World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:1, No:3, 2007

# Investigate the Relation between the Correctness and the Number of Versions of Fault Tolerant Software System

Pham Ba Quang, Nguyen Tien Dat, Huynh Quyet Thang



***Abstract*—**In this paper, we generalize several techniques in developing Fault Tolerant Software. We introduce property "Correctness" in evaluating N-version Systems and compare it to some commonly used properties such as reliability or availability. We also find out the relation between this property and the number of versions of system. Our experiments to verify the correctness and the applicability of the relation are also presented.

***Keywords*—**Correctness, Fault Tolerant Software, N-version Systems

## I. Introduction

TODAY, most industries are highly dependent on computers for their day-to-day functioning. Safe and reliable software operations are significant requirement for many types of systems. For instance, in air traffic control, nuclear safety, high-speed rail, electronic banking, automated manufacturing... The cost and consequence s of these systems failing can range into catastrophic, with serious injury occurring or lives lost. Software becomes more complex and more significant to the overall system performance and dependability. Unfortunately, software could not be developed without errors. Even if the best people, practices and tools were used, it would be very risky to assume the software developed is error-free. Software does not physically deteriorate, it has only logical faults that are difficult to visualize, classify, detect and correct. To protect against these faults, we cannot simply add redundancy as typically done for hardware faults, because doing so will duplicate the problem. So, to provide protection against these faults, we turn to software fault tolerance. There are many techniques available today for implementing software fault tolerance. These techniques are divided into two categories: single version and multi-version techniques. This paper reviews some techniques in developing fault tolerant software. Our aim is to survey and present one new property in evaluating N-version Systems. We also investigate the relation between this property and the number of versions of system. Part 2 presents some single-version and multi-version techniques for implementing fault tolerant software. In part 3 and part 4 we introduce more details about Correctness and find out the relation between this property and the number of versions of multi-version system. Part 5 presents our experiments to verify the accuracy and the applicability of the founded relations. The final part is some conclusions and our future work.

Manuscript received January 15, 2006. This work was supported in part by the Ministry of Education and Training of Vietnam under grant No. B2005-28-180 and Ministry of Science and Technology of Vietnam under grant No. KHCB2.034.06.

Pham Ba Quang received the M.S (1998) and B.S (2001) degrees in Computer Science from Hanoi University of Technology. He now works for Bac A Bank, Hanoi, Vietnam. (phambaquang@yahoo.com).

Nguyen Tien Dat: received the M.S (June 2006) degrees in Computer Science from Hanoi University of Technology. Now he is PhD student of University of Iowa, Iowa, USA (tinguyen@cs.uiowa.edu)

Huynh Quyet Thang: received B.S degree (1990) from Varna University, Bulgaria and Dr Degree (1995) in Bulgaria. He is Head of Software Engineering Department, Hanoi University of Technology (thanghq@it-hut.edu.vn).

## II. Techniques in Software Fault Tolerance

In this section we present some fault tolerance techniques for implementing software fault tolerance.

### A. Single-Version Software Fault Tolerance Techniques

Single-version fault tolerance is based on the use of redundancy applied to a single version of a piece of software to detect and recover from faults. Among others, single-version software fault tolerance techniques include considerations on program structure and actions, error detection, exception handling, checkpoint and restart, process pairs, and data diversity [6].

Normally, we use multi-version techniques instead of single-version techniques because multi-version techniques have higher "fault tolerant ability" than single-version techniques.

### B. Multi-Version Software Fault Tolerance Techniques

The multiple version software techniques are to provide diversity in the design and implementation of the software. The goal of design diversity is to make the components as diverse and independent as possible. The overriding principle is implemented using redundant software components called variants. The designer assumes that coincident components failure is rate and results are different enough to enable error detection and to distinguish a correct result or "best" answer.

Multi-version fault tolerance is based on the use of two or more versions (or "variants") executed either in sequence or in parallel. The versions are used as alternatives (with a separate

means of error detection), in pairs (to implement detection by replication checks) or in larger groups (to enable masking through voting). The rationale for the use of multiple versions is the expectation that components built differently (i.e., different designers, different algorithms, different design tools, etc) should fail differently [1, 2]. Therefore, if one version fails on a particular input, at least one of the alternate versions should be able to provide an appropriate output. Multi-version software fault tolerance techniques include Recovery Blocks (RcB), N-version Programming (NVP), N Self-checking Programming (NSP), Consensus Recovery Blocks (CRB) and t/(n-1)-Variant Programming (t/(n-1)VP) [7,9]. Among them, RcB and NVP techniques are the original design diverse software fault tolerance techniques. The other techniques are the extension (NSP or t/(n-1)VP) or the combination (CRB) of them. Therefore we will focus on two basic techniques: RcB and NVP.

*1) Recovery Blocks*

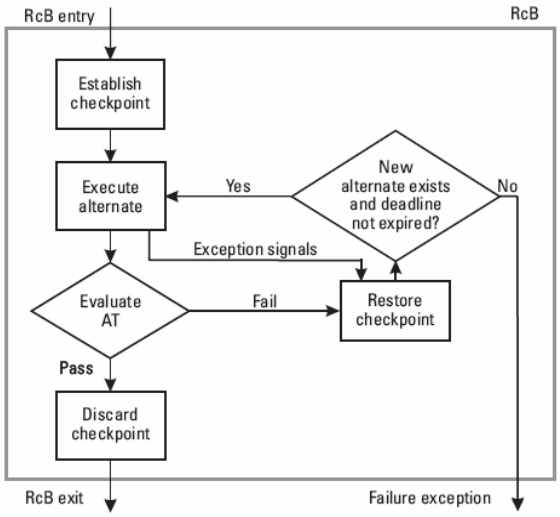


Fig. 1 Recovery Blocks Model

The basic RcB scheme is one of the two original designs diverse software fault tolerance techniques. RcB uses an AT to accomplish fault tolerance. We know that most program functions can be performed in more than one way, using different algorithms and designs. These differently implemented function variants have varying degrees of efficiency in terms of memory management and utilization, execution time, reliability, and other criteria. RcB incorporates these variants such that the most efficient module is located first in the series, and is termed the primary alternate or primary try block. The less efficient variant(s) are placed serially after the primary try block and are referred to as (secondary) alternates or alternate try blocks. Thus, the resulting rank of the variants reflects the graceful degradation in the performance of the variants.

*2) N-version Programming (NVP)*

N-Version programming [2] is a multi-version technique in which all the versions are designed to satisfy the same specification and the decision mechanism (DM) examines the results and selects the "bets" result, if one exists. There are many alternative DM available for use with NVP. Since all the versions are built to satisfy the same specification, the use of N-version programming requires considerable development effort but the complexity (i.e., development difficulty) is not necessarily much greater than the inherent complexity of building a single version. Design of the voter can be complicated. Much research has gone into development of methodologies that increase the likelihood of achieving effective diversity in the final product. The NVP processes can run concurrently on different computers or sequentially on single computer.

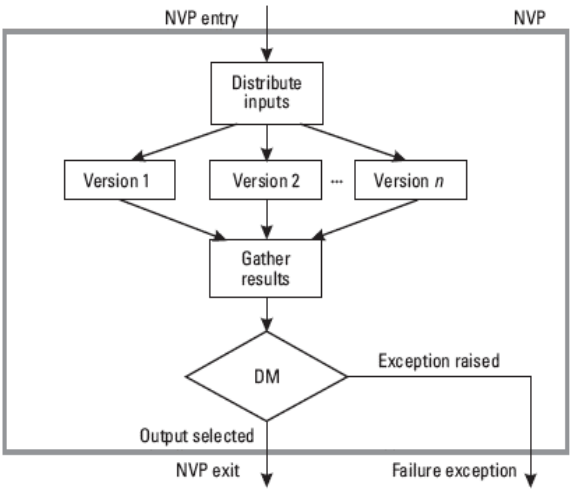


Fig. 2 N-version Programming Model

## III. The Relation Between "Correctness" and the Number of Versions in N-version System

The property "Correctness" is very important in evaluating Fault Tolerant Software. The higher it is, the safer the system is. It may be more important than the availability or the reliability because if we know it, we can make sure how many results are truly right, and only use a number of results that is less than accuracy. And of cause we see that the higher accuracy is, the higher the availability and the reliability are. But there is a question that: for a specific system, how many versions are needed to archive the desired ability to return right results. To answer this question, we have to find the relation between "Correctness" and the number of versions in N-version system. In Multi-version techniques we have two fundamental techniques: RcB and NVP. So we only need to find the relations in two techniques, the relations in other techniques can be referred from these relations.

### *A. The relation in RcB technique*

RcB technique will return rights result if one version returns a right output and the Acceptance Test (AT) Module makes a right decision.

Assume that we have n versions with the reliabilities $r_1$, $r_2$, …,$r_n$ respectively. The reliability of AT Module is B and

Correctness of system is $S_n$. We will find the relation between $S_n$ and n by inductive method.

n=1: the system will return right result if the only one version returns right output and the AT Module makes right decision. Therefore $S_1 = r_1 B$

n=2: the system will have one of these cases:

Case 1: the first version returns right output (1 right for short) and the AT Module makes right decision (B right for short). The probability of this case is $r_1 B$

Case 2: the second version is called because 1 right B wrong or 1 wrong B right. If 2 right and B right, the system will returns right results.

Therefore:

$$S_2 = r_1 \times B + (r_1 \times (1-B) + (1-r_1) \times B) \times r_2 \times B =$$
$$= S_1 + (r_1 \times (1-B) + (1-r_1) \times B) \times r_2 \times B$$

n = 3: The system will have one of these cases

Case 1: the first two versions returns a right output and this result is accepted, the probability of this case is $S_2$

Case 2: the third version is called because of one of following reasons:

1 right, B wrong, 2 right, B wrong: $r_1 \times (1-B) \times r_2 \times (1-B)$
1 wrong, B right, 2 right, B wrong: $(1-r_1) \times B \times r_2 \times (1-B)$
1 right, B wrong, 2 wrong, B right: $r_1 \times (1-B) \times (1-r_2) \times B$
1 wrong, B right, 2 wrong, B right: $(1-r_1) \times B \times (1-r_2) \times B$

So we have:

$$S_3 = S_2 + [(r_1 \times (1-B) \times r_2 \times (1-B) + (1-r_1) \times B \times r_2 (1-B) +$$
$$+ (r_1 \times (1-B) \times (1-r_2) \times B + (1-r_1) \times B \times (1-r_2) \times B] \times r_3 \times B$$
$$= S_2 + [r_1 \times (1-B) + (1-r_1) \times B] \times [r_2 \times (1-B) + (1-r_2) \times B] \times r_3 \times B$$

Continue do in the same way we finally have:

$$S_n = S_{n-1} + [(r_1 \times (1-B) + (1-r_1) \times B] + \ldots\ldots +$$
$$+ [(r_{n-1} \times (1-B) \times (1-r_{n-1}) \times B] \times r_n \times B$$

If all versions have the same reliability r, we have:

$$S_1 = r \times B$$

$$S_n = S_{n-1} + [r \times (1-B) + (1-r) \times B]^{n-1} \times r \times B$$

We will prove by inductive method that:

$$S_n = r \times B \times \{1 - [r \times (1-B) + (1-r) \times B]^n\} / \{1 - [r \times (1-B) + (1-r) \times B]\}$$
$$n = 1 : S_1 = r \times B$$

With n=1 the formula is true.

Suppose the formula is true with n, we will prove that it is also true with (n+1)

We have:

$$S_{n+1} = S_n + [(r \times (1-B) \times (1-r) \times B]^n \times r \times B$$
$$= r \times B \times \{1 - [r \times (1-B) + (1-r) \times B]^n\} / \{1 - [r \times (1-B) + (1-r) \times B]\} +$$
$$+ [r \times (1-B) + (1-r) \times B]^n \times r \times B$$
$$= r \times B \times \{1 - [r \times (1-B) + (1-r) \times B]^{n+1}\} / \{1 - [r \times (1-B) + (1-r) \times B]\}$$

Proved)

When n is great, increasing n will not affect this ability so much. Therefore when developing a fault tolerant software, we have to consider Correctness and the cost to construct all versions.

Some special cases:

When B = 1: $S_n = 1 - (1-r)^n$

It is easily seen that $S_n \rightarrow 1$ when $n \rightarrow \infty$. So when B = 1 we can have a system with the arbitrary ability to return right results without considering the reliability of versions, all we need is the large enough number of versions.

When r = 1: $S_n = 1 - (1-B)^n$

Similarly, we have $Sn \rightarrow 1$ when $n \rightarrow \infty$. So when r = 1 we also can have a system with the arbitrary ability to return right results without considering the reliability of the AT Module, all we need is the large enough number of versions.

The relation in NVP technique

In NVP technique, the design of voters affects Correctness very much. They are generally divided into two main categories: type A (agreement-based) voters which produce an output from redundant inputs if there is agreement between a particular number of voter inputs (e.g., majority and plurality voting), and type B voters that always produce an output regardless of the agreement, or otherwise, between redundant inputs.

Type B voters either amalgamate the inputs or simply select one of them based on a particular metric (e.g., weighted average voter and mid-value selector respectively) [5,7,8]. Because of this efficient effect, we will find the relation in two types of voters. We choose two main techniques to survey: Majority (Type A) and Weighted Average (Type B).

The relation in NVP techniques with Majority Voter (NVP-M)

In Majority Voter, version outputs are compared for equality. If more than half of the version outputs agree this common output becomes the output of the N-version system. The definition of "agree" is really case-dependent and requires significantly different methods to apply (see [5] for more information).

In NVP-M, we will have a right result if more than half of versions return right outputs and the Voter selects one of them. Suppose that all versions have the same reliability r, the reliability of Voter is B and Correctness of system is $S_n$. The number of versions is n (n odd, n = 2k+1). If in one case, we have j versions return right outputs, and (n-j) remaining versions return wrong outputs, so the probability of this case is:

$P_n(j) = C^j_n r^j (1-r)^{n-j}$

It is easily seen that Correctness is the sum of probabilities of cases that have more than (k+1) versions return right outputs and the Voter also makes right decision. Therefore we have:

$$S_{2k+1} = B \times \sum_{j=k+1}^{2k+1} C^j_{2k+1} \times r^j \times (1-r)^{2k+1-j}$$

We can see that Correctness in NVP-M technique is covariant with n. And we also see that with the same value of r and B, Correctness in RcB technique is higher than the ability in NVP-M technique.

Some special cases:

When B = 1: $S_{2k+1} = \sum_{j=k+1}^{2k+1} C_{2k+1}^{j} \times r^{j} \times (1-r)^{2k+1-j}$

So we can conclude that when B = 1, we can have a system with the arbitrary ability to return right results without considering the reliability of versions, all we need is the large enough number of versions.

When r = 1: $S_n = B$. It means that Correctness equals to the reliability of the Voter, not depend on the number of versions.

The relation in NVP technique with Weighted Average Voter

In this technique, all outputs are combined to produce a new, possibly distinct output. Suppose n versions of software with outputs in x produce the outputs $x_1, x_2, \ldots, x_n$. Let $w_1, w_2, \ldots, w_n$ be non-negative real numbers satisfying:

$\Sigma w_i = 1$

Define a new element of x by: $x = \Sigma w_i x_i$

Then x is the output produced by an N-version fault tolerant system. There are some methods to determine $w_i$, see [4] for more information. We can see that with Weighted Average Voter, the result x is surely true if all outputs are true. If there are some outputs wrong, the result x may still be true. We say x may be true because it may happen like that: there are only two wrong outputs $x_i$ and $x_j$, all remaining outputs are right and equal to x*. If $w_i = w_j$ and $x_i = x^* + \Delta$ and $x_j = x^* - \Delta$ t then the result x still equals to x* and it is true. Because we can't calculate the probability of these cases, we can't make any relation between Correctness and the number of versions in NVP techniques with Weighted Average Voter.

## IV. Experiment

We have executed two experiments: the first one is to verify the Correctness and the second one is to verify the applicability of the founded relation.

### A. The first experiment

#### 1) Experimental Apparatus

Input Generator: This module will create the file DI.TXT that contains 10,000 random integer numbers.

Error Maker: This module is to create errors for versions. The errors have to be random. We create errors with assuming that our hardware has errors that change bits from 1 to 0.

Versions: Versions sort the correlative input files using different algorithms. There are six sorting algorithms: Insertion Sort, Bubble Sort, Selection Sort, Quick Sort, Heap Sort and Merge Sort. In NVP-M technique we need odd versions, so we have six versions with six algorithms and three remaining versions use three algorithms: Quick Sort, Heap Sort and Merge Sort (we have totally nine versions).

Voter: (This module appears only in NVP-M technique) The voter chooses a result from nine outputs of nine versions. If there are more than five identical outputs, it will return a result.

Acceptance Test: (This module appears only in RcB technique) This module takes an output of one version and XOR all numbers in it. If the result equals to the result of XOR all numbers in file DI.TXT, this output will be accepted and become the final result. If it doesn't equal, it will be rejected and the next version will be called. After all versions are executed unsuccessfully, the system will be determined as failure.

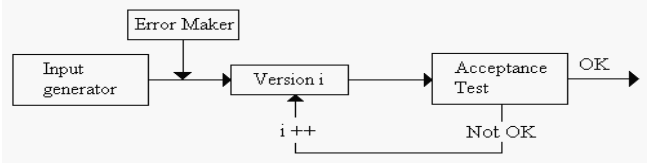


Figure 3. Experimental System with RcB technique

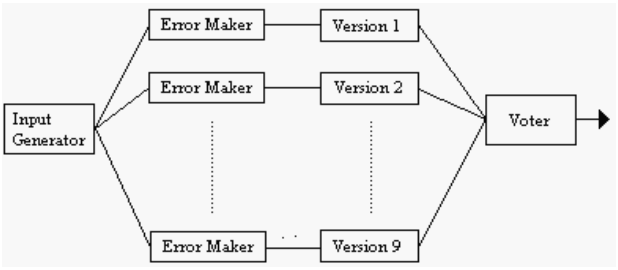


Fig. 4 Experimental System with NVP-M technique

#### 2) The result of Experiment

After executing the system 1000 times, we have the following result as shown on table 1 and table 2.

TABLE I.
RESULTS OF EXPERIMENTING WITH SORTING ALGORITHMS

| Element | Number of execution | Right output | Reliability |
|---|---|---|---|
| Version 1 | 1000 | 756 | 0.756 |
| Version 2 | 1000 | 774 | 0.774 |
| Version 3 | 1000 | 768 | 0.768 |
| Version 4 | 1000 | 749 | 0.749 |
| Version 5 | 1000 | 764 | 0.764 |
| Version 6 | 1000 | 765 | 0.765 |
| Version 7 | 1000 | 779 | 0.779 |
| Version 8 | 1000 | 762 | 0.762 |
| Version 9 | 1000 | 756 | 0.756 |
| Voter | 1000 | 1000 | 1.000 |
| Acceptance Test | 10964 | 10950 | 0.999 |

TABLE II.
RESULTS OF CALCULATIION "CORRECTNESS"

| Technique | The average reliability of Versions | Correctness | |
|---|---|---|---|
| | | *Experimental result* | *Theoretic result* |
| RcB | 0.764 | 0.999 | 0.99899 |
| NVP | 0.764 | 0.964 | 0.96118 |

The result of experiment shows that our theoretic calculate is true.

### B. The second experiment

#### 1) Experimental Apparatus

In the second experiment we apply fault tolerant models to our e-class system – BKEC (Bach Khoa e-class). BKEC is a system to control e-classes and developed by Software Engineering Department, Hanoi University of Technology. It contains all basic functions of an e-class controller such as desktop sharing, multimedia, chat and voice chat, file transfer, remote control … The following chart represents to the functional decomposition diagram (FDD) of BKEC system. In BKEC system, transferring exam question from teacher's computer to learners' computers need to be highly reliability and highly safe. Because transferring files via LAN or Internet always contains risk, fault tolerant mechanism is what needed to satisfy these requirements.

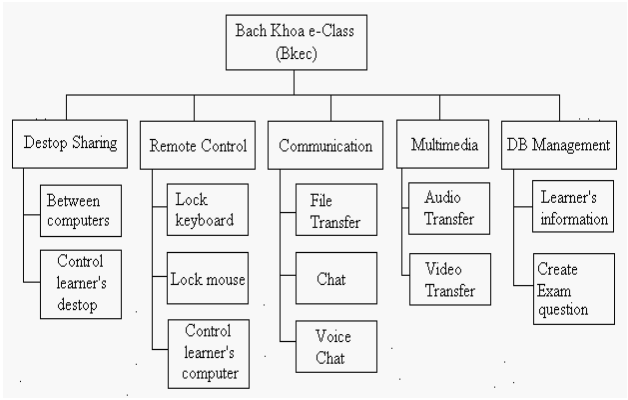


Fig. 5 The FDD of BKEC system

#### a) File transfer module with RcB technique

Figure 6 shows the model of module with RcB technique. To start a transfer session we begin with the 1st version. The 1st version compresses exam questions, transfers compressed file to learner's computer and compressed file will be decompressed in learner's computer. To check if the received file is unchanged or not, we send checking information with compressed file. To create checking information, we split compressed file in to 4-byte parts and XOR all parts to receive 4-byte result. 4-byte result will be checking information and be sent with compressed file. In learner's computer, we do the same process to have 4-byte result, compare it with checking information. If they are equal, the compressed file will pass Acceptance Test module. If they aren't equal means the compressed file doesn't pass AT module, the learner's computer will send a feedback to teacher's computer to inform that the file transfer didn't complete and need to resend files with another version. So the exchanged information between two computers will be as following: Information from teacher's computer to learners' computers: (1) compressed exam question + the ID of version used to compress + checking information; (2) Information from learners' computers to teacher's computer: the ID of version received + feedback to inform the file transfer process is complete or not.

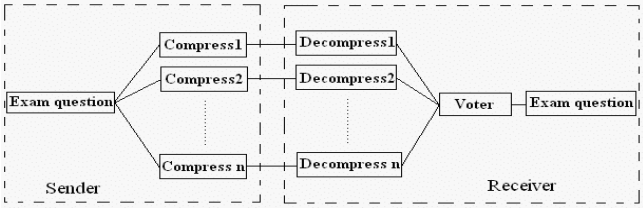


Fig. 7 BKEC system with NVP-M technique

#### b) File transfer module with NVP-M technique

Figure 7 shows the model of module with NVP-M technique. After executing all versions to have n compressed files, we transfer all of them to receiver's computer. In receiver's computer, we decompress all compressed files with corresponded decompressions, and n decompressed files will go through the Majority Voter to return final file. The final file, if existing, will be the final result of file transfer process.

#### 2) The result of Experiment

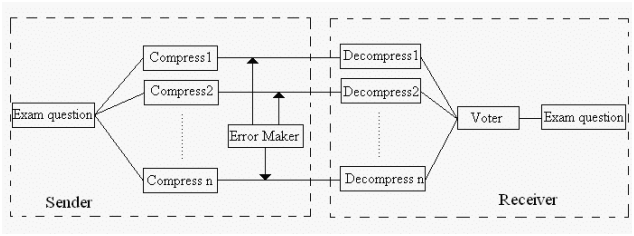


Fig. 8 NVP technique with error maker module

We implemented BKEC system with constructed model and evaluated its performance through two experiments: with and without making errors. The requirement is the ability to return right result of system must be higher than 0.999.

Experiment without making errors: In the normal conditions of our laboratory, when we executed the system to send files 1000 times, no fault happened. We concluded that we need only one version to satisfy the requirements.

We implemented BKEC system with constructed model and evaluated its performance through two experiments: with and without making errors. The requirement is the ability to return right result of system must be higher than 0.999.

Experiment without making errors: In the normal conditions of our laboratory, when we executed the system to send files 1000 times, no fault happened. We concluded that we need only one version to satisfy the requirements.

Experiment with making errors: Figures 8 and 9 show the models of system with error maker module. This module is to simulate some types of errors happening in file transfer process such as data errors or process errors. The purpose of this module is to evaluate the system's performance when errors happen. If the ability to return right result of system is higher than 0.999, it means that our relations are correct and highly applicable.

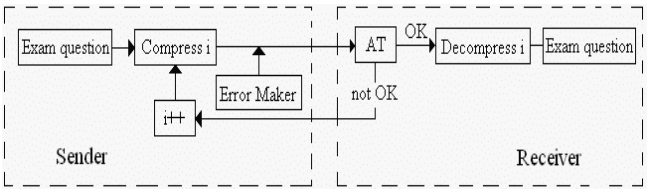


Fig. 9 RcB technique with error maker module

We use five lossless data compression techniques (Zip, Flate, Huffman, BZip and LZW, see www.wikipedia.org for more information about these techniques) to create five versions.

According to (1) when n = 5 and B = 1:

$S_5 = 1-(1-r)^5 \geq 0.999 \rightarrow 0.001 \geq (1-r)^5 \rightarrow 0.251 \geq 1-r \rightarrow r \geq 0.749$

According to (2) when n = 5 and B = 1:

$S_5 = 10r^3(1-r)^2 + 5r^4(1-r) + r^5 \geq 0.999 \rightarrow r \geq 0.953$

TABLE III.
EXPERIMENT RESULTS OF BKEC SYSTEM

| Element | Number of execution | Right output | Reliability |
|---|---|---|---|
| Version 1 | 1000 | 957 | 0.957 |
| Version 2 | 1000 | 954 | 0.954 |
| Version 3 | 1000 | 952 | 0.952 |
| Version 4 | 1000 | 951 | 0.951 |
| Version 5 | 1000 | 954 | 0.954 |

TABLE IV.
CALCULATIION OF BKEC SYSTEM "CORRECTNESS"

| Technique | The average reliability of Versions | Correctness | |
|---|---|---|---|
| | | Experimental result | Theoretic result |
| RcB | 0.9536 | 1.000 | 0.9999997 |
| NVP | 0.9536 | 0.999 | 0.9990692 |

We control the execution of error maker module in some ways to satisfy that the reliabilities of versions are approximate 0.953. Note that the reliability of a version is 0.953 means if you execute this version 1000 times, it will return 47 wrong results. In fact it is unacceptable if you have a piece of software with that wrong rate. But if our system still have Correctness higher than 0.999, it means that our relations are accurate and highly applicable. The results are shown on the table 3 and table 4. The results of experiments show that Correctness of system meets the requirements.

## V. CONCLUSION AND FUTURE WORK

### A. Conclusion

In this paper, we generalize several commonly used techniques in developing Fault Tolerant Software. We also introduce one new property in evaluating N-version Systems: "Correctness" and the relation between this property and the number of versions of system. Our experiments to verify the Correctness and the applicability of the relation are also presented. From our theoretic calculation and experimental results we have some conclusions:

- Correctness of fault tolerant software is covariant with the number of versions, but not linear.
- In RcB and NVP-M techniques, when the reliability of AT Module or Voter equals to 1, we can have a system with the arbitrary ability to return right results without considering the reliability of versions, all we need is the large enough number of versions.
- In RcB technique, when the reliabilities of all versions equal to 1, we also can have a system with the arbitrary ability to return right results without considering the reliability of the AT Module, all we need is the large enough number of versions.
- In NVP techniques with Weighted Average Voter, we can't make any relation between Correctness and the number of versions. This ability depends on the way to determine the weights of versions.

### B. Future work

We are in progress to find out the relation between the number of versions and two properties: reliability and availability. Once finding out these relations, we will finish our mission to determine the proper number of versions for a multi version fault tolerant system. Because you can see that with our calculating above, Correctness of system will increase when the number of versions increases. But in fact when increasing the number of versions, the system will be more complex and the reliability and the availability of system may decrease. So with a specific system, we need a proper number of versions that best satisfies the requirements of the reliability, the availability and Correctness.

## REFERENCES

[1] Algirdas Avizienis and L. Chen, On the Implementation of N-Version Programming for Software Fault Tolerance During Execution, Proceedings of the IEEE COMPSAC'77, November 1977, pp. 149 – 155.
[2] lgirdas Avizienis, The Methodology of N-Version Programming, in R. Lieu, editor, Software Fault Tolerance, John Wiley & Sons, 1995.
[3] EEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1998.
[4] G. Latif-Shabgahi, A. J. Hirst, and S. Bennett "A novel family of weighted average voters for fault-tolerant computer control systems" , 2004
[5] Lorczak, P.R., Caglayan, A.K., and Eckhardt, D.E., "A Theoretical Investigation of Generalized Voters", Digest of papers FTCS'19: IEEE 19th Ann. Int. Symp. on Fault- Tolerant Computing Systems, Chicago, IL, pp. 444-451, 1989.
[6] Michael R. Lyu, editor, Software Fault Tolerance, John Wiley & Sons, 1995.
[7] NASA Organization, "Software Fault Tolerance: A tutorial", 2000
[8] Brian Randell, System Structure for Software Fault Tolerance, IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975, pp. 220 –232.
[9] Brian Randell and Jie Xu, The Evolution of the Recovery Block Concept, in Software Fault Tolerance, Michael R. Lyu, editor, Wiley, 1995, pp. 1 – 21.