Rates ($e=6\%$, $d=0.1\%$ to 0.6%)	51
3-20	Comparison of System Failure Rates ($e=6\%$, $d=1\%$ to 6%) ..	53
3-21	Comparison of System Failure Rates ($e=6\%$, $d=0.01\%$ to 0.06%)	54
3-22	NVP System Reliability by Using Scott's Model	55
3-23	RB System Reliability with Error Type $t_1=0$	57
3-24	Comparisons of System Failure Rates Using Scott's Model ($e=2\%$)	58
3-25	Comparisons of System Failure Rates Using Scott's Model ($e=6\%$)	59
4-1	Homogenous Hybrid Fault-Tolerant Software Scheme	70
4-2	Hybrid Fault-Tolerant System Scheme (1)	79
4-3	Hybrid Fault-Tolerant System Scheme (2)	80
4-4	Hybrid Fault Tolerant System Scheme (3)	81

4-5	Hybrid Fault-Tolerant System Scheme (4)	82
5-1	Model 1 - Three Level Redundancy	90
5-2	Model 2 - Two Level Redundancy	91
5-3	Model 3 - One Level Redundancy	92
5-4	Comparison of System Reliability	93

CHAPTER I

INTRODUCTION

1.1 Motivation

In certain critical areas such as air traffic control [Aviz87], nuclear plants monitoring, financial management applications [Sims87], and in military applications, for instance in the so called "star wars" (Strategic Defense Initiative - SDI) project [Myer86], the reliability of the computer systems is of utmost concern. Fault-tolerant computer systems are capable of recovering from failures of their hardware or software components to provide uninterrupted service [Kimk89]. Due to the continuous decline of the cost of computer hardware, the reliability of computer systems can be improved by using redundant components. This redundancy can be static or dynamic [Aviz75]. Most of previous studies have concentrated on hardware redundancy mechanisms as the means to improve the computer system's reliability. However, it is the software reliability problem that has become more and more critical to the total reliability of the computer system. Any study of fault-tolerant systems must consider in a balanced way both hardware and software fault tolerance [Fern90].

Software fault tolerance is the study of design approaches to provide correct outputs in the presence of design faults. Two important issues which are related

to the design and analysis of software fault-tolerant systems are the reliability and the cost associated with various fault-tolerant mechanisms, and we concentrate on these issues in this thesis. Fault-tolerant software systems considered in this study are the two most commonly adopted schemes, Recovery Block (RB) [Rand75] and N-Version Programming (NVP) [Aviz77], [Chen78].

1.2 Basic Concepts

1.2.1 Software Fault Tolerance

It has been noticed that to completely remove all software defects is not possible for a complicated software system. In order to prevent the failure of a software system due to some unpredicted conditions, different programs (alternative programs) are developed separately, preferably based on different logic and/or algorithms (design diversity). The fault-tolerant program so obtained should be able to function correctly in the presence of most software design faults.

1.2.2 Recovery Blocks

The Recovery Block (RB) scheme [Rand75], is one of the basic fault-tolerant programming structures. In a RB system, a programming function is realized by n alternative programs. The computational result generated by an alternative program is checked by an acceptance test. If the result is rejected, another alternative program is then executed. The program will be repeated until an acceptable result is generated by one of the n alternatives or there are no more alternatives available.

1.2.3 N-Version Programming

The N-Version Programming (NVP) scheme [Chen78] also consists of n alternative programs and a decision algorithm, usually, a voting mechanism. Differently from the RB approach, all the n alternative programs are usually executed simultaneously and their results are sent to the decision algorithm which selects the final output.

1.2.4 Hybrid Fault-Tolerant Scheme and Cost Constraints

The hybrid fault-tolerant system considered in this study is a software system which combines the RB and NVP schemes for a given functional task. In the hybrid system RB and NVP are blended together by different arrangements of the n alternatives. The idea here is to take advantage of the fact that the reliability of RB and NVP fault-tolerant systems depend on the reliability of the components which form the system. Those components include the program module, the acceptance test module for RB and the decision module for NVP. For instance, when a voting mechanism cannot select a correct result from n alternative results due to lack of similar results, a recovery block can be applied in this case since an acceptance test could test individual results.

In general, the reliability of the fault-tolerant system is enhanced by using more redundant program modules and by selecting the right fault-tolerant strategies. If there is a limitation on the total cost of the fault-tolerant system, the complexity of selecting the right components and the right structure to achieve the best system reliability is substantial.

1.2.5 Hierarchical N-Version Programming

The original N-Version Programming method implies to develop redundant modules for the entire programming task. In other words, one needs to write n ($n \geq 2$) programs to solve a particular problem. However, normally a problem can be divided into several distinct modules, and the reliability of the system can then be improved by applying fault-tolerant programming to some or all of the modules instead of the entire system. The hierarchical N-Version Programming is based on the perception that software reliability can be improved by applying N-Version Programming on the subsystems rather than the entire system.

1.3 Contributions of the Thesis

Fault-tolerant programming methods improve software reliability using the principles of design diversity and redundancy. Design diversity and redundancy, on the other hand, escalate the cost of the software design and development. Therefore, the objective of this study is to analyze the reliability and cost of RB, NVP and hybrid schemes of those two original strategies.

Probability models based on fault trees are developed for the RB, NVP and hybrid schemes. Two heuristic methods are developed to construct hybrid fault-tolerant systems with total cost constraints. Mathematical programming methods, such as linear and nonlinear programming methods, are used in those proposed methods. Those heuristic methods provide a systematic approach to the design of hybrid fault-tolerant systems.

[Bell90], [Scot83] proposed various probability models to calculate the reliability of NVP and RB fault-tolerant systems. [Bell91] introduced an optimization model to design the RB and NVP schemes with total cost constraints. The model introduced in [Bell90] and [Bell91] is rather complicated and the optimal solution is obtained by using exhaustive searching. Extensive amount of calculations made this model not practical. Additional assumptions are introduced in this study to simplify the mathematical calculations. Using probability models and fault tree to study the hybrid fault-tolerant schemes and utilizing heuristic algorithms to design hybrid scheme with cost constraints are the unique contribution of the study.

1.4 Thesis Overview

Some basic concepts and previous research related to this study are presented in Chapter 2. The discussion includes a review of software failure behavior, RB and NVP, as well as methods for modeling fault-tolerant system such as fault trees and probability models.

Chapter 3 deals with mathematical models for the RB, NVP and hybrid schemes. Fault tree and probability models are developed to study the reliability of the RB, NVP as well as the hybrid fault-tolerant systems. Examples are given to analyze the general behavior of the different schemes under various input data.

The cost issues of hybrid fault-tolerant software systems are considered in Chapter 4. Two heuristic algorithms used for the design of hybrid fault-tolerant systems are presented. Algorithm I is for a symmetrical balanced hybrid structure. In a

symmetrical balanced system, the reliability of program modules are all identical, and the same as the reliability of acceptance test modules and decision making modules. A nonlinear program model is developed to optimize the design of the structure of the hybrid fault-tolerant system. Examples are constructed under various system structures and software failure conditions. Method II addresses a more general condition, in which the system structure is not necessarily balanced and the program version failure rates vary. The reliability of the testing and voting modules also vary. Heuristic methods are developed for the design of the system under cost constraints.

A hierarchical N-version fault-tolerant system is presented in Chapter 5. The ARIES 82 software system was used here to evaluate the reliability of different schemes.

Finally, some thoughts on the limitations of this study and future research are presented in Chapter 6.

CHAPTER II

BACKGROUND AND REVIEW

This chapter reviews basic fault tolerance concepts and recent developments in software fault tolerance.

2.1 Fault-tolerant Software

Hardware fault tolerance has been extensively studied [Siew82], [John89]. We concentrate here in software fault tolerance which is the objective of this thesis. We consider software failure behavior, the two basic constructs for fault tolerance: Recovery Blocks and N-Version Programming, and we present the use of fault trees for the evaluation of software.

2.1.1 Software Failure Behavior

In order to study software failure behavior we need to understand the life cycle of software development. Software development can be divided into three stages as shown in Figure 2-1:

- (1) Functional requirement specifications stage;
- (2) Logic/algorithm design stage; and
- (3) Programming/coding stage.

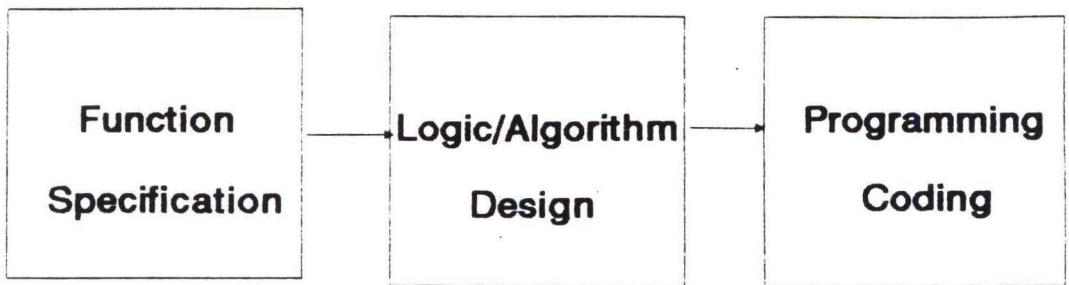


Fig. 2-1 Three Stages of Software Development Cycle

Usually, the human errors/faults occur in all three stages. Errors in the functional specifications can be reduced by careful planning and supervision. Using modern computer aided software engineering systems programming faults can be largely reduced. Most of the software failures come from the design stage. Design faults could be produced by the following reasons:

- (1) Misinterpretation of the specifications; or
- (2) Faults on design and selection of the logic and/or algorithms.

Most of the misinterpretations of the functional specifications are caused by inaccurate and/or incomplete specifications. This should be resolved by better planning and administration. Fault tolerant programming, therefore, should be addressed specifically to faults in the logic/algorithm design stage.

The logic/algorithm design faults imply more than just errors in themselves but faults usually occur with unpredicted input combinations or unpredicted data exchanges with other functional programs. Because of the complexity of software systems, complete testing of all the input combinations for a particular program is not possible. Using fault tolerance programming, n different versions will be coded for the same functional requirements. We hope that not all of the n different versions of the programs will fail under a particular input condition.

A general conceptual diagram of software fault tolerance is shown in Figure 2-2 [Lapr84]. Let us use I for input, O for output, V for program versions, subscript e for error set, and c for correct set. Then the notation I_{e_i} will be the input subset of those points which will cause $V(i)$ to fail. Failures are produced whenever

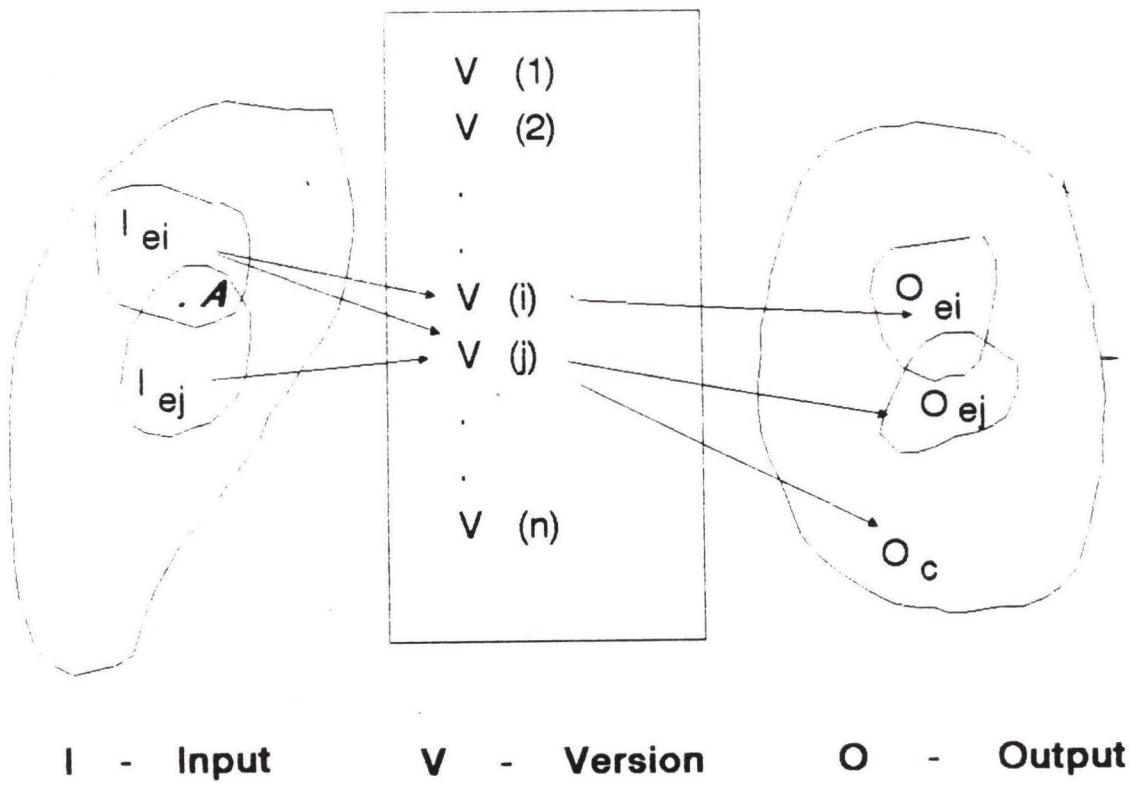


Fig. 2-2 Conceptual Diagram of Software Failure

inputs are selected from the subset I_{ei} within input space I , processed by program version $V(i)$, and an erroneous result in O_{ei} is generated. The input error sets overlap as shown by $I_{ei} \cup I_{ej}$. This type of errors, correlated errors, are an important source of failure for fault-tolerant software [Dhil89], [Eckh85].

2.1.2 Software Fault Tolerance

As said earlier, software fault tolerance is based on redundant diversity.

In general, program redundancy can be applied under three major aspects [Horn74]:

- (1) Acceptance test or error checks;
- (2) Alternate try routines; and
- (3) Restoration routines.

In the acceptance test approach, the intermediate results of the program are tested for reasonableness or acceptability during program execution. Alternate try routines use different approaches for the same objective. Restoration routines return the system to a previously determined state when the acceptance test rejects a result. Recent developments on fault tolerant system design are primarily focused on two approaches, Recovery Blocks (RB) and N-Version Programming (MVP) methods.

2.2 Recovery Blocks and N-Version Programming

Two of the most popular software fault tolerant programming methods, Recovery Blocks and N-Version Programming are discussed. The system reliability is used as quality criteria. Comparisons of RB and NVP are made.

2.2.1 Recovery Blocks

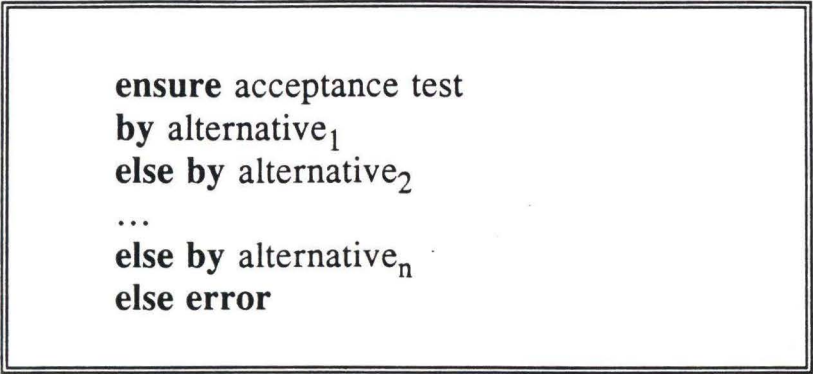
The general structure of the RB is shown in Figure 2-3 while Figure 2-4 shows its conceptual diagram where AT indicates the *Acceptance Test*, *Restore* is the program function which restores the input states (state i) of the RB. The A_i , ($i = 1, 2, \dots, n$) are the *Alternative Programs*. If the AT rejects an output produced by program A_i then the alternative A_{i+1} is activated. This process continues until a result is accepted or until all outputs are rejected. In the later case, an error signal will be generated.

Primary Alternative and Secondary Alternatives

The RB contains n alternative programs which are developed from the same set of specifications. They are arranged in a serial fashion comparable to the standby sparing technique used in hardware redundancy. Usually, the first alternative in the series is called the *primary alternative* which is the most reliable or most efficient program. The other alternatives are known as the *secondary alternatives*. The secondary alternatives could be degraded program modules; i.e., they can be simpler than the primary program and could generate degraded but acceptable service. (A reduction in the number of design faults should be expected by designing less complicated alternative programs.)

Acceptance Test

The *Acceptance test* applies some known conditions that the result should satisfy for error detection. It is invoked at the exit point of the alternatives. The acceptance test should be as simple as possible such that itself does not contain any



```
ensure acceptance test  
by alternative1  
else by alternative2  
...  
else by alternativen  
else error
```

Fig. 2-3 General Structure of the Recovery Block

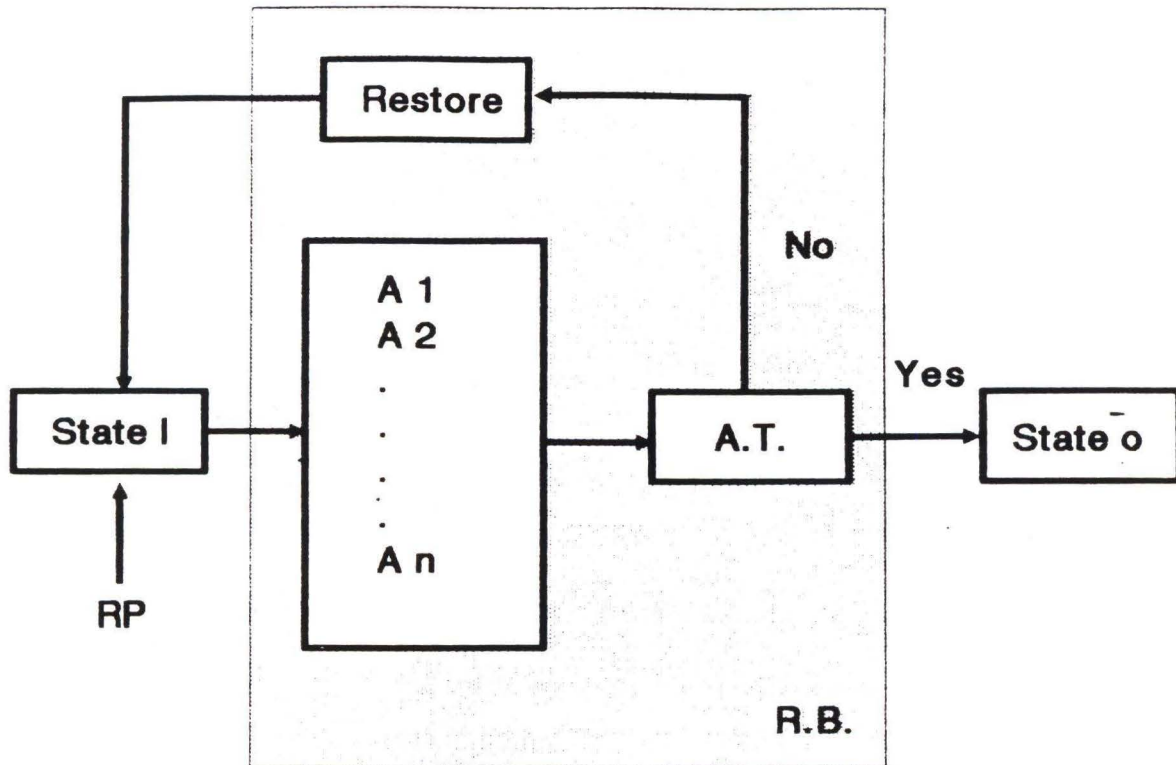


Fig. 2-4 Conceptual Diagram of the Recovery Block

design faults. In case the acceptance test contains design faults, it could then produce a failure by:

- (1) Rejecting an acceptable result, or
- (2) Accepting an unacceptable result. (This is really the most serious failure.)

Limitations of RB

The most important limitation of RB is in finding good acceptance tests. If there are not adequate their lack of coverage reduces the reliability of the system.

It is expected that by using different programmers, computer languages, and algorithms to produce several functional comparable programs from the same initial specification, the alternatives should not contain design faults. However, this expectation is not guaranteed, because of the possibility of correlated errors. However, in a RB those errors are not so significant as in N-Version Programming.

2.2.2 N-Version Programming (NVP)

The NVP method consists of n program versions and a voting mechanism. Figure 2-5 shows the conceptual structure of the NVP approach. State I and State O are the input state and the output state of the NVP module. $A_i (i=1, 2, \dots, n)$ are the alternative program modules. Decision selects the best solution out of n alternative solutions. Usually, a voter is used as the decision mechanism.

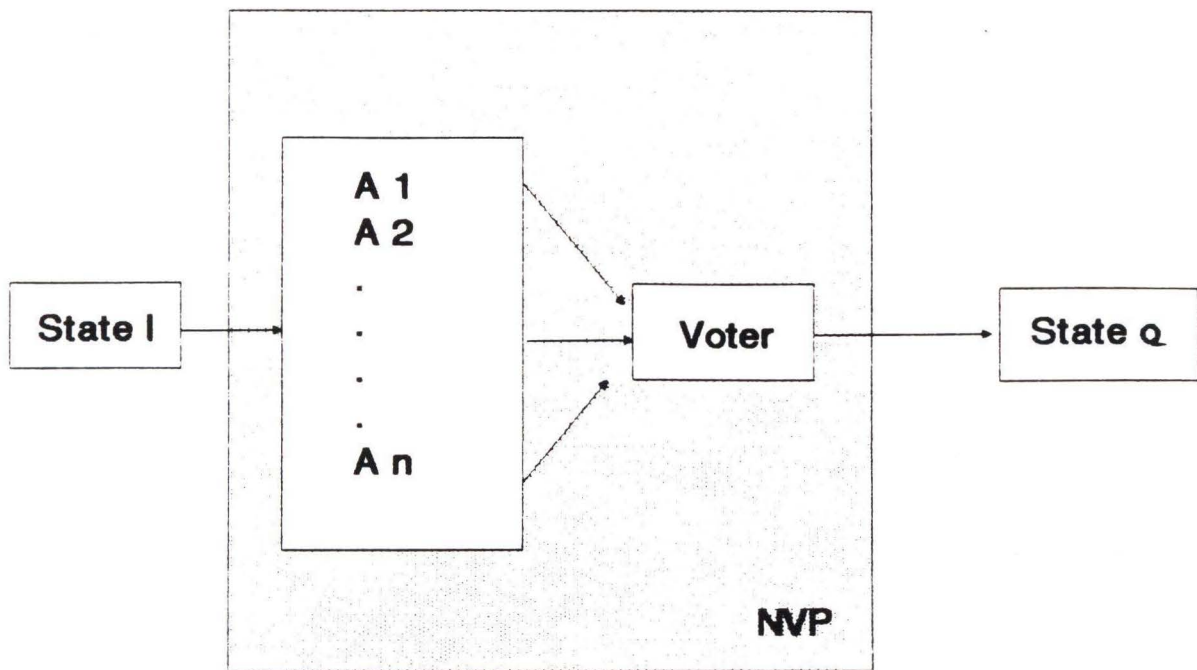


Fig. 2-5 Conceptual Structure of the N-Version Programming

Voting Mechanisms

Usually a voting mechanism is used to select the correct result from the n results generated by the n program versions. However, some applications generate identical results and some can produce slightly different but correct results. In the case that the versions generate identical results then a majority voting can select the correct result [Andt81]. For results which are slightly different due to precision voting can be done after a range check and correction.

Limitations of NVP

Similar to the RB, the NVP requires more design and programming work, and needs additional hardware to run those n program versions. Its effectiveness depends directly on the independence of the programs. Recent study reveals that the independently developed programs do not fail independently [Voum85], i.e., there are correlated errors among independently developed n programs. Because the versions must execute concurrently the effect of correlated errors is much more serious than for RBs. N-Version Programming is also wasteful of resources since the n versions must execute concurrently.

2.2.3 Methods

Various probability models have been developed for RB and NVP [Grna80a], [Grna80b], [Scor83]. Assumptions are used in developing those models. [Bell91] proposed a model for designing these fault tolerant schemes with a total cost limit. The fault tolerant schemes considered were simple RB or NVP schemes.

2.3 Fault Tree for Analyzing Fault Tolerant Systems

Many modeling techniques have been adopted to study the reliability of fault tolerance systems. In this section we describe the fault tree method since it is the only one we will use in this thesis. See [Leus90] for a discussion of several other methods.

Fault Trees

The fault tree is a modeling tool represents the conditions that result in a system or subsystem failure. It displays the possible events which cause the system failure. The fault tree is obtained from the system structure and functional requirements. Sometimes, the system reliability is calculated based on the tree representation but it cannot describe common failures.

A TMR system is used here to demonstrate the construction of fault tree and develop the system reliability from it [John88]. Figure 2-6 is the fault tree for a TMR system. Two type of logical gates are shown in the figure, the "OR" gate and the "AND" gate. The OR gate indicates that the output event will exist if one or more of the input events is present. The AND gate defines the situation when the coexistence of all input events is required to produce the output event. Additional discussions about fault trees can be found in [Arse80], [Barl75], and [Dhil78]. Some reliability modeling program packages use the fault tree as one of their input tools [Stif79], [Sahn87].

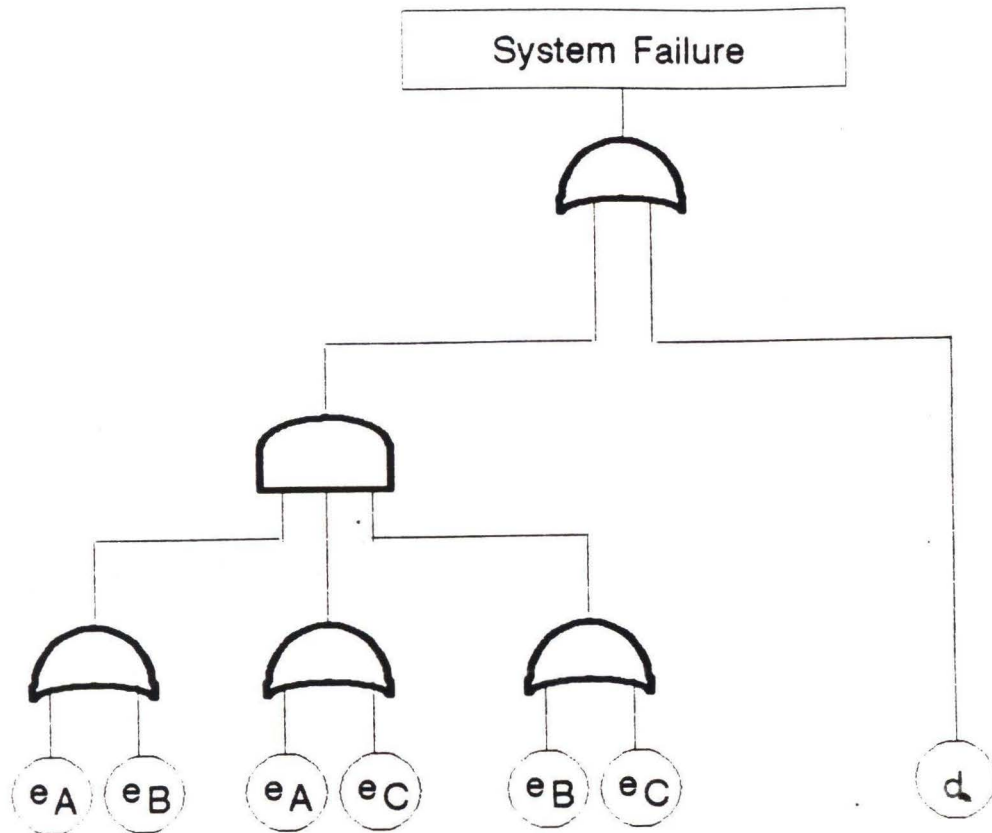


Fig. 2-6 Fault Tree for TMR System

2.4 Cost Analysis of Fault-tolerant Systems

In general, more redundancy in the computer program modules will increase the reliability of the software in both RB and NVP schemes (if these modules are carefully tested). However, additional program modules increase the software development cost. Extra programs also requires more computational power on the computer, especially for the NVP. A balance on reliability and cost should be achieved. [Bell91] seems to be the only work which addresses the relationship between cost and reliability of a fault tolerant system.

2.5 Summary

This chapter introduced the basic concepts of software fault tolerance and reviewed previous research in this field. The fault tree method will be applied in this thesis for analyzing hybrid fault tolerant systems and has been discussed here in some detail.

CHAPTER III

RELIABILITY ANALYSIS OF

HYBRID FAULT-TOLERANT SYSTEMS

In this chapter two of the most common approaches for fault tolerance, the Recovery Block (RB) and the N-Version Programming (NVP) method are evaluated. Then the reliability of a hybrid fault-tolerant system combining the RB and NVP is explored.

The analysis consists of two parts: First, reliability models of RB and NVP are developed in the form of fault-trees and analytical modeling. Then, a reliability expression for the hybrid mechanisms is developed. A program containing six modules and a number of testing and voting modules is used to demonstrate the proposed reliability analysis method. Numerical calculations are analyzed to give a general understanding of the hybrid approach.

3.1 Recovery Block and N-Version Programming Models

Recovery Block

A generalized RB structure with n_{RB} modules is described in Figure 3-1. We assume one acceptance test module T is used for all the versions. If the acceptance

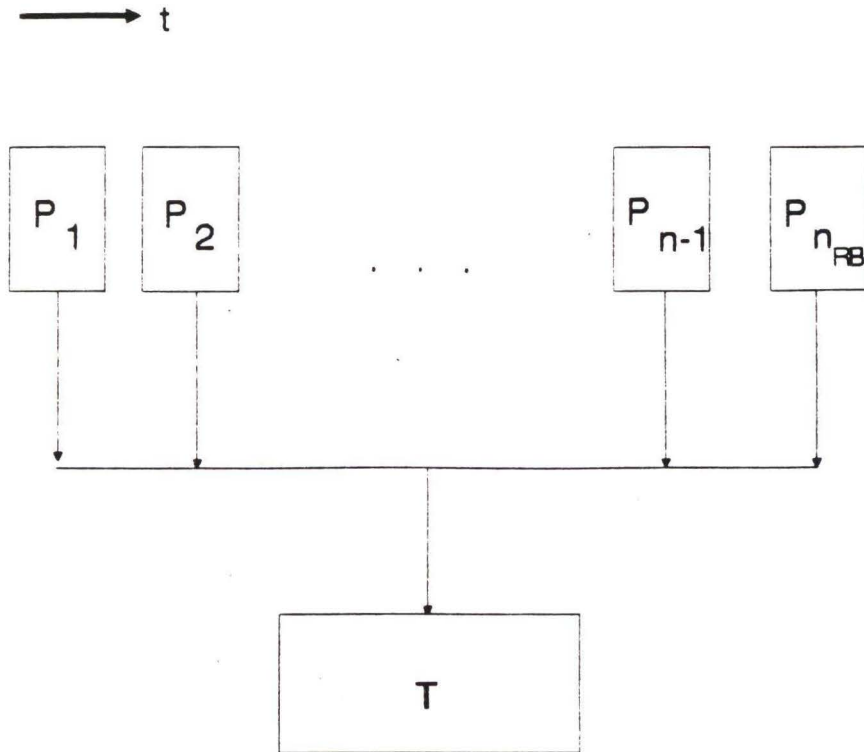


Fig. 3-1 Generalized RB Structure

test detects an erroneous output in module i then the input state is recovered and module $(i+1)$ is activated. This procedure is repeated until success or lack of versions.

A fault tree for this RB model is shown in Figure 3-2 [Bell90]. The reliability of the fault-tolerant system depends on the reliability of program modules P_i as well as on the reliability of the acceptance test T . The test module can fail in two modes, type 1 failure (t_1) and type 2 failure (t_2) described below. The following parameters are used in the analysis:

e_i = Probability of failure for program module i ($i=1, 2, \dots, n$).

n = Number of program modules.

n_{RB} = Number of RB modules in a scheme.

t_{1i} = Probability of failure in RB when acceptance test i judges an incorrect result as correct.

t_{2i} = Probability of failure in RB when acceptance test i judges a correct result as incorrect.

The probability model of the generalized RB scheme is calculated as follows:

$$P_{RB} = \prod_{i=1}^{n_{RB}} (e_i + t_{2i}) + \sum_{i=1}^{n_{RB}} t_{1i} \left(\prod_{j=1}^i (e_j + t_{2j}) \right) \quad (3-1)$$

where:

P_{RB} = Probability of failure of the RB scheme.

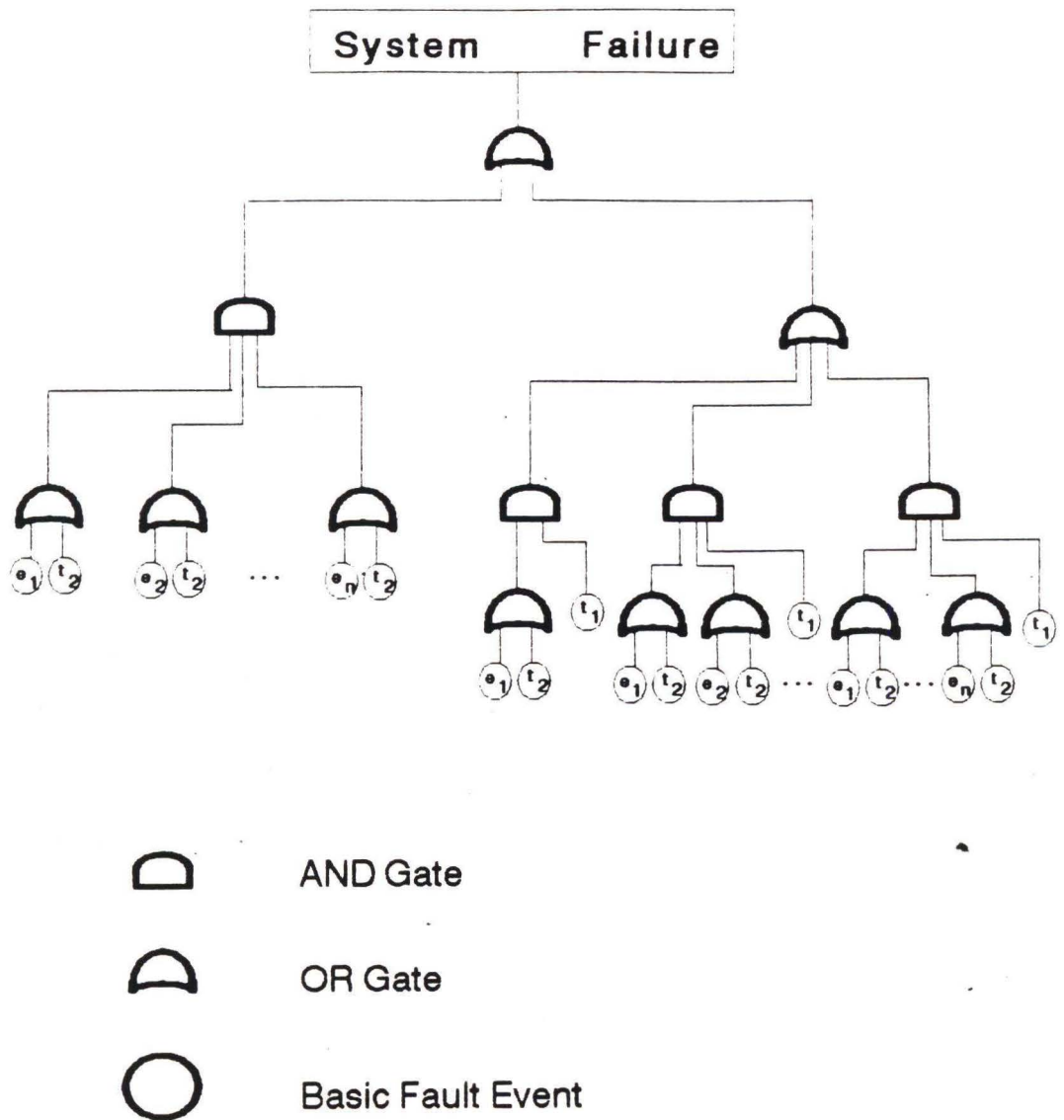


Fig. 3-2 Fault Tree for Generalized RB [Bell90]

There are two parts in equation 3-1. The first part reflects the situation when program versions fail or when the acceptance test erroneously judges correct results as failures. The second part represents the situation when the test accepts an incorrect result. Both conditions cause the fault-tolerant system failure.

N-Version Programming

Figure 3-3 shows a generalized NVP structure. The decision algorithm may itself fail by not being able to select the correct result. Figure 3-4 is the fault tree for a generalized NVP [Bell90]. [Scot83] described that there are three types of errors related to NVP:

- (1) all of the n versions disagree
- (2) more than one version has an incorrect result
- (3) voting procedure has error.

Assuming that the correlated errors among program versions and the error of type (2) are ignored, the probability of system failure P_{NV} is obtained from the fault tree as:

$$P_{NV} = \prod_{i=1}^{n_{NV}} e_i + d \quad (3-2)$$

where:

e_i = Probability of failure for program module i ($i = 1, 2, \dots n$). In NVP, the program module failure is defined as producing no output.

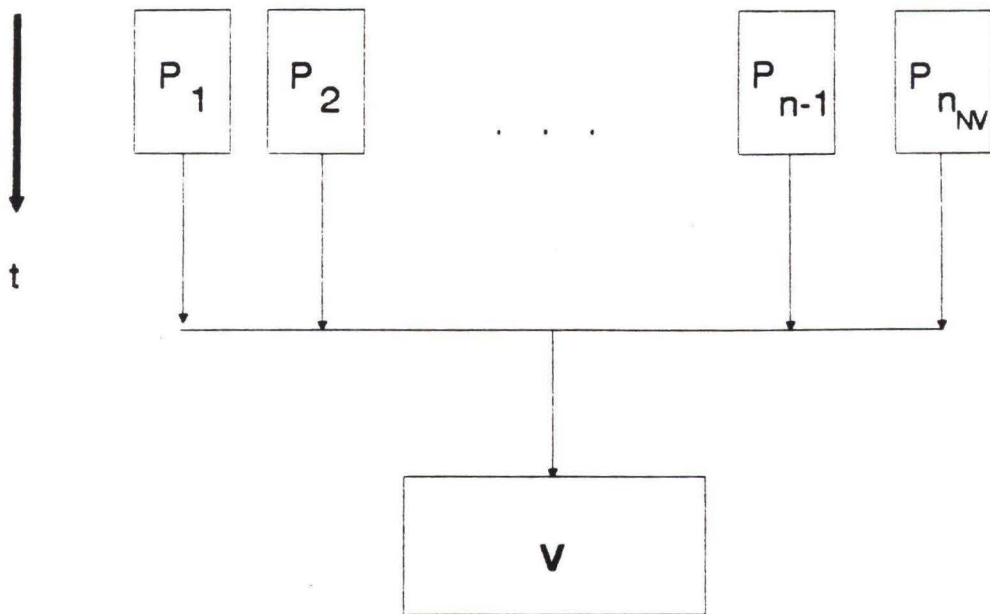


Fig. 3-3 Generalized NVP Structure

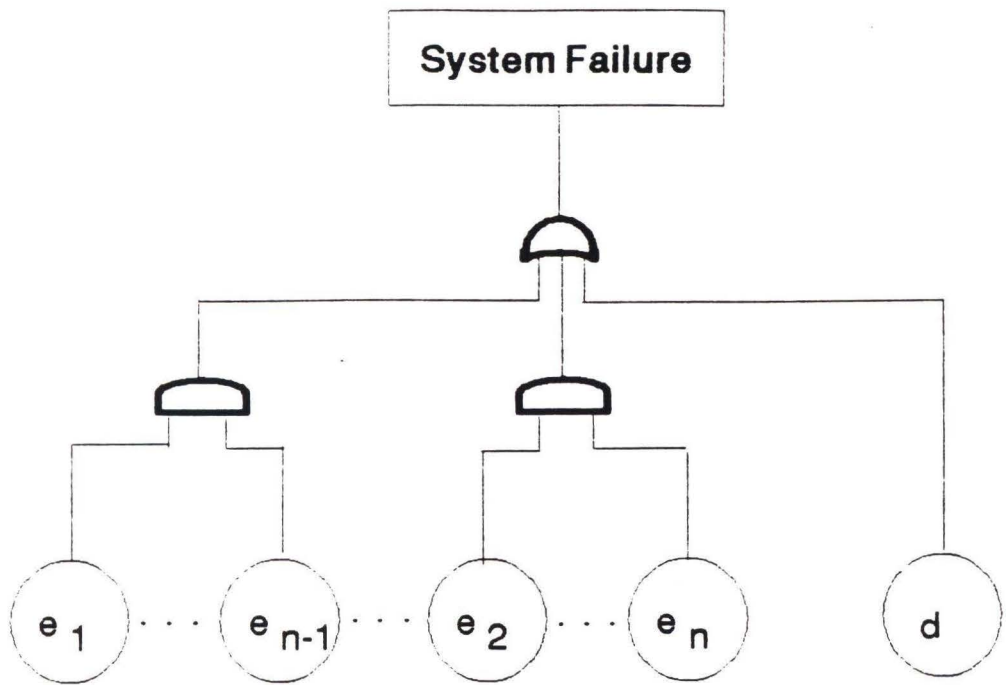


Fig. 3-4 Fault Tree for Generalized NVP [Bell90]

$d =$	Probability in NVP that the decision algorithm cannot select the result out of at least 2 correct results
$n_{NV} =$	Number of NVP versions in a scheme.

3.2 Hybrid RB and NVP Fault-Tolerant Systems and Their Mathematical Models

A hybrid fault-tolerant system combines the RB and NVP schemes. The idea here is to take advantage of good aspects of both RB and NVP. In general, a hybrid fault-tolerant system consists of many small subsystems. Each subsystem may include even smaller subsystems. To simplify the discussion, we consider here hybrid fault-tolerant systems with only two levels: RB embedded in NVP or NVP embedded in RB. Figure 3-5 shows the basic structure of a hybrid RB and NVP fault-tolerant system. The first level consists of P_{n_i} basic program modules which form the second level program versions P_{v_i} , $1 \leq i \leq m$. If RB (or NVP) is used at the first level, NVP (or RB) is used at the second level. Failure rates for the basic program modules and program versions are e_i and e_{v_i} , respectively. The program versions failure rates e_{v_i} are calculated based on the structure of the version and the failure rates of the program modules (e_i), acceptance test error probabilities (t_1 , t_2), and decision error probability (d). The total hybrid fault-tolerant scheme's reliability is obtained by first calculating the reliability of the lower level program versions, and then use the lower level program versions reliability as the input to the higher level versions. This process is repeated until the total system reliability is obtained. Mathematically, the hybrid system's reliability is calculated

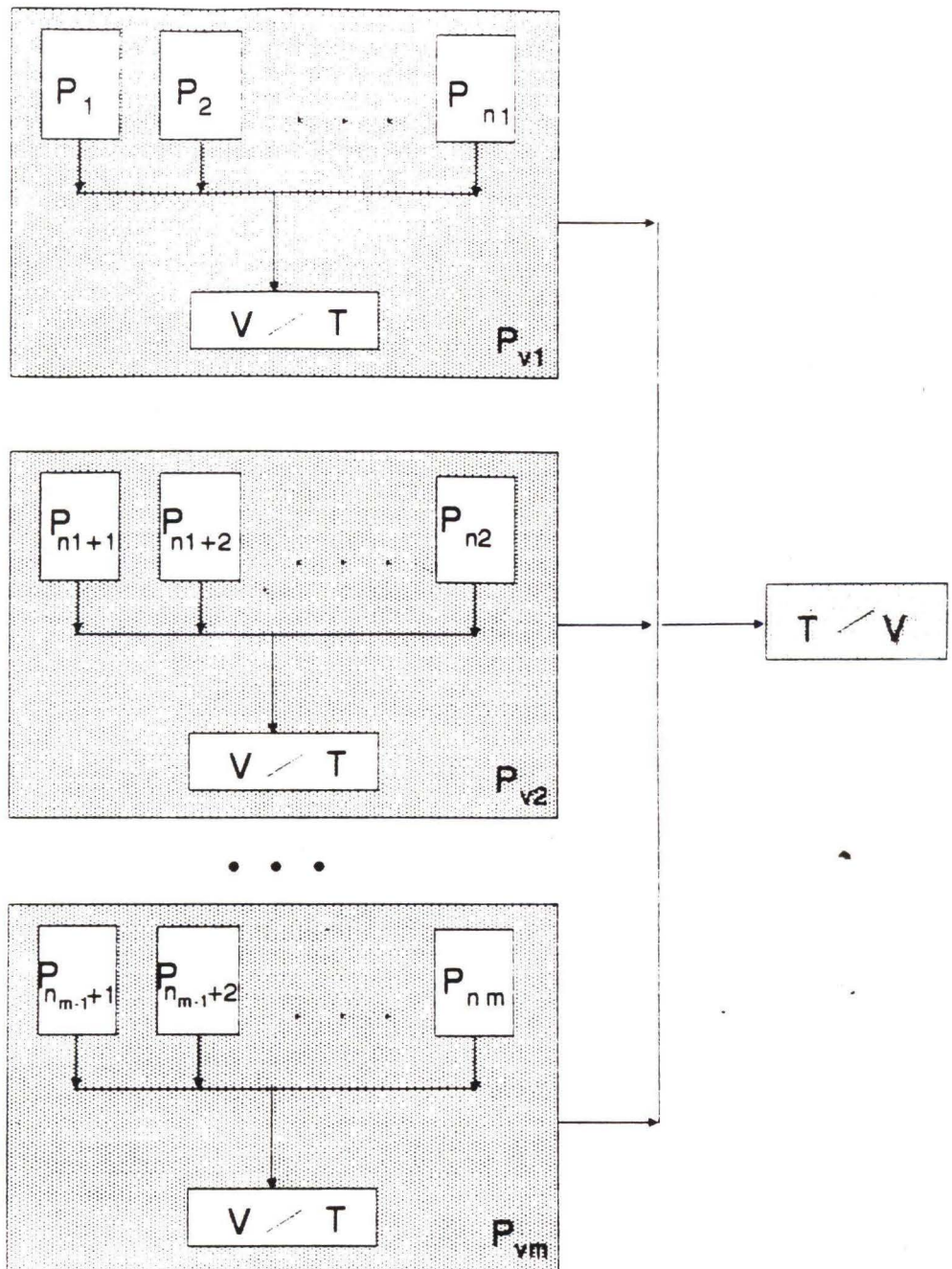


Fig. 3-5 Structure of Hybrid RB & NVP Fault Tolerance System

by using equations (3-1) and (3-2) where the program module's failure rates e_i is substituted by e_{v_i} . We have:

$$P_{RB} = \prod_{i=1}^{n_{RB}} (e_{v_i} + t_{2i}) + \sum_{i=1}^{n_{RB}} t_{1i} \left(\prod_{j=1}^i (e_{v_j} + t_{2j}) \right) \quad (3-3)$$

$$P_{NV} = \prod_{i=1}^{n_{NV}} e_{v_i} + d \quad (3-4)$$

The following definitions are defined for hybrid fault-tolerant scheme:

e_i = Probability of failure for program module i ($i = 1, 2, \dots, n$)

e_{v_i} = Probability of failure for version i .

m = Number of hybrid versions.

T_i = Test module i in the RB scheme.

V_i = Voting module i in the NVP scheme.

P_i = Program module i ($i=1, 2, \dots, n$).

n_i = Number of program modules in version i ($n_1 + n_2 + \dots + n_m = n$)

P_{vi} = Hybrid program version i ($i = 1, 2, \dots, m$).

The probability of system failure P_F :

$$P_F = f(P_{RB}, P_{NV}) \quad (3-5)$$

The function \mathcal{F} defines the approach used in the higher level of the system, which is given by equation (3-3) for RB or equation (3-4) for NVP.

3.3 Static Modeling of the Hybrid Systems

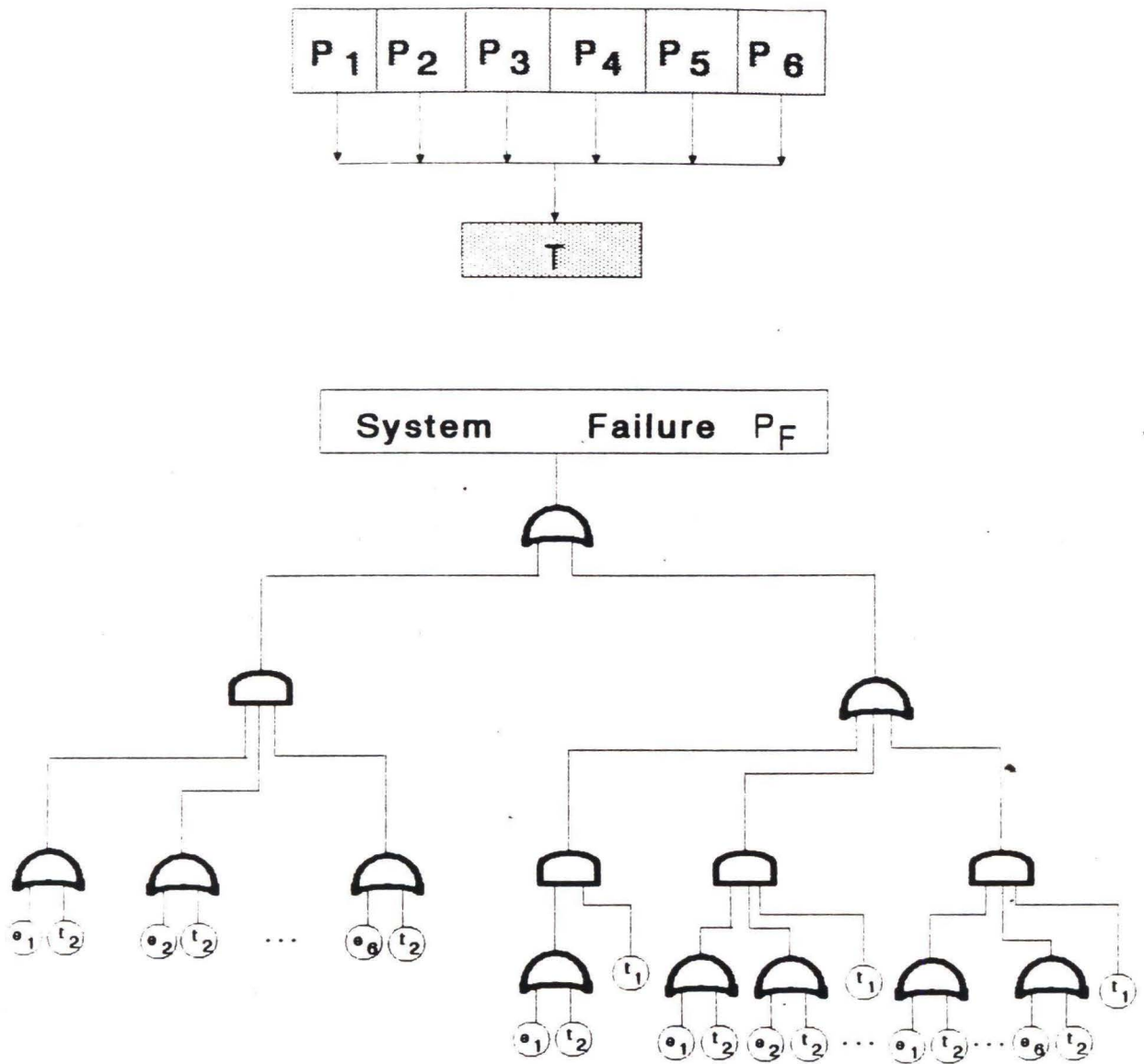
In this section, a fault-tolerant scheme with six (6) program modules which represent the same function in different ways is used to study different fault tolerance characteristics. The RB, NVP as well as the hybrid RB and NVP schemes are studied. The following simplifications are used:

- (1) All program module's failure rates are same ($e_i = e$, $e_{vi} = e_v$). The probabilities of failure for hybrid program versions are also the same since we assume the number of program versions and their structure (RB or NVP) in hybrid versions are the same.
- (2) The two types of errors t_1 and t_2 are the same ($t_1 = t_2$) in a Recovery Block structure.
- (3) The probabilities of acceptance test error t_1 and t_2 are greater than the decision error d in a NVP.

3.3.1 Recovery Block Structure

Scheme 1: 6^1 RB (6-modules, 1-acceptance test RB scheme, Figure 3-6)

With six program modules and one acceptance test, several Recovery Block schemes can be formed. Figure 3-6 shows its system structure and its fault tree

Fig. 3-6 Scheme (1) $6^1 1$ RB

form. According to equation (3-3) the probability of system failure P_F for the 6^1 RB scheme is as follows:

$$\begin{aligned}
 P_F &= \prod_{i=1}^6 (e_i + t_2) + t_1 \sum_{i=1}^6 (e_i + t_2)^i |_{e_i=e} \\
 &= (e + t_2)^6 + t_1 \sum_{i=1}^6 (e + t_2)^i
 \end{aligned}
 \tag{3-6}$$

3.3.2 N-Version Programming Structure

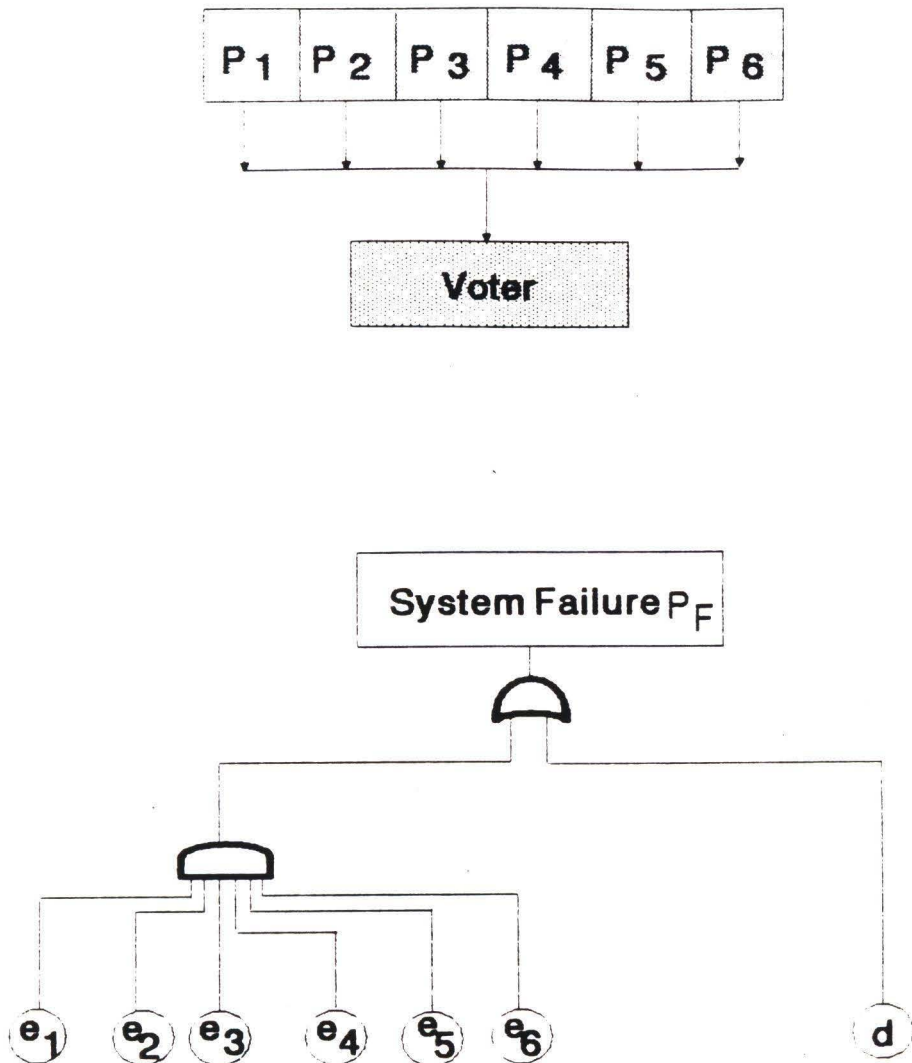
Scheme 2: 6^1 NVP (Six modules, one voter NVP scheme, Figure 3-7)

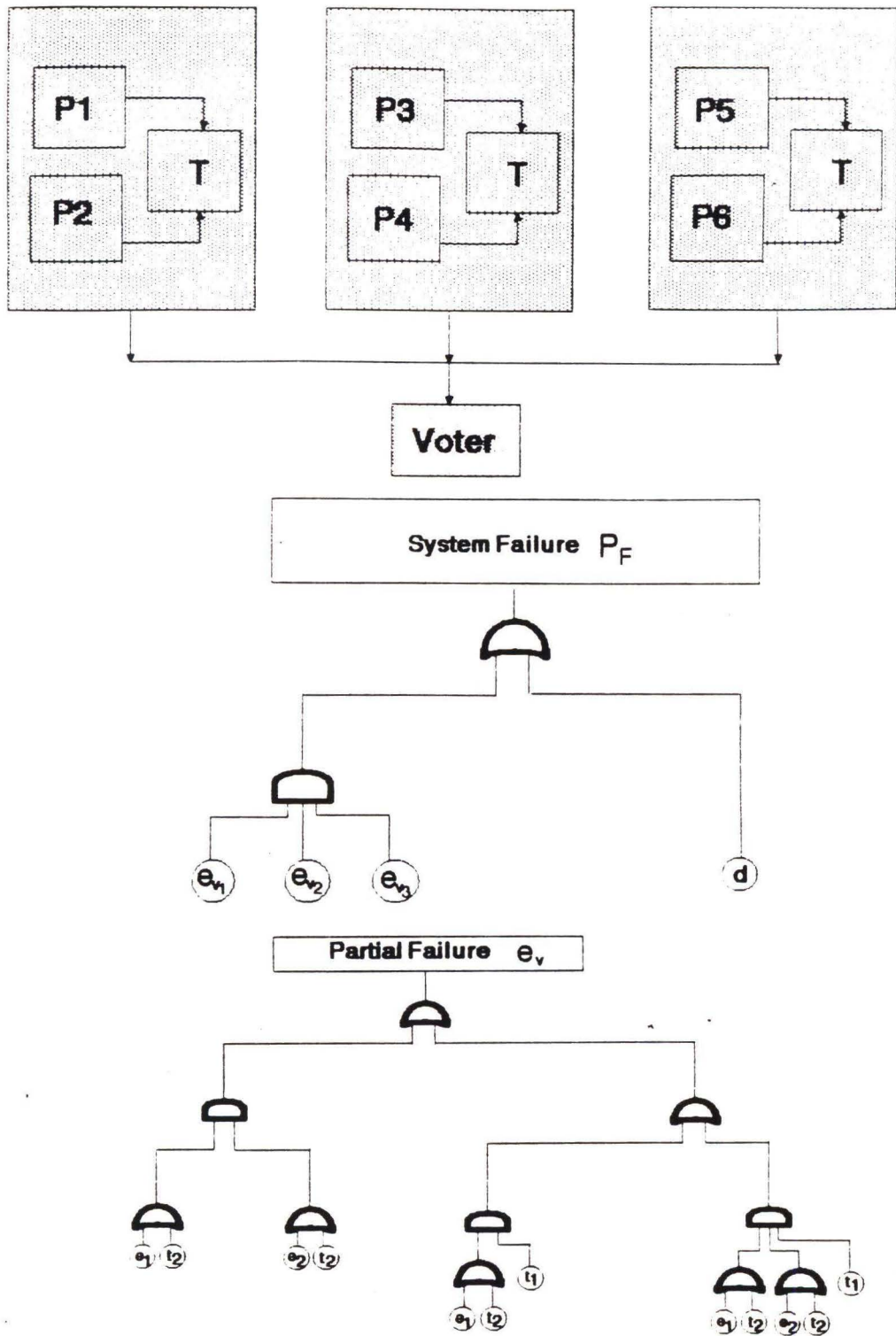
In this NVP scheme the system is unable to produce an output under two conditions: either all six program modules fail or the decision (voter) mechanism fails. Figure 3-7 displays the scheme and its corresponding fault tree. From the equation (3-4), the probability model is as follows:

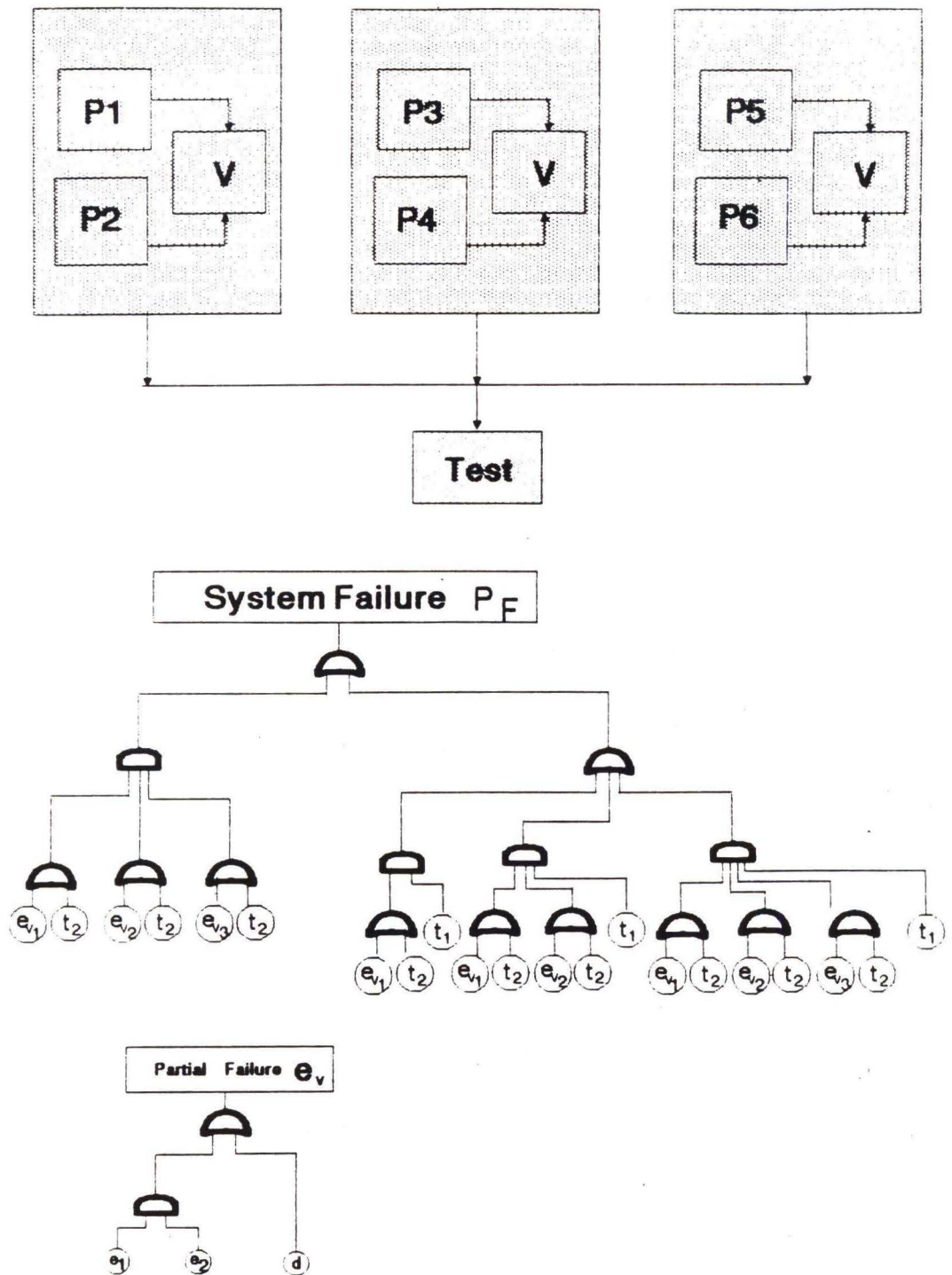
$$\begin{aligned}
 P_F &= \prod_{i=1}^6 e_i + d |_{e_i=e} \\
 &= e^6 + d
 \end{aligned}
 \tag{3-7}$$

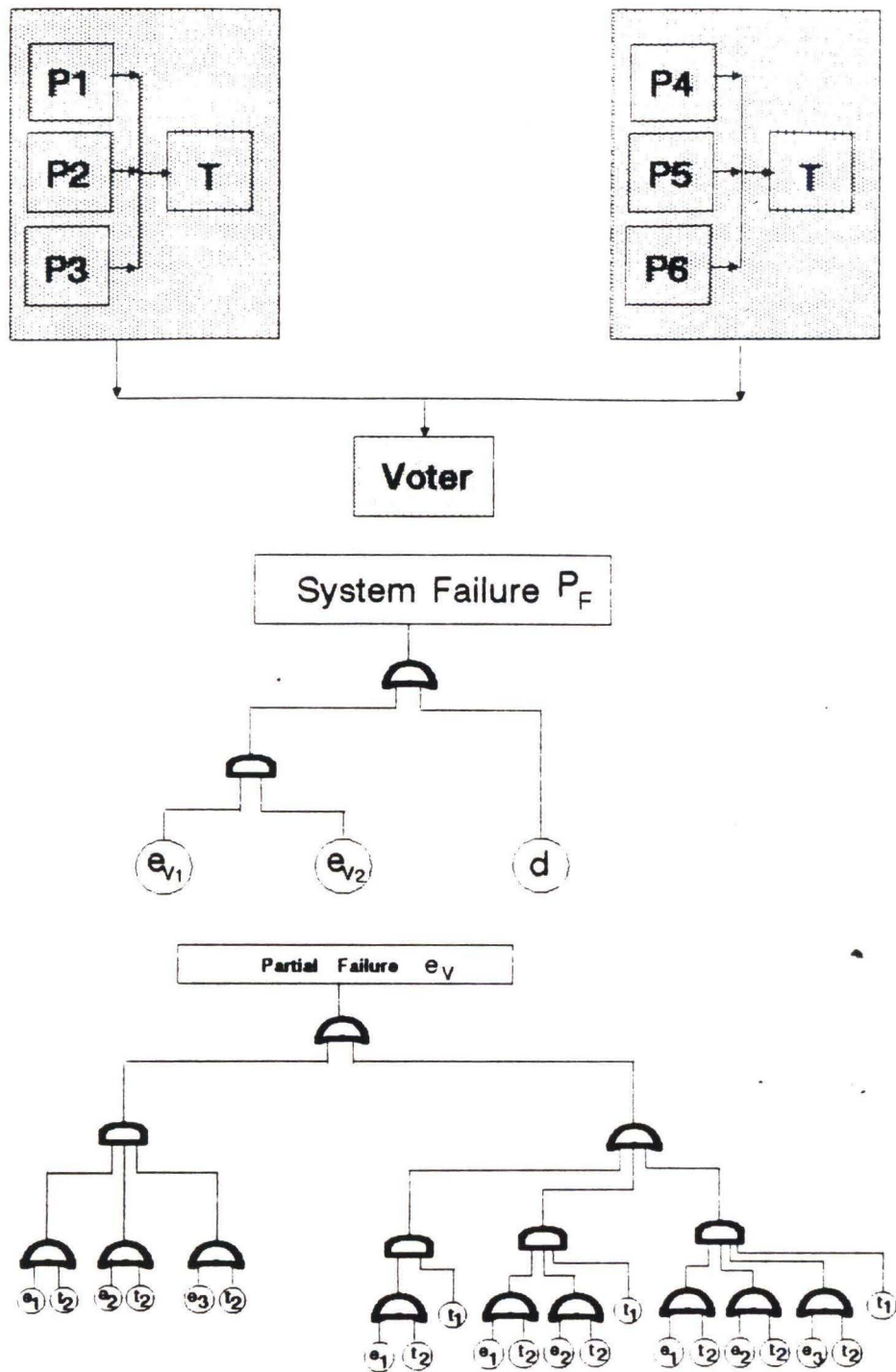
3.3.3 Hybrid Structure

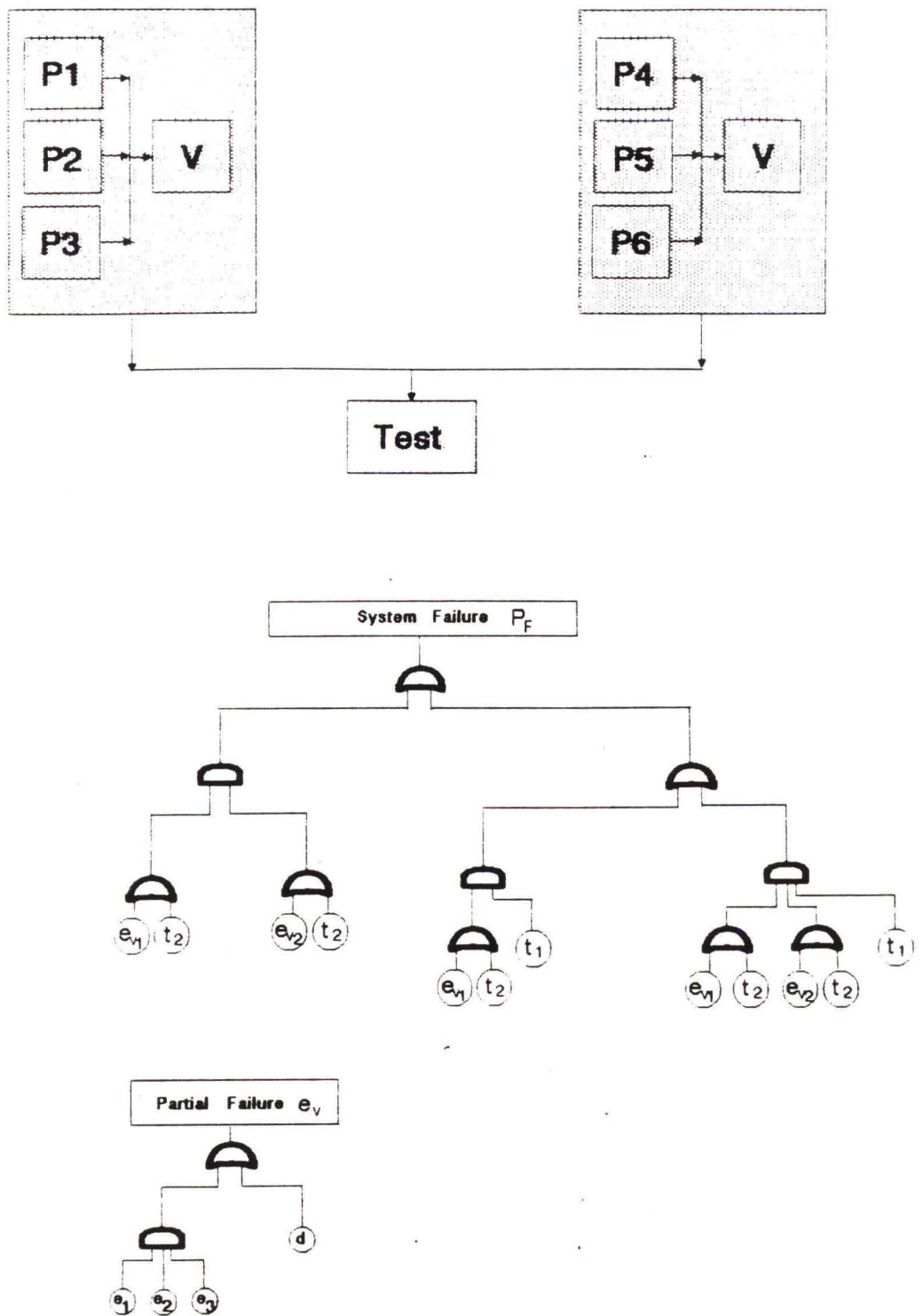
The hybrid structure combines both RB and NVP concepts. With six modules, four combinations are possible. Figures 3-8, 3-9, 3-10, and 3-11 are the schemes and fault trees for the schemes 3, 4, 5 and 6 relatively. The probability models are the following:

Fig. 3-7 Scheme (2) 6^1 NVP

Fig. 3-8 Scheme (3) $2^1 \text{ RB } 3^1 \text{ NVP}$

Fig. 3-9 Scheme (4) 2^1 NVP 3^1 RB

Fig. 3-10 Scheme (5) 3¹ RB 2¹ NVP

Fig. 3-11 Scheme (6) 3^1 NVP 2^1 RB

Scheme 3: 2¹ RB 3¹ NVP (Figure 3-8)

$$\begin{aligned} e_{v_i} &= \prod_{i=1}^2 (e_i + t_2) + t_1 \sum_{i=1}^2 (e_i + t_2)^i \big|_{e_i=e} \\ &= (e + t_2)^2 + t_1 \sum_{i=1}^2 (e + t_2)^i \end{aligned}$$

(3-8)

$$\begin{aligned} P_F &= \prod_{i=1}^3 e_{vi} + d \big|_{e_{vi}=e_v, i=1,2,3} \\ &= e_v^3 + d \end{aligned}$$

(3-9)

Where:

P_F = Total system failure rate

e_{v_i} = Program version failure rate

e_i = Program module failure rate

Scheme 4: 2¹ NVP 3¹ RB (Figure 3-9)

$$\begin{aligned} e_v &= \prod_{i=1}^2 e_i + d \big|_{e_i=e} \\ &= e^2 + d \end{aligned}$$

(3-10)

$$\begin{aligned}
P_F &= \prod_{i=1}^3 (e_{v_i} + t_2) + t_1 \sum_{i=1}^3 (e_{v_i} + t_2)^i \Big|_{e_{v_i}=e_v, i=1,2,3} \\
&= (e_v + t_2)^3 + t_1 \sum_{i=1}^3 (e_v + t_2)^i
\end{aligned}
\tag{3-11}$$

Scheme 5: 3¹ RB 2¹ NVP (Figure 3-10)

$$\begin{aligned}
e_v &= \prod_{i=1}^3 (e_i + t_2) + t_1 \sum_{i=1}^3 (e_i + t_2)^i \Big|_{e_i=e} \\
&= (e + t_2)^3 + t_1 \sum_{i=1}^3 (e + t_2)^i
\end{aligned}
\tag{3-12}$$

$$\begin{aligned}
P_F &= \prod_{i=1}^2 e_{v_i} + d \Big|_{e_{v_i}=e_v, i=1,2} \\
&= e_v^2 + d
\end{aligned}
\tag{3-13}$$

Scheme 6: 3¹ NVP 2¹ RB (Figure 3-11)

$$\begin{aligned}
e_v &= \prod_{i=1}^3 e_i + d \Big|_{e_i=e} \\
&= e^3 + d
\end{aligned}
\tag{3-14}$$

$$\begin{aligned}
P_F &= \prod_{i=1}^2 (e_{v_i} + t_2) + t_1 \sum_{i=1}^2 (e_{v_i} + t_2)^i \Big|_{e_{v_i}=e_v, i=1,2} \\
&= (e_v + t_2)^2 + t_1 \sum_{i=1}^2 (e_v + t_2)^i
\end{aligned}
\tag{3-15}$$

3.3.4 Simulation Models and Modeling Results

There are many different types of simulation techniques [John88]. Simulation involves conducting experiments with a model in order to understand how a system will behave and obtaining numerical evaluations of the various operational strategies. In this study, all six schemes are simulated under different failure rates of the program modules, the acceptance tests, and the decision (voting) mechanisms. The calculation program was developed under the Lotus 123 environment.

The simulation results for the schemes shown in Figures 3-6 to 3-11 are presented in Figures 3-12 to 3-17. Test data ranges are chosen as:

$$e = 1\% \text{ to } 6\%$$

$$t = 1\% \text{ to } 6\%$$

$$d = 0.0001\% \text{ to } 0.0006\%$$

We assume that the decision module has a higher reliability than the testing module. Some observations that can be obtained by analyzing the output are the following:

Scheme 1 (Figure 3-12): 6¹ RB

Under the given test ranges of e , t , and d , the scheme failure rate $P_F = 0\% \text{ to } 1.3\%$ (Max. when $e = 6\%$, $t = 6\%$).

The 6¹ scheme is a pure RB scheme and its reliability is used as the reference for the other schemes. The 6¹ RB scheme is fairly reliable. If the average program module failure rate (e) is 1% and the test error rate is 1% then the system reliability is better than 99.95%. If the computer program module reliability dropped to

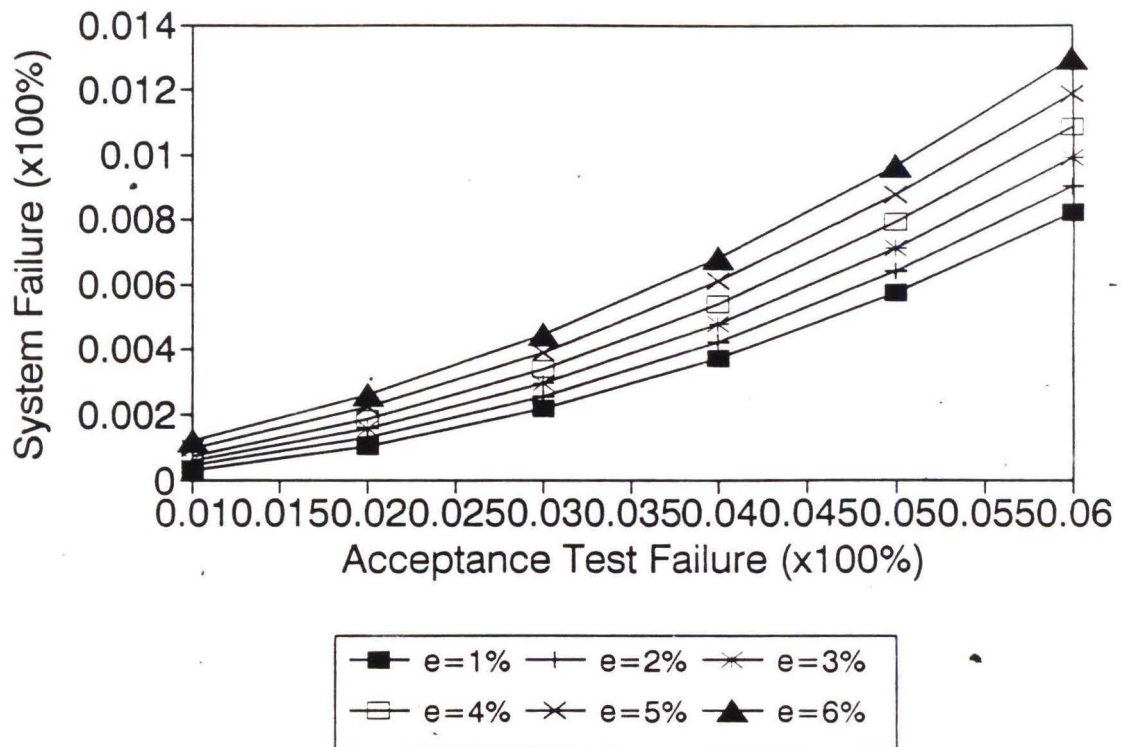
Scheme 1 (6^1) Failure Rates

Fig. 3-12 Failure Rates for Scheme (1)

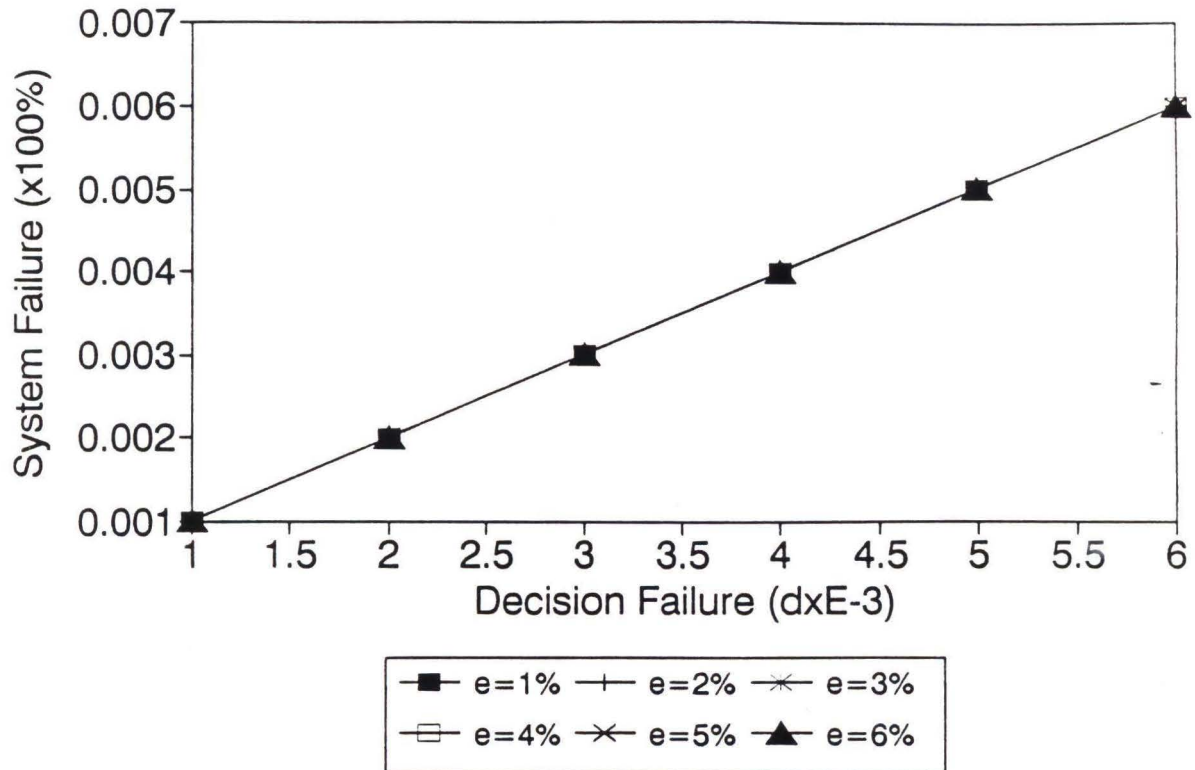
Scheme 2 (6^1 NVP) Failure Rates

Fig. 3-13 Failure Rates for Scheme (2)

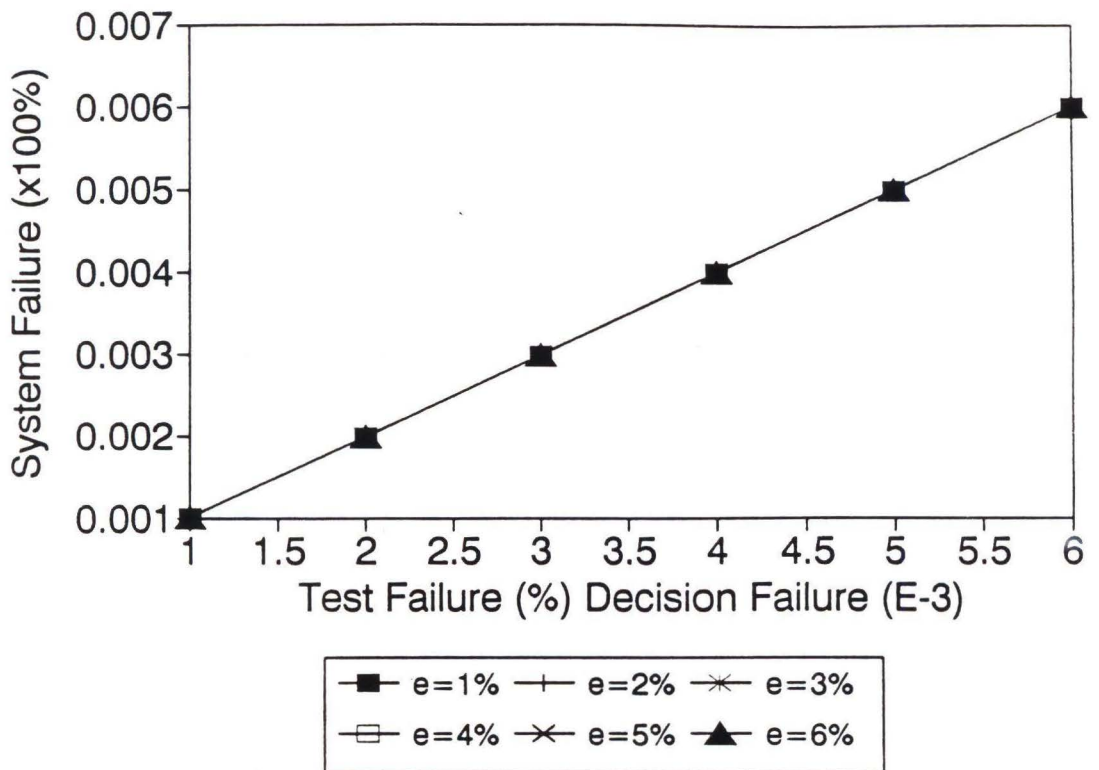
Scheme 3 (2^1 RB 3^1 NVP)

Fig. 3-14 Failure Rates for Scheme (3)

Scheme 4 ($2^1 NVP 3^1 RB$) Failure Rate

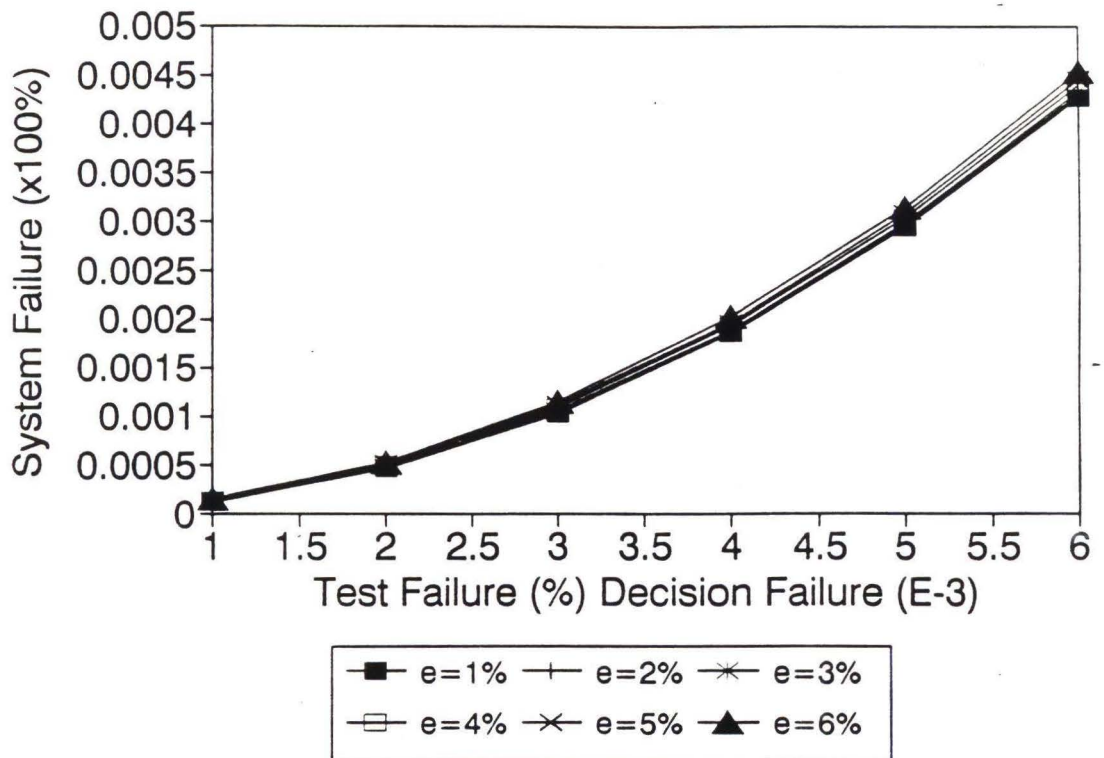


Fig. 3-15 Failure Rates for Scheme (4)

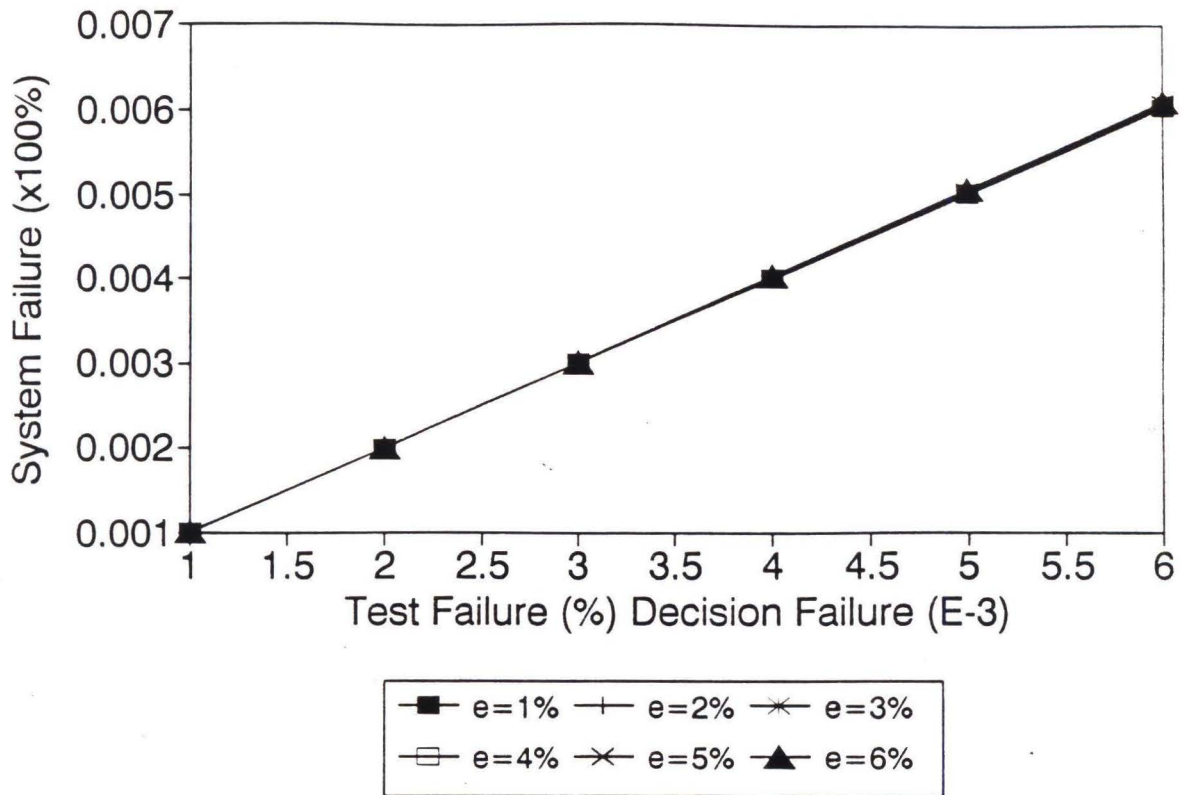
Scheme 5 (3^1RB2^1NVP) Failure Rate

Fig. 3-16 Failure Rates for Scheme (5)

Scheme 6 ($3^1 NVP 2^1 RB$) Failure Rate

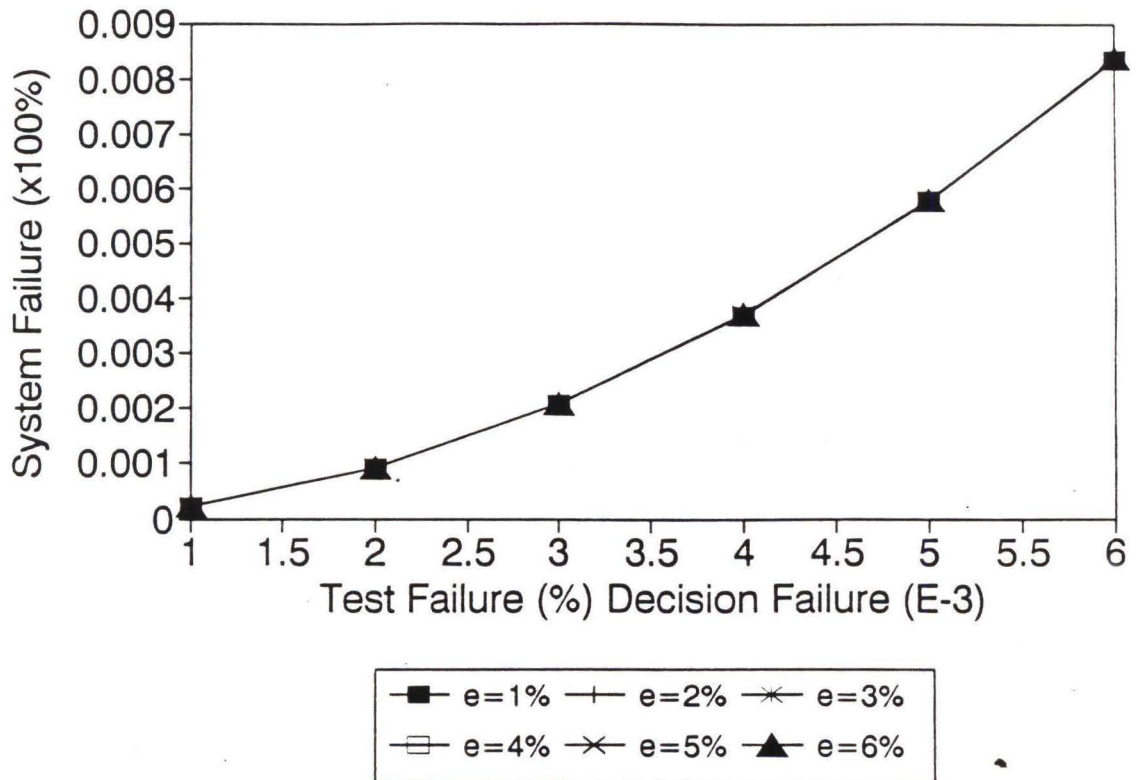


Fig. 3-17 Failure Rates for Scheme (6)

94% (6% failure rates) and the test error rate increased to 6%, the system reliability is still better than 98.6%. The system is not very sensitive to the program module failure rate and test failure rate: with a 6% difference in failure rates the system reliability decreases by 1.1%.

Scheme 2 (Figure 3-13): 6¹ NVP

The probability of system failure for the 6¹ NVP scheme depends strongly on the reliability of the voting mechanism. When the program module failure rate e varies from 1% to 6%, there is no clear effect on the system reliability. However, there is a clear relation between the system failure P_F and the voter error rate d . In other words, the NVP can tolerate not so reliable modules as far as it has a reliable decision mechanism.

Scheme 3 (Figure 3-14): 2¹ RB 3¹ NVP

Similar to the 6¹ NVP scheme, the system's failure rate is related to the decision failure rate d . Lower level redundancy is not necessary if a decision module with failure rate d is going to be used as a final determination of the scheme.

Scheme 4 (Figure 3-15): 2¹ NVP 3¹ RB

This scheme takes the advantages of both NVP and RB. Under the same testing data, the system's failure rate P_F is between 0.01% to 0.45%. The system's reliability is not very sensitive to the program module failure rates e .

Scheme 5 (Figure 3-16): 3¹ RB 2¹ NVP

When a decision/voting module is used as the final judgement of the scheme, it resembles the characteristics of NVP. In other words, the system's failure rate is strongly dependent on decision failure rate d .

Scheme 6 (Figure 3-17): 3¹ NVP 2¹ RB

The system reliability is very much independent of the program module failure rate e . It can be discovered from the plot that the system failure is related to decision or testing failure rates exponentially.

3.3.5 Discussion

Figures 3-18 and 3-19 show the comparisons of the six schemes when the program failure rates are $e=2\%$ and $e=6\%$. The testing failure rate t is set between 1% to 6% and the decision/testing failure rate d is assigned from 0.1% to 0.6%. The following are some observations from Figure 3-18 and 3-19.

- (1) Scheme 4 2¹NVP3¹RB has the best system reliability under the given testing data.
- (2) A pure RB or NVP scheme has higher system failure rates than most of the hybrid schemes.
- (3) Under the assumption that $d = 0.1t$, the schemes 2, 3, and 5 generate better system reliability than other three schemes when $t > 4\%$ and $d > 0.4\%$ approximately.

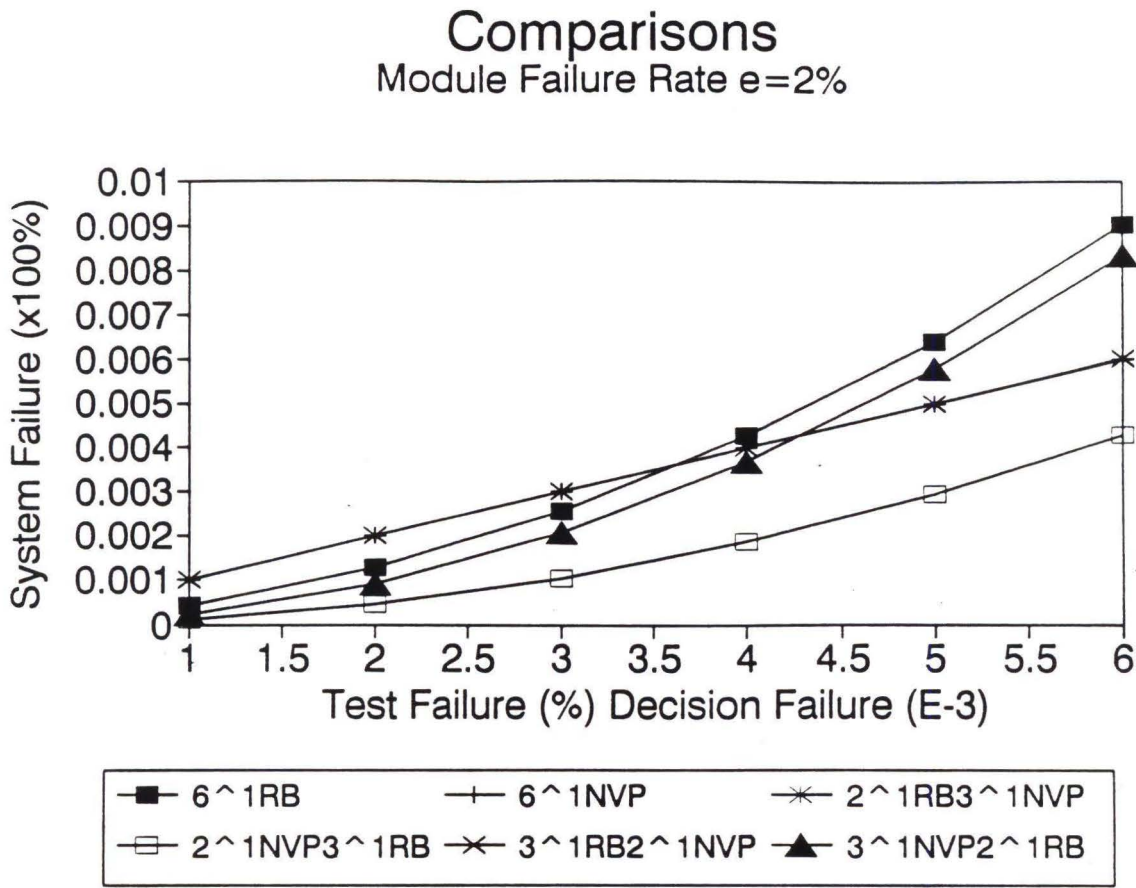


Fig. 3-18 Comparison of System Failure Rates ($e=2\%$, $d=0.1\%$ to 0.6%)

Comparisons

Module Failure Rate $e=6\%$

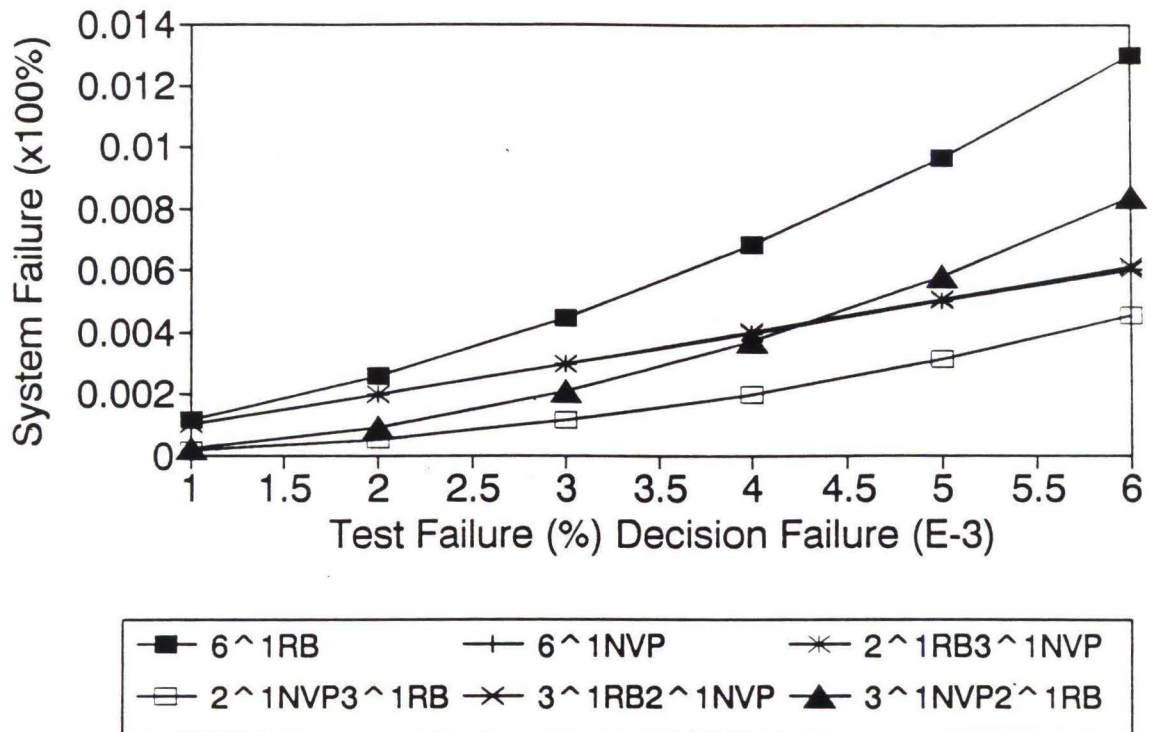


Fig. 3-19 Comparison of System Failure Rates ($e=6\%$ $d=0.1\%$ to 0.6%)

Figures 3-20 and 3-21 show the comparison of system failure rates of the six hybrid schemes with decision failure rate $d=1\%$ to 6% (Figure 3-20) and $d=0.01\%$ to 0.06% (Figure 3-21). It can be clearly seen that the system's reliability for a NVP dominated scheme, such as schemes 2, 3 and 5, will have a better system's reliability if $d < 0.1t$ or worse reliability if $d > 0.1t$ when compared with a RB dominated scheme.

[Scot87] proposed a reliability model to calculate the NVP system's reliability, as well as other mechanisms. The assumptions used in his model are:

- (1) The only type of error considered is that when all outputs disagree (called type one error by them).
- (2) Type two and type three errors, that is, when an incorrect output appears more than once (type 2) and errors in the voting procedure (type 3) are ignored.

Under those assumptions, the system failure rate P_F becomes:

$$P_{NV} = 1 - R_{NV} = 1 - \sum_{i=2}^{n_{NV}} \binom{n_{NV}}{i} (1-e_i)^i e_i^{n_{NV}-i} \quad (3-16)$$

Figure 3-22 shows the system failure rate when $e=1\%$ to 6% . Under Scott's model the NVP system reliability does not increase very large by using fault tolerance especially when e increases. According to Scott's model using the same way to define the type of errors, we have equation (3-17) from [Bell90]:

Comparisons

Module Failure Rate $e=6\%$

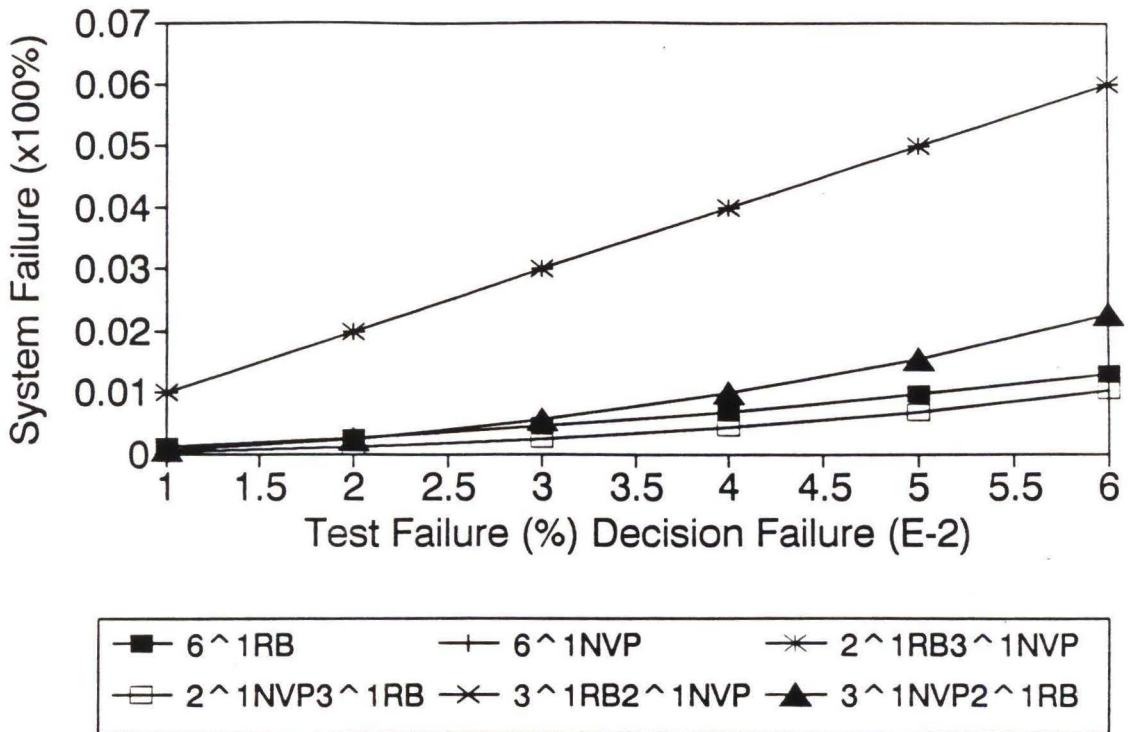


Fig. 3-20 Comparison of System Failure Rates ($e=6\%$, $d=1\%$ to 6%)

Comparisons

Module Failure Rate $e=6\%$

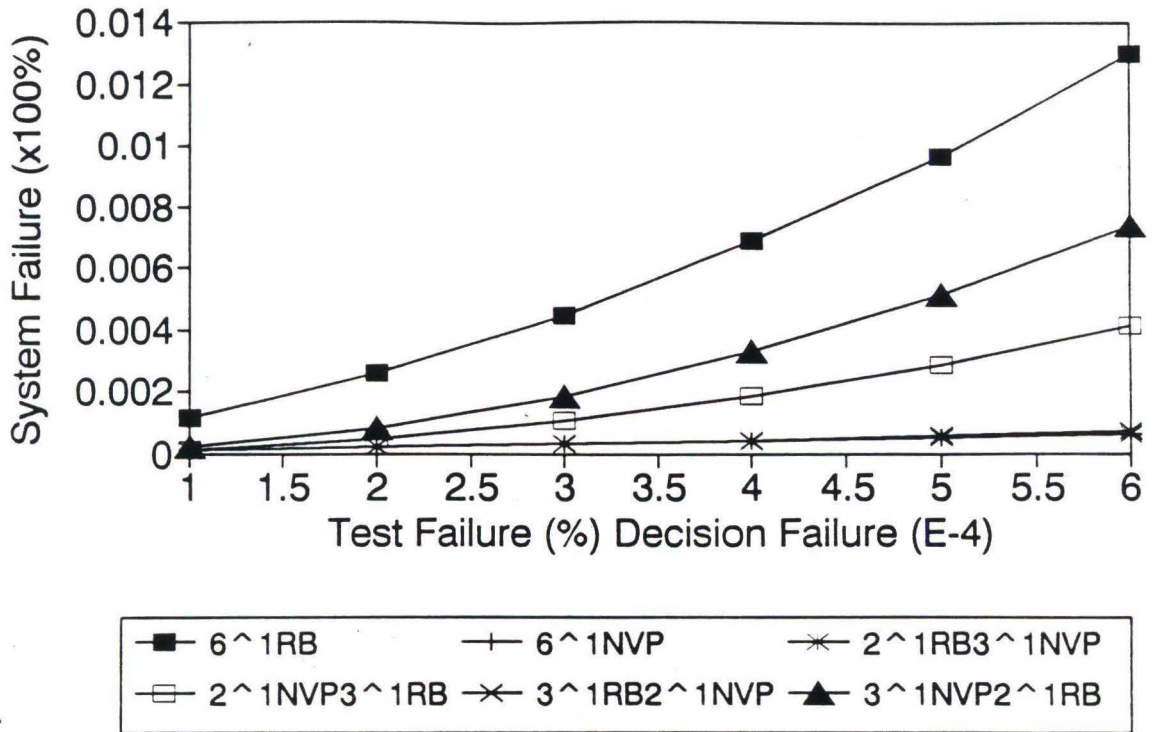


Fig. 3-21 Comparison of System Failure Rates ($e=6\%$, $d=0.01\%$ to 0.06%)

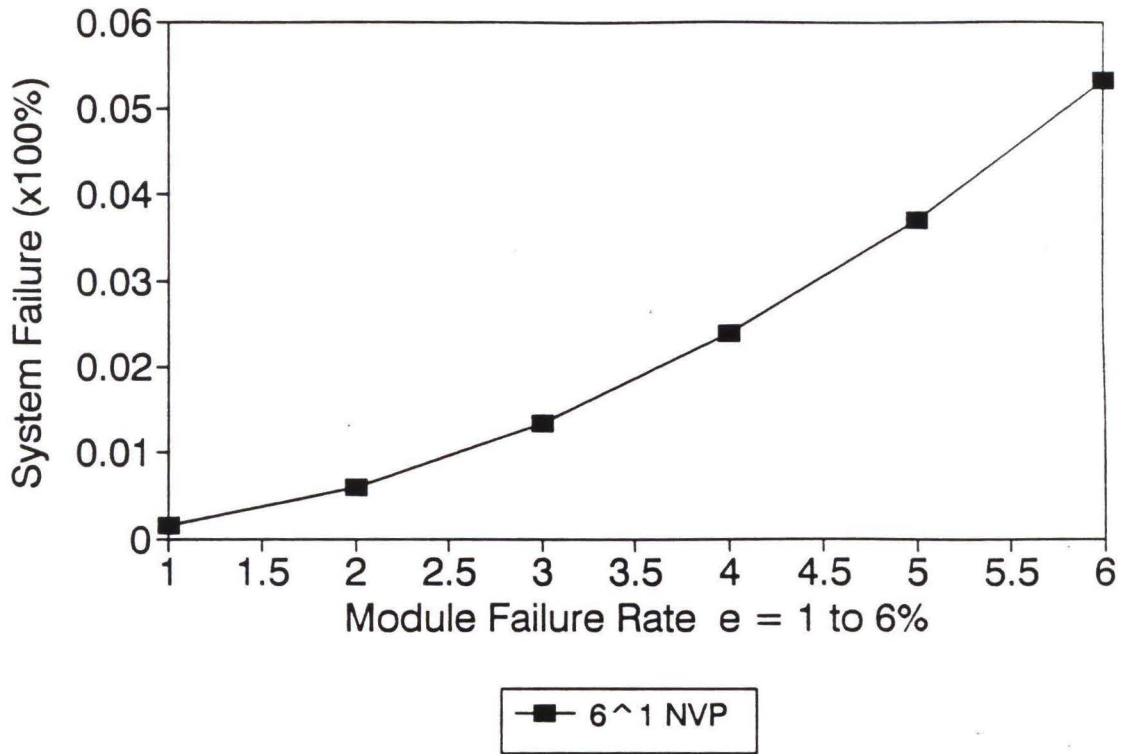
6^1 NVP (Scott's Model: $d=0$)

Fig. 3-22 NVP System Reliability by Using Scott's Model

$$P_{RB} = \prod_{i=1}^{n_{RB}} (e_{v_i} + t_{2i})$$

(3-17)

where:

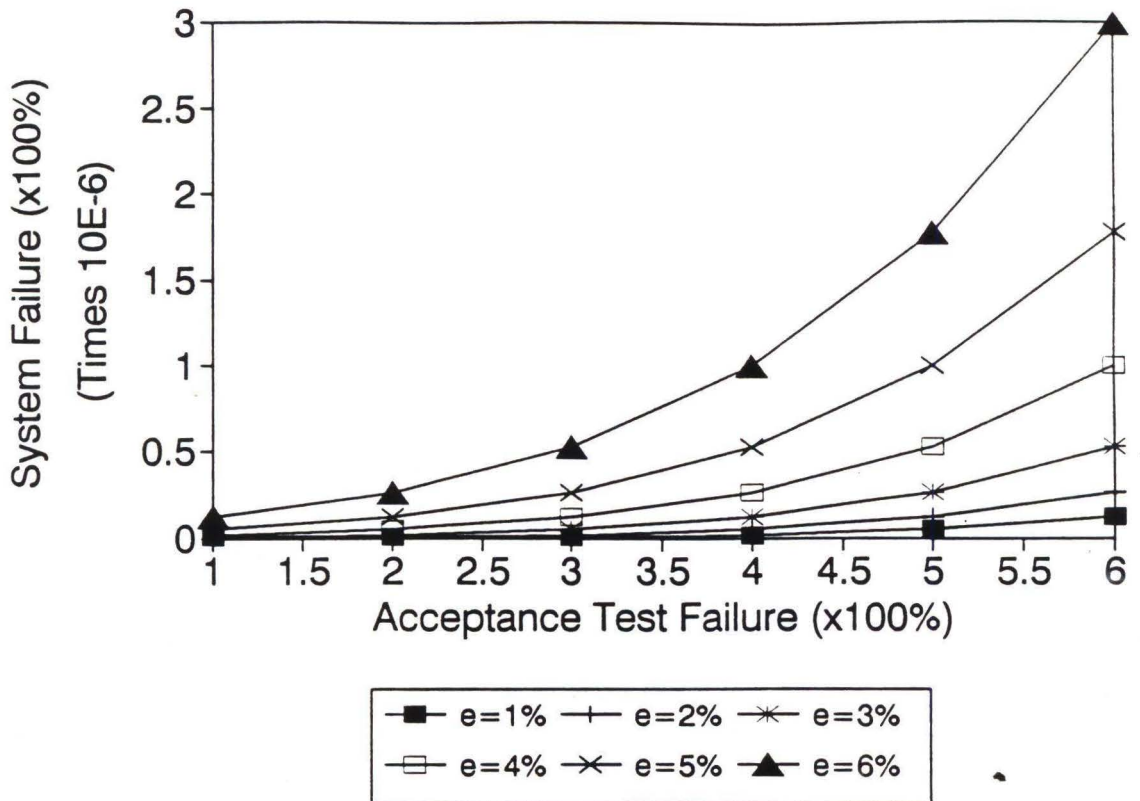
$t_1 = 0$ and t_2 is the only type of error of the acceptance test in RB.

Figure 3-23 shows the system failure rate when $e=1\%$ to 6% under the pure RB. Figures 3-24 and 3-25 show the comparisons of the system failure rates for all six hybrid schemes when Scott's model is used to calculate the failure rates of NVP modules. Except for the 6^1 NVP scheme, the other schemes yield similar system failure rates to the ones calculated by the mathematical model developed in this study.

[Scot83] proposed a consensus RB model which starts with an NVP scheme and if there is no output result an RB scheme is applied. That is also a combined NVP and RB scheme of the type used in this study. The difference is that the consensus RB combines NVP and RB within one level of redundancy and the hybrid scheme in this study applies these methods in two levels. The hybrid fault-tolerant systems proposed in this study generate better system reliability than the pure RB or NVP systems. The hybrid fault-tolerant scheme could be extended to more than two levels but the high number of versions makes this idea impractical.

3.4 Summary

A simulation model has been used to study the behavior of hybrid fault-tolerant schemes. Six fault-tolerant programs were constructed using Recovery Block

6^1 RB (Scott's Model: $t_1=0$)Fig. 3-23 RB System Reliability with Error Type $t_1=0$

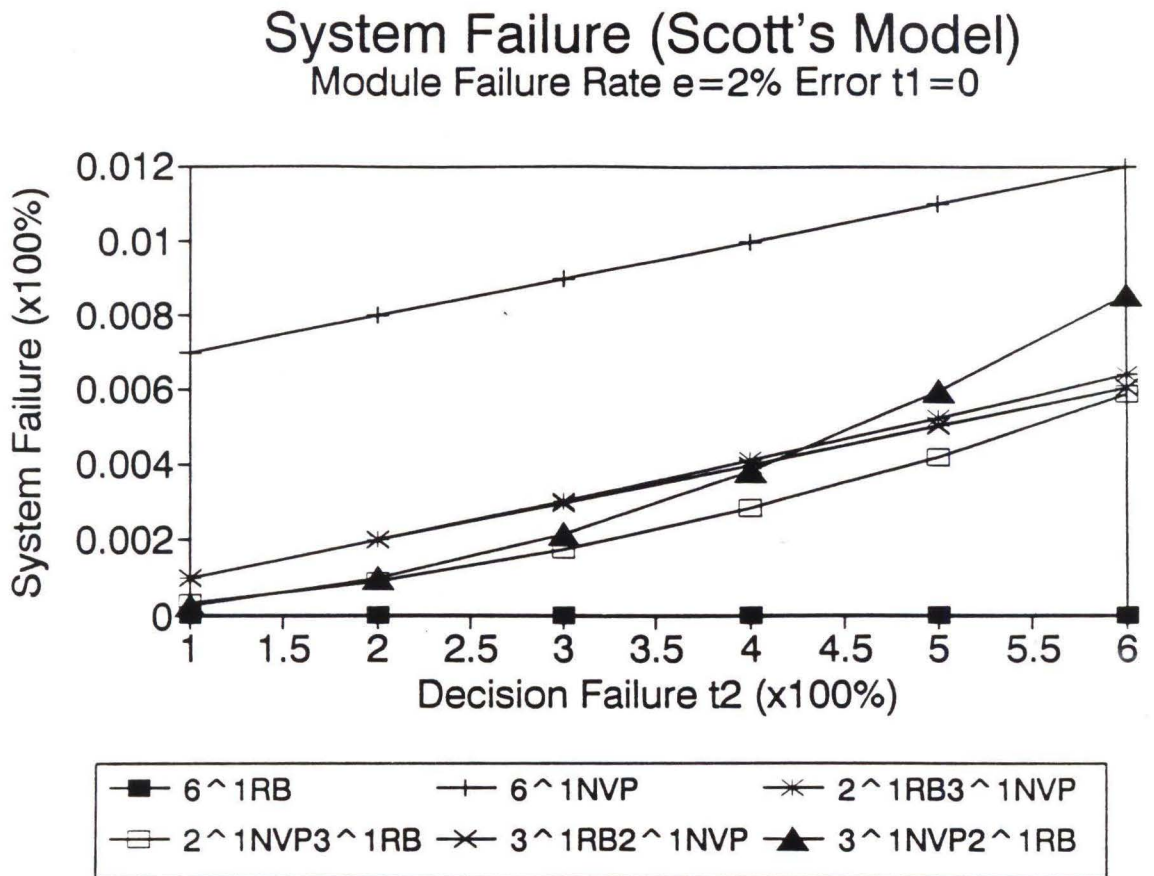


Fig. 3-24 Comparisons of System Failure Rates Using Scott's Model ($e=2\%$)

System Failure (Scott's Model)

Module Failure Rate $e=6\%$ Error $t_1=0$

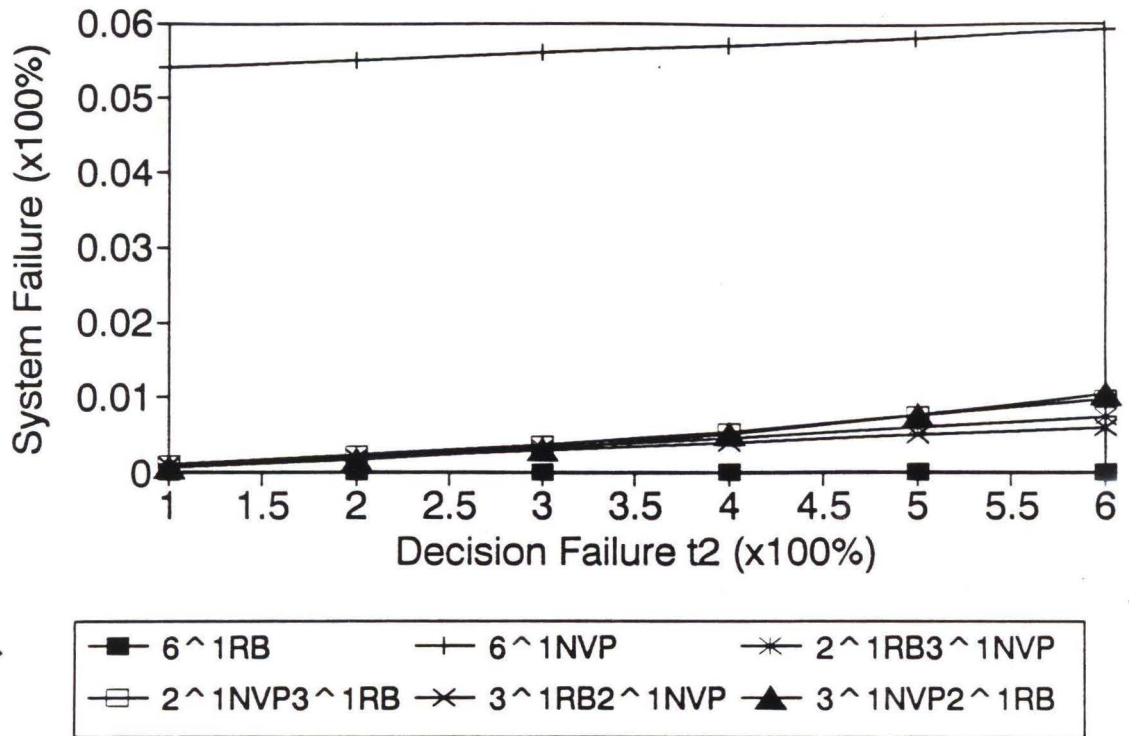


Fig. 3-25 Comparisons of System Failure Rates Using Scott's Model ($e=6\%$)

and N-Version Programming including their combinations. The study contains four main parts: development of the fault-tolerant schemes, fault tree for these schemes, probability models, and simulation.

A sample fault-tolerant system which consists of six redundant versions was utilized for building the various schemes and for numerical testing. Four hybrid configurations of the basic RB and NVP were compared with the original RB and NVP design. The results show that the combined schemes can produce better system reliability than that of the original RB and NVP methods. Also studied were the system reliability under different schemes and different probabilities of program failure, acceptance testing failure and decision failure. Guidance to select the best system configurations was also presented.

CHAPTER IV

HYBRID FAULT-TOLERANT SYSTEM DESIGN

WITH COST CONSTRAINTS

4.1 Hybrid Fault-tolerant System

Two important issues which are related to the design and analysis of fault-tolerant software are the reliability and the cost associated with various fault-tolerant mechanisms. This chapter presents two heuristic methods for the design of hybrid Recovery Block (RB) and/or N-Version Programming (NVP) systems under cost constraints. The first one is an homogeneous model where all program module's failure rates, acceptance test or voter's failure rates and costs are the same. The second algorithm deals with a general model in which all failure rates and costs are variable. As we saw in Chapter III, hybrid RB and NVP schemes, can further improve reliability. Design diversity and redundancy, on the other hand, escalate the cost of software design and development. [Bell90] proposed an optimization algorithm to design RB and NVP systems under total cost constraints. In this chapter, we propose two heuristic methods for the design of fault-tolerant software using two levels of hybrid RB and NVP schemes under cost constraints.

The cost of fault-tolerant software includes design, implementation, testing, and operation costs. In general, the reliability of software can be increased by adding more redundant programs or units. However, in practice cost limits are usually imposed in designing such a system. [Bell91] reported optimization models for systems using separated RB and NVP systems under constraints on the total cost.

System Failure Rates of the Hybrid System

In general, using the notation of Chapter III, the failure rates for the RB and NV module can be written as (Equations 3-3 and 3-4):

$$P_{RB} = \prod_{i=1}^{n_{RB}} (e_{v_i} + t_2) + t_1 \sum_{i=1}^{n_{RB}} (e_{v_i} + t_2)^i \quad (4-1)$$

$$P_{NV} = \prod_{i=1}^{n_{nv}} e_{v_i} + d \quad (4-2)$$

The probability of system failure P_F is:

$$P_F = f(P_{RB}, P_{NV}) \quad (4-3)$$

In this equation, P_F is a function f defined by the configuration of the hybrid system.

Since P_F is the probability of system failure, then $(1 - P_F)$ is equal to the system reliability. The cost models of the proposed schemes are then a nonlinear function as follows:

Objective:

$$\max_{j \in S} (1 - P_{F_j})$$

(4-4)

Subject to:

$$\begin{aligned} \sum_{i=1}^{n_{ei}} C_{ei} + \sum_{j=1}^{n_{rb}} C_{tj} + \sum_{k=1}^{n_{nv}} C_{vk} &\leq C \\ P_{F_j} &\geq 0 \\ C_{ei} &\geq 0 \\ C_{tj} &\geq 0 \\ C_{vk} &\geq 0 \end{aligned}$$

(4-5)

where:

P_{F_j} = failure rates of the scheme j

C = the total resources available

C_{ei} = resources needed for program module i

C_{vi} = resources needed for voting module i

C_{ti} = resources needed for testing module i

S = the complete set of possible schemes can be constructed

s = the scheme that gives the optimal system reliability

j = scheme index

The configuration which generates the best system reliability within the total resource limits will be the optimal solution.

Two methods are presented here which will generate optimal fault-tolerant system structure under given cost limits. The first method developed is for homogeneous fault-tolerant systems in which all the program modules have the same reliability and cost. A more general method applied to systems in which the program modules as well as the voting and acceptance test all have dissimilar reliability and associated costs. Both algorithms are based on two results from our previous study (Chapter III). The relevant facts are the following:

- (1) A fault-tolerant system which consists of more program modules has higher reliability. This is true for both RB and NVP schemes if the modules have been carefully tested. Since the probabilities of the program failures are less than one ($e_i, t, d < 1$), more multiplication of them will generate a smaller number.
- (2) In general, software system reliability is improved by using more test and vote program modules. As our previous study shows that for a system consist of same number of functional redundant programs, the systems with least testing and voting programs have lowest reliability. The system's reliability can be improved by using more testing and voting modules.

4.2 Fault-Tolerant System Design with Equal Program Reliability and Costs

Based on the observations stated above, the algorithm calculates possible system configurations under the cost limit. It first confines the maximum number of

redundant programs supported by the cost limit. The system reliability will be used as the lower bound. Then more voting and testing modules are added into the scheme. Program modules will be traded for the testing and voting modules if necessary. The configuration with the highest reliability will be selected. Following is a step by step description of the method.

4.2.1 Method I: Symmetrical Balanced Fault-Tolerant System Design

Step 1: Calculate the total number of program modules, n_p .

$$\begin{aligned} n_p &= \text{max integer (Total cost / cost for each program module)} \\ &= \text{max int } [C/C_i] \end{aligned}$$

(The assumption used here is that all the program modules have the same cost $C_{ei}=C_i$).

Step 2: Calculate the remaining resources C'

$$C' = (C - C_i \cdot n_p)$$

C' represents the resources remaining for the voting and testing programs when a number n_p of programs are utilized.

Step 3: Select hybrid scheme

If $C' \geq r \cdot C_t$ or $C' \geq r \cdot C_v$, go to step 4. This indicates that there are enough resources remaining for the testing and voting programs after the resources have been committed to the n_p number of functional programs. (Here r is the iteration number which in initially 1).

Else, $n_p = n_p - 1$, repeat step three.

If there are not enough resources left for the voting and/or testing programming, then one of the functional programs has to be traded for the testing/voting programs.

Step 4: Calculate the system reliability

Calculate the system reliability under the given number of testing and voting modules. This is accomplished by using equations (1), (2), and (3). There could be several different configurations under a given number of program modules, acceptance test and voting modules.

Step 5: Stopping Rule

If, $r < \frac{1}{2} \cdot n_p + 1$, then, $r = r + 1$, go to step 3;

else, go to step 6.

The stopping rule is used to decide when to continue or terminate the iteration procedure. It is obvious that in the hybrid system, the number of testing and voting programs should not exceed half of the number of functional programs. This assumes that there should be at least two functional programs in a RB or NVP structure.

Step 6: Select the scheme with the maximum system reliability.

4.2.2 Illustration of Method I

The following example illustrates this method. It is assumed that all programs and their testing modules have the same reliability and costs. The input data are:

Cost of the program module $C_e = 15$ (thousand dollars)

Cost of the testing module $C_t = 85\% \cdot C_e = 12.75$ (thousand dollars)

Cost of the voting module $C_v = 10\% \cdot C_e = 1.50$ (thousand dollars)

Total amount of resources available $C = 120$ (thousand dollars)

Probability of program module failure $e = 5\%$

Probability of testing module failure $t = 2\%$

Probability of voting module failure $d = 0.2\%$

Iteration 1:

Step 1: Total number of program modules $n_p = \lceil C/C_e \rceil = \lceil 120/15 \rceil = 8$ (modules)

Step 2: if $n_p = 8$, remaining $C' = 0$,

Step 3: There are no resources for testing and voting, we have to adjust the n_p and R as follows:

$$n_p = n_p - 1 = 7 \text{ (modules)}$$

$$C' = (C - C_e \cdot n_p) = 120 - 15 \cdot 7 = 15 \text{ (unit \$)}$$

Step 4: Calculate the reliability for this number of modules.

Reliability for 7¹ RB (Scheme 1)

Reliability for 7¹ NVP (Scheme 2)

Step 5: $r = 1 < \frac{1}{2} \cdot n_p + 1 = 4.5$, go to step 3.

Iteration 2:

Step 3: $n_p = n_p - 1 = 7 - 1 = 6$

Step 4: $C' = 120 - (6 \cdot 15) = 30$ (unit \$)

Step 5: Calculate the systems reliability under the possible system configurations.

There are:

2^1 NVP 3^1 RB ($n_p = 6$, $n_t = 1$, $n_v = 3$) (Scheme 4)

3^1 RB 2^1 NVP ($n_p = 6$, $n_t = 2$, $n_v = 1$) (Scheme 5)

3^1 NVP 2^1 RB ($n_p = 6$, $n_t = 1$, $n_v = 2$) (Scheme 6)

Note: Scheme 3 (2^1 RB 3^1 NVP) is not feasible.

Go to step 3 until $r \leq \frac{1}{2} \cdot n_p + 1$.

The final systems reliability and their costs are as shown in Table 4-1.

Step 6: Select the system structure with the least failure probability. Scheme 4 will be selected. The scheme consists of three NVP versions each of which has two functional programs and a voting mechanism. Then those three NVP versions are combined using a RB testing module. The reliability of the system is 99.95% under the given conditions. Figure 4-1

Table 4-1: System Reliability of Different Hybrid Fault-Tolerant Systems

Schemes	Total Cost	System Reliability
1	117.75	99.78
2	106.50	299.80
4	109.50	99.95
5	116.50	99.80
6	111.00	99.91

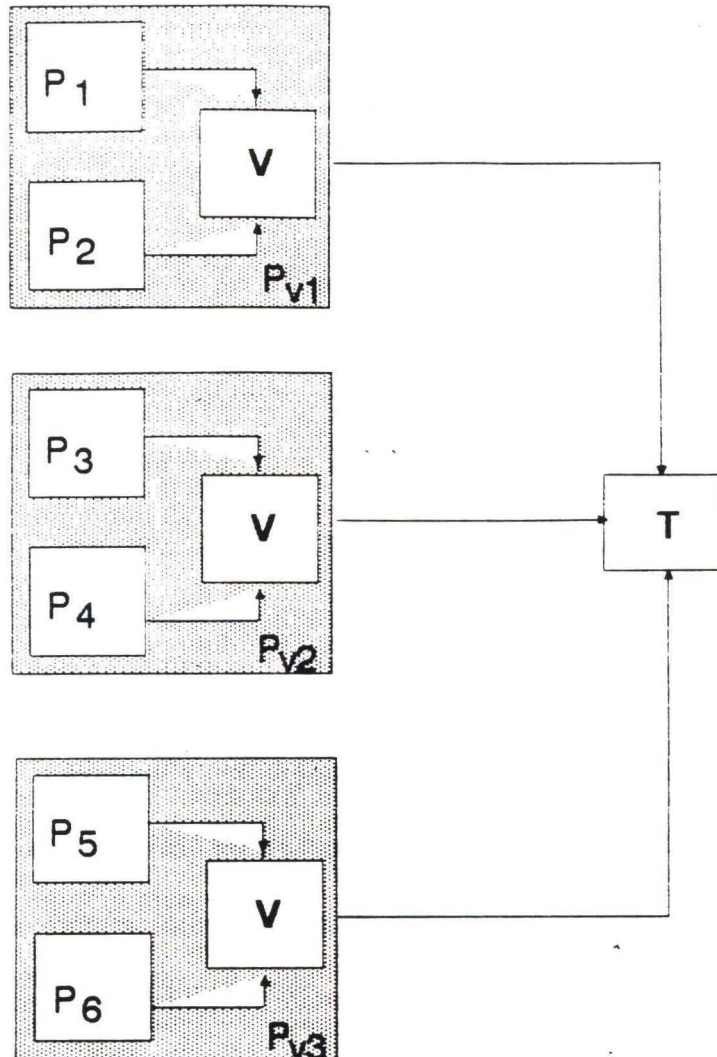


Fig. 4-1 Homogenous Hybrid Fault Tolerant Software Scheme

shows the structure of the hybrid design and Table 4-1 shows the result of the variable schemes (those scheme numbers related with same method in Chapter III).

4.3 Hybrid Fault-Tolerant System Design with Cost Constraints: General Model

The previous method assumes that all modules have the same reliability and cost. We now show a generalization of the previous algorithm which handles the situation when costs and reliability are variable.

4.3.1 Fault-Tolerant System Design with Different Program Reliability and Costs

The two assumptions used in the previous method are still valid. They allow us to simplify the method into two stages, one to design the best system structure using an iterative procedure and the second is select the best combinations of program modules. This method selects tests and voters first, then the program versions, while method I selects version first followed by the selection of tests and voters.

Step 1: Initialization:

The procedure starts with one voter or one test module. According to the observation that the reliability of the voting and testing has the biggest influence on the system reliability, one should select the voting or testing module with the highest reliability to start the procedure.

Initially, n_t or $n_v = 1$

Step 2: Remaining Resources for program modules:

$$C' = C - \sum_{i=1}^{n_k} C_{ti} - \sum_{i=1}^{n_v} C_{vi}$$

Step 3: Program Module Selection Rules:

Assume there are n_p program modules and that the costs of the program modules C_{ei} are different. Each program module can be used only once. The procedure of selecting the program modules is given by the following integer program module:

$$\min \prod_{ies} e_i$$

Subject to:

$$\sum_{ies} C_{ei} \leq C'$$

$$0 \leq e_i \leq 1$$

$$C_{ei} \geq 0$$

The program modules i can now be selected so as to give the best system reliability under the cost constraints.

This is a nonlinear programming problem. In this particular case, it can be converted into a linear programming problem by the following procedure.

Since:

$$\begin{aligned}\ln\left(\prod_{i=1}^{n_{ei}} e_i\right) &= \ln e_1 + \ln e_2 + \cdots + \ln e_n \\ &= \sum_{i=1}^{n_{ei}} \ln e_i\end{aligned}$$

Then:

$$\min \prod_{i \in S} e_i$$

is equivalent to:

$$\min \sum_{i=1}^{n_{ei}} \ln e_i$$

Letting: $x_i = \ln e_i$,

the original problem becomes a linear problem:

$$\min \sum_{i \in S} x_i \quad ,$$

subject to:

$$\sum_{i \in S} C_{ei} \leq C'$$

$$x_i \geq 0 \quad .$$

$$C_{ei} \geq 0$$

Step 4: Hybrid Rules:

It can be proved that the system reliability is maximized by building a balanced hybrid structure. In other words, the hybrid system should have versions (RB or NVP or a combination of them) with similar reliabilities.

Mathematical Model:

Given:

e_i = the probability of failure for the individual program module i .

$0 \leq e_i \leq 1, \quad i \in S.$

t_i = failure rates of the testing module

d_i = failure rates of the voting module

The system failure rate is minimized by:

$$\min \sum_{i=1}^{m_p} (P_{mi} - P_{mi-1})$$

Where:

m_p = number of versions in the hybrid system

P_{mi} = failure rates of the version i , P_{mi} is calculated according to the equations 1, 2, and 3 given before.

Step 5: Calculate the system reliability accordingly.

Step 6: Stopping Rule:

if

$$r \leq \frac{1}{2} \cdot n_p + 1$$

then go to step 7 else go to step 8.

Step 7: Adding Rules:

When the remaining C' is less than the cost for adding a test or a vote module, one program module will be selected to be deleted from the system. In order to do so, two ratios are considered here, the cost efficiency ratios for program module and the testing/voting module.

In general, since the reliability of the testing and voting module are vital to the system reliability, a testing/voting module with the highest reliability usually will be added to the system. The choice of adding a testing or voting module depends on the initial selection of the type of decision module. We assume that in a two layer structure, if one layer is RB then the other layer is NVP, i.e. one voter is always combined with a number of acceptance tests to form a hybrid system; and one acceptance test is always combined with number of voters to form a hybrid system. The program module with the least cost efficiency and enough cost to cover the added testing or voting module will be removed from the previous design.

Program cost efficiency:

$$E_{e_i} = (1 - e_i) / C_{e_i}$$

Testing/Voting efficiency:

$$E_{t_i} = (1 - t_i) / C_{t_i} ,$$

or

$$E_{v_i} = (1 - v_i) / C_{v_i} .$$

Addition Rules:

$$\max (1 - t_i) , \quad \text{or}$$

$$\max (1 - v_i) \quad \text{for all } i, i \text{ is the testing/voting module index.}$$

$$i \in (\text{remaining testing or voting modules})$$

Deletion Rule:

$$\min E_{e_i}$$

$$\text{for } i \text{ that } C_{e_i} \geq C_{t_s} \text{ or } C_{v_s}$$

Go to step 2.

Step 8: Select the design with the maximum system reliability.

4.3.2 Illustration of Method II

The following example illustrates the proposed method. Table 4-2 shows the failure rates and costs of the modules. It is assumed that the voter has higher reliability and less cost when compared with the acceptance test.

The procedure to solve the problem is quite lengthy, only the final results are presented here. Four iterations generate four system designs according to the given

Table 4-2: Input Data for Illustration II

Data for the Illustration Example II						
	1	2	3	4	5	6
Cost of Program Module	10	12	14	16	18	20
Failure Rates of Program Module	.06	.05	.04	.03	.02	.01
Cost of Testing Module	10	12	14			
Failure Rates of Testing Module	.05	.03	.01			
Cost of Voting Module	1	2	3			
Reliability of Voting Module	.005	.003	.001			

data. Figures 4-2 through 4-5 show the system structure. The first design is a pure NVP design with a system failure rate of 5.01% . Design number two is a hybrid scheme. Three program modules with a testing module form a RB version. Two of such versions are then connected with a voter to form a NVP scheme. This design gives a failure rate of 5.22%. A third design is generated in the form of a hybrid NVP with RB. Three program modules with a voter form an NVP. Two of those NVPs form a RB scheme. The best design consists of three voters and a test. Each two of the six programs with a voter forms an NVP. Three of those NVPs are then linked using an RB scheme. This design yields the lowest failure rate of 1.69%. Considering the average program failure rate is 3% to 4%, the hybrid design made a big impact in terms of improving the system reliability. Table 4-3 summarizes these results.

4.4 DISCUSSION

The mathematical form of the hybrid fault-tolerant system design is in a rather complicated nonlinear programming form. It may be very difficult to obtain an analytical optimal solution. Therefore, using approximate methods are a practical and more efficient approach.

In a realistic environment, method II might be more useful than method I since the assumption of equal cost and reliability for all the modules is not realistic. However, it does supply a simple approach to design the hybrid system.

Method II is a more complex procedure. The method applies several linear and nonlinear programming methods to achieve some local optimization goals. It should be brought to our attention that comparing the two examples used in this thesis, method

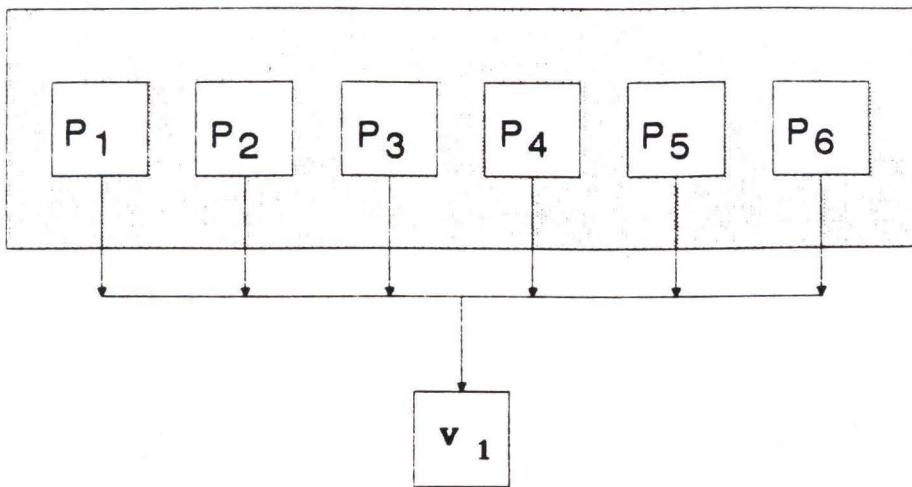


Fig. 4-2 Hybrid Fault Tolerant Software Scheme (1)
System Reliability = 94.99 %
Total Cost = 91 (unit of dollars)

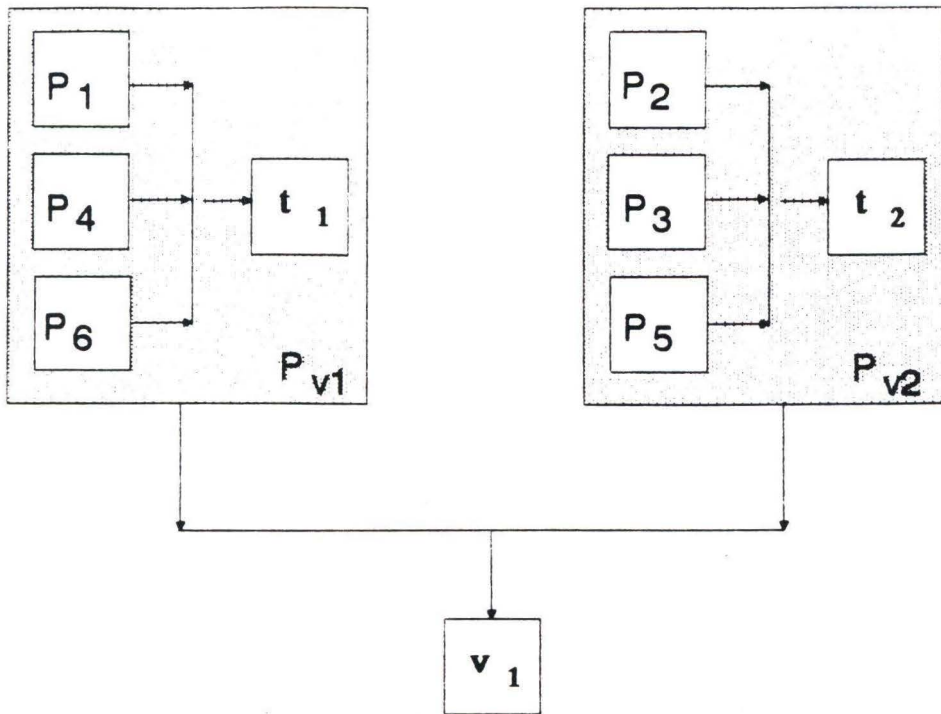


Fig. 4-3 Hybrid Fault Tolerant System Scheme (2)
System Reliability = 94.78 %
Total Cost = 113 (unit of dollars)

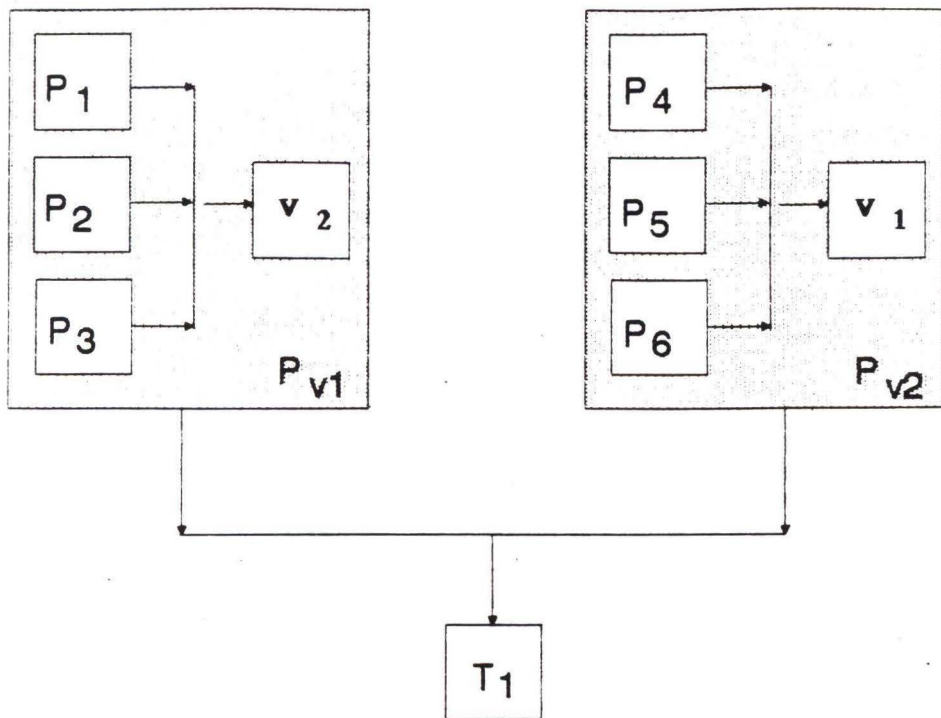


Fig. 4-4 Hybrid Fault Tolerant System Scheme (3)
System Reliability = 94.48 %
Total Cost = 103 (unit of dollars)

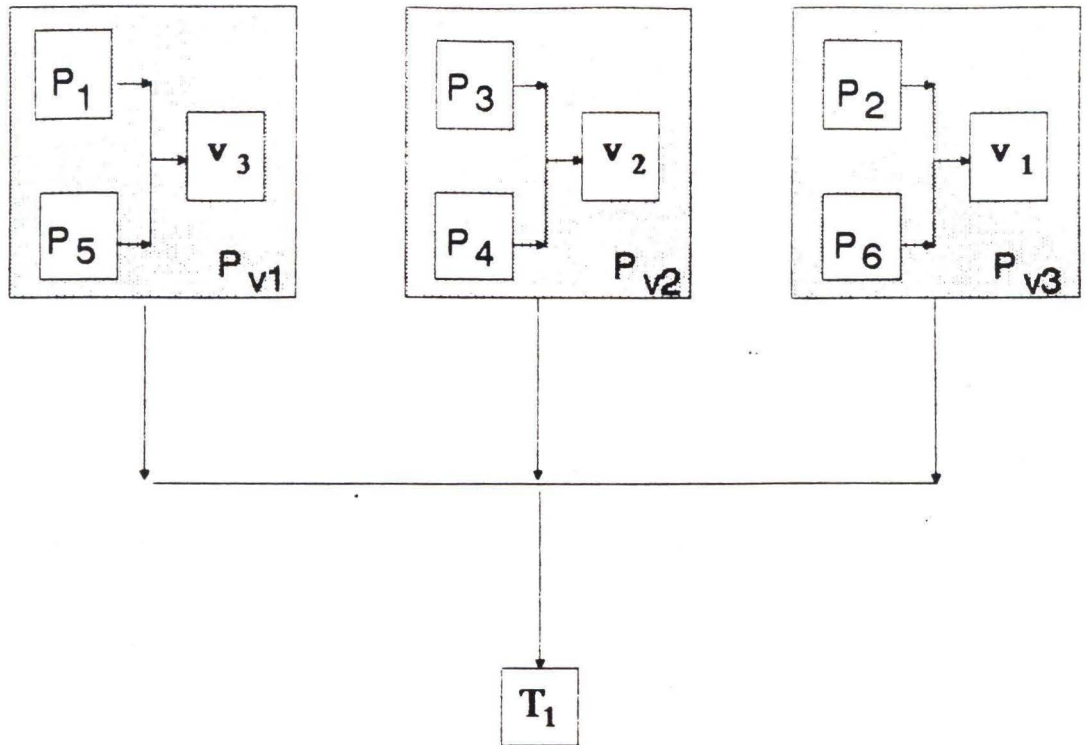


Fig. 4-5 Hybrid Fault Tolerant System Scheme (4)
System Reliability = 98.3 %
Total Cost = 106 (unit of dollars)

Table 4-3: Feasible Solutions of Illustration II

Feasible Solutions for Illustration II			
Solution No.	Hybrid Structures	System Reliability	Cost
1	6 ¹ NVP	0.9499	91
2	3 ¹ RB & 2 ¹ NVP	0.9478	113
3	3 ¹ NVP & 2 ¹ RB	0.9448	103
4	2 ¹ NVP & 3 ¹ RB	0.9831	106

II generates a better hybrid system compared to method I under similar circumstances. This heuristic method does not necessarily generate a system with maximum possible reliability. An analytical explanation for this result requires for further study.

[Bell91] proposed an approach to optimize system reliability with cost constraints. In his approach, system reliability is calculated through a nonlinear program and the optimal solutions obtained through exhaustive searching. Under our different assumptions, the method proposed in this study is much simpler.

4.5 SUMMARY

In this study, we proposed two algorithms for design hybrid RB and NVP fault- tolerant systems. Both procedures were executed in an spreadsheet environment. Provided with input data such as the reliability and cost of program modules as well as testing and voting modules, these procedures will determine the hybrid structures with highest system reliability within the cost constraints. In this thesis we showed examples with two layer hybrid configurations. In fact the number of layers could be more than two, although in general this is not a practical approach because of the high number of versions required.

CHAPTER V

MODELING OF HIERARCHICAL N-VERSION SOFTWARE FAULT-TOLERANT SYSTEMS

Wu [WuJi91] presented a hierarchical N-Version method where a problem is viewed as a set of objects which can be hierarchically organized into several levels. The reliability of hierarchical N-Version fault-tolerant software is now studied using a reliability modeling software called ARIES 82 [ARIE91]. Simulation models are used here to test the system reliability and performance of that approach. The system behavior under various failure patterns is also analyzed.

5.1 Hierarchical N-Version Programming

In general, a software system can be modeled as the superposition of four levels:

Level 1	application software
Level 2	module
Level 3	procedure
Level 4	data structure

The reliability of the whole application depends on the reliability of the subsystems at each level. Reliable subsystems can enhance the reliability of the whole system. By implementing hierarchical NVP, it is also possible that the cost of the software development

can be reduced. One can spend more resources on the subsystems which have the least reliability rather than on the entire system.

5.2 ARIES 82 Reliability Modeling Software

The ARIES (Automated Reliability Interactive Estimation System) 82 [ARIES91] was developed to assist designers of fault-tolerant systems. ARIES 82 provides a general mathematical framework of analysis which allows extensions to new models and new classes of systems.

ARIES 82 is a set of more than 100 C-language procedures developed by a research group at the Department of Computer Science, UCLA. The system is capable to model transient fault recovery, graceful degradation, off-line repair, periodic renewal, as well as user defined subsystems. The system supports seven types of systems:

- (1) Closed FT systems
- (2) Closed FT systems with transient fault recovery
- (3) Mission-oriented repairable systems
- (4) Repairable systems with transient fault recovery
- (5) Repairable systems with restart
- (6) Periodically renewed closed FT systems
- (7) User defined systems

The functions of ARIES 82 include three groups: system/subsystem configuration functions, reliability analysis functions, and system utility commands. The notation used in building system models and output analysis is as follows:

- $Y[0]$ Initial number of active program modules in NVP
- $Y[1]$ Minimum number of faultfree program modules
- D Number of degradations allowed
- λ Probability of failure for program module

5.3 Simulation Models

As said earlier, a software system (whole application) can be divided into three levels: the data structure level, the procedure level, and the module level. The systems reliability can be improved by utilizing N-Version Programming at one, two, or all three levels, and these simulation studies intend to evaluate this using three models.

Model one uses NVP on all three levels; model two applies NVP on two levels; and model three applies NVP to one of the three levels. The simulation type specified by ARIES has been selected as type 7 systems which corresponds to closed fault-tolerant systems. In this particular case spare module recovery is not considered. The descriptions of those models are given below.

Model One: N-Version Programming on all three levels

Assume there are three program modules at each level (number of initial active program modules $Y[0]=3$). Only of the three program modules can fail at a time ($D=1$). The minimum number of faultfree program modules $Y[1]$ is equal to $Y[0] - D = 2$. The failure rates λ are between 1% to 25%. The failure rates are too high for practical usages. They have been exaggerated here to show the trends of the model. The input parameters are as follows:

$$D = 1$$

$$Y[0] = 3$$

$$Y[1] = 2$$

$$\lambda = 1\% \text{ to } 25\%$$

Model Two: N-Version Programming on two of three levels

The two NVP sublevels have the following input parameters:

$$Y[0] = 3$$

$$Y[1] = 2$$

$$D = 1$$

The level without redundant programming has the following parameters:

$$D = 0$$

$$Y[0] = 1$$

$$Y[1] = 1$$

Model Three: N-Version Programming on one of the three levels

This model studies the hierarchical NVP by applying NVP on one of the three levels. For instance, NVP is only applied to the data structure level, while the procedure and module levels do not use any fault-tolerant programming.

The system is configured as having three ($n=3$) NVPs at one level and the other two have one program each. The failure rate of an active module, λ , is set from 1% to 25%. The maximum number of failed modules is set to one.

Using the ARIES 82, the system reliability for all three models is obtained. Figure 5-1, 5-2. and 5-3 show the input and output values of the models one, two and three, respectively.

5.4 Analysis of Results

Figure 5-4 shows the system reliability for all three models. As we can see from the figure the systems reliability follows closely a linear pattern, i.e. when program module failure probability (λ) increases, the system reliability decreases.

Figures 5-4 also shows the quantitative relations of the hierarchical NVP vs. regular NVP. It is interesting to notice that the effect of implementing one more level NVP is similar to reduce the model failure rates by half. For instance, in Figure 5-4, the system reliability is equal to 85% when probability of module failure is equal to 25% and all three levels are implemented using NVP; the value is about the same if the failure rates are reduced to 12.5% and only one level is using NVP. This gives us an quantitative indicator for NVP software design. The effect of reducing the module failure rates can be compared with the use of program redundancy.

5.5 Summary

Hierarchical NVP has simulated on ARIES82 and was studied under three basic models: NVP on one level, two levels, and all three levels. The results show that the effect of hierarchical NVP can improve system reliability. It also indicates the quantitative relationship

(1) Redundant at All Three Levels

Input:

$$Y[0] = 3$$

$$Y[1] = 2$$

$$D = 1$$

$\lambda = \mu$	System Reliability $T = 0.6$	System Reliability $T = 1.0$
0.01	0.999679	0.999115
0.05	0.992313	0.979432
0.10	0.970949	0.925593
0.15	0.938462	0.849812
0.20	0.897322	0.761891
0.25	0.849812	0.669598

$\lambda = \mu$	System MTTF	MTTF to t
0.01	38.6825	83.3333
0.05	7.9365	16.6667
0.10	3.9684	8.3333
0.15	2.6457	5.5556
0.20	1.9841	4.1667
0.25	1.5873	3.3333

Fig. 5-1 Model 1 - Three Level Redundancy

(2) Redundant at Two Levels

Input:

Level without redundancy:

$$D = 0$$

$$Y[0] = 1$$

Levels with redundancy:

$$Y[0] = 3$$

$$Y[1] = 2$$

$$D = 1$$

$\lambda = \mu$	System Reliability $T = 0.6$	System Reliability $T = 1.0$
0.01	0.993805	0.989466
0.05	0.965466	0.938141
0.10	0.923435	0.859378
0.15	0.876042	0.772214
0.20	0.825155	0.682972
0.25	0.772214	0.596077

$\lambda = \mu$	System MTTF
0.01	37.1429
0.05	7.4286
0.10	3.7142
0.15	2.4726
0.20	1.8571
0.25	1.4857

Fig. 5-2 Model 2 - Two Level Redundancy

(3) One Level Redundant

Input:

Level without redundancy:

$$D = 0$$

$$Y[0] = 1$$

Levels with redundancy:

$$Y[0] = 3$$

$$Y[1] = 2$$

$$D = 1$$

$\lambda = \mu$	System Reliability $T = 0.6$	System Reliability $T = 1.0$
0.01	0.987966	0.979907
0.05	0.939345	0.898591
0.10	0.878247	0.797899
0.15	0.817773	0.701702
0.20	0.758727	0.612228
0.25	0.701702	0.530629

$\lambda = \mu$	System MTTF
0.01	35.0000
0.05	7.0000
0.10	3.5000
0.15	2.3333
0.20	1.7500
0.25	1.4000

Fig. 5-3 Model 3 - One Level Redundancy

Comparison of System Reliability

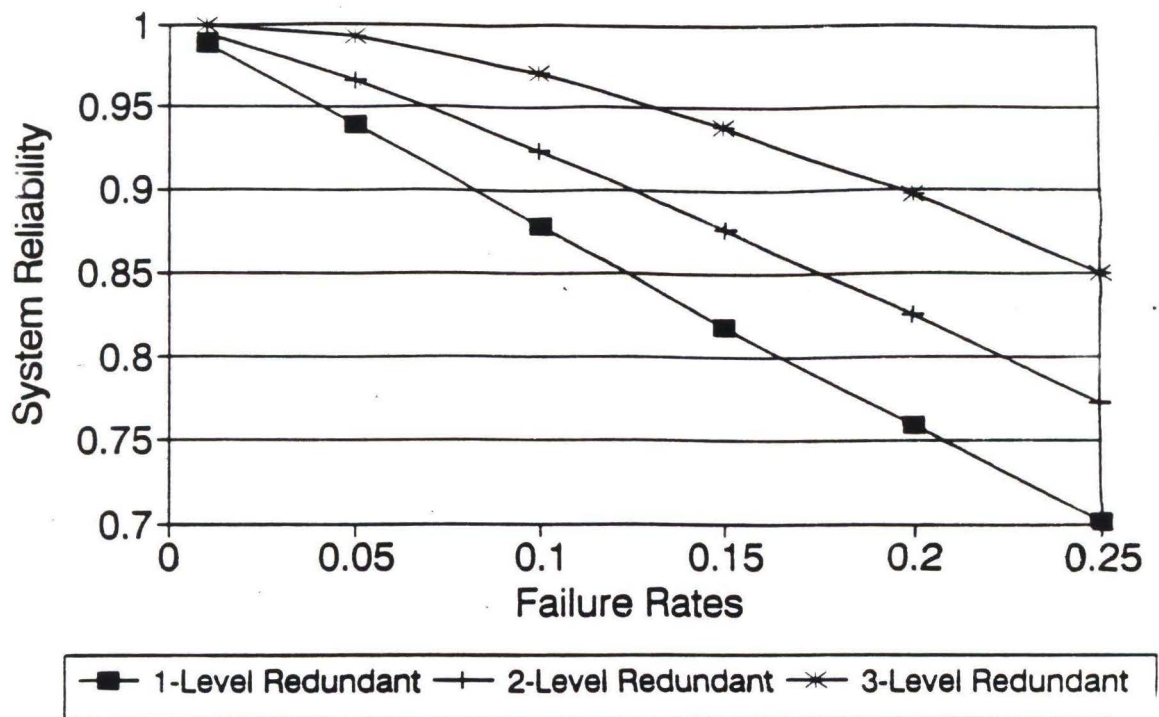


Fig. 5-4 Comparison of System Reliability

between the redundancy and the probability of program module failure. We believe that this information can be used to design hierarchical fault-tolerant software systems.

Future study should extend the proposed work by testing more combinations of failure rates on different levels, increase the number of program versions (increase n in NVP), and probably to explore further the quantitative relationship among the number of program modules, the system reliability, the software development cost, and the execution time.

CHAPTER VI

CONCLUSIONS AND FUTURE STUDY

6.1 Conclusions

The objective of this study was to analyze the reliability of hybrid fault-tolerant software systems as well as to design such systems with total resource cost limitations.

Mathematical models for the hybrid RB and NVP were developed. ARIES 82 software was used to model the hierarchical NVP. Simulation results reveals that the reliability of a hybrid software system depends on the parameters of the system, such as the program module failure rates, as well as the failure rates of acceptance test and voting modules. The numerical results show that the reliability of the acceptance test in the RB model and the voter in the NVP model have much larger influence in the reliability of a hybrid system. The results also show that the relationship between cost and system reliability is not a linear relation, i.e. the system reliability does not increase when more program modules (alternates, voter, and testing module) are added into the system.

Two heuristic methods were developed to supply a step by step approach to design hybrid fault-tolerant software systems. These procedures can be easily converted into computer programs.

6.2 Limitations and Future Study

There are several limitations which could be the topic of future study.

- (1) The probability model used in the study is a simplified version of the general probability model [Bell91]. The reliability of the hybrid system could be further investigated based on an more generic probability model.
- (2) The design methods proposed in this study do not necessary generate the optimal solution for hybrid fault-tolerant system design. They do produce a standard approach to the problem. Searching for an optimal or suboptimal solution to the problem should continue.
- (3) We assumed that the cost and reliability of the software modules are known. However, in most cases estimating the development cost and predicting the reliability of software is difficult.
- (4) Hardware cost is not taken into consideration in this study. N-Version Programming utilizes more hardware resources than RB due to the parallel processing nature of the NVP. The hardware cost should be taken into consideration.
- (5) Since the timing factor (calculation time and the distribution of the faults) is not considered in this study, the hybrid schemes generate better system reliability than that of pure RB and NVP systems. However, if the timing is a factor especially in real time situations, the results need to be reconsidered.
- (6) The static simulation does not handle the statistical (dynamical) features of a fault-tolerant system. Most of the simulation studies have assumed that the failures in

the output will follow a Poisson process [Leus90] [Musa87]. The theoretical foundation of this assumption is that the times between failures are assumed to be exponentially distributed. In fact the faults of the software should follow a binomial distribution. It is necessary to study fault-tolerant systems using a general statistical modeling environment.

- (7) Correlated faults are not concerned in this study. Coorelated faults could have big effects on the system reliability, especially the NVP systems.
- (8) Other analytical methods, such as Markov chain and Petri Nets, can be applied to analyze the hybrid fault-tolerant systems.

REFERENCES

- [Arse80] Arsenault, J.E. and J.A. Roberts, *Reliability and Maintainability of Electronic Systems*, Computer Science Press, Rockville, MD, 1980.
- [Aviz75] Avizienis, A., "Architecture of fault-tolerant computer systems," *Digest 1975 International Conference on Fault-Tolerant Computing (FTCS-5)*, IEEE, June 1975, pp.3-16.
- [Aviz77] Avizienis, A. and L. Chen, "On the implementation of N-version programming for software fault-tolerance during program execution," *Proceedings COMPSAC 77*, Chicago, IL, 1977, pp.149-155.
- [Aviz84] Avizienis, A. and J. P. Kelly, "Fault tolerance by design diversity: concepts and experiments," *IEEE Computer*, Vol. 17, No. 8, August 1984, pp.67-80.
- [Aviz85] Avizienis, A., "The n-version approach to fault-tolerant software," *IEEE Transactions on Software Engineering*, Vol. 12, No. 13, Dec. 1985, pp.1491-1510.
- [Aviz87] Avizienis, A., "On the achievement of a highly dependable and fault-tolerant air traffic control system," *IEEE Computer*, Vol. 20, No. 2, February 1987.

- [Barl75] Barlow, R.W., and H.E. Lambert, "Introduction to fault tree analysis," *Reliability and Fault Tree Analysis, Theoretical and Applied Aspects of System Reliability and Safety Assessment*, R.E. Barlow, J.B. Fussell, N.D. Singurwalla, Ed., SIAM, Philadelphia, 1975, pp.7-35.
- [Bell90] Belli F., and P. Jedrzejowicz, "Fault-tolerant programs and their reliability," *IEEE Transactions on Reliability*, Vol. 39, No. 2, 1990, pp.184-192.
- [Bell91] Belli, F., and P. Jedrzejowicz, "An approach to the reliability optimization of software with redundancy," *IEEE Transaction on Software Engineering*, Vol. 17, No. 3, 1991, pp.310-312.
- [Bhar81] Bhargava, B., and C. Hua, "Cost analysis of recovery block scheme and its implementation issues," *International Journal of Computer and Information Sciences*, Vol. 10, No. 6, 1981, pp.359-383.
- [Bour69] Bouricius, W.G., et al., "Reliability modeling techniques for self-repairing computer systems," *Proceeding ACM 1969 Annual Conference*, NY, 1969, pp.295-309.
- [Chen78] Chen, L., and A. Avizienis, "N-version programming: a fault tolerance approach to reliability of software operation," *Digest of the 8th Annual International Conference on Fault-Tolerant Computing (FTCS-8)*, IEEE, June 1978, pp.3-9.
- [Cost81] Costes, A., J.E. Doucet, C. Landrault, and J.C. Laprie, "SURF A program for dependability evaluation of complex fault-tolerant computing

systems, " *Digest of the 11th Annual Symposium of Fault-Tolerant Computing (FTCS-11)*, IEEE, 1981, pp.72-78.

- [Dhil78] Dhillon, B.S., and C. Singh, "Bibliography of literature on fault-trees," *Microelectrical Reliability*, Vol. 17, 1978, pp.501-503.
- [Dhil89] Dhillon, B.S., et al., "Modeling human errors in repairable systems," *Proceedings of the Annual Reliability and Maintainability Symposium*, January 1989, pp.418-424.
- [Doln83] Dolny, L.J., R.E. Fleming, and R.L. DeHoff, "Fault-tolerant computer system design using GRAMP," *Proceedings of the 1983 Annual Reliability and Maintainability Symposium*, IEEE, 1983, pp.417-422.
- [Eckh85] Eckhardt, D.E., and L.D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Transactions on Software Engineering*, Vol. 11, December 1985, pp.1511-1517.
- [Fern90] Fernandez, E.B., *Fault-Tolerant Computer Systems*, Class Notes, Department of Computer Science and Engineering, Florida Atlantic University, 1990.
- [Geis83] Geist, R. et al., "Design of the hybrid automated reliability predictor," *Proceedings IEEE/AIAA 5th Digital Avionics Systems Conference*, November 1983.
- [Grna80a] Grnarov, A., J. Arlat and A. Avizienis, "Modelling of software fault-tolerance strategies," *Proceedings 1980 Pittsburgh Modeling and Simulation Conference*, Pitt., Pa., May 1980.

- [Grna80b] Grnarov, A., J. Arlat and A. Avizienis, "On the performance of software fault-tolerance strategies," *Digest of the 10th Annual International Conference on Fault-Tolerant Computing (FTCS-10)*, Japan, October 1980, pp.251-253.
- [Heip84] Heidelberger, P., and S. Lavensberg, "Computer performance evaluation methodology," *IEEE Transactions on Computers*, Vol. C-33, No. 12, December 1984.
- [Horn74] Horning, J.J., and H.C. Lauer, P.M. Melliar-Smith, and B. Randell, "Program structure for error detection and recovery," *Lecture Notes in Computer Science*, Vol. 16, Springer-Verlag, New York, 1974, pp.171-187.
- [John88] Johnson, A.M., Jr. "Survey of software tools for evaluation, reliability, availability, and serviceability," *ACM Computing Surveys*, Vol. 20, No. 4, December 1988, pp.227-269.
- [KimK78] Kim, K.H., "An approach to program-transparent coordination of recovering parallel processes and its efficient implementation rules," *Proceedings Int. Conf. Parallel Processing*, 1978, pp. 58-68.
- [KimK84] Kim, K.H., "Software fault tolerance," *In Handbook of Software Engineering*, C.R. Vick and C.V. Ramamoorthy, Eds., Van Nostrand Reinhold Co. Inc., 1984, pp.437-455.

- [Lala83] Lala, J.H., "Interactive reductions in the number of states in Markov reliability analysis," *Proceedings of the AIAA Guidance and Controls Conference*, AIAA, 1983.
- [Lapr84] Laprie, J.C., "Dependability evaluation of software systems in operation," *IEEE Transactions Software Engineering*, Vol.6, SE-10, No.11, 1984, pp.701-714.
- [Laze84] Lazowska, E.D., et al., Quantitative system performance, Prentice-Hall Inc., Englewood Cliffs, NJ, 1984.
- [LeuS90] Leu, S., "Reliability modeling of fault-tolerant software," Ph.D. Thesis, Dept. of Computer Engineering, Florida Atlantic University, 1990.
- [Lice86] Liceage, C.A., and D.P. Siewiorek, "Towards automatic Markov reliability modeling of computer architectures," *NASA Technical Memorandum*, 89009, 1986.
- [Ajmo84] M.Ajmone Marsan, Balbo,G., and Conte,G., "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Transactions Computer Systems*, Vol. 2, No. 2, 1984, pp.94-122.
- [Math70] Mathur, F.P. and A. Avizienis, "Reliability analysis and architecture of a hybrid-redundant digital system: generalized triple modular redundancy with self-repair," *AFIPS Spring Joint Computer Conference*, 1970, pp.375-383.

- [Math75] Mathur, F.P. and P.T. deSousa, "Reliability modeling and analysis of general modular redundant systems," *IEEE Transactions on Reliability*, Vol. R-24, No. 5, December 1975, pp.269-299.
- [Musa87] Musa J.D., A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, New York, NY, McGraw-Hill, 1987.
- [Myer86] Myers, W., "Can software for the strategic defense initiative ever be error-free?", *IEEE Computer*, November 1986.
- [Pete77] Peterson, J.L., "Petri nets," *ACM Computer Surveys*, Vol. 9, No.3, 1977, pp.223-252.
- [Rand75] Randell, B., "System structure for software fault tolerance," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp.220-232.
- [Sahn86] Sahner, R.A., and K.S. Trivedi, "A hierarchical, combinatorial-Markov method of solving complex reliability models," *Proceedings of the 1986 Fall Joint Computer Conference*, AFIPS, 1986, pp.817-825.
- [Sahn87] Sahner, R.A., and K.S. Trivedi, "Reliability modeling using SHARPE," *IEEE Transactions on Reliability*, Vol.R-36, No.2, Feb. 1987, pp.186-193.
- [Scot83] Scott, R.K., et al., "The consensus recovery block," *Proceedings of the IEEE workshop on Hardware-software System Reliability*, December 1983, pp.74-85.

- [Scot87] Scott, R.K., et al., "Fault tolerant software reliability modeling," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 5, May 1987, pp.582-592.
- [Siew82] Siewiorek, D.P. and R.S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982.
- [Shoo87] Shooman, M.L., and A.E. Laemmel, "Simplification of Markov models by state merging," *Proceedings of the 1987 Annual Reliability and Maintainability Symposium*, IEEE, 1987, pp.159-164.
- [Stif79] Stiffler, J.J. L.A. Bryant, and L. Guccione, "CARE III final report phase I volume I & II," *NASA Contractor Reports 159122 & 159123*, 1979.
- [Triv82] Trivedi, D.S., *Probability & Statistics with Reliability Queuing, and Computer Science Applications*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.
- [WuJi89] Wu, J., *The Design of Reliable Decentralized Computer Systems*, Ph.D. Thesis, Department of Computer Engineering, Florida Atlantic University, 1989.
- [WuJi91] Wu, J., *Software Fault Tolerance Using Hierarchical N-Version Programming*, Working Paper, Department of Computer Engineering, Florida Atlantic University, 1991.
- [Zhan91] Zhang, M., and J. Wu, "Modeling and analysis of fault tolerant systems," *Second International Conference of Young Computer Scientists*, Beijing, China, 1991.

