Methodology for Reliability Evaluation of N-Version Programming Software Fault Tolerance System

Ping Guo\textsuperscript{1,2}, Xin Liu\textsuperscript{1,2}  
\textsuperscript{1}College of Information Science and Technology  
Beijing Normal University, 100875  
Beijing, China,  
pguo@ieee.org, Liux_skylark@tom.com

Qian Yin\textsuperscript{1,2}  
\textsuperscript{2}Laboratory of Computer Science, Institute of Software  
The Chinese Academy of Sciences, 100080  
Beijing, China  
yinqian@bnu.edu.cn

\textbf{Abstract}—Software reliability can be improved by tolerating software faults, such as using N-version programming technique. Reliability evaluation is focused on the modeling and analysis techniques for fault prediction purpose. In this paper, a straightforward analysis method for evaluating reliability of software system established by N-version programming is proposed. The dependent failure parameters are assumed as random variables instead of constant. A case study is presented of the analysis of failure data from two software projects; the effectiveness of proposed evaluation methodology is demonstrated.

\textbf{Keywords—}N-version programming; reliability evaluation; dependent failures; software fault tolerance

I. Introduction

While software is executed in sorts of critical applications, such as flight control and weapon systems, software failure may cause catastrophic results, therefore, software reliability is expected to be very high. That is, software should be able to survive failures. Fault-tolerant techniques are introduced in these circumstances which desired extremely high system reliability. The most widely used fault-tolerant techniques are N-version programming (NVP) [1] and recovery block scheme (RBS) [2]. While in this paper, we mainly concern the NVP technique, which was proposed by Chen and Avizienis [1].

Although NVP has been adopted in some mission-critical applications, the effectiveness of this approach is still an open question. How to estimate the fault correlation between multiple versions and how to predict the system’s reliability are main issues. Theoretical as well as empirical investigations have been conducted by many researchers. Empirical studies can be found in references [3, 4, 5], to evaluate reliability with modeling can be found in references [3, 6, 7, 8, 9, 10, 11, 12]. and optimal models for fault tolerance system are studied in references [13, 14, 15, 16, 17]. Littlewoods [6] addresses the reliability problem using Bayesian inference. Wattanapongsorn [7] and Zafiropoulos [8] consider the reliability as random variable rather than constant. To estimate the system failure affected by coincide failure rate between versions, Ege et al [9] introduced dependency concept in case software versions has the same behavior in producing incorrect output with probability $\alpha$. Applying subdomains, Popov [10] analyzes the bounds of final reliability.

In reference [9], it is assumed that all versions have the same failure rate. Due to the independence of development, the failure rate of all the versions must not be exactly equal. Furthermore, by applying distinct programming technique and using different commercial-off-the-shelf software, each version’s failure rate can not be known exactly but must be estimated with some uncertainty. This software failure rate uncertainty can be modeled assuming a probability distribution function with appropriate parameters. The stochastic nature of the failure rate is propagated to the system failure rate and results in more realistic estimations of the system reliability. In this paper, on consideration of uncertainty of dependent failure parameter $\alpha$ and version failure rate $p$, a straightforward method for evaluating reliability of software system establishing by N-version programming is proposed. By calculating the mean and variance of system reliability, we achieve the final bounds of the reliability.

\footnote{This work was fully supported by a grant from the National Natural Science Foundation of China (Project No. 60675011). Dr. Qian Yin is the corresponding author.}
II. Reliability Evaluation Method

In this section, we present the reliability evaluation method with dependent failure. In order to illustrate our method, let us define the following events first, which is similar with that in reference [9] except three our defined events.

\( I_i \): event that version \( i \) executes its function incorrectly.

\( C_i \): event that version \( i \) executes its function correctly.

\( A_{j\leq i\leq n} \): event that only version \( i \) executes its function correctly and the others execute their function incorrectly.

\( A_{n+1} \): event that all the versions execute their function incorrectly.

\( A \): event that occurs when any event \( A_i \; (2 \leq i \leq n) \) occurs.

\( B \): event that the voting procedure cannot select the correct result from at least 2 correct outputs.

\( F \): event that the N-version programming system fails.

\( n \): the number of all possible inputs.

\( S(x) \): the number of the inputs associated with the event ‘x’.

\( \alpha_i \): dependent failure parameter; measures the similarity percentage of the input sets on pair of version \( i \) and version \( j \)’s fail. In other words, pair of version \( i \) and version \( j \) produce incorrect output on the fraction \( \alpha_i \) of inputs that cause both versions to fail.

Following the analysis in reference [9], the probabilities of these events can be calculated as

\[
P(A_i) = P(I_i)P(C_1/I_1)\cdots P(C_n/I_n)\cdot \frac{P(\sum_{i=1}^{n} C_i - I_i)}{1 - P(\sum_{i=1}^{n} C_i - I_i)}
\]  \( 1 \leq i \leq n \) \ldots \ldots \ldots \ldots \ldots (1)

\[
P(A_{n+1}) = P(I_n)P(I_{n+1}/I_n) - P(I_n)/I_n - I_{n+1}/I_n \cdot C_{n+1}\ldots C_n \ldots C_1
\] \( 1 \leq i \leq n \) \ldots \ldots \ldots \ldots \ldots (2)

\[
P(A) = \sum_{i=1}^{n+1} P(A_i)
\] \ldots \ldots \ldots \ldots \ldots (3)

\[
P(F) = P(A)(1 - P(B)) + P(B)
\] \ldots \ldots \ldots \ldots \ldots (4)

A. Improving method

In reference [9], it is assumed that all versions have the same failure rate \( p \), and only the case that dependent failure parameter \( \alpha \) takes several values is investigated. As discussed in first section, the uncertainty of failure rate \( p \) can be represented by random variable. Similarly, dependent failure parameter \( \alpha \) between each two versions must not be equal and can be represented by random variable. In our work, the probability distribution is used to represent the stochastic nature of these two parameters.

Firstly, we assume that the failure rate of 1st version is random variable \( p_1 \). The density function of \( p_1 \) is represented as \( f_1(x) \) and the mean as \( m_1 \). Assume the \( \alpha_{ij} \), which is independent failure parameter between version 1 and version \( j \; (2 \leq j \leq n) \), as independent random variable with certain distribution. The density function of \( \alpha_{ij} \; (2 \leq j \leq n) \) is represented as \( f_j(x) \) and the mean as \( m_2 \). For simplicity, we write \( \alpha_{ij} \) as \( \alpha_i \). We also assume that \( p_1 \) and \( \alpha_i \; (2 \leq j \leq n) \) are independent.

With aforementioned hypothesis, the mean \( E \) and variance \( Var \) of probability of event \( A_i \; (2 \leq i \leq n) \) can be calculated as follows.

\[
E(P(A_i)) = \left( \frac{\alpha_i \alpha_{i+1} \cdots \alpha_n p_{n+1}}{N} - \frac{\alpha_i \alpha_{i+1} \alpha_{i+2} \cdots \alpha_n p_{n+1}}{N} \right)
\] \( N \) \ldots \ldots \ldots \ldots \ldots (5)

\[
Var (P(A_i)) = E((\alpha_i \cdots \alpha_n p_{n+1} - (1 - \alpha_i))^2)
\] \ldots \ldots \ldots \ldots \ldots (6)

\[
E(P(A_{n+1})) = E((\alpha_n p_{n+1}) = m_{n+1} m_1
\] \ldots \ldots \ldots \ldots \ldots (7)

Under the condition that each of \( P(A_i) \) is independent, the mean and variance of probability of event \( A_{n+1} \) can be calculated as follows.

\[
E(P(A_{n+1})) = \sum_{i=1}^{n+1} P(A_i)
\] \ldots \ldots \ldots \ldots \ldots (9)

When calculating the probability of event \( A_i \), we derive the following equation:

\[
P(A_i) = P(C_1)\cdots P(C_n) - \frac{1}{n} \sum_{i=1}^{n} P(A_i)
\] \ldots \ldots \ldots \ldots \ldots (10)

Under the condition that each of \( P(A) \) is independent, the mean and variance of probability of \( A_i \) can be obtained as

\[
Var (P(A_i)) = \frac{1}{n} \sum_{i=1}^{n} Var (P(A_i))
\] \ldots \ldots \ldots \ldots \ldots (11)

Now, with equations (5) to (11), the characteristic of event \( F \) can be obtained as follows.

\[
E(P(F)) = (nm \int_{1-m_1}^{1-m_2} \frac{f(x)dx}{(1-m_2)} + m_2 \int_{1-m_1}^{1-m_2} \frac{f(x)dx}{(1-m_2)} + P(B))
\] \ldots \ldots \ldots \ldots \ldots (12)

\[
Var (P(F)) = (1-P(B)) \sum_{i=1}^{n+1} Var (P(A_i))
\] \ldots \ldots \ldots \ldots \ldots (13)
By applying 3-sigma principle [10], the upper bound can be calculated as follows.

\[
P_{\text{upper}} = E(P(F)) + 3\sqrt{\text{Var}(P(F))}.
\]  

(14)

Normal and the uniform probability functions, which are given in equation (15) and (16), were selected to describe the distribution of 1-\(p_1\) and \(\alpha_i\), respectively.

\[
f_+ (x) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left(-\frac{(x - \mu)^2}{2\sigma^2}\right)
\]

(15)

\[
f_- (x) = \begin{cases} 0, & x < L \text{ or } x > U \\ \frac{1}{U - L}, & L \leq x \leq U \end{cases}
\]

(16)

Equation (12) and (13) are rewritten as follows.

\[
E(P(F)) = \left(n \left(\frac{U + L}{2}\right)^{n-1}(1-\mu)\left(1 - \frac{U + L}{2}\right) + \frac{U + L}{2}\right)
\]

(17)

\[
\text{Var}(P(F)) = (1 - P(B))^2 \sum_{k=1}^{n} \text{Var}(P(A_k)).
\]

(18)

III. Numerical Experiments

In the experiments, two real-world software projects, which are conducted by Cai [5] and Eckhardt [4] before, are adopted to assess our method. The first one is NASA 4-University Project studied in 1985 [4]. The second project sponsored in 2002 [5]. The quantitative results of the two experiments are listed in Table I.

**TABLE I. QUANTITATIVE RESULT IN OPERATIONAL TEST**

<table>
<thead>
<tr>
<th>Item</th>
<th>First Project</th>
<th>Second Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of test cases</td>
<td>920,746</td>
<td>100,000</td>
</tr>
<tr>
<td>Average failure rate of single version</td>
<td>0.06881</td>
<td>0.00139</td>
</tr>
<tr>
<td>2-version coincide failures</td>
<td>21173</td>
<td>57</td>
</tr>
<tr>
<td>Coincide failure rate of 2-version</td>
<td>1.2103×10^-4</td>
<td>1.0200×10^-6</td>
</tr>
<tr>
<td>Average failure rate of 3-version</td>
<td>2.0171×10^-5</td>
<td>9.5000×10^-8</td>
</tr>
<tr>
<td>3 or more versions coincident failures</td>
<td>372</td>
<td>0</td>
</tr>
<tr>
<td>Average failure rate of 4-version</td>
<td>8.7118×10^-9</td>
<td>0</td>
</tr>
</tbody>
</table>

In the first experiment, the parameters are selected as follows. \(\mu=0.93119, \sigma=0.02, U=0, L=2.4206e^{-4}\). \(P(B)=0\), here, \(\mu\) equals to 1-Average failure probability of single version. With 3-sigma principle, selected \(\sigma\) can ensure to cover the probabilities of all versions. Because \(E(\alpha)=(U+L)/2\), which is the average coincide failure rate of 2-version, and \(U\) equals to 0, \(L\) is set to 2.4206e-4. By using equation (17), (18) and (14), the mean and the upper bound of reliability are calculated. The calculated values with our method are compared with those obtained by the method which set the versions as independent [3], those obtained by the method used in reference [9] and those obtained by realistic software test, which is listed in Table II.

**TABLE II. COMPARISON OF CALCULATED VALUES, REALISTIC SOFTWARE TEST AND INDEPENDENT VERSIONS**

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>Upper Bound</th>
<th>Real</th>
<th>Independent</th>
<th>Method in ref.[9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>failure rate of 3-version system</td>
<td>2.498×10^2</td>
<td>2.0171×10^-5</td>
<td>3.2580×10^-6</td>
<td>2.5010×10^-5</td>
<td></td>
</tr>
<tr>
<td>failure rate of 4-version system</td>
<td>4.031×10^-3</td>
<td>8.711×10^-9</td>
<td>3.2580×10^-6</td>
<td>3.2580×10^-6</td>
<td></td>
</tr>
</tbody>
</table>

The values presented in Table II illustrated that the failure rate calculated by our method are the same order with realistic test value, both the mean and the upper bound. Compared to the independent versions method, obtained results with ours are closer to real results. Compared to the result using method in reference [9], obtained mean value with ours approximates their method. When number of versions is larger than 3, obtained upper bound value using our method is closer to realistic value than method in reference [9]. Figure 1 illustrates the relationship between the system failure rate and number of versions. From Figure 1, we can see that the failure rate will trend to 0 when increasing the number of versions. This shows that our method is better than that of independent versions method. Furthermore, the order values obtained from our method are equal to results obtained by method in reference [9].

![Figure 1. The number of versions on failure rate in the first (left) and second (right) experiment](image)

In the second experiment, the parameters are \(\mu=0.99861, \sigma=0.0001, U=0, L=2.046e^{-6}\), \(P(B)=0\), which are selected in the same way as the first experiment. The results are shown in Table III and Figure 1.

**TABLE III. COMPARISON OF CALCULATED VALUES, REALISTIC SOFTWARE TEST AND INDEPENDENT VERSIONS**

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>Upper Bound</th>
<th>Real</th>
<th>Independent</th>
<th>Method in ref.[9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>failure rate of 3-version system</td>
<td>4.253×10^-4</td>
<td>6.0256×10^-8</td>
<td>9.5000×10^-8</td>
<td>2.6856×10^-9</td>
<td></td>
</tr>
<tr>
<td>failure rate of 4-version system</td>
<td>5.784×10^-6</td>
<td>8.2085×10^-14</td>
<td>3.7330×10^-12</td>
<td>5.9678×10^-15</td>
<td></td>
</tr>
</tbody>
</table>

The values presented in Table III illustrated that the failure rate calculated by our method are the same order with realistic test value, both the mean and the
upper bound. Compared to the independent versions method, obtained results with ours are more close to real results. Compared to the result using method in reference [9], obtained mean value with ours approximates their method. When number of versions is larger than 2, obtained upper bound value using our method is closer to realistic value than method in reference [9]. Figure 1 illustrates the relationship between the system failure rate and number of versions. From Figure 1, we can see that the failure rate will trend to 0 when increasing the number of versions. This shows that our method is better than that of independent versions method.

Different from the first experiment, when \( n = 3 \), reliability obtained by our method is not as high as that obtained by independent one. This may be caused by the high reliability of each version in the second experiment. With high quality function module, the increase of version number may not be so impressively improving the system’s reliability as lower ones. Furthermore, we can see that the upper bound values and mean values as well as result using method in reference [9] are always in different order. The upper bound values are closer to realistic test value. This shows that when value of \( \alpha \) is little and version is high quality, which is a general situation, adapting upper bound value calculated by our method as the failure rate is better than result using method in reference [9].

IV. Conclusion

In this paper, the stochastic approach of both failure rate of single version \( p_i \) and coincide failure rate \( \alpha \) between versions are considered. We derive the equations to calculate mean and variance of entire system’s reliability.

Using these results, we can evaluate the upper bound of entire system’s reliability. Two real-world software projects are adopted to assess our method. The results show that both the mean and upper bound values are closer to realistic value than independent model. Also, the upper bound value is closer to realistic test value than result obtained by using method in reference [9], which demonstrates the effectiveness of our method.

REFERENCE


