Semi-Annual Technical Report Submitted to the

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Langley Research Center, Hampton, Va.

for research entitled

# MULTI-VERSION SOFTWARE RELIABILITY THROUGH FAULT-AVOIDANCE AND FAULT-TOLERANCE

(NAG-1-983)

from

Mladen A. Vouk, Co-Principal Investigator, Assistant Professor
David F. McAllister, Co-Principal Investigator, Professor

Department of Computer Science
North Carolina State University
Raleigh, N.C. 27695-8206
(919) 737-2858

Report Period
Beginning Date: September 1, 1989.
Ending Date: March 31, 1990.

# Table of Contents

# Project Progress Summary

In this project we have proposed to investigate a number of experimental and theoretical issues associated with the practical use of multi-version software in providing dependable software through fault-avoidance and fault-elimination, as well as run-time tolerance of software faults. In the period reported here we have worked on the following:

- We have continued collection of data on the relationships between software faults and reliability, and the coverage provided by the testing process as measured by different metrics (including data flow metrics). We continued work on software reliability estimation methods based on non-random sampling, and the relationship between software reliability and code coverage provided through testing.

- We have continued studying back-to-back testing as an efficient mechanism for removal of uncorrelated faults, and common-cause faults of variable span. We have also been studying back-to-back testing as a tool for improvement of the software change process, including regression testing.

- We continued investigating existing, and worked on formulation of new fault-tolerance models. In particular, we have partly finished evaluation of Consensus Voting in the presence of correlated failures, and are in the process of finishing evaluation of Consensus Recovery Block (CRB) under failure correlation. We find both approaches far superior to commonly employed fixed agreement number voting (usually majority voting). We have also finished a cost analysis of the CRB approach.

This report describes the results obtained in the period September 1, 1989 to March 31, 1990. Detailed reports are attached on "Preliminary Report on Consensus Voting in the Presence of Failure Correlation", and on "Modeling Execution Time of Multi-Stage N-Version Fault-Tolerant Software".

# Preliminary Report on Consensus Voting in the Presence of Failure Correlation*

Mladen A. Vouk
David F. McAllister

Department of Computer Science, Box 8206
North Carolina State University
Raleigh, NC 27695-8206

## Key Words

Voting, Consensus Voting, N-version programming, System reliability, Software fault-tolerance

## Reader Aids

**Purpose:** Present empirical evaluation of Consensus Voting scheme.
**Special math needed for explanations:** Very basic probability and statistics
**Special math needed to use results:** None
**Results useful to:** Reliability analysts, software reliability engineers, software system designers, designers of fault-tolerant software

## Abstract

The effect of failure correlation is to reduce the output space in which a voter makes decisions. A voting strategy called consensus voting may in part compensate for the problems that arise from this when classical, fixed agreement number, voting is employed. Consensus voting automatically adapts to different component reliability and output space cardinality characteristics. Theory predicts that in small output spaces consensus voting on the average performs as well or better than majority voting, while in large output spaces its performance compares with 2-out-of-n voting. Because consensus voting is auto-adaptive it will perform better than majority voting whenever effective space cardinality exceeds two. This work empirically explores this voting strategy by applying it to a large number of functionally equivalent software components. It is confirmed that majority voting strategy provides reliability which is a lower bound, while ideal 2-out-of-n voting strategy provides reliability which is an upper bound, on the reliability by consensus voting.

*Research supported in part by NASA Grant No. NAG-1-983

# 1. Introduction

In a fault-tolerant system based on software diversity and a voting strategy [e.g. 1, 2], there is a difference between correctness and agreement. McAllister, Sun and Vouk [7] distinguish between agreement and correctness and develop and evaluate an auto-adaptive voting strategy called Consensus Voting. This strategy is particularly effective in small output spaces because it automatically adjusts the voting to the changes in the effective output space size. They show that majority voting strategy provides a lower bound on the reliability provided by consensus voting, and that an ideal 2-out-of-n voting strategy gives an upper bound on this reliability.

The theory developed in [7] was derived under the assumption of failure independence. In this paper we show that the effect of failure correlation is similar to a reduction in the output space cardinality, and therefore under some conditions, consensus voting can considerably improve reliability of multiversion systems even in the presence of failure correlation. We will argue that if choosing a wrong answer or having no answer has the same impact on the system, then consensus voting should be preferred to majority voting even in the presence of correlated failures. We empirically explore this issue by using 20 functionally equivalent programs developed in a multiversion experiment [4]. The primary aim of this study is investigation of the properties of consensus voting, and not of software diversity and the faults that may be associated with such a process. The versions are used only as a medium for testing the theoretical hypotheses about consensus voting.

Throughout this paper we shall use the terms software "component(s)" and "version(s)" interchangeably. When two or more functionally equivalent software components fail on the *same* input case we shall say that a *coincident* failure has occurred. k failing components give a coincident failure of span k. If a coincident failure of k versions is caused by an identical or similar fault we say that the fault spans k versions. When two or more versions give the same incorrect response, to a given tolerance, we say that an *identical-and-wrong* (IAW) answer was obtained. If the measured probability of the coincident failures is significantly different from what would be expected by random chance (using independent failures model) then we say that the observed coincident failures are *correlated* or *dependent.*

# 2. Voting Strategies

In an m-out-of-n fault-tolerant software (FTS) system the number of functionally equivalent independently developed versions is n, and m is the agreement number, or the number of matching

outputs, which the voting or adjudication algorithm requires for system success [e.g. 5, 6]. In the past, because of cost restrictions n was rarely larger than 3, and m was traditionally chosen as $\frac{n+1}{2}$ for odd m. In general, in **majority voting** $m = \lceil \frac{n+1}{2} \rceil$, where $\lceil\ \rceil$ denotes the ceiling function.

Scott, Gault and McAllister [3] showed that, if the output space is large, and true statistical independence of version failures can be assumed, there is no need to choose $m > 2$ regardless of the size of n, although larger m values offer additional benefits. We will use the term **2-out-of-n voting** for the case where agreement number m=2.

For voting in small output spaces McAllister, Sun and Vouk [7] suggest a third voting technique called **consensus voting.** Let the output space have cardinality r, and let all the output classes have non-zero probability of occurring. Assume that the output classes, j, are labeled 1..r such that output 1 represents the only correct output, while outputs 2..r the possible incorrect output classes. Situations where multiple correct answers are possible are not considered explicitly. All the outputs in one class are assumed to be identical. Let the vector $(n_1,n_2,n_3,..,n_r)$ represent the event where output j occurs $n_j$ times such that

$$n_1 + n_2 + ... + n_r = n.$$

Because the effective output space cardinality is $r < \infty$, answer agreement does not imply correctness. For example, if r=3,and n=5, then majority is m=3. The events where the maximum number of agreeing versions is 3 are (3,2,0), (3,0,2), (2,0,3), (2,3,0), (0,2,3), (0,3,2), (3,1,1), (1,3,1), and (1,1,3). An obvious strategy is to choose the output with the largest frequency. The problem is that only three of the events offer outputs where the agreeing majority is correct. Furthermore, majority voting will fail to deliver an output for events (2,2,1), (2,1,2) and (1,2,2). However, if choosing a wrong answer, or having no answer, has the same impact on the system, then choosing one result with two identical outputs at random is a better strategy (on the average) than declaring system failure. In the example there is a 50 percent chance that the correct output will be selected when this formal strategy is used.

## 2.1 Consensus Voting Algorithm

In **consensus voting** the voter uses the following algorithm to select the "correct" answer:

- If there is a majority agreement ($m \geq \lceil \frac{n+1}{2} \rceil$, $n>1$) then this answer is chosen as the " correct" answer.
- Otherwise, if there is a unique maximum agreement, but this number of agreeing versions is less than $\lceil \frac{n+1}{2} \rceil$, then this answer is chosen as the "correct" one.
- Otherwise, if there is a tie in the maximum agreement number from several output classes then one set is chosen at random and the answer associated with this set is chosen as the "correct" one.

The theory of consensus voting is described in [7]. It is shown that the strategy is equivalent to majority voting when the output space cardinality is 2, and to the 2-out-of-n voting when the output space cardinality tends to infinity provided agreement number is not less than 2. It is also proved in [7] that, in general, the boundary probability below which the system reliability begins to deteriorate as more versions are added is $\frac{1}{r}$. This makes the binary space (and majority) voting a special case with r=2.

## 2.2. Independent failures

Several versions may fail coincidentally because of one or more faults. In an ideal situation the versions would fail independently, and in addition the probability of identical and wrong answers from two or more versions would be zero, or very close to zero. Then, if two versions agreed on an answer that agreement could be equivalenced with correctness, and 2-out-of-n voting would give excellent results because it could tolerate coincident (but not identical) failures of all but two versions. Under the same conditions consensus voting performs at least as well as 2-out-of-n voting. In fact, it may yield marginally better results because the algorithm (as defined in 2.1) allows tie breaking even in the case when all classes offered to a voter contain only one item. Hence, there is a finite probability that in the cases where 2-out-of-n voting fails ($m < 2$) consensus voting will return the correct answer.

The other extreme is the case of binary output space (r=2). An answer is either correct, or incorrect and no distinction is possible among incorrect answers. Majority voting then provides a way of distinguishing the output class that is chosen as the correct answer. In binary output space, consensus voting reduces to majority voting and cannot improve on it.

The problem that majority voting does not solve are the small space situations where the vote fails because a voter is offered more than two classes to select the "correct" answer from but there is no

majority so voting cannot return a decision. The events are the ones where there is no agreement majority but one of the outputs occurs more frequently than any other, and the situation where there is a tie between the maximum number of outputs in two or more output classes.

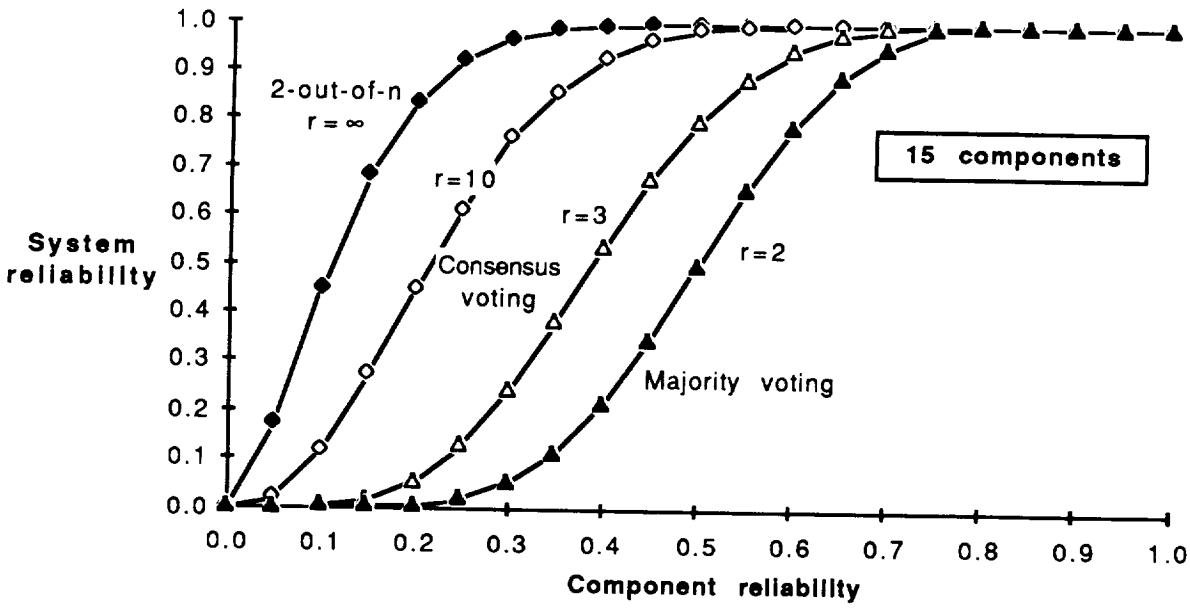


**Figure 2.1** System reliability under different voting strategies vs. component reliability for n=15. The probability of each j=2,..,r failure state is $\frac{1-p}{r-1}$.

If version failures are independent then for $r \geq n > 1$ the lower limit on the agreement number for such events is m=1 because, at worst, all the outputs are different. When $1 < r < n$, there is a finite probability that at least one output of every class will be available, and that the remainder are equally distributed among two or more classes. Hence, the lower limit on m is $\lfloor \frac{n}{r} \rfloor + 1$, where $\lfloor \ \rfloor$ represents the floor function. For i smaller than this limit, there are always more than i outputs in at least one output class. In general, the above special events will occur when i lies between $\lfloor \frac{n+r-1}{r} \rfloor$ and $\lceil \frac{n+1}{2} \rceil$. In the simplest case let the reliability of each version be p, and let each of the wrong outputs occur with equal probability $q_j = q = \frac{1-p}{r-1}$. It can be shown [7] that the probability of obtaining exactly i identical and wrong outputs, Pr{E(i,n)} where E(i,n) = "exactly i outputs from n versions are wrong", is then no larger than

$$(r-1)_n C_i \, q^i \, (1-q)^{n-i}.$$

When r tends to infinity q tends to zero, and the probability of obtaining identical and wrong answers by chance also tends to zero. This fact is the basis for the 2-out-of-n voting strategy. When i is not smaller than the majority this incorrect output will be voted as correct in N-version programming. But, depending on the voting strategy, it may or may not be voted as the correct answer when i is a number between $m = \lfloor \frac{n+r-1}{r} \rfloor$ and $m = \lceil \frac{n+1}{2} \rceil$.

The theoretical relationship between r and the voting strategies derived in [7] is illustrated in Figure 2.1 for n=15. In the figure we plot the reliability of an N-version system based on consensus voting versus the reliability of an average component for different output space cardinality values. Also shown are the 2-out-of-n and majority voting boundary curves. It is important to note that both majority voting and 2-out-of-n voting are effectively output space insensitive. For n odd, the former behaves as if r=2 since agreement number is $m = \frac{n+1}{2}$ which is equivalent to letting r=2 in $m = \lfloor \frac{n+r-1}{r} \rfloor$ for consensus voting. 2-out-of-n is a viable strategy only when r >>1. Consensus voting is r sensitive and therefore will perform better than majority voting for $r > 2$ since $\lceil \frac{n+1}{2} \rceil \geq \lfloor \frac{n+r-1}{r} \rfloor$. The reliability based on majority voting is a lower limit on the reliability of consensus voting, while 2-out-of-n voting reliability is an upper limit on the reliability of consensus voting.

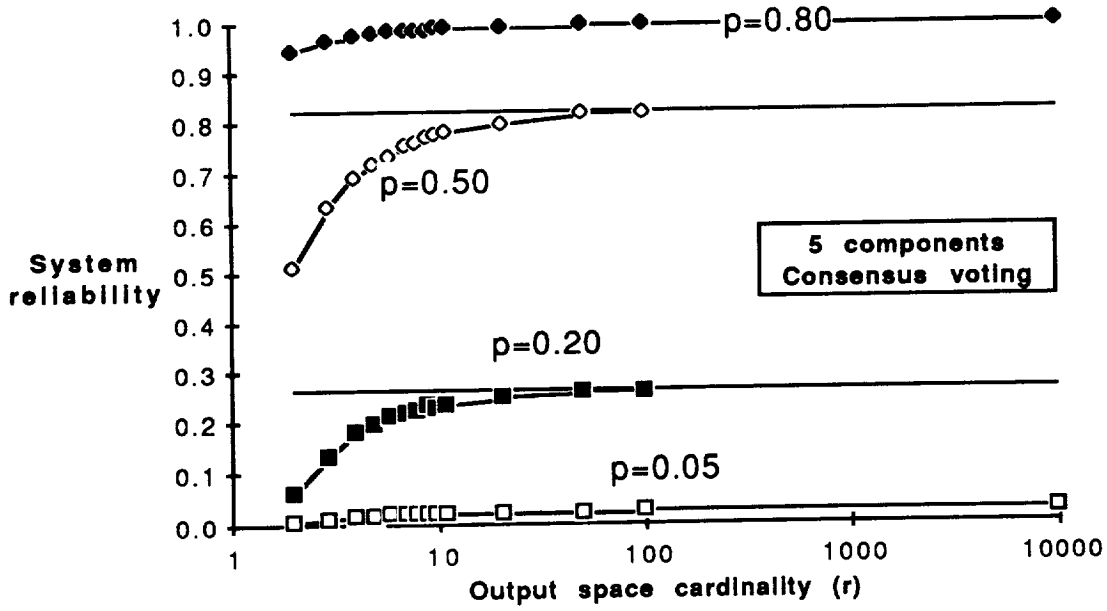


**Figure 2.2.** System reliability vs. output space cardinality for **n=5** using consensus voting. All components have the same reliability, p. The probability of each j=2,..,r failure state is $\frac{1-p}{r-1}$.

The dependence of consensus voting reliability on the output space cardinality and the meaning of $r >> 1$ is further illustrated in Figure 2.2 for n=5. The data points were obtained by simulation. Failure state probabilities were assumed to be the same for all j=2..r incorrect outputs. We note that for a given p the asymptotic system reliability corresponds to the 2-out-of-n voting approach, while the r=2 point corresponds to majority voting. For example, in the figure the asymptotic behavior is observed when $r > 10$ for component reliability $p > 0.8$. In general, larger n values will reduce the influence of small r values.

In practice, failure probabilities of individual versions will be scattered around some mean value. Increasing scatter up to a certain point increases reliability obtained by voting. When the scatter is excessive the system reliability can actually be lower than the reliability of one or more of its best component versions. This effect is illustrated in Figure 2.3. Data shown there were obtained by simulation (100,000 case runs for each point shown). We plot the system reliability based on consensus voting and majority voting against the standard deviation of the sample of version failure probabilities (the mean value being constant). Also plotted for each point is the reliability of the best single version involved in the simulation. The feature to note is the very sharp step in this reliability once some critical value of the standard deviation of the sample is exceeded (about 0.03 in this example).

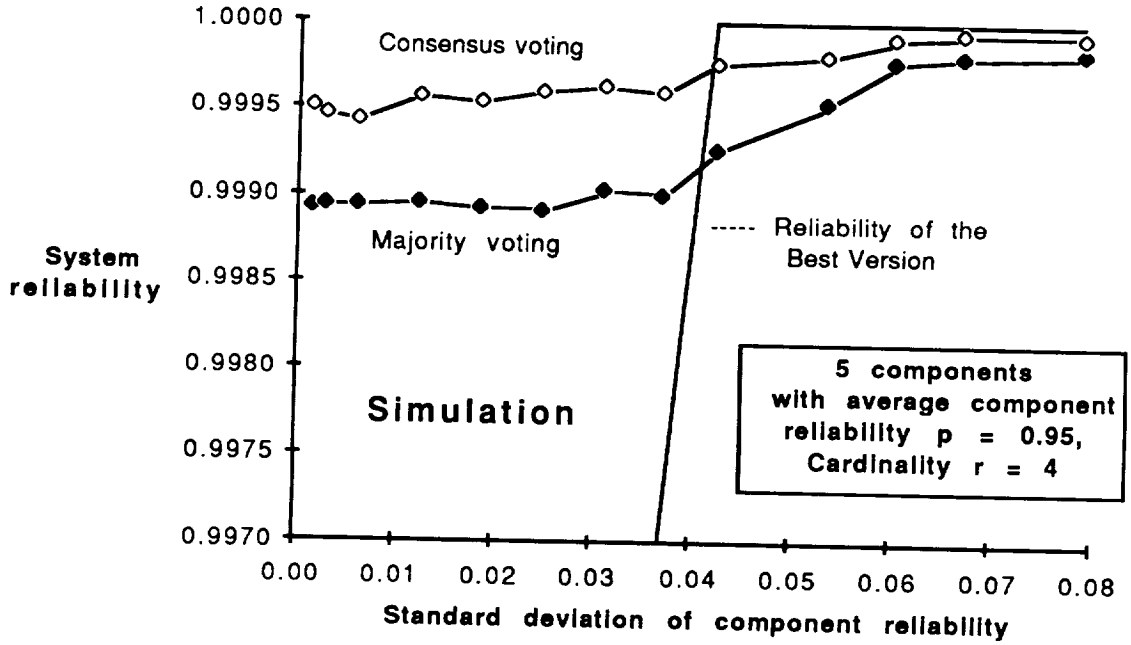


**Figure 2.3.** System reliability vs. standard deviation of version reliability for **n=5** using consensus voting. The probability of each j=2,..,r failure state is $\frac{1-p}{r-1}$.

Similarly it can be shown [7] that increased variability in the failure state probability, i.e. conditional probability that an output is in a particular failure class given that a program has failed, reduces advantages that consensus voting has over majority voting and for sufficiently large scatter around an effective r value consensus voting reduces to majority voting.

## 2.3. Correlated failures

When version failures are correlated [e.g. 2, 8, 9, 10] for some reason the probability of obtaining identical and wrong answers may be considerably increased even for very large r values. In the extreme this reduction may be all the way down to binary output space.

For example, let k out of n functionally equivalent versions have an identical fault which, when excited, results in identical and wrong responses from all k versions. Let probability that the input data profile excites the fault be Pr{f}. Then the probability of k identical and wrong answers has a lower bound of

$$Pr\{E(k,n)\} \geq Pr\{f\}$$

regardless of how large r is. If there are no other faults in the system then whenever a failure occurs there will be k outputs in the output class belonging to fault f, and n-k outputs in the j=1 class (correct answers). For each test case where there is no failure all outputs will be in the j=1 class. This means that in effect the output space cardinality will be reduced to two. Voting algorithms are then presented with two choices (k, or n-k versions) with probability Pr{f}, and with only one choice (n versions) with probability 1-Pr{f}. On the average, the voting algorithm will be offered

$$1 * (1-Pr\{f\}) + 2 * Pr\{f\}$$

classes to chose its response from.

On the other hand, if a fault causes correlated coincident failures of k versions but all failing versions return different responses, then in the presence of only that fault the effective decision space for voting will not exceed k+1. In practice, versions may contain several correlated faults with different spans, different excitation probabilities, and different properties regarding IAW answers. Furthermore, note that $n \geq k$, where n is the number of versions, represents a hard upper bound on the decision space cardinality. So, in general, presence of faults resulting in correlated failures will produce an effect which is equivalent to a reduction of the output space cardinality. In situations like

that a voting strategy which can adapt to changes in the effective output space cardinality may have definite advantages over non-ideal 2-out-of-n (which is fast, but performs worse in reduced space) and majority voting (which returns an incorrect answer as the correct one less often, but fails if there is no majority).

The questions that we wished to explore by experimenting with actual multi-version software containing genuine faults which give rise to correlated version failures, were:

a) Are there situations and version combinations where consensus voting would improve system reliability with respect to majority voting?

b) What are the conditions under which this happens, and do these conditions agree, and if so to what extent, with the theoretical expectations about consensus voting derived using an independent model?

From the discussion in 2.2 we would expect consensus voting:

1. To always offer reliability at least equivalent to majority voting.
2. To offer higher average system reliability than majority voting if the average decision space in which voters operate per test case is large (exceeding 4 or 5 based on Figures 2.1 and 2.2).
3. To offer average system reliability which is not much better than that of majority voting, but may be worse than reliability of one or more of the versions in situations where average version reliability has large standard deviation (unmatched versions).
4. To offer average system reliability that is not much better than that offered by majority voting in situations where the average decision space "seen" by the voters per test case has large standard deviation (e.g. preferential excitation of particular faults, drastically different visibility of different faults to employed testing profile).

## 3. Results

We have used 20 functionally equivalent programs described in [4] to construct an evaluation environment (see Appendix I). The versions and test cases we used were known to result in relatively high intensity correlated program failures. The choice of output variables involved in the voting, and the number and the choice of versions, all influence the mix and intensity of both uncorrelated and correlated failures observed during testing (for a given test set profile). In conducting our analyses we considered a number of combinations of versions and output variables.

We present the results for the 16 combinations described in Appendix II. Twelve combinations represent selections made solely on the basis of individual version reliabilities. For example, combination 3 consists of five versions that exhibited lowest failure rate based on all output values. Combination 8 is the same version set re-evaluated considering only three most critical output variables, while set 11 are the five versions that showed lowest failure rate based on three most critical output variables. Three combinations (11, 14,16) were selected with the intent of forming well balanced sets, i.e. combinations for which the scatter of component version reliabilities around the mean was low.

**Table 3.1** Version characteristics

| Version | Failure Rate (all)* | Failure Rate (best.acc)** |
|---|---|---|
| 1 | 1.00 | 0.58 |
| 2 | 0.15 | 0.07 |
| 3 | 0.30 | 0.13 |
| 4 | 0.28 | 0.07 |
| 5 | 0.26 | 0.11 |
| 6 | 1.00 | 0.63 |
| 7 | 0.90 | 0.07 |
| 8 | 1.00 | 0.35 |
| 9 | 0.63 | 0.40 |
| 10 | 0.84 | 0.004 |
| 11 | 0.40 | 0.09 |
| 12 | 1.00 | 0.58 |
| 13 | 0.29 | 0.12 |
| 14 | 1.00 | 0.37 |
| 15 | 1.00 | 0.58 |
| 16 | 1.00 | 0.58 |
| 17 | 0.88 | 0.10 |
| 18 | 0.13 | 0.004 |
| 19 | 1.00 | 0.58 |
| 20 | 0.50 | 0.34 |

(*) using all 63 output variables, (**) using only 3 best.acceleration values to estimate failure rate. Estimates were obtained using 500 random cases.

Version failure rates measured using all output variables, and using only the three most critical variables (best acceleration values), are shown in Table 3.1. Note that the generally high failure intensities stem from the fact that we used programs which have not been debugged in order to retain the original mix of faults. The characteristics of different combinations of versions and variables are summarized in Tables 3.2. We use standard deviation of the sample to show the extent of scatter of component version failure rates around the sample mean value. Note that sets 14 and 16 are

reasonably well balanced and show small sample standard deviation. Set 15 is set 14 but with all variables taken into consideration which increased the failure rate and scatter.

**Table 3.2** Combination characteristics

| Combination | Versions | Failure Rate per Version | | Variables |
|---|---|---|---|---|
| | | Mean | SampleStD* | |
| 1 | 1-20 | 0.68 | 0.34 | 63 |
| 2 | 2-5,7,9-11,13,17,18,20 | 0.46 | 0.28 | 63 |
| 3 | 2,4,5,13,18 | 0.22 | 0.07 | 63 |
| 4 | 2,5,18 | 0.18 | 0.07 | 63 |
| 5 | 2,4,20 | 0.31 | 0.17 | 63 |
| 6 | 1-20 | 0.29 | 0.23 | 3 |
| 7 | 2-5,7,9-11, 13,17,18,20 | 0.12 | 0.12 | 3 |
| 8 | 2,4,5,13,18 | 0.07 | 0.04 | 3 |
| 9 | 2,5,18 | 0.06 | 0.05 | 3 |
| 10 | 2,4,20 | 0.16 | 0.16 | 3 |
| 11 | 2,4,7,10,18 | 0.04 | 0.03 | 3 |
| 12 | 7,10,18 | 0.02 | 0.04 | 3 |
| 13 | 7,10,18 | 0.62 | 0.43 | 63 |
| 14 | 3,5,11,13,17 | 0.11 | 0.02 | 3 |
| 15 | 3,5,11,13,17 | 0.43 | 0.26 | 63 |
| 16 | 3-5,13 | 0.28 | 0.02 | 63 |

(*) based on a sample of 500 random cases.

The measured coincident failure profiles for all the combinations are shown in Appendix III. The limited number of test cases we have used to get the tables shown in Appendix I did not detect IAW spans exceeding 5. However, the mix of faults and correlated failures that we produced just by using a uniform random testing profile and by monitoring different output variables in different version sets was sufficient to illustrate consensus voting effects.

The voting activity took place for each test case. The outcome was compared to the "golden" answer and the frequency of voting successes and failures was recorded. The results are summarized in Table 3.3.

**Table 3.3** Frequency of voting events.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | **Success Frequency** | | | | | |
| **Best Version** | 434 | 434 | 434 | 434 | 424 | 498 | 498 | 498 | 498 | 466 |
| **2-of-N*** | 457 | 457 | 424 | 424 | 406 | 500 | 500 | 467 | 466 | 466 |
| **Majority** | 4 | 270 | 421 | 424 | 406 | 404 | 466 | 466 | 466 | 466 |
| **Consensus** | 450 | 450 | 423 | 424 | 413 | 482 | 482 | 466 | 466 | 466 |
| | | | | | **Mean Value** | | | | | |
| **D-Space** | 11.9 | 4.7 | 1.7 | 1.3 | 1.8 | 5.9 | 2.2 | 1.2 | 1.1 | 1.4 |
| **Std. Dev.** | 0.10 | 0.08 | 0.04 | 0.02 | 0.03 | 0.17 | 0.05 | 0.03 | 0.01 | 0.02 |
| | | | | | **Success Frequency of CV Sub-Events** | | | | | |
| **S-Majority** | 4 | 270 | 421 | 424 | 406 | 404 | 466 | 466 | 466 | 466 |
| **F-Majority** | 0 | 0 | 58 | 59 | 31 | 0 | 0 | 32 | 33 | 33 |
| **S-Plurality** | 435 | 169 | 2 | 0 | 0 | 77 | 16 | 0 | 0 | 0 |
| **F-Plurality** | 41 | 42 | 18 | 0 | 0 | 17 | 17 | 1 | 0 | 0 |
| **S-Random** | 11 | 11 | 0 | 0 | 7 | 1 | 0 | 0 | 0 | 0 |
| **F-Random** | 7 | 7 | 1 | 1 | 16 | 1 | 1 | 1 | 1 | 1 |
| **F-Fiat** | 2 | 1 | 0 | 16 | 40 | 0 | 0 | 0 | 0 | 0 |
| **F-Total** | 50 | 50 | 77 | 76 | 87 | 18 | 18 | 34 | 34 | 34 |

(*) Assuming that effective error output space has infinite cardinality, i.e. there are no coincident identical and wrong answers from two or more versions.

We voted using majority, consensus and 2-out-of-n strategies. The last strategy had to be simulated because effective output space was not sufficiently large for it to function properly on its own. This was accomplished by checking, for each test case, the frequency of versions that were

successful. If that frequency was two or more, then in an ideal situation where all coincident wrong answers recorded for the test case are unique, 2-out-of-n strategy would have succeeded in selecting the correct answer. The count of these hypothetical successes provided simulated 2-out-of-n voting reliability bound. In the case of consensus voting we also recorded the frequency of sub-events that yielded the consensus decision.

**Table 3.3 (continued)** Frequency of voting events.

| | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|
| | | | **Success Frequency** | | | |
| **Best Version** | 498 | 498 | 434 | 454 | 354 | 372 |
| **2-of-N*** | 499 | 498 | 77 | 483 | 388 | 407 |
| **Majority** | 482 | 498 | 77 | 447 | 330 | 354 |
| **Consensus** | 482 | 498 | 154 | 476 | 379 | 412 |
| | | | **Mean Value** | | | |
| **D-Space** | 1.1 | 1.1 | 2.4 | 1.5 | 2.8 | 1.8 |
| **Std. Dev.** | 0.01 | 0.01 | 0.03 | 0.04 | 0.05 | 0.04 |
| | | | **Success Frequency of CV Sub-Events** | | | |
| **S-Majority** | 482 | 498 | 77 | 447 | 330 | 354 |
| **F-Majority** | 18 | 2 | 174 | 0 | 27 | 26 |
| **S-Plurality** | 0 | 0 | 0 | 19 | 18 | 53 |
| **F-Plurality** | 0 | 0 | 0 | 16 | 2 | 34 |
| **S-Random** | 0 | 0 | 77 | 10 | 31 | 5 |
| **F-Random** | 0 | 0 | 143 | 7 | 74 | 14 |
| **F-Fiat** | 0 | 0 | 29 | 1 | 18 | 14 |
| **F-Total** | 18 | 2 | 346 | 24 | 121 | 88 |

(*) Assuming that effective error output space has infinite cardinality, i.e. there are no coincident identical and wrong answers from two or more versions.

We recorded the number of times consensus was a successful majority (S-Majority), an unsuccessful majority (F-Majority), a successful plurality[1] (S-Plurality), an unsuccessful plurality (F-Plurality), a successful (S-Random) and an unsuccessful (F-Random) attempt at breaking a tie, and a failure by fiat[2] (F-Fiat). By F-Fiat we mean a situation where a tie existed but all the classes involved contained wrong answers so any choice made to break the tie led to failure. The sum of S-Majority, S-Plurality and S-Random comprises consensus voting success total, while the sum of F-Majority, F-Plurality, F-Random and F-Fiat is equal to the total number of cases where voting failed (F-Total).

Another quantity which we measured was the average size of the decision space in which the voters operated. For each test case we counted the number of different classes of outputs offered to the voters. At the end we computed the sample mean and standard deviation of the sample and the mean. The mean and its standard deviation are given in the table rows D-Space and Std. Dev. respectively. The number of times the best version was correct was also counted (Best Version).

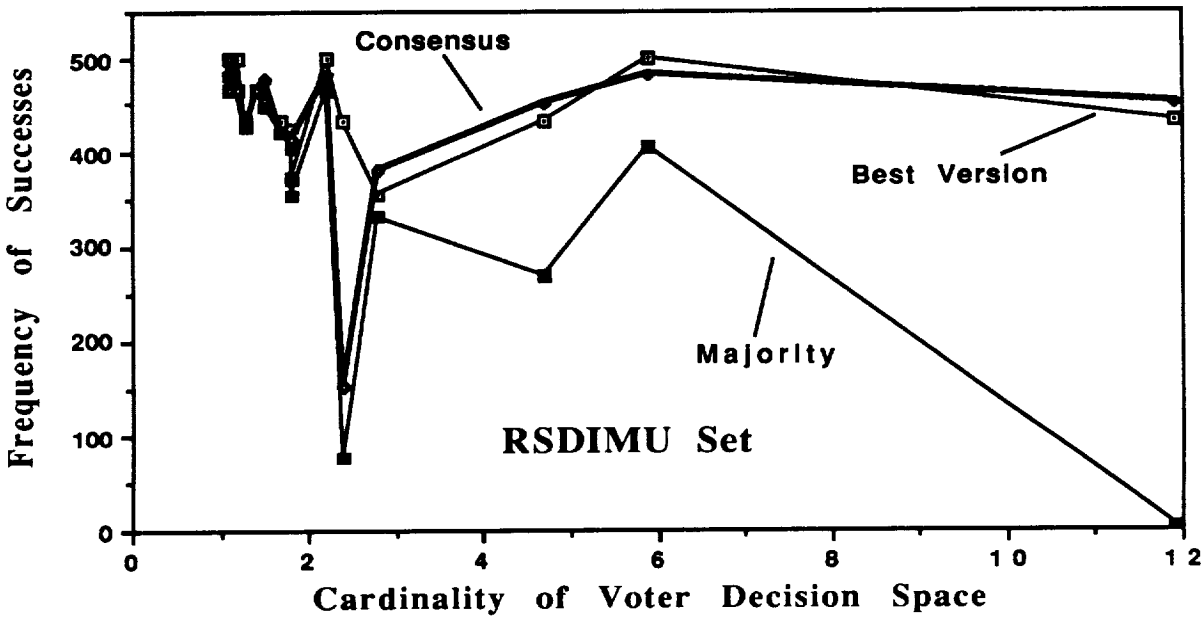


**Figure 3.1** Influence of voter decision space cardinality: General trend.

[1] One of the definitions given by the Webster's New Collegiate Dictionary for "plurality" is "a number of votes cast for a candidate in a contest of more than two candidates that is greater than the number cast for any other candidate but no more than half the total votes cast." If the number of votes cast exceeds half of the total votes then this number is majority. In consesnus voting plurality corresponds to the situation where there is a uniqe maximum of identical outputs but that maximum is not majority.

[2] One of the definitions given by the Webster's New Collegiate Dictionary for "fiat" is "A command or act of will that creates something without or as if without further effort".

The success frequency of majority and consensus voting from Table 3.3 is plotted against the voter decision space in Figure 3.1. Also plotted is the corresponding "best version" success frequency. It is interesting to note that the plotted data sets are very mixed (number of versions varies, average reliability per version decreases with increasing decision space, etc.) consensus voting always performs as well or better than majority voting, and most of the time consensus voting performance is close to that of the best version.

In Figure 3.2 we plot voter success frequency against average failure probability of an n-tuple version. Again we note the general trend which shows that that consensus voting performance is closer to the best version performance than majority voting.

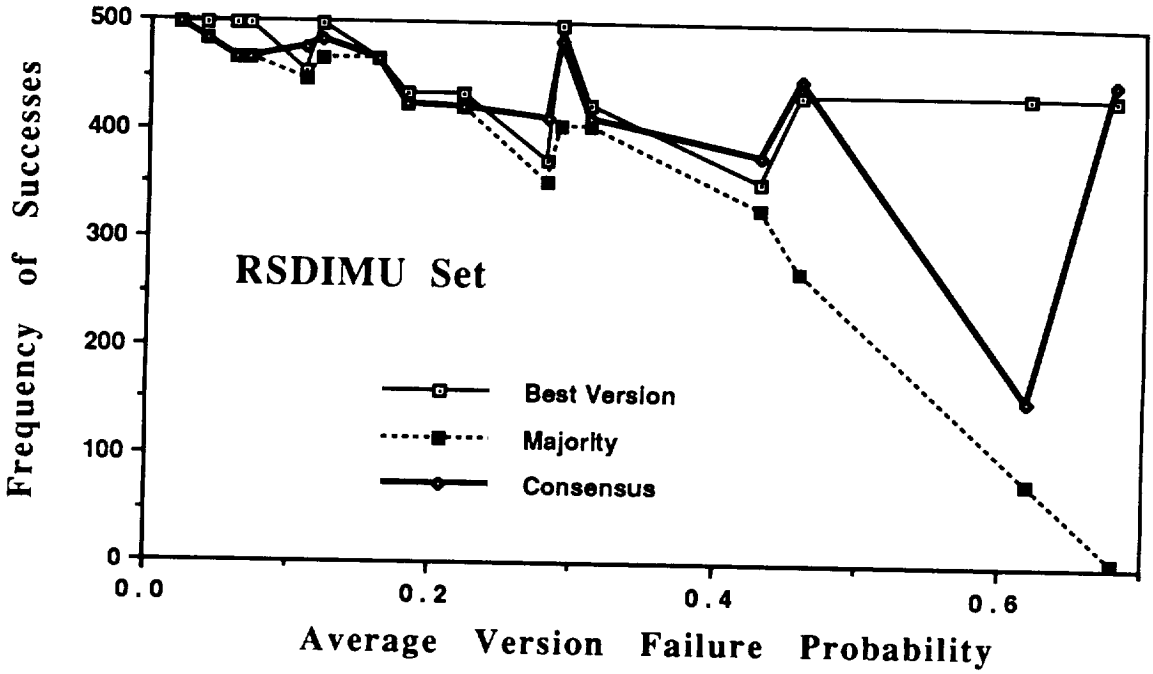


**Figure 3.2** Influence of average version failure probability: General trend.

In Figure 3.3 we plot the data for combinations which consisted of five versions only (n = 5). The vertical axis is the normalized success frequency, i.e. ratio between the observed success frequency and the frequency expected on the basis of the average version reliability. Note the smoothing of the performance, and the fact that consensus voting performs very close to the 2-of-N upper bound but in this case does not exceed that performance. It is obvious that the closer the decision space is to its upper bound (of 5 in this cases), the better is the performance of consensus voting.

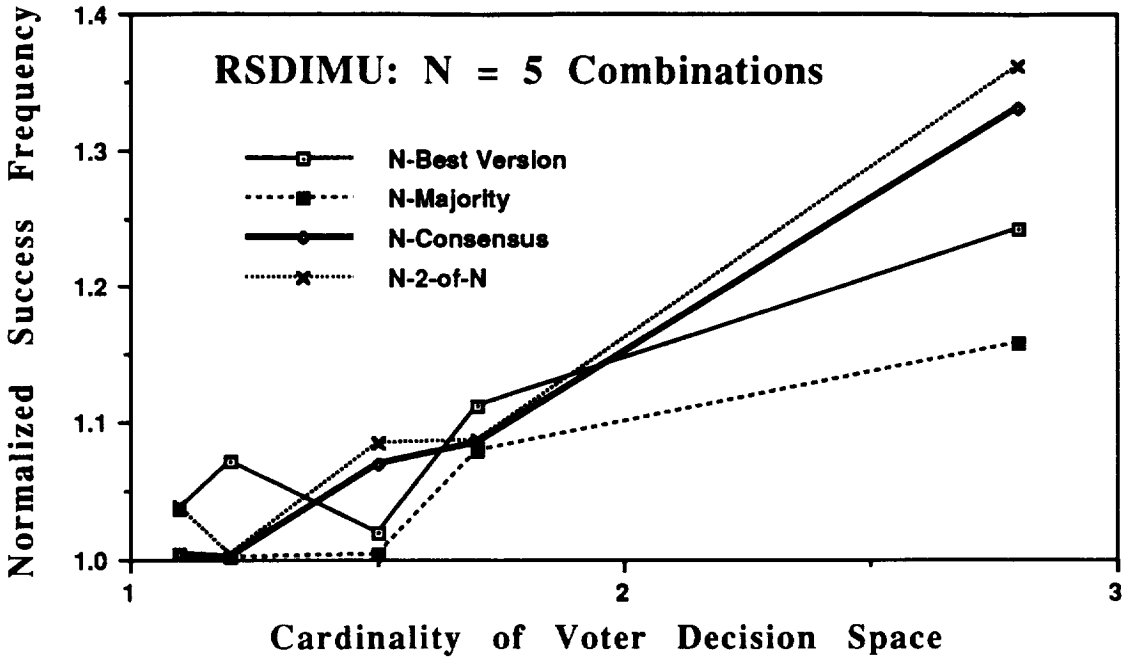


**Figure 3.3** Influence of decision space cardinality for a fixed n = 5.

Inspection of table 3.3 and the figures confirms that in many important respects consensus voting behaves very much like its model based on failure independence in small output spaces predicts. For example,

1. Consensus voting always offers reliability at least equivalent to majority voting. This is a sanity check.
2. Consensus voting offers a higher number of successes than majority voting when the average decision space in which voters work is sufficiently large. In the presented cases this is when average decision space exceeds about 1.5. Particularly striking is the advantage consensus voting has over majority voting in situations where the effective decision space is a large fraction of the maximum decision space (e.g. combinations 1, 2, 6, and 13).
3. Consensus voting and majority voting perform worse, or no better, than at least one of component versions in situations when the ratio between average version failure probability and the sample standard deviation exceeded certain critical value (unmatched versions). For example, the critical value is about 1.5 for failure probability of 0.6 , but this ratio increased to over 3 for average component failure probability less than about 0.2.
4. In the case of relatively well balanced versions (e.g. combinations 14, 15, and 16) consensus voting does better than majority voting.

5. Most of the time the 2-of-N voting represents an upper bound on consensus voting. However, under special circumstances the fact that consensus voting attempts to randomly select an answer even when there was only one answer per decision space category makes consensus voting better than even 2-of-N voting (e.g. in combinations 5 and 16 consensus voting is better than 2-of-N voting exactly by the count from successes deriving from random selection S-Random).

## 6. Summary and Conclusions

We have used 20 functionally equivalent software versions to empirically investigate some properties of consensus voting algorithm. Consensus voting provides automatic adaptation of the voting strategy to varying component reliability and output space characteristics.

Our results indicate that in many important respects consensus voting behaves very much like its model based on failure independence in small output spaces predicts. It is therefore conjectured that because of its auto-adaptive nature, even in the presence of considerable failure correlation, reliability performance of consensus voting is on the average better than that of any fixed agreement number voting algorithm, and is most of the time far better than that of majority voting.

This is a preliminary report and complete analysis, including that based on randomly selected n-tuples rather than special combinations, is underway. Recommendations will be given when the full analysis has been completed.

## References

[1] A. Avizienis and L. Chen, "On the Implementation of N-version Programming for Software Fault-Tolerance During Program Execution", Proc. COMPSAC 77, 149-155, 1977.

[2] A. Avizienis and P.A. Kelly, "Fault-Tolerance by Design Diversity: Concepts and Experiments", Computer, Vol. 17, pp. 67-80, 1984.

[3] R.K Scott, J. W. Gault and D. F. McAllister, "Fault-Tolerant Reliability Modeling", IEEE Trans. Soft. Eng. Vol. SE-13, No. 5, pp. 582-592, 1987

[4] J. Kelly, D. Eckhardt, A. Caglayan, J. Knight, D. McAllister, M. Vouk, "A Large Scale Second Generation Experiment in Multi-Version Software: Description and Early Results", Proc. FTCS 18, pp 9-14, June 1988.

[5] K.S. Trivedi, "Probability and Statistics with Reliability, Queueing, and Computer Science Applications, Prentice-Hall, New Jersey, 1982.

[6] D.E. Eckhardt, Jr. and L.D. Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors", IEEE Trans. Soft. Eng., Vol. SE-11(12), 1511-1517, 1985.

[7] D.F. McAllister, C.E. Sun, and M.A. Vouk, "Reliability of Voting in Fault-Tolerant Software Systems for Small Output Spaces", North Carolina State University, Department of Computer Science, Technical Report, TR-87-16, to appear in IEEE Trans. Reliability, 1990.

[8] R.K. Scott, J.W. Gault, D.F. McAllister and J. Wiggs, "Investigating Version Dependence in Fault-Tolerant Software", AGARD 361, pp. 21.1-21.10, 1984

[9] P.G. Bishop, D.G. Esp, M. Barnes, P Humphreys, G. Dahl, and J. Lahti, "PODS--A Project on Diverse Software", IEEE Trans. Soft. Eng., Vol. SE-12(9), 929-940, 1986.

[10] J.C. Knight and N.G. Leveson, "An Experimental Evaluation of the assumption of Independence in Multiversion Programming IEEE Trans. Soft. Eng., Vol. SE-12(1), 96-109, 1986.

# Appendix I: Experimental environment

Empirical information on the consensus voting was collected using 20 functionally equivalent programs developed in an earlier experiment [4]. The programs solved a problem in inertial navigation. The requirement was to interpret and analyze part of sensor signals (accelerometers only) received from a redundant strapped down inertial measurement unit (RSDIMU). It was also required that the code be written in Pascal, and developed and tested in a UNIX environment on VAX hardware. The problem specification was new, written for the experiment, and was not debugged via a "pilot" version of the code, or prototyping, prior to the production of the redundant versions.

Following their unit testing all programs were subjected to an extensive acceptance test. In the process over sixty distinct faults were identified among the versions. Less than 12% of the faults exhibited varying degrees of positive correlation. The common-cause (or similar) faults spanned as many as 13 components. However, majority of these faults were trivial, and easily detected by proper unit and/or system testing. System testing of the versions that followed acceptance testing included over 1,000,000 simulated operational (flight) test cases and revealed several more faults. In this paper we use the test case profiles that are effective for detection of the faults detected during acceptance testing, and program versions that incorporate all the errors present just prior to acceptance testing.

When making comparisons of version responses we used tolerances compatible with the accuracy of the input data. All 11 output variables (63 individual values) were checked for each input vector. A difference was signaled whenever any one of these values differed from the corresponding "golden" value which was used to adjudicate the correctness of the answers. The "golden" program has been very extensively tested and inspected on its own, and we believe that right now no faults remain in that code. However, as has been the case in all the experiments published so far [e.g. 2, 8, 9, 10] we have not proved the "golden" code correct. The results presented here were obtained using a tolerance of 0.000244 relative (for 12 bits of input data accuracy) for real numbers larger than 0.1 in magnitude, and 0.0000244 absolute otherwise (requirements specification assumed that the cockpit display size was five digits), and 0 for integers.

For evaluation of consensus voting we used random data. The generation profile we used was uniform over each individual variable. For each test case we generated a 21x20 response matrix. Each element of this matrix contained a vector of 63 values describing the relationship between the responses of a row version and a column version. The vector contained one entry for each variable (or its part, if it was a compound variable) with 0 denoting agreement, and 1 disagreement between

the answers. Row zero carried the responses of the versions to comparisons with the "golden" answers. We then combined the analysis of the matrices with the inspections of the actual response values, and the inspections of the code.

## Appendix II: Version and variable mix combinations

1. All output values, and all versions. All 20 versions are involved in the voting. A version disagrees with another version if any of the 63 output values disagrees with the corresponding value from that other version. A version fails, with respect to golden version, if any of its 63 output values is in disagreement with the corresponding golden value.
2. All output values, but only 12 versions. When all output variables are considered, eight out of 20 versions failed in at least one variable for all test cases we used. The twelve versions chosen for this cut were the ones that the testing profile we used exhibited failure probabilities less than one.
3. All output values, but only 5 versions. High failure probabilities provide a very poor working medium for majority voting. The five versions chosen for this cut were the ones that for all output variables exhibited the five lowest failure probabilities.
4. All output values, but only 3 versions, choice I. The three versions chosen for this cut were the ones that exhibited the three lowest failure rates when all output variables were considered in comparisons.
5. All output values, but only 3 versions, choice II. The three versions in combination 4 showed very high affinity towards triple coincident failures, about 18 times what would be expected by random chance. We therefore selected a triplet which showed a lowered affinity towards coincident failures.
6. Only three output values of the best acceleration estimates, but all versions. Best acceleration vector was chosen illustrate the system behavior in the extreme where only the most critical variable is considered. A version was declared as differing from another version only if one of the best acceleration estimates differed from the other version's values by more than the tolerance. A versions was declared as failed if any of its three best acceleration estimates differed form the golden answers.
7. Only three output values of the best acceleration estimates, but 12 versions. The same 12 versions were used as in the second combination above.
8. Only three output values of the best acceleration estimates, but 5 versions. The same versions were used as in the third combination above.

9. Only three output values of the best acceleration estimates, but only 3 versions, choice I. The same versions were used as in the fourth combination above.
10. Only three output values of the best acceleration estimates, but only 3 versions, choice II. The same versions were used as in the fifth combination above.
11. Only three output values of the best acceleration estimates, and 5 versions. This time we chose the best five versions based on the failure rates observed using the three acceleration values.
12. Only three output values of the best acceleration estimates, and 3 versions. We use the best three versions based on the failure rates observed using three acceleration values.
13. All output values, and 3 versions. The same versions as in 12 above.
14. Only three output values of the best acceleration estimates, and 5 versions. This set of versions was chosen so that the versions had similar failure rates using best acceleration estimates. This set's average was about 0.11.
15. All output values, and 5 versions from 14 above.
16. All output values, and 4 versions, This set of versions was chose so that that the versions had similar failure rates using all variables. This sets' average was bout 0.28.

## Appendix III: Coincident failure profiles

In Tables A3.1 (a-d) we present the coincident failure profiles measured for the 16 combinations. Column k denotes the k-tuple category. The rows represents the number of k-tuples in a particular column class. Categories with k>0 refer to version tuples that have failed. For example, a 3-tuple is a set of three versions. Category k=0 refers to programs that have not failed. Columns marked CF give the observed frequency of coincident failure events. For a CF category, k=0 shows the observed frequency of test cases in which none of the versions involved in the voting failed. For example, for combination 3 (CF3) there are 259 such events over the 500 random test cases. The corresponding IAW column gives the cumulative frequency of events where a correct answer was returned (out of a possible total of n*500, where n is the number of versions). For category k=1 the CF column contains the frequency of test cases in which only one version out of n failed. The corresponding IAW column contains the count of events where the version response to a test case was wrong and unique. Note that there may be up to n such events per test case. For categories k>1 each CF column shows the frequency of test cases where k versions failed coincidentally for a test case. The corresponding IAW columns contain the frequency of events where k versions failed with an identical and wrong answer. Again note that several IAW k-tuples may be possible per test case so long as the sum of their spans does not exceed n.

**Table A3.1a** Frequency of coincident and identical and wrong failures.

| k | CF1 | IAW1 | CF2 | IAW2 | CF3 | IAW3 | CF4 | IAW4 | CF5 | IAW5 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3216 | 0 | 3216 | 259 | 1945 | 353 | 1230 | 172 | 1036 |
| 1 | 0 | 4664 | 4 | 1300 | 129 | 317 | 71 | 152 | 234 | 402 |
| 2 | 0 | 420 | 26 | 378 | 33 | 19 | 29 | 59 | 52 | 31 |
| 3 | 0 | 362 | 102 | 178 | 3 | 32 | 47 | 0 | 42 | 0 |
| 4 | 0 | 16 | 108 | 16 | 29 | 26 | | | | |
| 5 | 0 | 26 | 30 | 26 | 47 | 0 | | | | |
| 6 | 0 | 0 | 85 | 0 | | | | | | |
| 7 | 0 | 0 | 20 | 0 | | | | | | |
| 8 | 0 | 0 | 63 | 0 | | | | | | |
| 9 | 4 | 0 | 18 | 0 | | | | | | |
| 10 | 26 | 0 | 1 | 0 | | | | | | |
| 11 | 102 | 0 | 2 | 0 | | | | | | |
| 12 | 108 | 0 | 41 | 0 | | | | | | |
| 13 | 30 | 0 | | | | | | | | |
| 14 | 85 | 0 | | | | | | | | |
| 15 | 20 | 0 | | | | | | | | |
| 16 | 63 | 0 | | | | | | | | |
| 17 | 18 | 0 | | | | | | | | |
| 18 | 1 | 0 | | | | | | | | |
| 19 | 2 | 0 | | | | | | | | |
| 20 | 41 | 0 | | | | | | | | |

**Table A3.1b** Frequency of coincident and identical and wrong failures.

| k | CF6 | IAW6 | CF7 | IAW7 | CF8 | IAW8 | CF9 | IAW9 | CF10 | IAW10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0* | 89 | 7123 | 137 | 5251 | 421 | 2318 | 446 | 1411 | 296 | 1260 |
| 1 | 79 | 2258 | 236 | 519 | 44 | 79 | 20 | 22 | 170 | 174 |
| 2 | 38 | 167 | 44 | 15 | 1 | 3 | 33 | 32 | 32 | 33 |
| 3 | 0 | 14 | 28 | 15 | 1 | 31 | 1 | 1 | 2 | 0 |
| 4 | 3 | 0 | 5 | 15 | 32 | 1 | | | | |
| 5 | 0 | 16 | 16 | 19 | 1 | 0 | | | | |
| 6 | 46 | 2 | 16 | 0 | | | | | | |
| 7 | 45 | 0 | 1 | 0 | | | | | | |
| 8 | 16 | 2 | 16 | 0 | | | | | | |
| 9 | 88 | 15 | 0 | 0 | | | | | | |
| 10 | 30 | 0 | 1 | 0 | | | | | | |
| 11 | 14 | 0 | 0 | 0 | | | | | | |
| 12 | 2 | 0 | 0 | 0 | | | | | | |
| 13 | 16 | 0 | | | | | | | | |
| 14 | 16 | 0 | | | | | | | | |
| 15 | 1 | 0 | | | | | | | | |
| 16 | 16 | 0 | | | | | | | | |
| 17 | 0 | 0 | | | | | | | | |
| 18 | 1 | 0 | | | | | | | | |
| 19 | 0 | 0 | | | | | | | | |
| 20 | 0 | | | | | | | | | |

**Table A3.1c** Frequency of coincident and identical and wrong failures.

| k | CF11 | IAW11 | CF12 | IAW12 | CF13 | IAW13 |
|---|---|---|---|---|---|---|
| **0*** | 449 | 2395 | 466 | 1463 | 46 | 564 |
| **1** | 16 | 17 | 32 | 33 | 31 | 588 |
| **2** | 17 | 17 | 1 | 2 | 364 | 174 |
| **3** | 17 | 18 | 1 | 0 | 59 | 0 |
| **4** | 1 | 0 | | | | |
| **5** | 0 | 0 | | | | |

**Table A3.1d** Frequency of coincident and identical and wrong failures.

| k | CF14 | IAW14 | CF15 | IAW15 | CF16 | IAW16 |
|---|---|---|---|---|---|---|
| **0*** | 350 | 2226 | 42 | 1433 | 234 | 1436 |
| **1** | 97 | 208 | 176 | 898 | 120 | 418 |
| **2** | 0 | 33 | 112 | 44 | 53 | 34 |
| **3** | 36 | 0 | 58 | 27 | 34 | 26 |
| **4** | 16 | 0 | 67 | 0 | 59 | 0 |
| **5** | 1 | 0 | 45 | 0 | | |

# Modeling Execution Time of Multi-Stage N-Version Fault-Tolerant Software[3]

Mladen A. Vouk, Amitkumar M. Paradkar, and David F. McAllister

North Carolina State University
Department of Computer Science, Box 8206
Raleigh, N.C. 27695-8206

Tel: (919)-737-7886
Fax: (919)-737-7382
e-mail: vouk@cscadm.ncsu.edu

## Abstract

N-version systems may be subdivided into stages for the purpose of forward error recovery through voting after each stage. The simplest is the approach where all components in a stage have to furnish their results before the vote takes place and the execution continues. An obvious problem with this approach is that the whole system waits for the slowest version to finish before a result is furnished. A better solution is to use a scheme we call Expedient Voting in which the voting takes place as soon as an adequate number of components have finished the stage. The concept of a "runahead" is introduced — the faster versions are allowed to run ahead of the rest of the slower versions by one or more stages, with synchronized re-start in the event of a failure. If the difference between the fastest and the slowest successful components in each stage is large, the versions are highly reliable, and the failure dependence is small, then the execution speed-up that Expedient Voting yields for successful may be substantial. The speed-up decreases for runaheads over about 3 stages, and also deteriorates with reduction in the version reliability and independence. The advantages and the limitations of using Expedient Voting are discussed.

**Key Words:** Software fault-tolerance, multi-stage voting, Expedient Voting, execution time performance, Community Error Recovery.

## 1. Introduction

The issue of reliability and safety has assumed great importance in the design and implementation of software. Basically, there are two approaches to assuring adequate reliability of software during its operation. One is fault-avoidance and fault-elimination prior to release of the software, and the other is fault-tolerance in the operational phase.

There are two major schemes for implementing fault tolerance in software viz. Recovery Block [Ran75] and N-version programming [Avi77, Avi85]. The former uses a primary module, an

[3]Research supported in part by NASA Grant No. NAG-1-983

acceptance test and (N-1), usually sequentially executed, alternate modules developed independently. The latter requires N independently developed versions, which usually run concurrently, with results from all compared and voted upon to select the *correct* answer. There are also hybrid schemes that employ combinations of the above two schemes, *e.g.* Consensus Recovery Block [Sco87].

The reliability and execution time performance of the basic software fault-tolerance schemes has been extensively studied. For example, Grnarov and Avizienis [Grn80] used queuing models for performance evaluation of both Recovery Block and N-version programming schemes, Laprie [Lap84] developed a Markov model for the Recovery Block, Eckhardt and Lee [Eck85] and Littlewood and Miller [Lit87] developed probabilistic models for analysis of the effectiveness of multi-version software subject to coincident failures, and Deb and Goel [Deb86, Deb88] used Markov chain models to arrive at closed form solutions to various system parameters, such as the reliability and the mean time spent in a system, and analyze the performance issues involved in the recovery block (RB) and N-version (NV) strategies.

The N-version approach is expected to mask design faults through the consensus of results from $N \geq 3$ diverse versions. Unlike hardware failures, software failures are usually input dependent and a faulty version may work quite well in segments, and with other than the inputs from its fail set [Mus87]. It can, thus, retaining the capability for masking failures of other components over faults that are not common. The key to re-use of versions, in full or in segments, is an error recovery scheme which transforms the erroneous state of a failed version to an error-free state from which normal executions can continue.

A mechanism that uses staged local and global forward error recovery has been proposed by Tso et al. [Tso86, Tso87]. Their method is called Community Error Recovery (CER). It assumes that at any given time during execution there exists a majority of good versions which can supply information to recover the failed versions. Each version is subdivided into stages and either a CER check-point or a recovery point is established after each stage. At each CER point every version submits its state vector to a supervisor which oversees the execution of the versions, compares their results, and decides on the correctness of the intermediate states. The result of the decision is sent back to the versions judged incorrect in order to initiate their recovery. The error recovery is fully effective only if a minority of versions fail between two successive CER points, and if all the state variables that are not part of the state vector are not corrupted. If there is no consensus among the versions, recovery will not be performed and the system will be shut down safely. If a majority of versions produces similar errors at the CER point, then the minority of good versions will be

forced to an erroneous state. The usual, and controversial, assumption is that the likelihood of such an occurrence is small.

Most of the analytical work in the area of performance of N-version software systems has been done for single-stage systems, i.e. systems equivalent to only one CER check-point, while multi-stage systems have been investigated to a much smaller extent. For example, Deb [Deb88] has suggested a method to compute the execution time requirements of a staged system, where each of the stages may be implemented as a recovery block or an N-version model. The analysis was carried out for the case where the voting takes place and execution continues only after all the components in the stage have furnished their results. Tso et al. [Tso86] developed Markov reliability models for the CER scheme, and more recently, Nicola and Goyal [Nic90] analyzed the effect of inter-version failure correlation on the reliability performance of Community Error Recovery.

In this paper, we study multi-stage N-version software timing performance for a forward error recovery scheme we call Expedient Voting (EV). In section 2 we discuss the multi-stage N-version model. In section 3 we describe and analyze the Expedient Voting scheme. Expedient Voting takes place as soon as an adequate number (e.g. majority) of components have finished the stage. Furthermore, the faster versions are allowed to run ahead of the rest of the slower versions by one or more stages, with re-start from the synchronization stage in the event of a failure. We show that in some situations EV may considerably improve execution time of staged systems. For example, given low inter-version failure dependence and exponential execution time of version stages, simulations show that Majority EV can produce results two or more times faster than a scheme where all stage elements are required to furnish their results before the vote takes place and any execution continues. We also discuss conditions under which there are minimal or no speed-up gains. Results of simulation experiments on EV are presented in Section 4. Summary and conclusions are given in Section 5.

## 2. Multi-Stage Model

A true concurrent N-Version programming strategy for software fault-tolerance requires N independently developed but functionally identical programs running in parallel. We use each of the N programs as monolithic units, or modules, and vote on the output without taking into account the intermediate program values (or states) [AVI 77]. This scheme may improve the overall system reliability over that of a single component, provided certain conditions are met [Tri82, Eck85, Lit87]. An alternative scheme is subdivision of each version into segments, and testing for

correctness of the intermediate results computed by each segment [Tso86, Tso87, Deb88]. These, supposedly, *correct* values can then be passed on to the remaining stages. The two strategies are illustrated in figures 1 and 2.

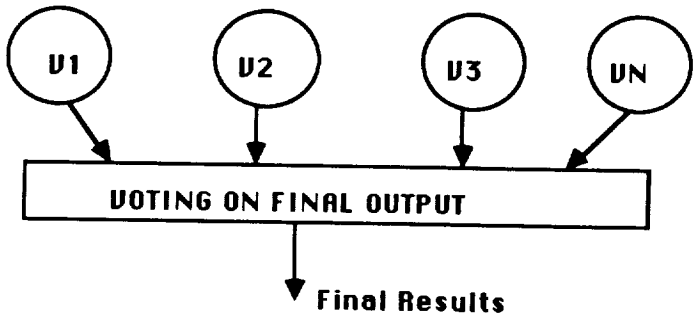


**Figure 1.** Single-stage 1xN system

The following terminology will be used in the remainder of the paper:

- *element* denotes one of the M version segments in a M-by-N (or MxN) system.
- *stage* denotes a 1-by-N horizontal slice of a M-by-N system.
- *version* denotes a M-by-1 vertical slice of a M-by-N system.

## 2.1 Reliability of a Multi-Stage System

Reliability of Community Error Recovery has been investigated theoretically [Tso86, Nic90, Par90] and experimentally [Tso87]. Nicola and Goyal [Nic90] results indicate that the effectiveness of the technique is highly sensitive to, and easily degraded by, errors in CER check-points and inter-version failure dependence, particularly when very high system reliabilities are targeted (e.g. 0.99999 or more). CER is more robust at lower target system reliabilities [Par90].

Consider a multi-stage N-version software system in Figure 2. Let, for tractability, p be the success probability (reliability per test case) of each of the M times N elements comprising this system, where M is the number of stages, and N is the number of versions. Let decision on the correctness of the intermediate results be derived by voting. Since voting is carried out after each stage, what at each stage are believed to be the correct results are passed on to all subsequent stages. If inter-version failure independence is assumed, it can be shown that the reliability, $R_{MS}$, of this MxN system for an arbitrary (m-of-N) voting approach is

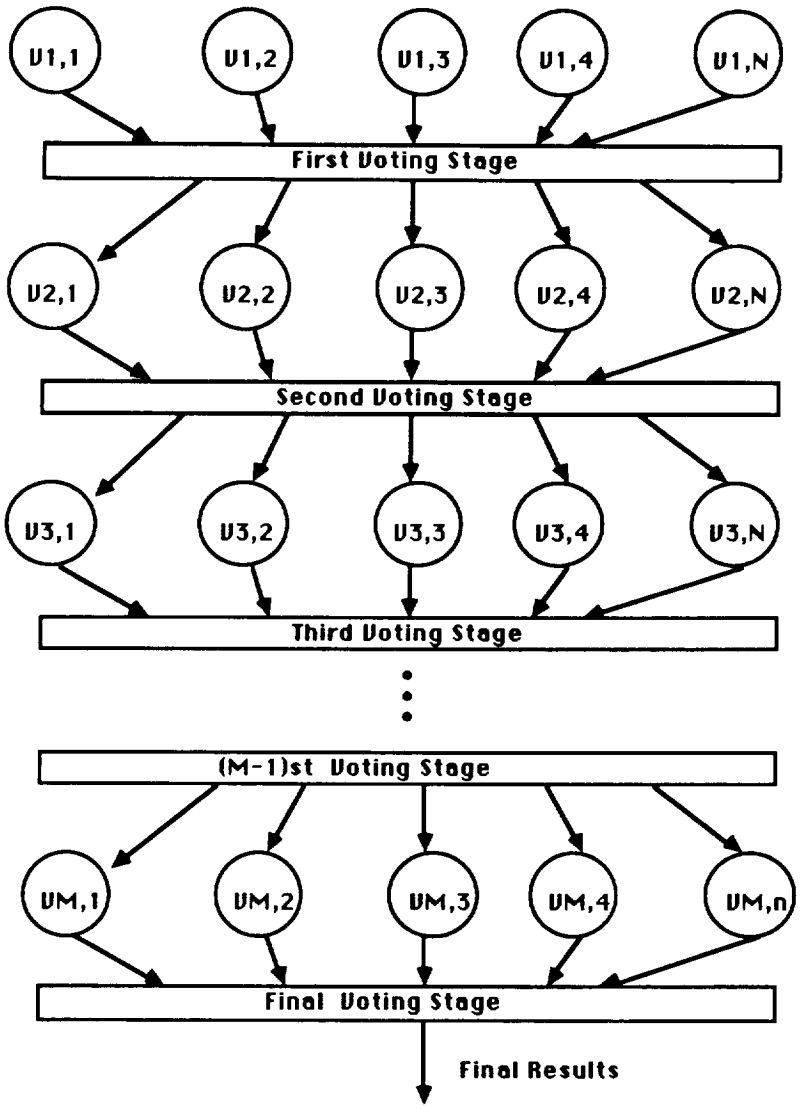


**Figure 2.** Multi-stage MxN system

$$R_{MS} = (R_S)^M = (R_C)^M \left[ \sum_{j=m}^{N} \binom{N}{j} (p)^j (1-p)^{N-j} \right]^M \quad (1)$$

where $R_S$ is the reliability of a single-stage (including the check-point), $R_C$ is the probability that the stage check-point operates correctly and also correctly resets any failed versions [Par90, Nic90], and m is the number of versions that have to agree on a result before it is accepted as correct. Again for tractability we have assumed that $R_C$ is same for all check-points. We will also assume that p is sufficiently large so that single-stage reliability, $R_S$, is larger than p [Tri82]. For Two-of-N Voting

m = 2, and for Majority Voting $m = \lceil \frac{N+1}{2} \rceil$, where m is the agreement number and $\lceil\ \rceil$ denotes the ceiling function. Voter imperfections ($R_C < 1$), and inter-version failure correlation, reduce the overall system reliability, and for high p values can easily make $R_{MS}$ worse than the reliability offered by an equivalent single-stage system [Nic90].

This is illustrated in Figure 3 where we compare the ideal ($R_C = 1$, and no failure correlation) reliability gain offered by a 3x3 multi-stage system, the ideal reliability of an equivalent 1x3 monolithic system[4], the ideal reliability of an equivalent single-stage one version system ($R_{ss1}=p^3$), and the reliability of a 3x3 system where $R_C = 0.9$ but there is still no correlation. The effect of correlation can be simulated through the single-stage single-version system curve. One possible extreme case of failure correlation is when all elements in a stage fail coincidentally with identical (and wrong) answers, but with negligible inter-stage failure correlation. The reliability behavior is then similar to that of a single-stage single version system. Under ideal conditions, the more stages there are, the higher the gain offered by a multi-stage system. In reality, voter imperfections, voter execution overheads, and failure dependence, will dictate the optimal placement and number of stages [Nic90, Par90].

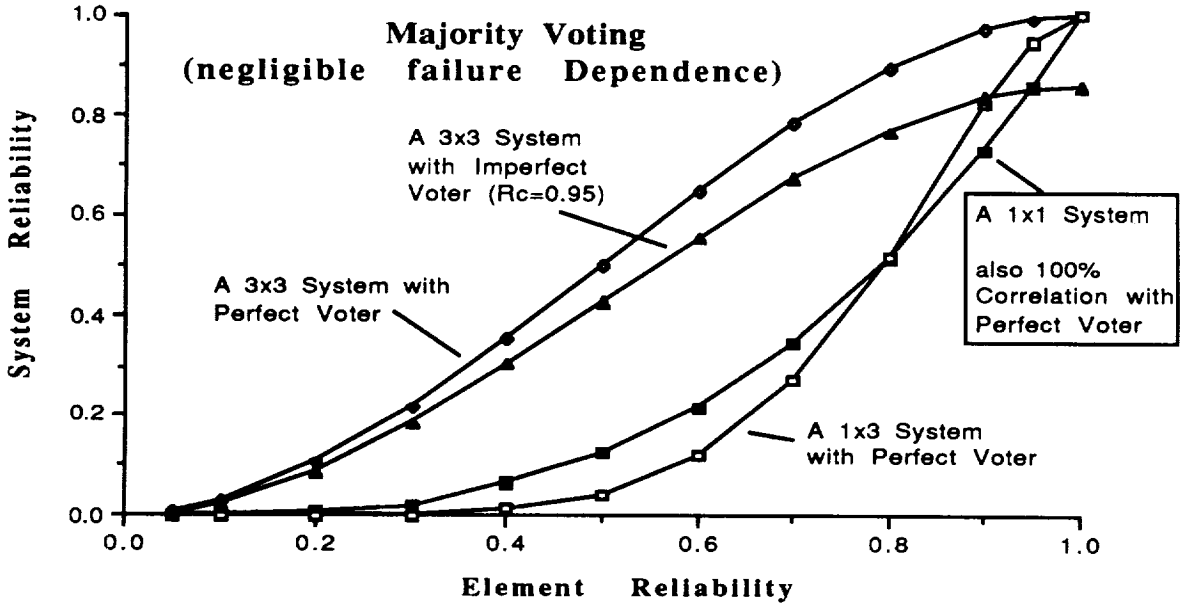


**Figure 3.** Illustration of the Reliability of Multi-Stage N-Version Software

[4] $R_{SS} = \sum_{j=m}^{N} \binom{N}{j} (p^M)^j (1-p^M)^{N-j}$

From Figure 3 we see that even small imperfections in the check-point mechanism can drastically degrade reliability of a multi-stage system. Similarly, correlation will reduce the effectiveness of the approach and, in combination with voter imperfections, may make the system performance worse than that offered by an equivalent single-stage one version system.

Experiments indicate that under current software development strategies it is quite likely that at least some of the residual faults in multi-version software will be correlated [Bis86, Kni86, Shi88, Avi88, Bis88, Vog88]. However, the prevalence of these faults and the degree of correlation remain an issue of considerable controversy. Experimental information on the reliability offered by CER is sparse and mixed. Results from a partial analysis [Tso87] of the functionally equivalent software developed in a multi-version experiment [Kel88] indicate that the approach can be successful, at least in part, in increasing reliability of medium to high reliability versions. On the other hand, analyses using a model based on Knight and Leveson [Kni86] data and reported in [Nic90] paint a far grimmer picture of the effectiveness of CER in the face of inter-version failure dependence, and check-point imperfections.

However, in truly parallel computing environments, under conditions of (very) low correlation inter-version failure correlation, high check-point reliability, and with appropriate voting scheme a multi-stage N-version system may not only be more reliable than an equivalent, more traditional, single-stage system, but it may also be run faster [Par90]. We now explore the latter possibility.

## 3. Execution Time Model

In this section we discuss analytical models for the times to completion of multi-stage N-version fault-tolerant software. For tractability, the analytical solutions are derived under a set of simplifying assumptions. The behavior of the model under less restrictive conditions is discussed. The assumptions are:

1. Probability of success, p, of each of the M times N stage elements is a constant, *i.e.*, p does not change with time.
2. The program output space cardinality is infinite.
3. The probability of m elements producing identical and wrong answers is negligible, i.e., agreement between the required number of elements implies correctness of the results.
4. The stage check-points are perfect ($R_c$=1).

5. Time taken by the voting algorithm is negligible compared to the execution time of each stage element.
6. The *restart* and *recovery* time in the case of a failed element is negligible compared to the execution time of each stage element.
7. The execution times of each of the M times N elements are independent of each other.
8. The software runs on a dedicated multi-processor hardware which has no other tasks to perform.

## 3.1 A Numerical Example

To motivate the analysis and illustrate the principles we will first discuss a numerical example. Consider an N-version fault tolerant software system with M stages as illustrated in Figure 4. The numbers inside the circles (stage elements) are the times required for each element to produce its results. The zeros or ones beneath each element represents the failure (0) or success (1) of the corresponding element.

In the simplest case we would wait for all the elements to produce their results before we start voting with them. In this situation the time required before the results are available at each stage is the time required for the slowest element in each stage. The time required before final results are available is obviously the sum of the times for each stage. In the example shown this time is given by the sum (150 + 170 + 160 + 100 = 580). Obviously, this scheme is very inefficient.

In fact, under our assumptions we can actually proceed with the voting as soon as first m components have produced their results (e.g. m is the majority, provided majority voting is used; otherwise whatever agreement number is used). If there is a failure among the first m components, we must wait for the $(m+1)^{st}$ module to produce its results before voting can produce a decision based on, say, majority, and so on. Thus we can vote on available results as soon as the sufficient number of correct results is available. In the example, the fastest element also yields wrong results, so the voter can make a decision only after the fourth element has finished running. This eliminates the need to wait for the fifth element to produce its results. The application of this scheme produces results from the first stage in time (125). The total time is the sum of times for each stage, so the final results are available in the time (125 + 100+100+85=410).

It is obvious that this scheme offers a saving of (580 - 410 = 170, or ~30%) time units over the previous one without sacrificing the reliability of the system. In the worst case, we have to wait

for the slowest component to produce its results, i.e. for (N-m-1) failures in prior (N-1) versions. For highly reliable systems, probability of such an event is expected to be very small. We call the above scheme Expedient Voting (EV). If correctness decision is based on the majority we will call the scheme Majority EV. Similarly, we can have Two-of-N EV.

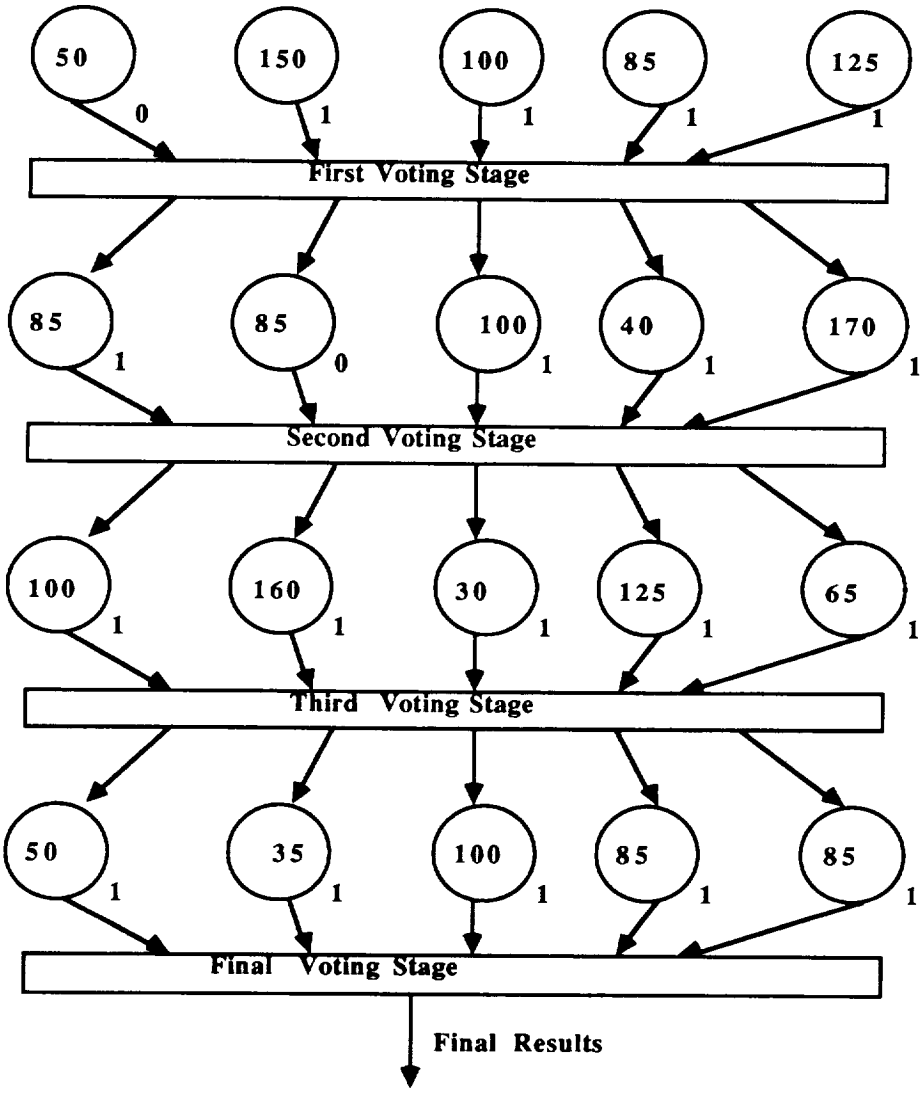


**Figure 4.** A 5x4 system with timing information.

Expedient Voting can be further improved. We have assumed that each of the N processors in the system is dedicated to the same version of the current software till the task is finished. Given this, the fastest (m - 1) processors in each stage are idle and waiting for the slower processors to finish the job. For example, element 1 in stage 1 with time requirements of 50 units is waiting for other processors to produce their results. Assuming that the results from this element are correct, we can let this version run ahead with the next computational stage for 35 time units, the time available

before the second fastest component in stage 1 produces its results. This reduces the time requirements of this version in the second stage from 85 to 50.

In this example the results produced by the first component in the first stage are wrong, so there is no advantage in letting it run ahead as it will need to be re-injected with the correct values after the first voting stage and re-started. But the advantages with high reliability versions, in general, should be apparent. The faster processes can be allowed to continue on to the next stage, temporarily assuming that their results for the current stage are correct. If later on it is found that these elements did produce correct results, they already have a head start for the next stage. On the other hand, if they are found to be wrong by the voting station, they can be restored to *'correct'* state, and re-started at the beginning for that stage. It should be noted that the overhead is minimized because the assumption is that the processor would be idle anyway. Only the supervising module has to keep track of the status of each processor and restore its states if its results from the previous stage are found to be wrong.

The above scheme can restrict the *'synchronization'* of each processor to either every other stage, or can impose no restriction on how far ahead a version is allowed to execute. The times for completion for one synchronizing stage (one *runahead* ), are illustrated in figure 5. The time required for the first stage is 125 because we still need to wait for the majority to furnish correct results. But, since component number 4 produces its results in 85 units, its version has (125 - 85 =) 40 units of spare time. It can proceed with stage number 2 and dedicate these 40 units towards stage 2. This reduces the time for stage 2 from 40, as would be required without the runahead, to 0. Similarly, time required for version 3 in stage 2 reduces from 100, to 100 - (125-100) = 75. Thus, the revised time vector for stage 2 looks like (85, 85, 75, 0, 170), and the results for stage 2 are available in 85 instead of 100 time units. Since, only one stage *'runahead'* is allowed, the same process is repeated from stage 3 and faster versions in stage 3 are allowed to run further into stage 4. With this scheme, the times required for stage 3 stay at 100, but the revised time vector and the revised time for stage 4 are (50, 35, 30, 85, 50) and 50, respectively. Thus, the total time taken is (125 + 85 + 100 + 50 = 360). It can be seen that although the savings are not very large, they are there nevertheless, and in some applications may be important.

In this example, there is nothing to be gained by letting the versions run beyond one stage before synchronization is carried out. For example, both versions 4 and 3 in stage one, with excess times of 40 and 25 units respectively, have these spare times used up in stage 2. So there is no excess time available to run ahead in stage 3.

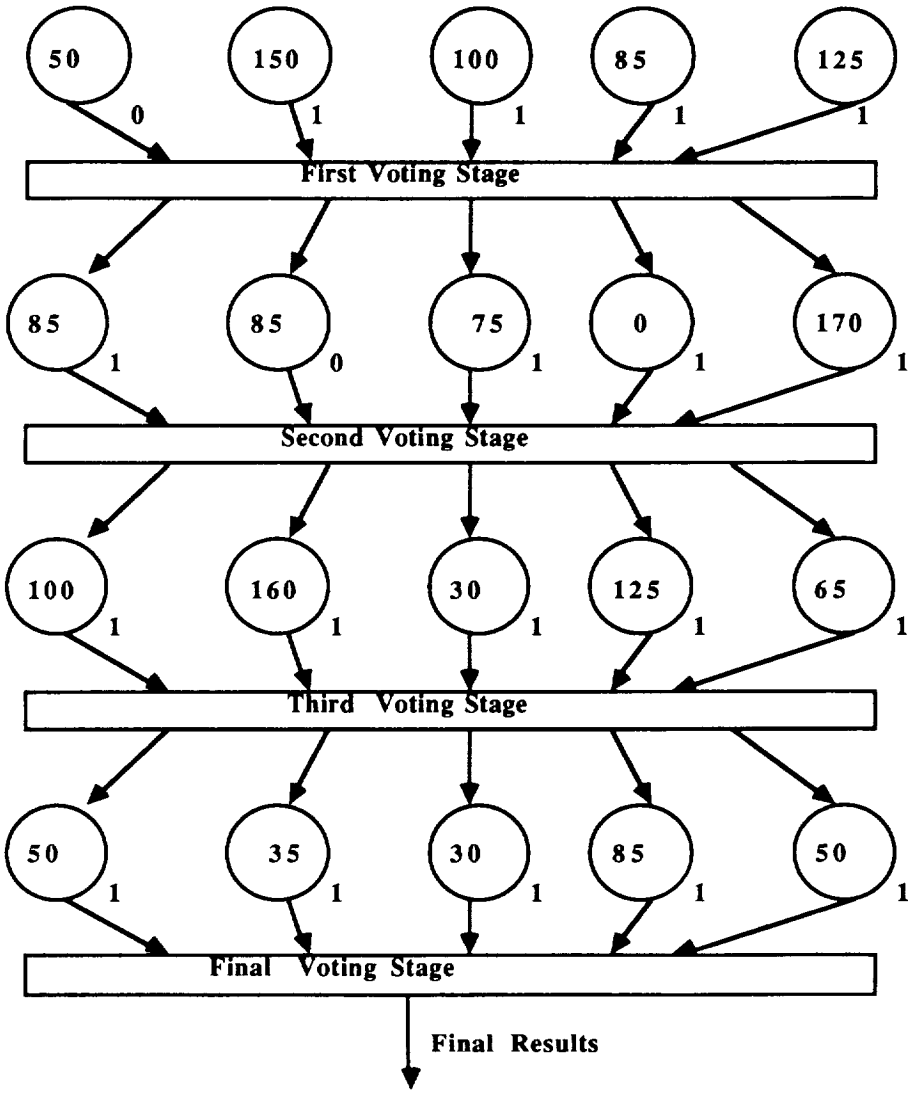


**Figure 5.** The Figure 4 system with timing information appropriate to Majority Expedient Voting with one stage runahead.

But, were the time required for the fourth version in the second stage reduced from 40 to 30, an additional 10 time units would be available in stage 3. These 10 units can be applied towards the execution of the third stage and there would be a net gain of 10 units. It thus reduces the time requirement for version 4 in stage 3 from 125 to 115. With two stage synchronization (all the versions are allowed to go up to 2 stages ahead) the time taken for the final results is 350 units. Even if there is an overhead in restoring the system states for failed versions, it can be seen that there can be savings with such a scheme.

## 3.2 Simple Voting Solution

Assume an MxN system as described in section 2.1. Let the execution times for each element within a stage be represented by mutually independent, identically distributed continuous random variables $t_{i,j}$, each having distribution function $F(t)$, where $t_{i,j}$ refers to the element of version j in stage i ($1 \leq j \leq N$). Let the execution time vector for a stage i be denoted by $\Theta_i(t) = (t_{i,1}, t_{i,2}, ..., t_{i,N})$. Let $T_{i,1}, T_{i,2}, ..., T_{i,N}$ be random variables obtained by permuting the set $\Theta_i(t)$ so as to be in increasing order. To be specific: $T_{i,1} = \min \{\Theta_i(t)\}$, $T_{i,N} = \max \{\Theta_i(t)\}$, etc. The random variable $T_{i,k}$ is then called the $k^{th}$ order statistic [Tri82].

Let the distribution function of $T_{i,k}$ be denoted by $F_{T_{i,N}}(t)$. Let P{...} denote probability of event {...}. From [Tri82] it follows that

$F_{T_{i,N}}(t) = P\{T_{i,k} \leq t\}$ = P {'at least k of the $t_{i,j}$'s in a stage lie in the interval (0,t]'}

$$= \sum_{j=k}^{N} \binom{N}{j} F^{j}(t) [1 - F(t)]^{N-j}, \quad 0 < t < \infty. \tag{2}$$

We now compute the time required to complete each stage when all N versions are allowed to finish before voting is performed. The time needed by each stage is the time needed by the slowest version, or it is the maximum of the times needed by each version in that stage. Let the symbol $\tau_i$ denote the completion time for stage i. Note that we have assumed that at each stage the time taken by the voter, and other overhead, is negligible compared to $\tau_i$. Hence, $\tau_i$, is given by $T_{i,N}$ of $\Theta_i(t)$. Thus, from equation (2), the distribution function of $F_{T_{i,N}}$ is

$$F_{T_{i,N}}(t) = [F(t)]^N \qquad 0 < t < \infty.$$

Before we continue we will make one more simplifying assumption, that $F(t)$ is exponential with parameter $\lambda$. In reality element execution time distributions may not be exponential. We discuss that in Section 4 and show how, for example, Uniformly and Normally distributed execution times affect the results.

If $F(t)$ is exponential then, according to Theorem 3.5 in Trivedi [Tri82], $F_{T_{i,N}}(t)$ is hypoexpontial with parameters [Nλ, (N - 1)λ, ...., kλ, ..., λ]. Its expected value is

$$E\left[F_{T_{i,N}}(t)\right] = \sum_{a=1}^{N} \frac{1}{a\lambda}, \tag{3}$$

and its variance is $\sum_{a=1}^{N} \frac{1}{(a\lambda)^2}$. The total time required before the final results are available after M stages is the sum of the times required in each stage $\{\sum_{i=1}^{M}\tau_i\}$, so the expected value of total time required is

$$\sum_{i=1}^{M}\sum_{a=1}^{N} \frac{1}{a\lambda} = M\sum_{a=1}^{N} \frac{1}{a\lambda}. \tag{4}$$

Since we have assumed that the execution times of different stages are independent from each other the variance is

$$\sum_{i=1}^{M}\sum_{a=1}^{N} \frac{1}{(a\lambda)^2} = M\sum_{a=1}^{N} \frac{1}{(a\lambda)^2}. \tag{5}$$

It should be noted that this completion time does not depend on success or failure of any individual stage since the results are voted upon only after **all** the versions in that stage have produced results. This completion time does not depend either upon the voting strategy used (Majority-of-N or Two-of-N), or on the individual version reliability. However the fraction of time devoted to successful stage completions will be (3) multiplied by the probability that the stage succeeds.

## 3.3 Expedient Voting Solution.

As described in the numerical example, it is not necessary to wait for the slowest component to furnish its results before voting can take place. As soon as the number of components that have finished is equal to, or larger than, the agreement number required by the chosen voting strategy, voting can take place and an attempt can be made to ascertain the correctness of the received results. If all results agree, it is not necessary to wait for the remaining components to finish running. Instead, the consensus solution can be injected into subsequent stage elements immediately, and the next stage started[5].

For example, consider Majority Voting given that the fastest $m^{th}$ component is correct, and that there are no failures in the previous (m-1) versions. Then the time required for the completion of stage *i* is given by $T_{i,m}$ of $\Theta_i(t)$. This event will occur with probability $p^m$ because all the fastest m versions need to be successful, and we have assumed inter-version failure independence, and equal probability p of success for all elements. If exactly one version fails among the fastest m versions, but the $(m+1)^{st}$ version is successful, then the stage can be completed in $T_{i,m+1}$ of $\Theta_i(t)$. There are **m** different ways this event can occur. So the probability of this event is $mp^m(1-p)$. From this, a more general expression for the probability of an event where exactly *j* failures occur in the fastest (m+j-1) versions and the $(m+j)^{th}$ version is successful, is:

$$\binom{m+j-1}{j}(p)^m(1-p)^j .$$

It can be seen that $\tau_i$, the time required for the stage i to succeed, is a random variable which takes on the values $T_{i,m+j}$ with the following probability:

$$\begin{aligned} P\{\tau_i = T_{i,m+j}\} &= \binom{m+j-1}{j}(p)^m(1-p)^j, \quad 0 \le j \le N-m, \\ &= 0, \quad \text{otherwise.} \end{aligned} \tag{6}$$

Hence, the expected value $E[\tau_i]$ for successes is :

$$\sum_{j=0}^{N-m} \left\{ E[T_{i,m+j}] \binom{m+j-1}{j}(p)^m(1-p)^j \right\} \tag{7}$$

[5]Note that under our assumptions this procedure is safe for both Two-of-N voting and Majority Voting because the probability of identical and wrong answers is zero. In practice this may not be the case, and only Majority Voting may be acceptable from the safety standpoint.

In all other situations (i.e. when the number of failures is such that m cannot be reached), the stage will wait for all versions to complete before a failure is declared[6]. The probability of this event is the probability that the stage will fail, i.e. $(1-R_S)$. If $F(t)$ is exponentially distributed with parameter $\lambda$, then $T_{i,m+j}$ is hypoexponentially distributed with (m+j) parameters [Tri82]. Its expected value is $\sum_{a=N+1-m-j}^{N} \frac{1}{a\lambda}$, and (7) transforms into

$$E[\tau_i] = \sum_{j=0}^{N-m} \left\{ \sum_{a=N+1-m-j}^{N} \frac{1}{a\lambda} \binom{m+j-1}{j} (p)^m (1-p)^j \right\} \qquad (8)$$

Because $\Theta_i(t)$ is a vector of continuous random variables, then $T_{i,1} < T_{i,2} < \ldots < T_{i,N}$ with probability one [Tri82]. Furthermore, because the maximum completion time (3) is part of only one term in (8), and is otherwise associated with stage failure probability $(1-R_S)$, it is obvious that successful Expedient Voting will, in general, produce results faster than the simple voting approach. For small values of p, however, it becomes more likely that all versions will have to be executed before a voter can make a decision, i.e. the most significant terms become the j=N-m in (8) and the stage failure term. Inter-version failure correlation has a similar effect. In practice however, we are usually only interested in the expected time required per successful stage because failure of any stage implies complete failure of the system. Furthermore, the stage check-points, re-starts and recovery will all add to the overhead of implementation and execution of the EV scheme. If the difference between the average per stage execution times of the simple voting solution (3) and the Expedient Voting is considerable, then it may outweigh the overheads and make the approach worthwhile. If version reliabilities are poor, then we could expect an increased number of re-starts, as well as a need to often wait for all versions to finish before a sufficient number is available for a voter decision.

[6]An alternative might be a strategy which preempts further execution, and fails the system or initiates a higher level recovery, once the number of yet unexecuted stage elements is not sufficient to make up the desired m even if they all succeeded.

# 4. Simulation

In order to validate analytical solutions and study situations not covered by them, we built a simulator for a multi-stage N-version system composed of elements of known success probability and execution time distribution. Unless stated otherwise, all examples shown below were run assuming exponentially distributed execution times with parameter $\lambda = \frac{1}{300}$. A typical simulation run was 500,000 cases long, and only the successful cases were considered in the timing statistics. The theoretical and the simulation results for the simple solution and EV with zero runaheads were identical within the error of simulation.

## 4.1 Expedient Voting with Runahead

Figure 6 shows the execution times for a 10x5 system against the number of runahead stages allowed for two element reliabilities of p=0.999 and p=0.55. At each stage the decision is made based on the majority (of N) agreement. A value of -1 for runahead stands for the simple voting solution where all version complete before voting takes place. The value of 0 denotes Expedient Voting with no runahead. The value of 1 means that the fastest version is allowed to execute only one stage ahead of the rest of the versions, and then it has to wait for the results from other versions to be made available for voting, and so on. This notation is used in all figures.

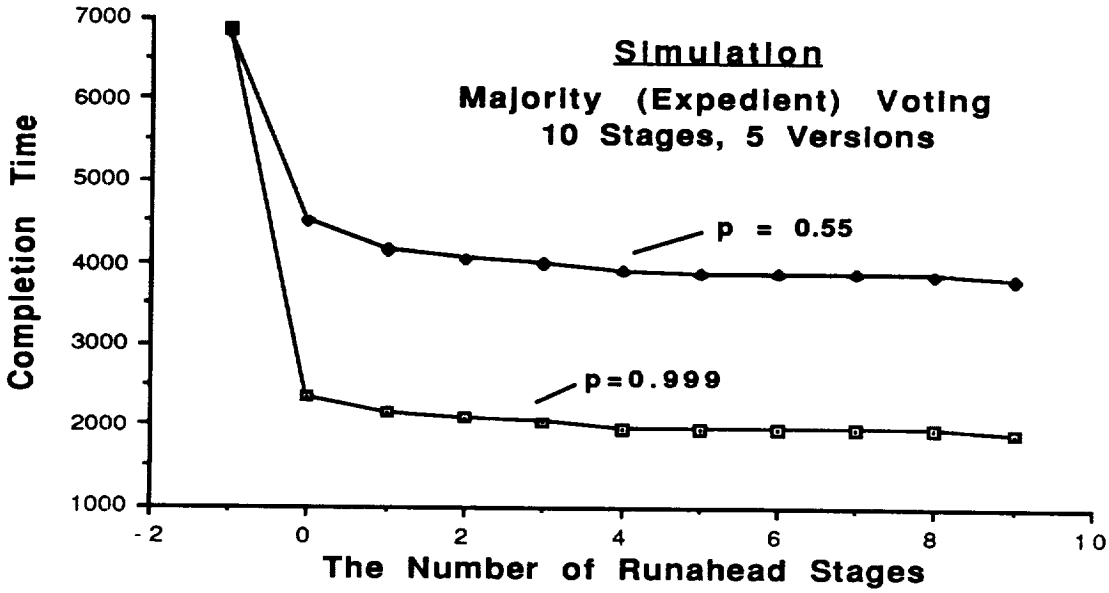


**Figure 6** Majority Expedient Voting with and without runahead.

As is to be expected, Figure 6 confirms that the simple voting solution (-1) does not depend upon the individual component success probabilities. The completion times are about 3 times larger than with Majority Expedient Voting at p=0.999, and about 1.5 times at p=0.55. A runahead of one increases the difference by a further 10%. The savings in time thereafter diminish rapidly, and there is virtually no gain after a runahead of 4 or 5 stages. This is also an anticipated result because the excess time available for the fastest versions quickly diminishes with longer runaheads. Another expected result is that for each runahead the completion times increase with the decrease in single component success probability. As the single component success probability decreases, the probability increases that more versions are needed before majority agreement can be achieved, so the voter waits longer before furnishing its decision.

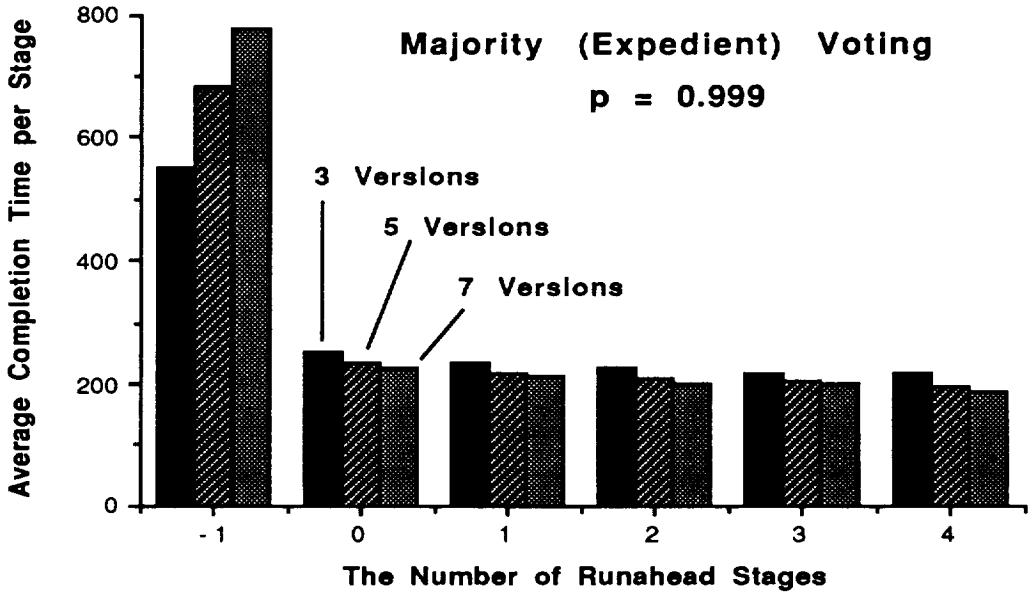


**Figure 7** Majority Expedient Voting with and without runahead.

To consistently compare the effect of the number of versions, N, on the execution times we compute the average times required per stage. These values are plotted in Figure 7 for element success probability of 0.999. For the simple voting solution (-1) the average time required per stage increases with the number of versions. This is to be expected because these times represent the maximum of 3, 5, and 7 independent and identically distributed random variables, and the probability of the maximum of 7 variables being larger than the maximum of 3 variables is higher. The situation is reversed with Expedient Voting. The larger the number of versions employed, the

faster are the results produced because the more likely it is that the requisite number of versions will agree. However, the differences in favor of larger N are not very big, and reduce further for smaller p values.

Simulations of Two-of-N Expedient Voting show trends similar to those exhibited for Majority EV except that the differences between the simple and Expedient Voting are more pronounced because the results are usually available as soon as the second fastest version finishes.

## 4.2 Influence of Execution Time Distribution

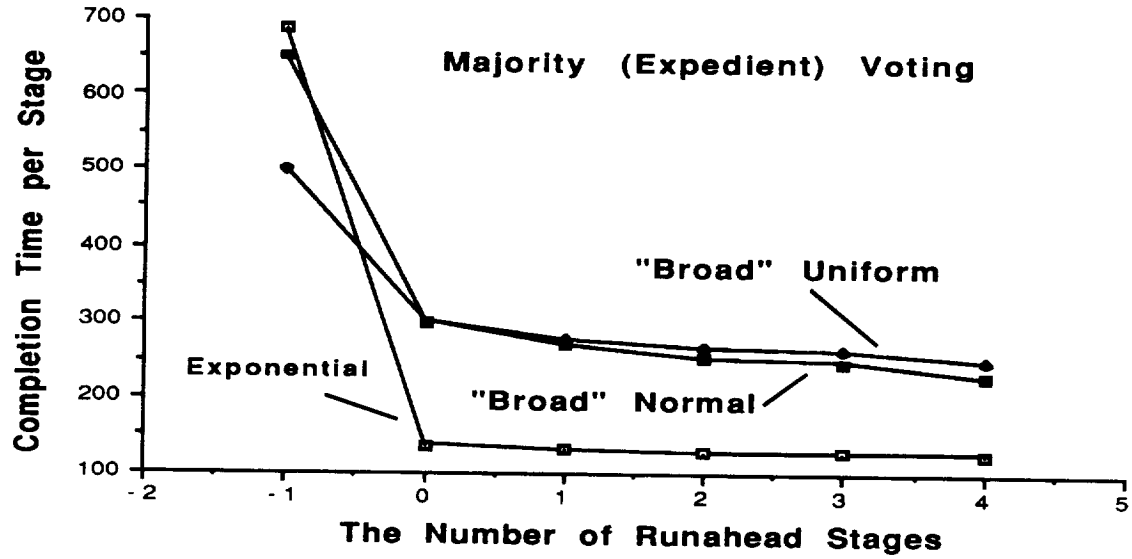


**Figure 8.** Influence of the element execution time distribution function on the stage completion times.

To influence of different execution time distribution functions is illustrated using the Normal and the Uniform distributions. In the following examples the average execution time ($\mu$) is 300 time units. Figure 8 shows the average one stage execution times for a 5x5 system using variances which give in "broad" or spread-out distributions, i.e. Exponential ($\mu = 300$, $\sigma^2 = 90000$), Normal ($\mu = 300$, $\sigma^2 = 300$), and Uniform ($\mu = 300$, minimum = 0, maximum = 600). Agreement number is majority of N, and p is 0.999. We see that the exponential distribution results in the longest stage times when the simple voting scheme (-1) is used. Uniform distribution, partly because of the a ceiling on its maximum, results in the shortest times. Normal gives intermediate results. With Expedient Voting all three distributions show substantial reductions over the simple voting. But,

because the Normal and the Uniform distributions have smaller differences between the average minimum and maximum values than the Exponential distribution has, they also offer smaller time savings with EV.

An interesting effect can be seen when, for the same average values, the distribution variance is made considerably smaller, i.e. for the Normal ($\sigma^2 = 10$) and for the Uniform (minimum = 270, maximum = 330) distributions. The results are summarized in Figure 9. The increased localization of the element execution times around the mean reduces the effective difference between the minimum and the maximum execution times within a stage. With simple voting this results in a substantial drop in the average times per stage for both the Normal and the Uniform distributions. In fact, there is not much difference between the timing requirements for simple voting and Expedient Voting with or without runaheads. This shows that in some practical situations the EV scheme may not be faster, even with high reliability high-independence versions, because this small difference could be easily annulled through voter imperfections, and low, but not negligible, voter and re-start execution overheads.

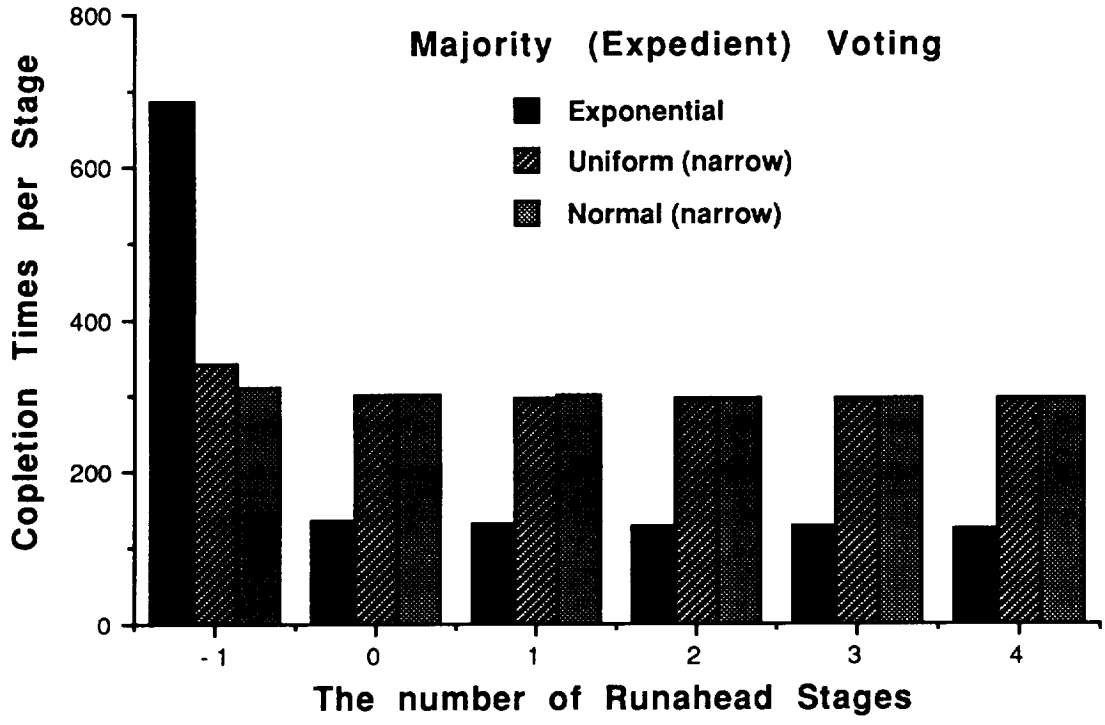


**Figure 9.** Influence of very localized element execution time distribution function on the per stage completion time.

The principal factor governing the speed-up is the average difference between the minimum and the maximum element execution times per stage. If the difference is small, then the savings in time provided by Expedient Voting (with or without runaheads) may be negligible.

## 5. Conclusions

We have analyzed the timing performance of N-version multi-stage software for a strategy we call simple voting and a strategy we call Expedient Voting. In simple voting all components in a stage have to furnish their results before the vote takes place and the execution continues. In Expedient Voting the voting takes place as soon as an adequate number of components have finished the stage. In EV the faster versions may be allowed to run ahead of the rest of the versions by one or more stages, with synchronized re-start in the event of a failure. The principal conclusions that can be drawn from this study are:

- Expedient Voting scheme may produce the final results faster than simple voting. The speed-up gain is a function of the difference between the fastest and the slowest elements in each stage, and of the reliability of stage elements. If the difference between execution of successful stage elements is large then the gains are substantial. The difference is governed by the "spread" (i.e. variance) of stage element execution time distributions. The speed-up advantages of EV over simple voting diminish as version reliability reduces.

- Expedient Voting scheme may be coupled with version runaheads. This variant may produce and additional speed-up, but this gain decreases rapidly for runaheads exceeding about 3 stages.

The authors are attempting to find solutions for systems with small output spaces, and in the presence of correlated and coincident failures. They are also investigating the incorporation of voter timing and rollback overhead on the results, and plan to experimentally verify above results in a true multiprocessor environment.

## References

[Avi77] A. Avizienis and L. Chen, "On the Implementation of N-version Programming for Software Fault-Tolerance During Program Execution", Proc. COMPSAC 77, 149-155, 1977.

[Avi85] A. Avizienis, "The N-Version Approach to Fault-Tolerant Software," IEEE Transactions on Software Engineering, Vol. SE-11 (12), pp 1491-1501, 1985.

[Bis86] P.G. Bishop, D.G. Esp, M. Barnes, P Humphreys, G. Dahl, and J. Lahti, "PODS--A Project on Diverse Software", IEEE Trans. Soft. Eng., Vol. SE-12(9), 929-940, 1986.

[Bis88] P.G. Bishop, and F.D. Pullen, "PODS Revisited--A Study of Software Failure Behaviour", Proc. FTCS 18, pp 2-8, June 1988.

[Deb86] A.K. Deb, and A.L. Goel, "Model for Execution Time Behavior of a Recovery Block,", Proc. COMPSAC 86, 497-502, 1986.

[Deb88] A.K. Deb, "Stochastic Modelling for Execution Time and Reliability of Fault-Tolerant Programs Using Recovery Block and N-Version Schemes," Ph.D. Thesis, Syracuse University, 1988.

[Eck85] D.E. Eckhardt, Jr. and L.D. Lee, "A Theoretical Basis for the Analysis of Multi-version Software Subject to Coincident Errors", IEEE Trans. Soft. Eng., Vol. SE-11(12), 1511-1517, 1985.

[Grn80] A. Grnarov, J. Arlat, and A. Avizienis, "On the Performance of Software Fault-Tolerance Strategies," Proc. FTCS 10, pp 251-253, 1980.

[Kel88] J. Kelly, D. Eckhardt, A. Caglayan, J. Knight, D. McAllister, M. Vouk, "A Large Scale Second Generation Experiment in Multi-Version Software: Description and Early Results", Proc. FTCS 18, pp 9-14, June 1988.

[Kni86] J.C. Knight and N.G. Leveson, "An Experimental Evaluation of the assumption of Independence in Multi-version Programming", IEEE Trans. Soft. Eng., Vol. SE-12(1), 96-109, 1986.

[Lap84] J.-C. Laprie, "Dependability Evaluation of Software Systems in Operation," IEEE Trans. Soft. Eng., Vol. SE-10 (6), 701-714, 1984.

[Lit87] B. Littlewood, and D.R. Miller, "A Conceptual Model of Multi-Version Software," FTCS 17, Digest of Papers, IEEE Comp. Soc. Press, pp 150-155, July 1987.

[Mus87] J. Musa, A. Iannino, and K. Okumoto, "Software Reliability: Measurement, Prediction, Application," McGraw-Hill Book Co., 1987.

[Nic90] V.F. Nicola, and Ambuj Goyal, "Modeling of Correlated Failures and Community Error Recovery in Multi-version Software," IEEE Trans. Soft. Eng., Vol. 16(3), pp, 1990.

[Par90] A.M. Paradkar, "Performance Analysis of Multi-Stage N-Version Fault-Tolerant Software," M.Sc. Thesis, North Carolina State University, 1990.

[Ran75] B. Randell, "System structure for software fault-tolerance", IEEE Trans. Soft. Eng., Vol. SE-1, 220-232, 1975.

[Sco87] R.K. Scott, J.W. Gault and D.F. McAllister, "Fault-Tolerant Software Reliability Modeling", IEEE Trans. Software Eng., Vol SE-13 (5), 582-592, 1987.

[Shi88] T.J. Shimeall and N.G. Leveson, "An Empirical Comparison of Software Fault-Tolerance and Fault Elimination," 2nd Workshop on Software Testing, Verification and Analysis, Banff, IEEE Comp. Soc., pp 180-187, July 1988.

[Tri82] K.S. Trivedi, "Probability and Statistics with Reliability, Queueing, and Computer Science Applications, Prentice-Hall, New Jersey, 1982.

[Tso86] K.S. Tso, A. Avizienis, and J.P.J. Kelly, "Error Recovery in Multi-Version Software," Proc. IFAC SAFECOMP '86, Sarlat, France, 35-41, 1986.

[Tso87] K.S. Tso and A. Avizienis, "Community Error Recovery in N-Version Software: A Design Study with Experimentation", Proc. IEEE 17th Fault-Tolerant Computing Symposium, pp 127-133, 1987.

[Vog88] U. Voges (ed.), *Software Diversity in Computerized Control Systems*, Springer-Verlag, Wien, Austria, 1988.