An Empirical Evaluation of Consensus Voting and Consensus Recovery Block Reliability in the Presence of Failure Correlation*

Mladen A. Vouk1, David F. McAllister1, David E. Eckhardt2, Kalhee Kim1

Key Words: Consensus Recovery Block, Consensus Voting, System Reliability, Software Fault-Tolerance, Correlated Failures

Abstract

The reliability of fault-tolerant software system implementations, based on Consensus Voting and Consensus Recovery Block strategies, is evaluated using a set of independently developed functionally equivalent versions of an avionics application. The strategies are studied under conditions of high inter-version failure correlation, and with program versions of medium-to-high reliability. Comparisons are made with classical N-Version Programming that uses Majority Voting, and with Recovery Block strategies. The empirical behavior of the three schemes is found to be in good agreement with theoretical analyses and expectations. In this study Consensus Voting and Consensus Recovery Block based systems were found to perform better, and more uniformly, than corresponding traditional strategies, that is, Recovery Block and N-Version Programming that use Majority Voting. This is the first experimental evaluation of the system reliability provided by Consensus Voting, and the first experimental study of the reliability of Consensus Recovery Block systems composed of more than three versions.

The contact author is:

Mladen A. Vouk
Department of Computer Science, Box 8206, North Carolina State University
Raleigh, NC 27695-8206
Tel: (919) 515-7886, Fax: (919) 515-5839 or 6497
e-mail: vouk@adm.csc.ncsu.edu

*Research supported in part by NASA Grant No. NAG-1-983
1Department of Computer Science, North Carolina State University, Box 8206, Raleigh, NC 27695-8206
2NASA Langley Research Center, MS 478, Hampton, VA 23665
AN EMPIRICAL EVALUATION OF CONSENSUS VOTING AND
CONSENSUS RECOVERY BLOCK RELIABILITY IN THE
PRESENCE OF FAILURE CORRELATION*

Abstract

The reliability of fault-tolerant software system implementations, based on Consensus Voting and
Consensus Recovery Block strategies, is evaluated using a set of independently developed
functionally equivalent versions of an avionics application. The strategies are studied under
conditions of high inter-version failure correlation, and with program versions of medium-to-high
reliability. Comparisons are made with classical N-Version Programming that uses Majority Voting,
and with Recovery Block strategies. The empirical behavior of the three schemes is found to be in
good agreement with theoretical analyses and expectations. In this study Consensus Voting and
Consensus Recovery Block based systems were found to perform better, and more uniformly, than
corresponding traditional strategies, that is, Recovery Block and N-Version Programming that use
Majority Voting. This is the first experimental evaluation of the system reliability provided by
Consensus Voting, and the first experimental study of the reliability of Consensus Recovery Block
systems composed of more than three versions.

Key Words: Consensus Recovery Block, Consensus Voting, System Reliability, Software
Fault-Tolerance, Correlated Failures

INTRODUCTION

Redundancy can be used to provide fault-tolerance in software systems. Several independently
developed, but functionally equivalent, software versions are combined in various ways in an
attempt to increase system reliability. Over the years, simple Majority Voting and Recovery Block
based software fault-tolerance has been investigated by a number of researchers, both

*Research supported in part by NASA Grant No. NAG-1-983
theoretically1,2,3,4,5,6,7 and experimentally8,9,10,11,12,13. However, studies of more advanced models such as Consensus Recovery Block5,6,8,14, Community Error Recovery15,16,17, Consensus Voting18 or Acceptance Voting14,19 are less frequent and mostly theoretical in nature. One of the principal concerns with all redundancy based software fault-tolerance strategies is their performance in the presence of failure correlation among the versions comprising the system.

In a recent study, Eckhardt et al.13 addressed the issue of reliability gains offered through classical majority-based N-Version Programming using high reliability versions of an avionics application under conditions of small-to-moderate inter-version failure dependence. In this paper we discuss system reliability performance offered by more advanced fault-tolerance mechanisms under more severe conditions. The primary goal of the present work is a mutual comparison of different experimental implementations of Consensus Recovery Block, in the presence of inter-version failure correlation, and a comparison of Consensus Voting and Consensus Recovery Block with more traditional schemes, such as N-Version Programming with Majority Voting. We report on the relative reliability performance of Consensus Voting and Consensus Recovery Block in an environment where theoretically expected effects could be easily observed, that is, under the conditions of strong inter-version failure coupling using medium-to-high reliability software versions of the same avionics application as was employed in the Eckhardt et al.13 study. To the best of our knowledge, this study is the first experimental evaluation of the Consensus Voting techniques, and the first experimental study of the reliability of Consensus Recovery Block systems composed of more than three versions.

In the first section of the paper we provide an overview of software fault-tolerance techniques of interest, and we discuss different voting approaches and the question of correlated failures. In the second section we describe the experimental environment and present the results. Summary and conclusions are given in last section.
SOFTWARE FAULT TOLERANCE

Recovery Block and N-Version Programming. One of the earliest fault-tolerant software schemes is Recovery Block\(^1,20\). In Recovery Block, independently developed functionally equivalent, versions are executed in sequence and the output is passed to an acceptance test. If the output of the first version fails the acceptance test, then the second, or first backup, software version is executed and its output is checked by the acceptance test, etc. In the case where the outputs of all versions are rejected, the system fails. One problem with this strategy is the sequential nature of the execution of versions. This was recently addressed by Belli and Jedrzejowicz\(^21\). Another is finding a simple, and highly reliable, acceptance test which does not involve the development of an additional software version. Another basic fault-tolerant software strategy, N-version Programming\(^2,22\), proposes parallel execution of, independently developed functionally equivalent, versions with adjudication of their outputs by a voter. One problem with all strategies based on voting is that situations can arise where, there is an insufficient number of agreeing versions and, voting fails simply because the voter cannot make a decision.

Consensus Recovery Block. Scot et al.\(^5\) developed a hybrid software fault-tolerance model called Consensus Recovery Block. The strategy is depicted in Figure\(^1\). The system executes N, independently developed but functionally equivalent, versions on the same input. Executions may be in series, or in parallel. Then, a vote is attempted on the returned results. The voter may choose the correct result, it may fail through a decision error by choosing an incorrect result as the correct answer, or, if the voting module cannot make a decision, the system reverts to Recovery Block. Recovery Block acceptance tests each version result in turn. It accepts the first result that passes the acceptance test. Recovery is successful if the correct result is accepted. It fails if an incorrect result is accepted as the correct answer. The system also fails if none of the N results pass the acceptance test. This can happen when the acceptance test correctly rejects N wrong answers, or when all answers are rejected including correct ones. It can be shown that, in general,
Consensus Recovery Block offers system reliability superior to that provided by N-Version Programming\textsuperscript{5,14}. However, Consensus Recovery Block, like N-Version Programming does not resolve the problem of a voter which returns a wrong answer because several versions produce identical-and-wrong answers or there is not a majority as might be the case when there are multiple correct outputs.

![Diagram](image)

**Figure 1.** Consensus Recovery Block model.

**Majority and 2-out-of-N Voting.** In an m-out-of-N fault-tolerant software system the number of versions is N, and m is the agreement number, or the number of matching outputs which the adjudication algorithm (such as, voting) requires for system success\textsuperscript{4,23}. In the past, N was
rarely larger than 3, and m was traditionally chosen as $\frac{N+1}{2}$ for odd N. In general, in Majority Voting, $m = \lceil \frac{N+1}{2} \rceil$, where $\lceil \cdot \rceil$ denotes the ceiling function.

Scott et al.\textsuperscript{5} 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 use the term 2-out-of-N Voting for the case where agreement number is $m=2$. In this experiment we do not have statistical independence of version failures. Hence, this voting technique is used only when showing upperbound on reliability of the systems. In a model, based on software diversity and a voting strategy, there is a difference between correctness and agreement. McAllister et al.\textsuperscript{18} distinguish between agreement and correctness, and develop and evaluate an 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 cardinality. They show that, for $m \geq 2$, Majority Voting provides an upperbound on the probability of failing the system using Consensus Voting, and 2-out-of-N provides a lowerbound.

**Consensus Voting.** The theory of Consensus Voting is given in McAllister et al.\textsuperscript{18}. 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 groups then
  - if Consensus Voting is used in N-Version Programming one group is chosen at random and the answer associated with this group is chosen as the "correct" one.
- else if Consensus Voting is used in Consensus Recovery Block all groups are subjected to an acceptance test which is then used to choose the "correct" output.

In McAllister et al. it is shown that the strategy is equivalent to Majority Voting, when the output space cardinality is 2, and to 2-out-of-N voting, when the output space cardinality tends to infinity provided the agreement number is not less than 2. It is also proved that, in general, the boundary probability below which the system reliability begins to deteriorate, as more versions are added, is \( \frac{1}{r} \), where \( r \) is the cardinality of the output space.

**Coincident Failures and Inter-Version Failure Correlation.** When two or more functionally equivalent software components fail on the same input case we say that a coincident failure has occurred. When two or more versions give the same incorrect response, to a given tolerance, we say that an identical-and-wrong answer was obtained. If the measured probability of the coincident version failures is significantly different from what would be expected by chance, assuming the failure independence model, then we say that the observed version failures are correlated or dependent.

Experiments have shown that inter-version failure dependence among independently developed functionally equivalent versions may not be negligible in the context of current software development and testing strategies. There are theoretical models of the classical majority based N-Version Programming model which incorporate inter-version failure dependence. However, most of the theory for advanced software fault-tolerance strategies is derived under the assumption of inter-version failure independence, and failure independence of acceptance tests with respect to versions and each other (if more than one acceptance test is used). Still, the behavior of the strategies in the presence of failure correlation can be deduced from these simple models by extrapolation from their behavior in extreme situations. Undesirable behavior observed in an uncorrelated situation provides a bound on the correlated behavior in the sense that, with correlation, the behavior is expected to be even less desirable. Therefore, it is interesting to see whether the effects, and special
events, that can be anticipated from analytical considerations, can actually be observed in real multiversion software.

For example, in the case of implementations involving voting, presence of correlated failures produces a change in the average cardinality of the space in which voting takes place. An increased probability of coincident but different incorrect answers will tend to increase the average number of distinct responses offered to a voter, while an increased probability of coincident identical-and-wrong failures will tend to decrease the voting space, from what would be expected based on the cardinality of the application output space, and version reliability (assuming versions are statistically independent). In a model based on failure independence the effects can be simulated, at least in part, through reduction, or increase, in the model output space size.

To see this, consider the following. Assume that all individual version failure probabilities in an N-tuple are mutually independent\(^2\), have identical failure probability \(1-p\) over the usage (test) distribution, and have the same probability of occurrence of each program output failure state given by \(\frac{1-p}{r-1}\), where: \(r\) is the size of the program output space, there is a unique success state \(j=1\), and there are \(r-1\) failure states, \(j=2,\ldots,r\). When \(r=2\) (binary output space), all failures, and what is more important all coincident failures, of the N-tuple versions result in identical-and-wrong answers. On the other hand, under the above assumptions, a large value of \(r\) translates into low probability that two incorrect answers are identical (in the analytical and simulation examples given later in this paper an "\(r=\infty\)" implies that the probability of obtaining identical-and-wrong answers is zero). This, in turn, implies higher probability that responses from coincidentally failing versions are different, and also increases the average size of the voting space when coincident failures occur. Of course, the voting space size is bounded by the number of versions in the N-tuple. An increase in the number of coincident version failures can be simulated, in part, by reduction in the value of \(p\). This shifts the peak of the envelope, of the independent coincident failures profile, closer to \(N\). However, in general, models based on the assumption of failure independence do not capture strong non-uniform failure coupling, that in practice can occur between two or more versions, such as, sharp spikes seen in the experimental trace in Figure 2, because the causes of the coupling are
different (e.g. identical-and-wrong responses are the result of a fault rather than a basic change in the output space of the problem, although the final effect may be similar).

An added dimension is failure correlation between an acceptance test and the N-tuple versions, or lack of mutual independence when two or more acceptance tests are used. The effects can, again only in part, be simulated by lowering reliability of the model acceptance test.

Nevertheless, we would expect that many of the effects in a high inter-version correlation environment, would in the simple theoretical models correspond to small output space \( (r) \) and low \( p \) effects. Similarly, we would expect that implementations composed of versions that exhibit low mutual inter-version failure correlation would exhibit many characteristics that correspond to model computations based on large \( r \) values.

**EMPIRICAL RESULTS**

In this section we discuss experimental data on reliability of N-Version Programming systems that use Consensus Voting (NVP-CV), and data on Consensus Recovery Block systems that use either Majority Voting (CRB-MV), or Consensus Voting (CRB-CV). Consensus Voting and Consensus Recovery Block are compared with N-Version Programming that uses Majority Voting (NVP-MV) and with Recovery Block (RB).

**Experimental Environment.** Experimental results are based on a pool of twenty, independently developed functionally equivalent, programs developed in a large-scale multiversion software experiment described in several papers\(^{13,24,25}\). Version sizes range between 2000 and 5000 lines of high level language code. We used the program versions in the state they were immediately after the unit development phase\(^{24}\), but before they underwent an independent validation phase of the experiment\(^{13}\). This was done to keep the failure probability of individual versions relatively high (and failures easier to observe), and to retain a considerable number of faults that exhibit mutual failure correlation, in order to high-light correlation based effects. The nature of the faults found in the versions is discussed in detail in two papers\(^{13,25}\). In real situations versions
would be rigorously validated before operation, so we would expect that in such situations any
undesirable events that we observed in our experiments would be less frequent, and less
pronounced. However, we expect that the mutual performance ordering of different strategies,
derived in this study relative to correlation issues, still holds under low correlation conditions.

For this study we generated subsets of program N-tuples with: 1) similar average§ N-tuple
reliability, and 2) a range of average N-tuple reliability. We use the average N-tuple reliability to
focus on the behavior of a particular N-tuple instead of the population (pool) from which it was
drawn, and to indicate approximate reliability of corresponding mutually independent versions. In
this paper we report on 3, 5 and 7 version systems. The subset selection process is described in
Appendix I.

In conducting our experiments we considered a number of input profiles, different combinations
of versions, and different output variables. Failure rate estimates, based on the three most critical
output variables (out of 63 monitored), are shown in Table 1. Two test suites each containing 500
uniform random input test cases were used in all estimates discussed in this paper. The sample size is
sufficient for the version and N-tuple reliability ranges on which we report here. One suite, which
we call Estimate-I, was used to estimate individual version failure rates (probabilities), N-tuple
reliability, select acceptance test versions, select sample N-tuple combinations, and compute expected
"independent model" response. The other test suite, Estimate-II, was used to investigate the actual
behavior of N-tuple systems, based on different voting and fault-tolerance strategies.

Recovery Block, and Consensus Recovery Block studies require an acceptance test. We used
one of the developed versions as an acceptance test. This provided correlation not only among
versions, but also between the acceptance test and the versions. Acceptance test versions were

---

§ Average N-tuple reliability estimate is defined as \( \bar{p} = \frac{1}{N} \sum_{i=1}^{N} \hat{p}_i \), and the corresponding estimate
of the standard deviation of the sample as \( \hat{\sigma} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (\hat{p}_i - \bar{p})^2} \), where \( \hat{p}_i = \frac{1}{k} \sum_{j=1}^{k} s_{i(j)} \)
is estimated reliability of version i over the test suite composed of k test cases, \( s_{i(j)} \) is a score function
equal to 1 when version succeeds and 0 when it fails on test case j, and \( 1 - \hat{p}_i \) is the estimated
version failure probability.
selected first, then N-tuples were drawn from the subpool of remaining versions. The fault-tolerance algorithms of interest were invoked for each test case. The outcome was compared with the correct answer obtained from a "golden" program\textsuperscript{2,25}, and the frequency of successes and failures for each strategy was recorded.

**Table 1.** Version failure rates.

<table>
<thead>
<tr>
<th>Version</th>
<th>Failure Rate* Estimate I</th>
<th>Failure Rate* Estimate II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.58</td>
<td>0.59</td>
</tr>
<tr>
<td>2</td>
<td>0.07</td>
<td>0.07</td>
</tr>
<tr>
<td>3</td>
<td>0.13</td>
<td>0.11</td>
</tr>
<tr>
<td>4</td>
<td>0.07</td>
<td>0.06</td>
</tr>
<tr>
<td>5</td>
<td>0.11</td>
<td>0.10</td>
</tr>
<tr>
<td>6</td>
<td>0.63</td>
<td>0.64</td>
</tr>
<tr>
<td>7</td>
<td>0.07</td>
<td>0.06</td>
</tr>
<tr>
<td>8</td>
<td>0.35</td>
<td>0.36</td>
</tr>
<tr>
<td>9</td>
<td>0.40</td>
<td>0.39</td>
</tr>
<tr>
<td>10</td>
<td>0.004</td>
<td>0.000</td>
</tr>
<tr>
<td>11</td>
<td>0.09</td>
<td>0.10</td>
</tr>
<tr>
<td>12</td>
<td>0.58</td>
<td>0.59</td>
</tr>
<tr>
<td>13</td>
<td>0.12</td>
<td>0.12</td>
</tr>
<tr>
<td>14</td>
<td>0.37</td>
<td>0.38</td>
</tr>
<tr>
<td>15</td>
<td>0.58</td>
<td>0.59</td>
</tr>
<tr>
<td>16</td>
<td>0.58</td>
<td>0.59</td>
</tr>
<tr>
<td>17</td>
<td>0.10</td>
<td>0.09</td>
</tr>
<tr>
<td>18</td>
<td>0.004</td>
<td>0.006</td>
</tr>
<tr>
<td>19</td>
<td>0.58</td>
<td>0.59</td>
</tr>
<tr>
<td>20</td>
<td>0.34</td>
<td>0.33</td>
</tr>
</tbody>
</table>

(*) Based on the 3 most important output variables, "best.acceleration". Each column was obtained on the basis of a separate set of 500 random cases.

The failure correlation properties of the versions can be deduced from their joint coincident failure profiles, and the corresponding identical-and-wrong response profiles. For example, Figure\textsuperscript{2} shows the profile for a 17 version subset (three versions selected to act as acceptance tests.
are not in the set). The abscissa represents the number of versions that fail coincidentally, and the ordinate is the frequency of the event over the 500 samples. Also shown is the expected frequency for the model based on independent failures, or the "binomial" model\textsuperscript{23}. Deviations from the expected "independent" profile are obvious. For instance, we see that the frequency of the event where 9 versions fail coincidentally is expected to be about 10. In reality, we observed about 100 such events. Table 2 summarizes the corresponding empirical frequency of coincident identical-and-wrong responses. For example, in 500 tries there were 15 events where 8 versions coincidently returned an answer which was wrong, yet identical within the tolerance used to compare the three most critical (real) variables. Both, Figure 2 and Table 2, are strong indicators of a high degree of inter-version failure dependence in the version set we used.

Figure 2. Example of a joint coincident failure profile.
Table 2. Frequency of empirical coincident identical-and-wrong (IAW) events over 500 test cases for the set of 17 versions shown in Figure 2. The span is the number of versions that coincidentally returned a IAW answer.

<table>
<thead>
<tr>
<th>The Span of IAW Events</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2049</td>
</tr>
<tr>
<td>2</td>
<td>164</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3. System reliability by voting (N=3).

Consensus Voting. Theory predicts that Consensus Voting is always either as reliable, or more reliable, than Majority Voting. In a binary output space, Consensus Voting reduces to Majority Voting, and it cannot improve on it. But for r > 2, Consensus Voting is expected to offer reliability higher than Majority Voting. Theory also predicts that, in N-Version Programming systems
composed of versions of considerably different reliability, both Majority Voting and Consensus Voting would have difficulty providing reliability that exceeded that of the most reliable, or "best", component, although Consensus Voting would still perform better than Majority Voting\textsuperscript{18}.

Figures 3 and 4 illustrate the observed relationship between N-Version Programming with Consensus Voting and Majority Voting. The figures show success frequency, for 3-version and 7-version systems, over a range of average N-tuple reliability. The "ragged" look of the experimental traces is partly due to the small sample (500 test cases), and partly due to the presence of very highly correlated failures. The experimental behavior is in good agreement with the trends indicated by the theoretical Consensus Voting model based on failure independence.

For instance, we see that for N=3 and low average N-tuple reliability, N-Version Programming has difficulty competing with the "best" version. Note that the "best" version was not pre-selected, based on Estimate-I data, but is the N-tuple version which exhibits the smallest number of failures during the actual evaluation run (Estimate-II). The reason N-Version Programming has difficulty competing with the "best" version is that the selected N-tuples of low average reliability are composed of versions which are not "balanced", that is, their reliability is very different, and therefore variance of the average N-tuple reliability is large. As average N-tuple reliability increases, N-Version Programming performance approaches, or exceeds, that of the "best" version. In part, this is because N-tuples become more "balanced," since the number of higher reliability versions in the subpool from which versions are selected is limited. This effect is further discussed in the text related to Table 3 and Figure 7. We also see that N > 3 improves performance of Consensus Voting more than it does that of Majority Voting. This is to a large extent because for N>3 plurality decisions become possible, that is, in situations where there is a unique maximum of identical outputs, the output corresponding to this maximum is selected as the correct answer even though it is not in majority.
Table 3 gives examples of the detailed behavior of selected individual N-tuples. In the table we first show the average reliability of the N-tuple (Avg. Rel.), its standard deviation (Std. Dev.), and the reliability of the acceptance test (AT Rel.). The table then shows the average conditional voter decision space (CD-Space), and its standard deviation of the sample. Average conditional voter decision space is defined as the average size of the space (that is, the number of available unique answers) in which the voter makes decisions, given that at least one of the versions has failed. We use CD-Space to focus on the behavior of the voters when failures are present. Of course, the maximum voter decision space for a single test case is N. We then show the count of the number of times the "best" version in an N-tuple was correct (Best Version), and the success frequency under each of the investigated fault-tolerance strategies. The best response is underlined with a full line, while the second best with a broken line.

Also shown in the table is the breakdown of the decision process for N-Version Programming with Consensus Voting (NVP-CV), that is, the frequency of sub-events that yielded the consensus
decision. We recorded the number of times consensus was a successful majority (S-Majority), an unsuccessful majority (F-Majority), a successful plurality (S-Plurality), an unsuccessful plurality (F-Plurality), a successful (S-Random) and an unsuccessful (F-Random) attempt at breaking a tie by random selection, and a failure by fiat (F-Fiat). F-Fiat denotes a situation where a tie existed but any choice made to break the tie led to failure, because all the groups of outputs involved contained wrong answers. The sum of S-Majority, S-Plurality and S-Random comprises the 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).

Table 3. Examples of the frequency of voting and recovery events.

<table>
<thead>
<tr>
<th>Column:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versions</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>9</td>
<td>11</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>18</td>
<td>13</td>
<td>8</td>
<td>12</td>
<td>8</td>
<td>5</td>
<td>11</td>
<td>7</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>11</td>
<td>8</td>
<td>13</td>
<td>8</td>
<td>11</td>
<td>12</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>14</td>
<td>13</td>
<td>20</td>
<td>20</td>
<td>11</td>
<td>17</td>
<td>20</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>20</td>
<td>16</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mean Value

<table>
<thead>
<tr>
<th>Column:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Rel.</td>
<td>0.59</td>
<td>0.75</td>
<td>0.91</td>
<td>0.67</td>
<td>0.58</td>
<td>0.86</td>
<td>0.70</td>
<td>0.86</td>
<td>0.86</td>
<td>0.92</td>
<td>0.71</td>
<td>0.84</td>
<td>0.83</td>
<td>0.61</td>
<td>0.63</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>0.36</td>
<td>0.21</td>
<td>0.03</td>
<td>0.26</td>
<td>0.21</td>
<td>0.12</td>
<td>0.20</td>
<td>0.11</td>
<td>0.13</td>
<td>0.05</td>
<td>0.21</td>
<td>0.13</td>
<td>0.13</td>
<td>0.22</td>
<td>0.25</td>
</tr>
<tr>
<td>AT Rel.</td>
<td>0.89</td>
<td>0.91</td>
<td>0.89</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>CD-Space</td>
<td>2.91</td>
<td>2.59</td>
<td>2.11</td>
<td>2.43</td>
<td>2.40</td>
<td>2.39</td>
<td>2.81</td>
<td>2.43</td>
<td>2.37</td>
<td>2.42</td>
<td>2.66</td>
<td>2.65</td>
<td>2.53</td>
<td>5.11</td>
<td>4.05</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>0.29</td>
<td>0.49</td>
<td>0.31</td>
<td>1.29</td>
<td>0.91</td>
<td>0.68</td>
<td>0.84</td>
<td>0.63</td>
<td>0.61</td>
<td>0.72</td>
<td>0.76</td>
<td>1.02</td>
<td>0.90</td>
<td>1.15</td>
<td>1.43</td>
</tr>
</tbody>
</table>
Table 3 (Continued). Examples of the frequency of voting and recovery events.

<table>
<thead>
<tr>
<th>Column:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Version</td>
<td>500</td>
<td>497</td>
<td>469</td>
<td>468</td>
<td>439</td>
<td>469</td>
<td>469</td>
<td>469</td>
<td>500</td>
<td>468</td>
<td>469</td>
<td>468</td>
<td>450</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>NVP-MV</td>
<td>206</td>
<td>341</td>
<td>486</td>
<td>321</td>
<td>285</td>
<td>468</td>
<td>405</td>
<td>468</td>
<td>454</td>
<td>466</td>
<td>430</td>
<td>465</td>
<td>455</td>
<td>280</td>
<td>282</td>
</tr>
<tr>
<td>NVP-CV</td>
<td>310</td>
<td>384</td>
<td>491</td>
<td>467</td>
<td>350</td>
<td>483</td>
<td>467</td>
<td>495</td>
<td>474</td>
<td>486</td>
<td>457</td>
<td>482</td>
<td>485</td>
<td>436</td>
<td>464</td>
</tr>
<tr>
<td>RB</td>
<td>443</td>
<td>461</td>
<td>443</td>
<td>454</td>
<td>454</td>
<td>454</td>
<td>441</td>
<td>463</td>
<td>478</td>
<td>493</td>
<td>458</td>
<td>485</td>
<td>462</td>
<td>441</td>
<td>458</td>
</tr>
<tr>
<td>CRB-MV</td>
<td>444</td>
<td>468</td>
<td>486</td>
<td>467</td>
<td>467</td>
<td>468</td>
<td>453</td>
<td>492</td>
<td>475</td>
<td>493</td>
<td>467</td>
<td>486</td>
<td>476</td>
<td>454</td>
<td>471</td>
</tr>
<tr>
<td>CRB-CV</td>
<td>444</td>
<td>468</td>
<td>486</td>
<td>467</td>
<td>465</td>
<td>481</td>
<td>466</td>
<td>494</td>
<td>475</td>
<td>493</td>
<td>467</td>
<td>482</td>
<td>484</td>
<td>433</td>
<td>467</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>S- Majority</td>
<td>206</td>
<td>341</td>
<td>484</td>
<td>321</td>
<td>285</td>
<td>468</td>
<td>405</td>
<td>468</td>
<td>454</td>
<td>466</td>
<td>430</td>
<td>465</td>
<td>455</td>
<td>280</td>
<td>282</td>
</tr>
<tr>
<td>F- Majority</td>
<td>1</td>
<td>32</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S- Plurality</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>146</td>
<td>42</td>
<td>13</td>
<td>61</td>
<td>18</td>
<td>14</td>
<td>16</td>
<td>23</td>
<td>17</td>
<td>29</td>
<td>146</td>
<td>171</td>
</tr>
<tr>
<td>F- Plurality</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>14</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>S- Random</td>
<td>104</td>
<td>43</td>
<td>5</td>
<td>0</td>
<td>23</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>F- Random</td>
<td>189</td>
<td>84</td>
<td>9</td>
<td>0</td>
<td>120</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>14</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>F-Fiat</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>28</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>F-Total</td>
<td>190</td>
<td>116</td>
<td>9</td>
<td>33</td>
<td>150</td>
<td>17</td>
<td>33</td>
<td>5</td>
<td>26</td>
<td>14</td>
<td>43</td>
<td>18</td>
<td>15</td>
<td>64</td>
<td>36</td>
</tr>
</tbody>
</table>
Columns 1 and 2, of Table 3, show the results for two unbalanced, low reliability, 3-tuples. Column 3 shows the results for a well balanced 3-tuple of higher reliability. We see, that in the former case, the highest reliability is that of the best version, while, in the latter, N-Version Programming with Consensus Voting offers the best result. An examination of Consensus Voting sub-events shows that, in the case of 3-tuples, most of the voting success came from majority agreements. The rest of the cases resulted in failures, because all three versions returned different results. Consensus Voting attempts to salvage this situation. For instance, for the 3-tuple in column 1, Consensus Voting attempted to recover 293 times by random selection of one of the outputs. As would be expected, it succeeded about 30% of the time. Notice that in column 3, N-Version Programming with Consensus Voting is more successful than Consensus Recovery Block with Consensus Voting. This is because N-Version Programming with Consensus Voting successfully broke ties five times by random selection, while at the same time Consensus Recovery Block with Consensus Voting unsuccessfully acceptance tested the answers.

Columns 4 to 11 illustrate behavior of 5-tuples, and columns 12 to 15 behavior of 7-tuples. When $N > 3$ the advantages of Consensus Voting, over Majority Voting, increase because plurality vote is now possible. One problem, that N-Version Programming with Majority Voting does not solve, are the small space situations where the vote fails because a voter is offered more than two non-majority groups of answers from which to select the "correct" output, that is, there is no majority so voting cannot return a decision. The events are those where there is no agreement majority, but one of the outputs occurs more frequently than any other, and those where there is a tie between the maximum number of outputs in two or more groups of outputs.

For example, consider the 5-version system from column 4, where N-Version Programming with Consensus Voting is more successful than N-Version Programming with Majority Voting. Correct majority was available in only 321 cases, while in 146 instances the correct output was chosen by plurality. In comparison, the 3-version N-Version Programming with Consensus Voting system from column 1 is more successful than the corresponding N-Version Programming with Majority Voting, primarily because of the random selection process (S-Random).
The theoretical relationship between voter decision space cardinality and voting strategies, assuming failure independence, is shown in Figure 5 for a simulated 5-version system composed of mutually independent versions of average N-tuple reliability of 0.85\textsuperscript{§}. We plot system reliability of N-Version Programming with Consensus Voting, and N-Version Programming with Majority Voting, against the average conditional voter decision space. The average conditional voter decision space was calculated as the mean number of distinct results available to the voter during events where at least one of the 5-tuple versions has failed. The illustrated variation in the average conditional voter decision space (\(v\)) was obtained by changing the output space cardinality from \(r=2\) to \(r=\infty\). This resulted in the variation in \(v\) in the range \(2 \leq v < 2.35\). Also shown is the N-Version Programming 2-out-of-N boundary (\(r=\infty\)).

Theory predicts that, as the decision space increases (\(v > 2\)), the difference, between the reliability of the systems using N-Version Programming with Consensus Voting and systems using N-Version Programming with Majority Voting, increases in favor of N-Version Programming with Consensus Voting\textsuperscript{18}. Figure 6 illustrates the observed relationship between system success frequency, and the average conditional voter decision space, for a subset of 5-version systems with N-tuple reliability close to 0.85. Note that in Figure 6 the variation in the voter decision space size is caused by the variation in the probability of obtaining coincident, but different, incorrect answers. The observed behavior is in good general agreement with the trend shown in Figure 5, except that in the experiment, as the decision space increases, the reliability of N-Version Programming with Consensus Voting increases at a slower rate, and reliability of N-Version Programming with Majority Voting appears to decrease.

\textsuperscript{§} Reliability of individual versions ranged between about 0.78 and 0.91, standard deviation of the sample was 0.061.
Simulation Sample = 100,000
Average Version Reliability = 0.85

Figure 5. Influence of voter space size on different voting strategies.
In practice, failure probabilities of individual versions have a non zero standard deviation about the N-tuple mean. Small scatter may, up to a point, appear to increase average reliability obtained by voting, because there may be enough versions on the "high" side of the mean to form a correct agreement number more often than would be expected from a set where all versions have the same reliability. But, when the scatter is excessive, the system reliability can actually be lower than the reliability of one or more of its best component versions\(^\text{18}\).

This effect is illustrated in Figure 7 (independent model simulation; 100,000 cases for each point shown). In the figure, we plot the reliability of N-Version Programming with Consensus Voting, and the reliability of N-Version Programming with Majority Voting, against the standard deviation of the N-tuple reliability (the mean value being constant and equal to 0.95). Also shown is the reliability of the best single version obtained from the simulation. The feature to note is the very sharp step in the best version reliability, once some critical value of the standard deviation of the sample is exceeded (about 0.03 in this example). The effect can be seen for some of the tuples shown in
Table 3 (for example, columns 1, 2, 4, 10, 11, and 15). Low average reliability systems, with a high standard deviation about the mean, tend to perform worse than the "best" version.

![Graph showing system reliability by Consensus Voting for 5-version systems vs. standard deviation of 5-tuple reliability. The probability of each j=2,...,r failure state is $\frac{1-\bar{p}}{r-1}$, where $\bar{p}$ is the average 5-tuple reliability.]

**Figure 7.** System reliability by Consensus Voting for 5-version systems vs. standard deviation of 5-tuple reliability. The probability of each $j=2,...,r$ failure state is $\frac{1-\bar{p}}{r-1}$, where $\bar{p}$ is the average 5-tuple reliability.

Experimental results indicate that N-Version Programming with Consensus Voting appears to behave like its models, based on failure independence, predict. The advantage of Consensus Voting is that it is more stable than Majority Voting. It always offers reliability at least equivalent to Majority Voting, and it performs better than Majority Voting when average N-tuple reliability is low, or the average decision space in which voters work is not binary. A practical disadvantage of Consensus Voting may be the added complexity of the voting algorithm (or hardware), since the strategy requires multiple comparisons and random number generation.
**Consensus Recovery Block.** Theory predicts that in an ideal situation (version failure independence, zero probability for identical-and-wrong responses) Consensus Recovery Block is always superior to N-Version Programming (given the same version reliability and the same voting strategy), or to Recovery Block$^{5,14}$ (given the same version and acceptance test reliability). This is illustrated in Figure 8, using 2-out-of-N voting and a perfect voter. It is interesting to note the existence of a cross-over point between Recovery Block and N-Version Programming reliability caused by the finite reliability of the Recovery Block acceptance test. In the figure the acceptance test reliability is $1-\beta = 0.9$, the probability that the acceptance test rejects a correct results is $\beta_1$, and the probability that it accepts an incorrect answer as correct is $\beta_2$. Of course, the behavior is modified when different voting strategies are used, or if inter-version failure correlation is substantial.

![Figure 8](image-url) **Figure 8.** System reliability for software fault tolerance schemes with 2-out-of-N voting, $N = 3$. 
Given the same voting strategy, and very high inter-version failure correlation, we would expect Consensus Recovery Block to do better than N-Version Programming only in situations where coincidentally failing versions return different results. We would not expect the Consensus Recovery Block to more than match N-Version Programming in situations where the probability of identical-and-wrong answers is very high, since then, many decisions would be made in a very small voting space, and the Consensus Recovery Block acceptance test would be invoked only very infrequently.

Some additional experimental results are shown in Figures 9, 10 and 11. The figures show the number of times that the result provided by a strategy was correct plotted against the average N-tuple reliability. The same acceptance test version was used by Consensus Recovery Block and Recovery Block. From Figure 9 we see that for N=3, Consensus Recovery Block with Majority Voting provides reliability always equal to or larger than the reliability by N-Version Programming with Majority Voting (given the same versions).

The behavior of the same 5-version systems using Consensus Voting, instead of Majority Voting, is shown in Figures 10 and 11. From Figure 10 we see that, at lower N-tuple reliability, N-Version Programming with Consensus Voting becomes almost as good as Consensus Recovery Block. Figure 11 shows that Consensus Recovery Block, with Consensus Voting, is quite successful in competing with the "best" version. We also see that the expected cross-over point between N-Version Programming and Recovery Block is present, and that reliability of Consensus Recovery Block with Consensus, or Majority, Voting is usually at least as good as that by Recovery Block (Figures 9, 11). However, it must be noted that, given a sufficiently reliable acceptance test, or binary output space, or very high inter-version failure correlation, all the schemes that vote may have difficulty competing with Recovery Block. Also observed were two, less obvious, events that stem from the difference between the way Consensus Voting is implemented. They are described in the next paragraph.
Figure 9. Consensus Recovery Block system reliability with majority voting.

Consensus Recovery Block with Consensus Voting is a more advanced strategy than N-Version Programming with Consensus Voting, and usually it is more reliable than N-Version Programming with Consensus Voting. However, there are situations where the reverse is true. Because Consensus Recovery Block with Consensus Voting employs the acceptance test to resolve situations where there is no plurality, while N-Version Programming with Consensus Voting uses random tie breaking, occasionally N-Version Programming with Consensus Voting may be marginally more reliable than Consensus Recovery Block with Consensus Voting. This will happen when the acceptance test reliability is low, or when acceptance test and program failures are identical-and-wrong. Examples of this behavior can be seen in columns 3, 6, 7, 8 and 13 of Table 3. The difference in favor of N-Version Programming with Consensus Voting is often exactly equal to S-Random.
Figure 10. Consensus Recovery Block with Consensus Voting compared with N-Version Programming with Consensus Voting and Recovery Block.

Figure 11. Comparison of Consensus Recovery Block with Consensus Voting with Recovery Block and best version successes.
Similarly, Consensus Recovery Block with Consensus Voting usually is more reliable than Consensus Recovery Block with Majority Voting. However, if the number of agreeing versions is less than the majority, the reverse may be true. For instance, if there is no majority, then Majority Voting will fail and the decision will pass to the acceptance test (which may succeed), while Consensus Voting will vote, and, if the plurality is incorrect because of identical and wrong answers, Consensus Voting may return an incorrect answer. Examples can be found in columns 5, 12, 14 and 15 of Table3.

A more general conclusion, based on the observed implementations, is that the Consensus Recovery Block strategy appears to be quite robust in the presence of high inter-version correlation, and that the behavior is in good agreement with analytical considerations based on models that make the assumption of failure independence5,14. Of course, the exact behavior of a particular system is more difficult to predict, since correlation effects are not part of the models. An advantage of Consensus Recovery Block with Majority Voting is that the algorithm is far more stable, and is almost always more reliable than N-Version Programming with Majority Voting. On the other hand, the advantage of using a more sophisticated voting strategy such as Consensus Voting, may be marginal, since the Consensus Recovery Block version of the Consensus Voting algorithm relies on the acceptance test to resolve ties. The Consensus Voting version of CRB may be a better choice in high correlation situations where the acceptance test is of poor quality. In addition, Consensus Recovery Block will perform poorly in all situations where the voter is likely to select a set of identical-and-wrong responses as the correct answer (binary voting space). To counteract this, we could either use a different mechanism, such as the Acceptance Voting algorithm, or an even more complex hybrid mechanism that would run Consensus Recovery Block and Acceptance Voting in parallel, and adjudicate series-averaged responses from the two14,19. A general disadvantage of all hybrid strategies is an increased complexity of the fault-tolerance mechanism, although this does not necessarily imply an increase in costs26.
SUMMARY AND CONCLUSIONS

In this paper we presented the first experimental evaluation of Consensus Voting, and an experimental evaluation of the Consensus Recovery Block scheme. The evaluations were performed under conditions of high inter-version failure correlation, and version reliability in the range between about 0.5 and 0.99.

The experimental results confirm, the theoretically expected, superior reliability performance of Consensus Voting over Majority Voting. The experiments also confirm that the Consensus Recovery Block strategy outperforms simple N-Version Programming, and is very robust in the presence of inter-version failure correlation. In general, our experimental results agree very well with the behavior expected on the basis of the analytical studies of the hybrid models, however additional experiments are needed to further validate these observation in different contexts. Of course, behavior of an individual practical system can deviate considerably from that based on its theoretical model average, so considerable caution is needed when predicting behavior of practical fault-tolerant software systems, particularly if presence of inter-version failure correlation is suspected.

REFERENCES


Appendix I

To select subsets of N-tuples with have certain properties such as approximately equal reliability we used the following approach.

We first select acceptance test versions based on Estimate I data (for example, one low reliability, one medium and one high reliability acceptance test). These versions are then removed from the pool of 20 versions. Also removed from the pool might be versions that have either very low or very high reliability to better balance reliability of the selected N-tuples. For a given N the remainder of the versions (a subpool) are then randomly sampled without replacement until an N-tuple that has not already been accepted for the subset is formed. The average N-tuple reliability is then computed, and if it lies within the desired reliability range the N-tuple becomes a member the subset. Then the N-tuple versions are returned to the subpool and the next N-tuple is selected in a similar manner, etc. Once the subset contains either all possible combinations, or at least 600 N-tuples (whichever comes first), the subset is sorted by the average N-tuple reliability and standard deviation of the average N-tuple reliability.

If a single reliability category is desired (e.g. between 0.8 and 0.9) then the first 30 versions with the smallest N-tuple standard deviation are chosen and run in the experiment.

If a range of reliability is desired, the range is divided into categories in such a way that members of the same category have identical first two digits after the decimal point. Then from each category we chose the combination that has the smallest standard deviation of the average N-tuple reliability.

We have thus selected a number of subsets. The following are mentioned in the text

N-Tuple Subset A, 5 version systems, average 5-tuple reliability in the range 0.8 to 0.9, acceptance test reliability of 0.67 (version 20), 0.93 (version 2) and 0.994 (version 18). Version 10 was not used (too reliable).

N-Tuple Subset B, 3 version systems, average 3-tuple reliability in the range 0.5 to 1.0, acceptance test reliability of 0.67 (version 20), 0.89 (version 3) and 0.91 (version 17). Version 10 was used.
N-Tuple Subsets C and D were chosen in a manner similar to set B except that they consisted of 5-tuples and 7-tuples, respectively.

N-Tuple Subset E, 5 version systems, average 5-tuple reliability about 0.91, no acceptance test.