Software Fault Tolerance: \( t/(n-1) \)-Variant Programming

Jie Xu  
University of Newcastle upon Tyne  
Brian Randell  
University of Newcastle upon Tyne

Key Words — Related software fault, Software fault tolerance, Software reliability, Software safety, System-level fault diagnosis.

Summary & Conclusions — This paper describes the software fault tolerance scheme, \( t/(n-1) \)-Variant Programming (\( t/(n-1) \)-VP), which is based on a particular system diagnosis technique used in hardware and thereby has some special advantages involving a simplified adjudication mechanism and enhanced capability of tolerating faults. The dependability of the \( t/(n-1) \)-VP architecture is evaluated and then compared with two similar schemes: N-version programming (NVP) and \( N \) self-checking programming (NSCP). The comparison shows that \( t/(n-1) \)-VP is a viable addition or alternative to present techniques.

Much of the classical dependability-analysis of software fault tolerance approaches has focused on the simplest architectural examples that tolerate only single software faults, without considering tolerance to multiple and/or related faults. The results obtained from such analyses are thus restricted. The dependability evaluation in this paper deals with more-complicated & general software redundancy: various architectures tolerating two or more faults. It is no surprise that we come to new conclusions: both \( t/(n-1) \)-VP and the NVP scheme have the ability to tolerate some related faults between software variants; in general, \( t/(n-1) \)-VP has higher reliability, whereas NVP is better from the safety viewpoint.

1. INTRODUCTION

Acronyms & Abbreviations

Dep-F \( s \)-dependent fault\(^1\)  
Indep-F \( s \)-independent fault  
NMR \( N \)-modular redundancy  
NSCP \( N \)-self-checking programming  
NVP \( N \)-version programming  
NVS sequential application of NVP  
RB recovery block(s)  
SFT software fault tolerance  
\( t/(n-1) \)-VP \( t/(n-1) \)-variant programming.

Hardware fault tolerance has been a common practice for many years and forms a vital part of any dependable computing system. A relatively new development is the fault tolerance techniques for coping with unanticipated faults such as design (typically software) faults [1]. In principle, simple replication of software components is insufficient because software design faults can be reproduced when redundant copies are made [2]. SFT usually requires design diversity. For design-diversity, two or more variants of a component for redundant computations are \( s \)-independently designed to meet a common service specification. Variants are aimed at delivering the same service, but implemented in different ways in the hope that they do not contain the same design faults. Since at least two variants are involved, tolerance to design faults necessitates an adjudicator [3] (a decision algorithm) that determines a single (assumed to be) error-free result based on the results produced by multiple variants. Several techniques have been proposed for structuring a software system, and providing SFT: RB [2], NVP [4], NSCP [5], and some intermediate or combined techniques e.g., [6, 7]. These techniques are complementary to (not a substitute for) those for achieving software fault avoidance such as verification & validation, software testing, and proof methodology.

RB was the first scheme developed for achieving SFT [2, 8]; variants are organized in a manner similar to standby-sparing [9] used in hardware. RB performs run-time fault detection by augmenting any conventional hardware/software error detection mechanism with an acceptance test applied to the results of execution of one variant. If the test fails, an alternate variant is invoked after backward error recovery is performed. Researchers at UCLA devised NVP [4] which directly applies the hardware NMR [9] to software. \( N \) variants (variants) of a program that have been \( s \)-independently designed are executed in parallel and their results compared by an adjudicator. By incorporating a majority vote, the system can eliminate erroneous results (the minority) and pass on the presumed-correct results (the majority). In simple cases, the voting can be based on tests for identity; in general, a more sophisticated & application-oriented test is needed. The \( N \) variants in NVP can be executed sequentially. Gnagnar, Arlat, Avizienis [10] sketched such a sequential application of NVP, called NVS. Laprie, et al [5, 11] developed NSCP which attains fault tolerance by the parallel execution of \( N \) self-checking software components. Each self-checking component consists of a pair of variants with a comparator; one self-checking component is regarded as the active component, and the others are considered hot-standby spares.

The success of a SFT scheme depends largely upon its adjudicator, and unreliability in the adjudicator dramatically reduces the system reliability [11]. The design for a highly reliable adjudicator generally requires that:

- the adjudication mechanism and variants being checked are as \( s \)-independent as possible, so that they cannot be affected by common faults or Dep-F;
- the mechanism itself must be simple enough to guarantee its reliability and the system performance.

The traditional mechanisms are not entirely satisfactory. In RB software, an acceptance test is used in its adjudication mechanism to provide a last line of detecting errors, but since

---

\(^1\)Also often referred to as: related fault.
the test is system-specific (and as such, very little specific guidance can be given for its construction), it is difficult to ensure that the acceptance test and variants are mutually
$s$-independent. To overcome this problem, some schemes adopt an adjudication mechanism that selects the results by comparing the outputs of multiple variants. However, a practical adjudicator used in NVP is much more sophisticated than the early simple majoritv vote, while adjudication mechanisms constructed in NSCP are too simple to detect effectively the Dep-
$F$ that can occur in the active self-checking components.

Section 2 develops an alternative, $t/(n-1)$-VP, which exploits several new research results in system diagnosis [12, 13] for designing a simplified adjudication mechanism. $t/(n-1)$-
VP has several favorable characteristics, eg,

- the potential ability to tolerate multiple Dep-$F$ among variants,
- simple adjudication mechanism that requires only $O(n)$ result comparison steps,
- delivery of correct service even when the number of faulty variants exceeds the bound $t$ in some fault situations,
- possible forms of graceful degradation.

It cannot be guaranteed that physically-independently designed variants will fail $s$-independently, despite the adoption of the design diversity [14, 15]. The dependability analysis of SFT systems must include the effect of Dep-$F$. Several papers devoted to such dependability analysis are [6, 14 - 17]. In particular, Arlat, Kanoun, Laprie [16] developed complete fault classifications and presented a detailed evaluation of NVP & RB. Their analysis concentrated on basic architectures able to tolerate a single fault and thereby the analytic conclusions hold only for those specific instances.

Section 3 augments published work by:

- analyzing more complex (more general) architectures that tolerate two or more software faults;
- carefully identifying the ability of various approaches to tolerate Indep-$F$ & Dep-$F$.

These results provide designers with:

- richer information about the SFT properties of various architectures than the results from traditional analysis;
- evidence that the $t/(n-1)$-VP approach is a viable addition or alternative to present schemes for coping with software faults.

Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_i$</td>
<td>state: adjudicator’s execution</td>
</tr>
<tr>
<td>$B, C$</td>
<td>state: [benign, catastrophic] failure caused by an undetected error</td>
</tr>
<tr>
<td>$C_i$</td>
<td>result comparator</td>
</tr>
<tr>
<td>$C_x$</td>
<td>$Pr{\text{catastrophic failure of approach } X}$</td>
</tr>
<tr>
<td>$D, U$</td>
<td>state: [detected, undetected] failure</td>
</tr>
<tr>
<td>$E$</td>
<td>state: software execution</td>
</tr>
<tr>
<td>$F_x$</td>
<td>$Pr{\text{failure of approach } X}$</td>
</tr>
<tr>
<td>$I$</td>
<td>state: software is idle during the specified exposure period</td>
</tr>
<tr>
<td>$N, n$</td>
<td>number of software variants</td>
</tr>
<tr>
<td>$p$</td>
<td>$Pr{\text{all variants produce the same correct results}}$</td>
</tr>
<tr>
<td>$q_i$</td>
<td>$Pr{\text{an Indep-$F$ in a variant, the adjudicator}}$</td>
</tr>
<tr>
<td>$q_{a,d}$</td>
<td>$Pr{\text{a detected, undetected Indep-$F$ in the adjudicator}}$</td>
</tr>
<tr>
<td>$q_{a,u}$</td>
<td>$Pr{\text{a catastrophic failure due to an undetected error}}$</td>
</tr>
<tr>
<td>$r_i$</td>
<td>result of $V_i$</td>
</tr>
<tr>
<td>$R(t), S(t)$</td>
<td>${\text{reliability, safety}}$ of approach $X$</td>
</tr>
<tr>
<td>$V$</td>
<td>state: variant execution</td>
</tr>
<tr>
<td>$\omega_{i+1}$</td>
<td>(comparison) test outcome</td>
</tr>
<tr>
<td>$\delta(\cdot)$</td>
<td>indicator function: $\delta(\text{True})=1$, $\delta(\text{False})=0$.</td>
</tr>
</tbody>
</table>

Other, standard notation is given in “Information for Readers & Authors” at the rear of each issue. Less frequently used notation is defined in the text where it appears.

2. $t/(n-1)$-VP

In the theory of system-level fault diagnosis, the $t/(n-1)$-
diagnosability measure was introduced in [18]. Its diagnosis goal is, for a system of $n$ units, to isolate the faulty units to a set of size at most $(n-1)$, under the condition that the number of faulty units is at most $t$. That is: at least 1 unit exists such that it is not in the set of size $(n-1)$ and can thus be unambiguously identified as fault-free, provided that the system itself is $t/(n-1)$-fault diagnosable and the number of faulty units in the system does not exceed the bound $t$. Thus the $t/(n-1)$-diagnosis technique can be used to select a single correct result from the results generated by $n$ replicated software modules (of $s$-independent design). We might benefit from the use of $t/(n-1)$-diagnosis since this special diagnosis measure appreciably reduces the requirement on the number of tests (number of result comparisons) relative to previous diagnosis schemes. It is thus possible to use the idea behind the $t/(n-1)$-diagnosis technique to construct a simple, dependable adjudication mechanism. Based on current theoretical results of $t/(n-1)$-diagnosis (see [19 - 24] and a subsequent discussion), we develop a new scheme for tolerating hardware and/or software faults. This scheme is described first in terms of application to SFT, but the approach can be implemented with hardware [22].

Two classes of software faults are distinguished: Indep-$F$ & Dep-$F$ [11, 16]. Dep-$F$ occur in single variants or in the adjudication mechanism, while Dep-$F$ can take place among multiple variants and among the adjudicator and one or more variants.

2.1 Description of $t/(n-1)$-VP and an Example

A general $t/(n-1)$-VP architecture can identify the correct result from a subset of the results of $n$ software modules
(or variants), provided that the number of faulty modules in the architecture does not exceed \( t \) (i.e., it can tolerate at least \( t \) software faults). The 4 steps of \( t/(n-1) \) VP are:

1. Each of \( n \) s-independently designed software variants is executed in parallel.
2. Some, but not all, of their results are compared to produce a syndrome.
3. Using the syndrome, a diagnosis program performs \( t/(n-1) \)-diagnosis and selects a presumably correct result as the system output (e.g., through switching of the results).
4. If no acceptable result is identified in step 3, the system invokes spare software variants, if they exist, or simply signals an exception.

2.1.1 Example

This example demonstrates the ability of \( t/(n-1) \) VP to tolerate both Indep-F & Dep-F.

![Diagram](image)

**Table 1. Possible Syndromes and Result Selections**

<table>
<thead>
<tr>
<th>( \omega_{1,2} )</th>
<th>( \omega_{2,3} )</th>
<th>( \omega_{3,4} )</th>
<th>Presumably Correct Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( r_1, r_2, r_3, r_4 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>( r_1, r_3, r_5 )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>( r_2, r_3 )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( r_5 )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>( r_5, r_4 )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( r_5 )</td>
</tr>
</tbody>
</table>

2.1.2 \( t/(n-1) \) VP architecture for given \( n, t \)

Unlike the NVP scheme and its variations, \( t/(n-1) \) VP does not have to make pair-wise comparisons among the results of \( n \) variants in order to identify a presumably correct result. We do calculate how many result comparisons (corresponding to the comparators illustrated in figure 1) are usually required for a general \( t/(n-1) \)-architecture.

In the simplest case, \( n=3 \) and \( t=1 \), one comparator is necessary & sufficient for \( t/(n-1) \)-diagnosis — the third result must be acceptable when the two compared results disagree; otherwise they can be identified as correct.
Larger \( n \) & \( t \) require a more deliberate comparison assignment among the results of multiple variants so as to guarantee \( t/(n-1) \)-diagnosability. For example, result comparisons of \( n \) variants can be organized into a form of chains, where \( C_i \) (\( 1 \leq i \leq n-1 \)) compares the results of \( V_i \) & \( V_{i+1} \). Alternatively, result comparisons can be organized into a more complex structure, \( H_{r,n} \), where the result of \( V_i \) (\( 1 \leq i \leq n \)) is compared with that of \( V_j \) if

\[
i-r \leq j \leq i + r \mod (n+1), \quad r = 1, 2, 3, \ldots .
\]

Ref [22; theorem 4] gives several sufficient conditions on the result comparison assignments of systems that are \( t/(n-1) \)-diagnosable; the result is reproduced here.

**Theorem 1.** A system \( S \) composed of \( n \) units (or software variants) is \( t/(n-1) \)-diagnosable if \( t \geq 2t + 1 \) and the assignment of result comparisons in \( S \) contains at least:

- \( 1 \leq t \leq 2 \): a chain of \( 2t \) units;
- \( 3 \leq t \leq 4 \): a chain of \( 2t + 1 \) units;
- \( 5 \leq t \leq 6 \): an \( H_{2,r,n} \) structure with \( r = 1 \);
- \( 7 \leq t \): an \( H_{2r,n} \) structure with \( r \geq \lceil t-1/5 \rceil \).

Because the major aim of this paper is to show how the \( t/(n-1) \)-diagnosis technique could be applied to the design of SFT schemes, this particular technique is not discussed further; see [20, 22 - 24] for more technical details. The diagnosis algorithms with respect to the testing assignments in theorem 1 have been developed in [23] for chains and in [24] for \( H_{2r,n} \)-type systems. For practical values of \( n \) (eg, \( 3 \leq n \leq 10 \)), a \( t/(n-1) \)-VP architecture uses only \( O(n) \) comparators and contains a simple diagnosis algorithm with linear complexity. The adjudicator in such an architecture is simpler than a voter used in NVP (which has to be based on \( O(n^2) \) result comparison steps).

For any SFT scheme, the correctness of results output by the system cannot always be guaranteed (eg, when more than \( t \) faults have occurred), and moreover such fault situations cannot be detected completely. However, this is not a severe deficiency; in practice, there are acceptable probabilities of catastrophic events, eg, an aircraft computer system usually accepts a failure-probability \( < 10^{-8} \)/hour in a 10-hour flight [25]. Dependability studies on practical fault-tolerant systems can be used to determine the \( \Pr \{ \text{occurrence of } t \text{ faults} \} \). This helps to make an appropriate design decision as to which scheme is likely to be most effective and how many variants are sufficient for a particular application. Additional fault-detection & exception-handling techniques [26] can be used to improve fault coverage & fault-tolerance. In the 2/(5-1)-architecture of figure 1, for example, exception-handlers can be incorporated into the variants. The function of the handlers in a variant is to handle any errors that are detected during the execution of the variant, and signaling an exception to the diagnostor. The diagnostor comes to final decision according to the value of the syndrome and the exception signals received so far: either it delivers a presumably correct result, or signals a failure exception.

### 2.2 Comparison with Other Schemes

The \( t/(n-1) \)-VP resembles other SFT techniques, especially those requiring the use of result comparisons such as NVP & NSCP. The fact that the variants are executed in parallel necessitates an input-consistency mechanism and a synchronization regime, based essentially on wait & send primitives, and incorporating a time-out mechanism. However, in each case there are important & fundamental distinctions. Correct results in \( t/(n-1) \)-VP are not obtained by majority vote (as in NVP), or by detecting & discarding erroneous results (as in NSCP), but by \( t/(n-1) \)-diagnosis.

#### 2.2.1 NVP

It could be argued that \( t/(n-1) \)-VP is only a variation of NVP; however, we believe that the majority voting check is an integral part of NVP, and each of \( N \) software versions in NVP is of equal importance. In marked contrast, the \( t/(n-1) \)-VP scheme does not try to find a majority of \( n \) results, but just to identify a presumably correct result. It can therefore deliver correct results with some probability even when the majority of results of \( n \) variants are incorrect. Moreover, \( t/(n-1) \)-VP has more flexible architecture. In the architecture of figure 1, the \( V_i \) can be considered as being active, actually delivering the system output in the absence of faults; the \( V_1 \) & \( V_2 \) are used as "hot" spares, and \( V_1 \) & \( V_2 \) are only exploited for detecting errors and producing test outcomes. In addition, NVP requires that all variants should be designed to produce the results that are essentially identical. This constraint can be loosened in the \( t/(n-1) \)-VP approach. While the primary variant \( V_1 \) in the \( 1/(5-1) \)-VP architecture should attempt to produce the desired output, the spare variant \( V_1 \) might only attempt to provide a degraded service. In this form, the \( t/(n-1) \)-VP architecture can be used to implement a type of graceful degradation.

#### 2.2.2 NVS

In principle, \( t/(n-1) \)-VP is also different from NVS (a form of sequential NVP) [10]. The \( t/(n-1) \)-VP method is based on hot-standby redundancy, whereas NVS uses cold-standby. More precisely, where the results of \( V_1 \) & \( V_2 \) disagree (assuming \( N = n = 3 \)), \( t/(n-1) \)-VP selects the result of \( V_3 \), which has been available, as the system output through the result switch. NVS however has to execute first the \( V_1 \) on the same set of input values and then makes a further decision by searching for a majority of the results. This validation process requires the extra execution time for \( V_1 \) and for the final decision. Clearly, in comparison with \( t/(n-1) \)-VP, NVS has relatively poor predictability of task completion time and can be inappropriate for certain time-critical applications.

#### 2.2.3 NSCP

It could be argued that \( t/(n-1) \)-VP is somewhat similar to NSCP. However, a fundamental distinction between the two schemes concerns their capacity for tolerating Dep-F. NSCP fails (and even causes catastrophic consequences) whenever the two variants that form the active self-checking component
produce identical, but incorrect results (no matter how many spares are still available). In contrast, \( t/(n-1) \)-VP can tolerate up to \( t \) Indep-F or Dep-F; that is, it can deliver the correct service even if \( t \) faulty variants compute identical incorrect results.

2.2.4 RB

Section 2.1 shows that \( t/(n-1) \)-VP is quite distinct from the RB concept [2]. Like NVP and its variations, \( t/(n-1) \)-VP is complementary in many respects to RB. RB can be more appropriate for those systems where hardware resources are limited and comparison-based adjudicators are inappropriate; see [1] for a very detailed discussion of the relative advantages & disadvantages of NVP & RB. For simplicity & brevity, we focus on the comparison of \( t/(n-1) \)-VP with NVP & NSCP, without further discussing RB.

3. DEPENDABILITY OF SFT APPROACHES

This section evaluates the dependability of \( t/(n-1) \)-VP and other similar approaches. Arlat, Kanoun, Laprie [16] analyzed some special architectures using RB, NVP, NSCP — mainly providing software redundancy able to tolerate single software faults. We

- use their modeling framework for investigating the software redundancy needed to tolerate two or more faults,
- establish a slightly different model to show the different impacts of Indep-F & Dep-F on software dependability.

Three architectures are analyzed that can tolerate at least two software faults:

- \( t/(n-1) \)-VP using 5 variants, adopting a simple diagnosis algorithm for result selection (see table 1).
- NVP using 5 variants, adopting the usual majority adjudication.
- NSCP using 6 variants organized as 3 self-checking components; the NSCP considered here can tolerate 2 faults in most fault situations except the Dep-F that occur in an active self-checking component.

Expressions for \( F_X \) & \( C_X \):

\[
X \in \{ t/(n-1)-VP \text{, NVP, NSCP} \},
\]

are derived using a Markov approach.

3.1 Assumptions

1. During the execution of scheme \( X \), Dep-F manifest themselves in the form of similar errors, whereas Indep-F cause only distinct errors. Similar errors lead to common-mode failures, and distinct errors cause only \( s \)-independent failures.
2. All variants have the same probability of fault manifestation (or error).
3. Only a single fault type, either Indep-F or Dep-F, may appear during the execution of the scheme and no compensation is determined by the adjudicator, \( \text{i.e.} \) either an error is detected or it causes an incorrect output.
4. Probabilities of Indep-F & Dep-F are importantly low such that \( p = 1 \) (as assumed by others in similar settings, \text{e.g.} [16]).

These assumptions are used only to simplify the notation and the complexity in modeling and do not alter the importance of analytic conclusions. In particular, assumption \#2 can be easily generalized to the case where the variants have respective fault characteristics. More complex models can be developed without applying assumption \#4, \text{i.e.} probabilities of Indep-F & Dep-F are allowed to be arbitrary; these models are described in [17, 27].

3.2 Detailed Reliability & Safety Models

Notation

\( t \) specified exposure time.

Two different but complementary attributes of dependability are considered:

- continuity of service (reliability),
- non-occurrence of catastrophic failure (safety) [28].

In general,

- software-reliability is a measure of time-to-failure,
- software-safety is a measure of the time to catastrophic failure [16, 28].

The time (or the specified exposure period) in this definition is a relative concept and can mean a single run, several runs, or time expressed in calendar or execution time units, of software. For multiple runs, software can be idle between its executions. However software faults can manifest themselves only when software is executed. We therefore focus on the execution process of software. Figure 2 shows a slight variation of the software behavior model [16]. In this behavior model, a detected failure (no service is delivered) is classified as benign; an undetected failure (an incorrect result is delivered) can be either benign or catastrophic. Since several runs are possible, service delivery can be restored from benign failures. Transitions from \( D \) or \( B \) to \( I \), and from \( U \) to \( B \) or \( C \), are applied only to the safety evaluation. Based on a Markov approach to modeling,

\[
R_X(t) = \exp(fc(\sigma \cdot F_X \cdot t)),
\]

for a detailed discussion see [16];

\[
S_X(t) = \exp(fc(\sigma \cdot C_X \cdot t)).
\]

3.3 \( t/(n-1) \)-VP Model for 2/(5-1)-Architecture

Figure 3 is a state-transition diagram for the 2/(5-1)-architecture. From state \( E \), the 6 execution states of the adjudicator are listed here.
For failure

Figure 2. A Modified Behavior Model

![Figure 2: A Modified Behavior Model](image)

Figure 3. \( t/(n-1) \)-VP Model

\[
A1. 5 \text{ variants produce the same correct results. According to assumption #4},
\]

\[
p = 1 - 5q_1 - 10q_2 - 10q_3 - 5q_4 - q_5 - 10q_{2,v} - 10q_{3,v} - 5q_{4,v} - q_{5,v} - q_{A,v} = 1.
\]

Given no fault in any variant, different types of adjudicator failure lead to states D & U with probabilities \( q_{A,D} \) and \( q_{A,U} \):

\[
A2 - A3: \text{Activation of 1 or 2 Indep-F in variants, given no Dep-F among the variants. These fault types can be tolerated by this } 2/(5-1)\text{-architecture.}
\]

\[
A4 - A6: \text{At least 3 Indep-F manifest themselves in variants. Since the number of faults has exceeded the bound 2, these states can lead to a failure state. However, a more precise analysis shows that } t/(n-1)\text{-VP can still deliver a correct result in some situations.}
\]

\[
A7: \text{Activation of Dep-F in any two variants. These faults can be tolerated.}
\]

\[
A8 - A10: \text{Dep-F manifest themselves in more than 2 variants, which are undetectable.}
\]

\[
A11: \text{Activation of Dep-F between the adjudicator and the variants. This is also regarded as undetectable (see assumption #3).}
\]

In this 2/(5-1)-VP model, there is the transition from state A4 (or A5) to state I: the architecture can still select a correct result as the system output even in the presence of more than two faults. Without loss of generality, take state A4 as an example. If 3 Indep-F affect only 3 of variants \( V_1, V_2, V_3 \), then by assumption #1 their results generate the syndrome where \( \omega_{1,2} = \omega_{2,3} = \omega_{3,4} = 1 \). The result of \( V_3 \) (a correct result) is then chosen as the system output. This class of events occurs with probability \( 4q_3^3 \). Similarly, if 3 Indep-F affect only \( V_1, V_2, V_3 \) (or only \( V_3, V_4, V_5 \)), according to table 1, the selected result can be correct, with probability \( 2q_3^3 \). In summary, the conditional probability of the transition from A4 to I is:

\[
(4q_3^3 + 2q_3^3)/10q_3^3 = 0.6.
\]

Therefore, the transition from A4 to a failure state can occur with the conditional probability:

\[
4q_3^3/10q_3^3 = 0.4.
\]

From figure 3, it follows that:

\[
F_{t/(n-1)-VP} = p \cdot (q_{A,D} + q_{A,U}) + 4q_1^3 + 4q_3^3 + q_5^3 + 10q_{3,v} + 5q_{4,v} + q_{5,v} + q_{A,v}.
\]

A close but pessimistic approximation is:

\[
F_{t/(n-1)-VP} = q_{A,D} + q_{A,U} + 4q_1^3 + 4q_3^3 + q_5^3 + 10q_{3,v} + 5q_{4,v} + q_{5,v} + q_{A,v}.
\]

(1)

For evaluation of safety, only state C is absorbing:

\[
C_{t/(n-1)-VP} = q_{C} \cdot [q_{A,U} + 4q_1^3 + 4q_3^3 + 5q_{4,v} + q_{5,v} + q_{A,v}].
\]

(2)

3.4 NVP Model for 5VP-Architecture

![Figure 4: NVP Model](image)
Figure 4 shows the NVP model for 5VP-architecture. The detailed analysis is similar to that for \( t/(n-1) \)-VP. A major difference is the case where multiple Indep-F have an impact on 3 or more variants. In NVP, this case is much simpler — these faults always lead to state D, assuming they are always detectable (but not tolerated). Thus, for reliability,

\[
F_{\text{NVP}} > F_{t/(n-1) \text{-VP}}.
\]

\[
F_{\text{NVP}} = q_{A,D} + q_{A,U} + 10q_{j} + 5q_{j}^2 + q_{j}^3 + 10q_{3,V} + 5q_{4,V} + q_{5,V} + q_{A,V}.
\]  

(3)

Due to the detectability of multiple Indep-F:

\[
C_{\text{NVP}} = q_{C}[q_{A,U} + 10q_{3,V} + 5q_{4,V} + q_{5,V} + q_{A,V}]
\]

(4)

\[
< C_{t/(n-1) \text{-VP}}.
\]

3.5 NSCP Model for 3SCP-Architecture

Figure 5 shows the NSCP model for the 3SCP-architecture. The interpretations of the states are similar to those of the \( t/(n-1) \)-VP model, though there are 13 states A1 - A13 to consider because there are 6 variants. Indep-F in 1 or 2 of variants can be tolerated. Indep-F in 3 or more variants can be either tolerated or detected, as indicated by states A4 & A5. Thus NSCP is quite effective for treating Indep-F.

However, cases where Dep-F manifest themselves among multiple variants become more complicated. On the one hand, NSCP is not fault-tolerant in the worst case — any Dep-F in active self-checking components could lead to certain failure states. On the other hand, some Dep-F can be tolerated or detected if they do not affect the pair of variants in an active self-checking component. Consider a representative case, state A9, in which Dep-F manifest themselves in 3 of the 6 software variants. The 3 sub-cases are:

1. If Dep-F occur only in the spare self-checking components or such faults affect just a variant in the active component but do not affect the first spare component, the 3SCP architecture can select a correct result and provide usual service. The conditional probability of this sub-case is:

\[
(6/20) \cdot 20q_{3,V}/20q_{3,V} = 0.3.
\]

2. If Dep-F affect exactly 1 variant in every self-checking component, they can be detected effectively; the corresponding conditional probability is:

\[
(2^3/20) \cdot 20q_{3,V}/20q_{3,V} = 0.4.
\]

3. The worst sub-case is that Dep-F influence:

- the pair of variants in the active component, or
- the pair of variants in the first spare component, given these Dep-F have affected a variant in the active one.

In this sub-case, the 3SCP architecture produces incorrect outputs; the corresponding conditional probability is:

\[
(6/20) \cdot 20q_{3,V}/20q_{3,V} = 0.3.
\]

A similar analysis can be applied to other states. Therefore, from the state-transition diagram:

\[
F_{\text{NSCP}} = q_{A,D} + q_{A,U} + 8q_{j}^3 + 12q_{j}^4 + 6q_{j}^5 + q_{j}^6 + q_{2,V} + 14q_{3,V} + 14q_{4,V} + 6q_{5,V} + q_{6,V} + q_{A,V}.
\]  

(5)

Since Indep-F can be either tolerated or detected, safety of the NSCP architecture concerns only Dep-F:

\[
C_{\text{NSCP}} = q_{C}[q_{A,U} + q_{2,V} + 6q_{3,V} + 14q_{4,V} + 6q_{5,V} + q_{6,V} + q_{A,V}].
\]  

(6)

3.6 Remarks

Table 2 summarizes the specific expressions for \( q_{j} \) & \( q_{U} \) and shows that s-independent failures of the variants have a relatively small influence upon \( t/(n-1) \)-VP, but a larger impact on NVP and even more on NSCP. This is because the \( t/(n-1) \)-VP possesses an important characteristics of the \( t/(n-1) \)-diagnosis technique: it is still possible in some fault situations for \( t/(n-1) \)-VP to identify the correct results, even though faulty variants are in the majority.

As assumed at the beginning of section 3, the adjudicators used in the 3 specific architectures are:

- \( t/(n-1) \)-VP: result comparison plus a diagnosis algorithm,
- NVP: a voter,
- NSCP: result comparison (plus the result switch).

According to their complexities, it is reasonable to rank \( q_{A} \) & \( q_{A,V} \) as:

\[
q_{A,\text{NSCP}} \leq q_{A,\text{t/(n-1)-VP}} \leq q_{A,NVP}.
\]  

(7)

\[
q_{A,V,\text{NSCP}} \leq q_{A,V,\text{t/(n-1)-VP}} \leq q_{A,V,NVP}.
\]  

(8)

It follows from table 2 that Dep-F among variants have the same influence upon \( t/(n-1) \)-VP & NVP, but more serious on NSCP.
This is because result-comparison used in the self-checking components and the NSCP architecture itself are not effective enough to detect (or further tolerate) the Dep-F that can affect both variants in a self-checking component. Generally, this cannot be overcome by incorporating more variants into a given architecture. In contrast, both $t/(n-1)$-VP and NVP can tolerate some Dep-F under the same bound; and their fault-tolerance capability can be enhanced, at least in principle, by involving more software variants.

3.6 General Conclusions

For reliability:

$$F_{t/(n-1)-VP} < F_{NVP} < F_{NSCP}. \tag{9}$$

Inequality (9) means that $t/(n-1)$-VP has the lowest failure-probability (highest reliability). Due to high detectability of Indep-F, NVP is less sensitive to undetected faults than is $t/(n-1)$-VP. The $q_{U}(t/(n-1)-VP)$ looks relatively high since this scheme can fail to detect some Indep-F when the bound on the number of faulty variants is violated. This probability can be reduced by using more software variants. The $q_{U}(NSCP)$ is high as well, but again the incorporation of more variants is of no effect on safety enhancement of NSCP. So for safety:

$$C_{NVP} < C_{t/(n-1)-VP} \leq C_{NSCP}. \tag{10}$$

The evaluation data obtained here are used only to uncover the relative advantages & disadvantages of these 3 schemes. For a given design using a particular scheme, the evaluation results also show how the design can be modified to improve further its dependability. Since the notion of software dependability captures many different concerns, including reliability & safety, our analysis demonstrates the need of a delicate balance between these complementary attributes. In practice, a software designer must make an objective decision as to which technique is likely to be most appropriate for a specific application.

ACKNOWLEDGMENT

This research was supported by the ESPRIT Basic Research Actions 3092 and 6362 on Predictably Dependable Computing Systems (PDCS and PDCS2) and by the ESPRIT Long Term Research Project 20072 on Design for Validation (DeVa).

REFERENCES

Changes to: Sensitivity of Reliability-Growth Models to Operational Profile Errors vs Testing Accuracy

1. [1: page 537, col 2, line 23] change: the reliability-growth estimated by GE for 5 cases: ...
   to: the reliability-growth estimated by GE for 4 cases: ...
2. [1: page 538, col 1, line 2] delete the whole line:
   d. OP4 with D(OP4) = 0.85 (solid lower line)
3. [1: page 538, col 2, line 23] change: sensitivity to errors in the OP. Data concerning the coverage to: sensitivity to errors in the OP. Data concerning the other coverage

REFERENCE