System-Level Reliability and Sensitivity Analyses for Three Fault-Tolerant System Architectures

Joanne Bechta Dugan and Michael R. Lyu†*
Department of Electrical Engineering
University of Virginia
Charlottesville, Virginia 22903-2442
jbd@Virginia.edu
†Bellcore
Morristown, New Jersey 07692-1910
lyu@bellcore.com

Abstract

This paper discusses the modeling and analysis of three major fault-tolerant software system architectures: DRB (Distributed Recovery Blocks), NVP (N-Version Programming) and NSCP (N Self-Checking Programming). In the system-level reliability modeling domain, fault tree analysis techniques and Markov modeling techniques are combined to incorporate transient and permanent hardware faults as well as unrelated and related software faults. These models are parameterized by a real-world fault-tolerant flight control computer application for evaluations and comparisons. In particular, a series of sensitivity analysis is performed to explore the critical components in each fault-tolerant architecture and display their quantitative impacts to the overall system reliability.

1 Introduction

Since the first computer was invented some forty years ago, human beings have been depending more and more on computers in their daily lives. When the requirements for and dependencies on computers increase, the crises of computer failures also increase. The impact of hardware and software failures range from inconvenience (e.g., malfunctions of home appliances), economic

F. Cristian et al. (eds.), *Dependable Computing for Critical Applications 4*
© Springer-Verlag/Wien 1995

loss (e.g., interceptions of banking systems) to life-threatening (e.g., failures of flight systems). Needless to say, reliability of computer systems becomes the major concern for our society for the 1990's and beyond. Consequently, computer systems that are used for critical applications are designed to tolerate both software and hardware faults by executing multiple software versions on redundant hardware. Many such examples exist in the aerospace industry [23, 7, 21], nuclear power industry [15, 2, 22], and ground transportation industry [6].

The system architectures incorporating both hardware and software fault tolerance are explored in three typical approaches. The Distributed Recovery Blocks (DRB) scheme [9] combines both distributed processing and Recovery Block (RB) [16] concepts to provide a unified approach to tolerating both hardware and software faults. Architectural considerations for the support of N-Version Programming (NVP) [1] were addressed in [10], in which the FTP-AP system is described. The FTP-AP system achieves hardware and software design diversity by attaching application processors (AP) to the byzantine resilient hard core Fault Tolerant Processor (FTP). N Self-Checking Programming (NSCP) [12] uses diverse hardware and software in self-checking groups to detect hardware and software induced errors. The NSCP concept forms the basis of the flight control system used on the Airbus A310 and A320 aircraft [3].

Sophisticated techniques exist for the separate analysis of fault tolerant hardware [5, 8] and software [11, 18, 19], and a few authors have considered their combined analysis [11, 20, 13]. This paper uses a combination of fault tree and Markov modeling as a framework for the analysis of hardware and software fault tolerant systems. The overall system model is a Markov model in which the states of the Markov chain represent the evolution of the hardware configuration as permanent faults occur. A fault tree model for each state in the Markov chain captures the effects of software bugs and transient hardware faults. This hierarchical approach simplifies the development, solution and understanding of the modeling process. In performing each model, the parameter values are derived from the analysis of data collected from an experimental NVP implementation [14]. A number of sensitivity analyses are conducted to study the quantitative behavior of the system reliability with respect to the parameter values.

2 Modeling Methodology

2.1 Assumptions

Task computation. The computation being performed is a task (or set of tasks) which is repeated periodically. A set of sensor inputs is gathered and analyzed and a set of actuations are produced. Each repetition of a task is independent. The goal of the analysis is the probability that a task will succeed in producing an acceptable output.

Software failure probability. Software faults exist in the code, despite rigorous testing. A fault is activated by some random input and produces an erroneous result. Each compu-

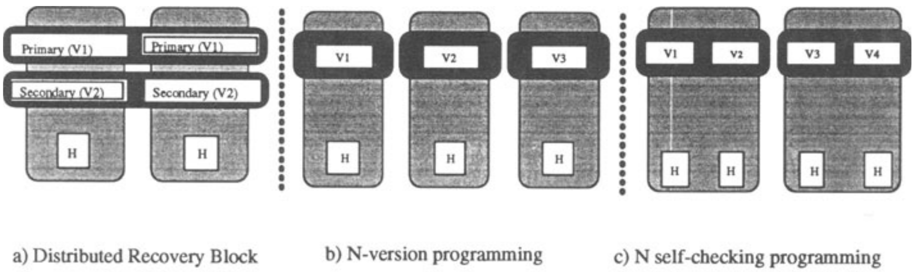


Figure 1: Structure of a) DRB, b) NVP and c)NSCP

tation of a task receives a different set of inputs which are independent. Thus, a software task has a fixed probability of producing an error for a given task execution.

Constant hardware failure rates. The arrival (activation) rate of *permanent* physical faults is constant and will be denoted by λ.

Transient hardware faults. Transient hardware faults are modeled separately from permanent hardware faults. A transient hardware fault is assumed to upset the software running on the processor and produce an erroneous result which is indistinguishable from an input-activated software error. We assume that the lifetime of transient hardware faults is shorter than a task computation, and thus assign a fixed probability to the occurrence of a transient hardware fault during a single computation.

Related software faults. A related software fault in two different variants produce similar erroneous results on the same input. The two erroneous results match, which will be undetected if the results are compared to each other.

For the comparisons drawn from this study, we assume that the systems are unmaintained. Repairability and maintainability could certainly be included in the Markov model; we have chosen not to include them to make the comparisons clearer. More interesting task computation processes could be considered within this modeling framework as well.

Figure 1 shows the hardware and software error confinement areas [12] associated with the three architectures being considered in this paper. The systems are defined by the number of software variants, the number of hardware replications, and the decision algorithm. The error confinement area covers the region of the system affected by faults in that component.

2.2 System reliability model

A reliability model of an integrated fault tolerant system must include at least three different factors: computation errors, system structure and coverage modeling. In this paper we concentrate on the first two, as coverage modeling has been addressed in detail elsewhere [4].

The computation process is assumed to consist of a single software task that is executed repeatedly, such as would be found in a process control system. The software component performing the task is designed to be fault tolerant. A single task iteration consists of a task execution on a particular set of input values read from sensors. The output is the desired actuation to control the external system. During a single task iteration, several types of events can interfere with the computation. The particular set of inputs could activate a software fault in one or more of the software versions and/or the decider. Also, a hardware transient fault could upset the computation but not cause permanent hardware damage. The combinations of software faults and hardware transients that can cause an erroneous output for a single computation is modeled with a fault tree. The solution of the fault tree yields the probability that a single task iteration produces an erroneous output. We note that in the more general case where more than one task is performed, the analyses of each task can be combined accordingly.

The longer-term system behavior is affected by permanent faults and component repair which require system reconfiguration to a different mode of operation. The system structure is modeled by a Markov chain, where the Markov states and transitions model the long term behavior of the system as hardware and software components are reconfigured in and out of the system. Each state in the Markov chain represents a particular configuration of hardware and software components and thus a different level of redundancy. The fault and error recovery process is captured in the coverage parameters used in the Markov chain [4].

The short-term behavior of the computation process and the long-term behavior of the system structure are combined as follows. For each state in the Markov chain, there is a different combination of hardware transients and software faults that can cause a computation error, and thus a different probability that an unacceptable result is produced.

The fault tree model solution produces, for each state i in the Markov model, the probability q_i that an output error occurs during a single task computation while the state is in state i. The Markov model solution produces $P_i(t)$, the probability that the system is in state i at time t. The overall model combines these two measures to produce $Q(t)$, the probability that an unacceptable result is produced at time t.

$$Q(t) = \sum_{i=1}^{n} q_i P_i(t)$$

We assume that the system is unable to produce an acceptable result while in the failure state, thus ($q_{fail} = 1$).

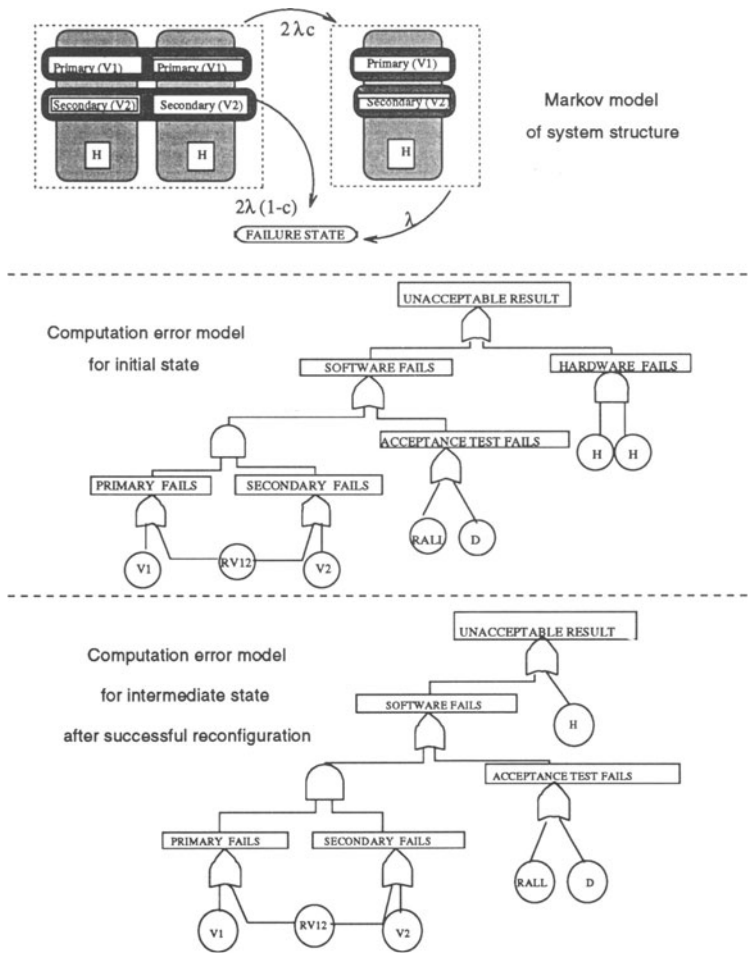


Figure 2: Reliability model of DRB.

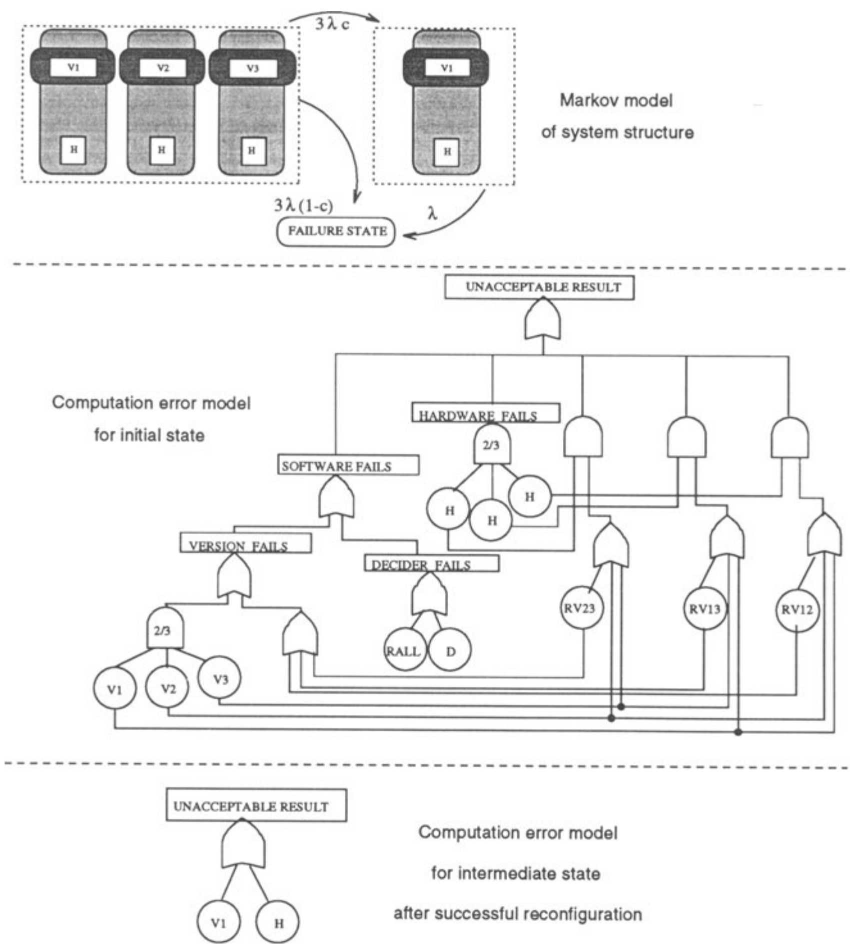


Figure 3: Reliability model of NVP.

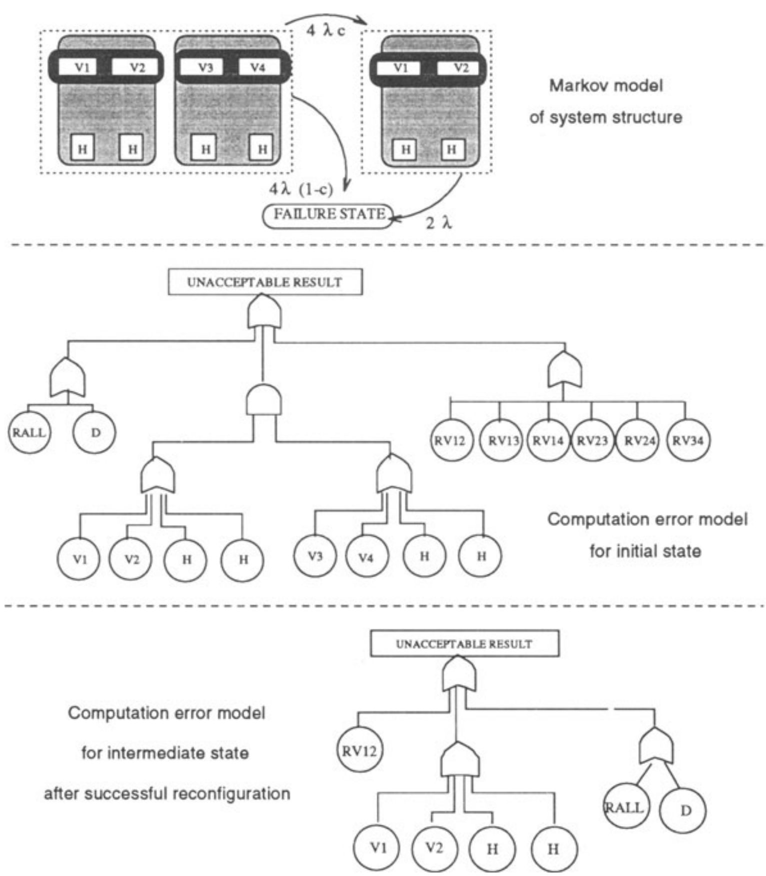


Figure 4: Reliability model of NSCP.

The three-part reliability models used for analysis of DRB, NVP and NSCP are shown in figures 2, 3 and 4, respectively. Each model consists of a three state Markov chain and two fault tree models. The Markov chain shows the evolution of the system structure as permanent physical faults are activated and handled. The fault tree models show the combinations of events which can upset a single task iteration in the full-up (initial) state and the intermediate state of the Markov model. In the Markov model, λ is the rate at which permanent hardware faults are activated and c is the probability that the system can automatically recover from a hardware fault. The basic events in the fault tree represent unrelated software failures (labeled Vi), related software failures between two versions (labeled Vij), related software failures in all versions ($RALL$), decider failures (D) and hardware transients (H).

3 Experimental Data Analysis

3.1 Description of experiment

The parameter values for the models in this paper were determined using actual data derived from an experimental implementation of a real-world automatic (i.e., computerized) airplane landing system, or so-called "autopilot." The software systems of this project were developed and programmed by 15 programming teams at the University of Iowa and the Rockwell/Collins Avionics Division. A total of 40 students (33 from ECE and CS departments at the University of Iowa, 7 from the Rockwell International) participated in this project to independently design, code, and test the computerized airplane landing system, as described in the Lyu-He study [14].

The application used in the Lyu-He study is part of a specification used by some aerospace companies for the automatic (computer-controlled) landing of commercial airliners. The specification can be used to develop the software of a flight control computer (FCC) for a real aircraft, given that it is adjusted to the performance parameters of a specific aircraft. All algorithms and control laws are specified by diagrams which have been certified by the Federal Aviation Administration (FAA). The *pitch control* part of the auto-landing problem, i.e., the control of the vertical motion of the aircraft, was selected for the project in order to fit the 14-week software development time.

By the end of the software development phase, 12 of the 15 programs passed the acceptance test successfully and were engaged in operational testing for further evaluations. The average size of these programs were 1564 lines of uncommented code, or 2558 lines when comments were included. The average fault density of the program versions which passed AT1 (the first step in the Acceptance Test) was 0.48 faults per thousand lines of uncommented code. The fault density for the final versions was 0.05 faults per thousand lines of uncommented code.

The operational environment for the application was conceived as airplane/autopilot interacting in a simulated environment. During the operational phase, 1000 flight simulations were con-

Version Id	Number of failures	Prob. by case	Prob. by time
β	510	0.51	0.000096574
γ	0	0.0	0.0
ϵ	0	0.0	0.0
ζ	0	0.0	0.0
η	1	0.001	0.000000189
θ	360	0.36	0.000068169
κ	0	0.0	0.0
λ	730	0.73	0.000138233
μ	140	0.14	0.000026510
ν	0	0.0	0.0
ξ	0	0.0	0.0
o	0	0.0	0.0
Average	145.1	0.1451	0.000027472

Table 1: Characteristics of accepted programs

ducted. Each flight simulation was characterized by the following five initial values regarding the landing position of an airplane: (1) initial altitude (about 1500 feet); (2) initial distance (about 52800 feet); (3) initial nose up relative to velocity (range from 0 to 10 degrees); (4) initial pitch attitude (range from -15 to 15 degrees); and (5) vertical velocity for the wind turbulence (0 to 10 ft/sec). One simulation consisted of about 5280 iterations of lane command computations (50 milliseconds each) for a total landing time of approximately 264 seconds. For a conservative estimation of software failures in the system, we took the program versions which passed the AT1 for study. The reason behind this was that had the Acceptance Test not included an extra test case after AT1, more faults would have remained in the program versions.

3.2 Failure data analysis

Table 1 shows the software failures encountered in each single version. We examine two levels of granularity in defining software execution errors and correlated errors: "by case" or "by time." The first level was defined based on test cases (1000 in total). If a version failed at any time in a test case, it was considered failed for the whole case. If two or more versions failed in the same test case (no matter at the same time or not), they were said to have coincident errors for that test case. The second level of granularity was defined based on execution time frames (5,280,920 in total). Errors were counted only at the time frame upon which they manifested themselves, and coincident errors were defined to be the multiple program versions failing at the same time in the same test case (with or without the same variables and values).

The accepted programs were then arranged in configurations of 2, 3 and 4 programs, and the error characteristics of each of the configurations is shown in tables 2, 3 and 4. Both the by-case

Category	BY CASE		BY TIME	
	Number of cases	Frequency	Number of cases	Frequency
1 - no errors	53150	0.8053	348259290	0.999192
2 - single error	11160	0.1691	281200	0.000807
3 - two coincident errors	1690	0.0256	230	0.000001
Total	66000	1.0000	1161802400	1.000000

Table 2: Error characteristics for two-version configurations

Category	BY CASE		BY TIME	
	Number of cases	Frequency	Number of cases	Frequency
1 - no errors	163370	0.7426	1160743690	0.999089
2 - single error	51930	0.2360	1056010	0.000909
3 - two coincident errors	4440	0.0202	2700	0.000002
4 - three coincident errors	260	0.0012	0	0.0
Total	220000	1.0000	1161802400	1.000000

Table 3: Error characteristics for three-version configurations

and by-time error detection methods were used. These characteristics were used to determine parameter values for the software failure models of DRB, NVP and NSCP.

Table 5 summarizes the parameters used for the software parameters of the system models. These parameters are derived from a single experimental implementation and so may not be generally applicable. Similar analysis of other experimental data will help to establish a set of reasonable parameters that can be used in models that are developed during the design phase of a fault tolerant system.

Category	BY CASE		BY TIME	
	Number of cases	Frequency	Number of cases	Frequency
1 - no errors	322010	0.65052	2611305000	0.998948
2 - single error	152900	0.30889	2719200	0.001040
3 - two coincident errors	16350	0.03303	31200	0.000012
4 - three coincident errors	3700	0.00747	0	0.0
5 - four coincident errors	40	0.00008	0	0.0
Total	495000	1.0000	2614055400	1.000000

Table 4: Error characteristics for four-version configurations

DRB model	**NVP model**	**NSCP model**
BY CASE DATA		
$P_V = 0.095$ $P_{RV} = 0.0167$	$P_V = 0.0958$ $P_{RV} = 0$ $P_{RALL} = 0.0003$	$P_V = 0.106$ $P_{RV} = 0$ $P_{RALL} = 0$
Predicted failure probability (perfect decider, no HW faults)		
0.0265	0.0262	0.0403
Observed failure probability (from the data)		
0.0256	0.0214	0.0406
Probability of decider failure used for system analysis		
0.001	0.0001	0.0001
BY TIME DATA		
$P_V = 0.0004$ $P_{RV} = 8.4 \times 10^{-7}$	$P_V = 0.0003$ $P_{RV} = 6 \times 10^{-7}$ $P_{RALL} = 0$	$P_V = 0.00026$ $P_{RV} = 0$ $P_{RALL} = 1.2 \times 10^{-5}$
Predicted failure probability (perfect decider, no HW faults)		
1×10^{-6}	2.07×10^{-6}	1.23×10^{-5}
Observed failure probability (from the data)		
1×10^{-6}	2.3×10^{-6}	1×10^{-5}
Probability of decider failure used for system analysis		
1×10^{-7}	1×10^{-7}	1×10^{-7}

Table 5: Summary of nominal software parameters used for system analysis

3.3 Hardware parameters

Typical permanent failure rates for processors range in the 10^{-5} *per hour* range, with transients perhaps an order of magnitude larger. Thus we will use $\lambda_p = 10^{-5}$ per hour for the Markov model.

In the by-case scenario, a typical test case contained 5280 time frames, each time frame being 50 ms., so a typical computation executed for 264 seconds. Assuming that hardware transients occur at a rate $\lambda_t = (10^{-4}/3600)$ *per second*, we see that the probability that a hardware transient occurs during a typical test case is

$$1 - e^{-\lambda_t \times 264 \ seconds} = 7.333 \times 10^{-6} \tag{1}$$

We conservatively assume that a hardware transient that occurs anywhere during the execution of a task disrupts the entire computation running on the host.

For the by-time data, the probability that a transient occurs during a time frame is

$$1 - e^{-\lambda_t \times 0.05 \ seconds} = 1.4 \times 10^{-9} \tag{2}$$

If we further assume that the lifetime of a transient fault is one second, then a transient can affect as many as 20 time frames. We thus take the probability of a transient to be 20 times the value calculated in equation 2, or 2.8×10^{-8}.

Finally, for both the by-case and by-time scenarios, we assume a fairly typical value for the coverage parameter in the Markov model, $c = 0.999$.

4 Reliability and Sensitivity Analysis

4.1 Reliability analysis

Figure 5 compares the predicted behavior of the three systems. Under both the by-case and by-time scenarios, the recovery block system is most able to produce a correct result, followed by NVP. NSCP is the least reliable of the three. It is noted, however, that the analysis performed in this paper is based on a *reliability* aspect (i.e., whether the system can deliver an acceptable result) rather than on a *safety* aspect (i.e., whether the system can deliver an acceptable result *or* conduct a safety shutdown after detecting an unacceptable condition). NSCP is expected to obtain a much better improvement with respect to the safety analysis. Of course, these comparisons are dependent on the experimental data used and assumptions made. More experimental data and analysis are needed to enable a more conclusive comparison.

Figure 6 gives a closer look at the comparisons between the NVP and DRB systems during the first 200 hours. The by-case data shows a crossover point where NVP is initially more reliable

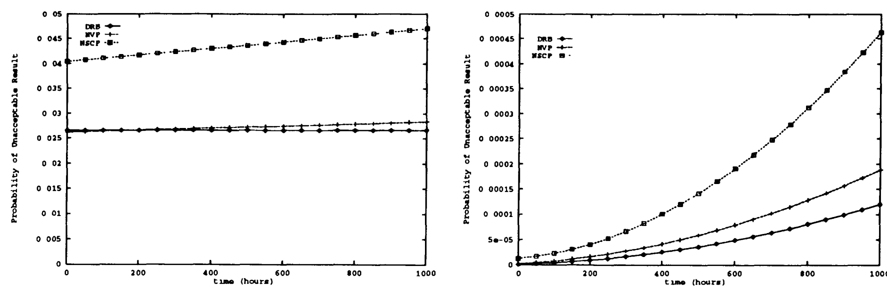


Figure 5: Predicted reliability, by-case data (left) and by-time data (right)

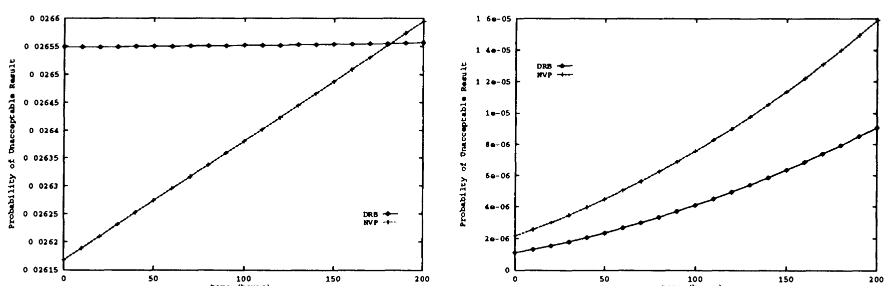


Figure 6: Predicted reliability, by-case data (left) and by-time data (right)

	By CASE Data		By TIME Data	
Parameter	Result	Percent Change	Result	Percent Change
Nominal	0.0265		1.10×10^{-6}	
$P_V + 10\%$	0.0284	7%	1.13×10^{-6}	2.8%
$P_{RV} + 10\%$	0.0282	6.2%	1.18×10^{-6}	7.6%
$P_D + 10\%$	0.0266	1.9%	1.11×10^{-6}	0.9%

Table 6: Sensitivity to parameter change for DRB model

	By CASE Data		By TIME Data	
Parameter	Result	Percent Change	Result	Percent Change
Nominal	0.02617		2.17×10^{-6}	
$P_V + 10\%$	0.03137	19.9%	2.23×10^{-6}	2.6%
$P_{RV} + 10\%$			2.35×10^{-6}	8.3%
$P_{RALL} + 10\%$	0.0262	0.1%		
$P_D + 10\%$	0.02618	0.04%	2.18×10^{-6}	0.5%

Table 7: Sensitivity to parameter change for NVP model

but is later less reliable than DRB. Using the by-time data, there is no crossover point, but the estimates are so small that the differences may not be statistically significant.

For all three systems the probability of producing an unacceptable result is initially much lower with the by-time data than with the by-case data. This analysis dramatizes the potential improvement associated with frequent comparisons (each time frame rather than each test case). The probability of producing an unacceptable result increases with time, as expected, but at 1000 hours is still far below even the initial by-case probability.

4.2 Sensitivity Analysis

To see which parameters are the strongest determinant of the system reliability, we increased each of the failure probabilities in turn by 10 percent and observed the effect on the predicted unreliability. The sensitivity of the predictions to a ten-percent change in input parameters for the DRB model is shown in table 6. It can be seen that the DRB model is most sensitive to a change in the probability of an unrelated fault for the by-case data, and to a change in the probability of a related fault for the by-time data.

Table 7 shows, the change in the predicted unreliability (at $t = 0$) when each of the NVP nominal parameters is increased. For the by-case data, a ten percent increase in the probability of an unrelated software fault results in a twenty percent increase in the probability of an unacceptable result. A ten-percent increase in the probability of a related or decider fault activation has an almost negligible effect on the unreliability. For the by-time data, the proability of a related

	By CASE Data		By TIME Data	
Parameter	Result	Percent Change	Result	Percent Change
Nominal	0.04041		1.237×10^{-5}	
$P_V + 10\%$	0.04833	19.6%	1.243×10^{-5}	0.5%
$P_{RALL} + 10\%$			1.357×10^{-5}	9.7%
$P_D + 10\%$	0.04042	0.02%	1.238×10^{-5}	0.08%

Table 8: Sensitivity to parameter change for NSCP model

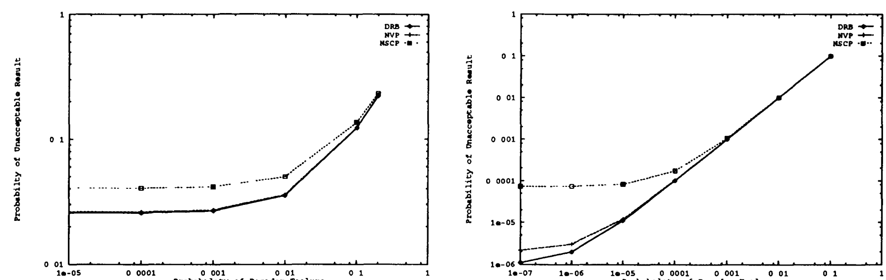


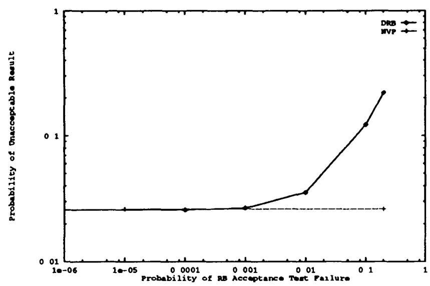
Figure 7: Effect of equal decider failure probabilities, by-case data (left) and by-time data (right)

fault has the largest impact on the probability of an unacceptable result. This is similar to the DRB model.

The sensitivity of the NSCP model to the nominal parameters is shown in table 8. The fault tree models and the sensitivity analysis show that NSCP is vulnerable to related faults, whether they involve versions in the same error confinement area or not.

5 Decider Failure Probability

The probability of a decider failure may be an important input parameter to the comparative analysis of the NVP and DRB systems. In this section we vary the decider failure probability in an attempt to demonstrate its importance. Figure 7 shows, for the by-case and by-time parameterizations, the unreliability of the three systems as the probability of decider failure is varied. For this analysis, we set the probability of failure for the decider to the same value for all three models, and show the probability of an unacceptable result at time $t = 0$.



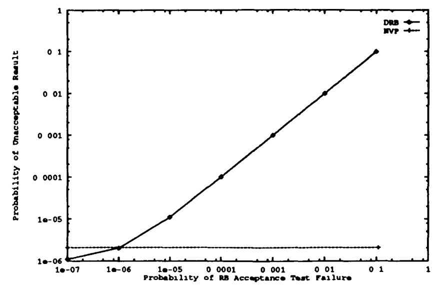


Figure 8: Effect of varying acceptance test failure probability, by-case data(left) and by-time data (right)

For the parameters derived from the experimental data, it seems that DRB and NVP are nearly equally reliable, if both have the same probability of decider failure. However, it is not reasonable for this application to assume equally reliability deciders for both DRB and NVP. The decider for the DRB system is an acceptance test, while that for the NVP is a simple voter and NSCP a simple comparator. For this application, it seems likely that an acceptance test will be more complicated than a majority voter. The increased complexity is likely to lead to a decrease in reliability, with a corresponding impact on the reliability of the system. In fact, reliability of DRB will collapse if the acceptance test in DRB is as complex and unreliable as its primary or secondary software versions. For example, if the probability of failure in acceptance test (P_D) is close to P_V, which is 0.095 by case or 0.0004 by time, then both Figure 7 indicates that DRB will initially perform the worst comparing with NVP and NSCP.

Figure 8 shows how the comparison between DRB and NVP is affected by a variation in the probability of failure for the acceptance test. The parameters for the NVP analysis were held constant, and the parameters (other than the probability of acceptance test failure) for the DRB model were also held constant. Figure 8 shows that the acceptance test for a recovery block system must be very reliable for it to be comparable in reliability to a similar NVP system.

6 Conclusions

We have proposed a system-level modeling approach to study the reliability behavior of three types of fault-tolerant architectures: DRB, NVP and NSCP. Using a recent fault-tolerant software project data, we parameterized the models and displayed the resulting system (un)reliability. The comparisons of the three fault-tolerant architectures were done not only from directly applying the estimated parameters, but from varying the baseline parameters as a sensitivity analysis. Several interesting results were obtained:

1. A drastic improvement of reliability could be observed if a finer and more frequent error detection mechanism could be performed by the decider for each architecture.

2. From the by-case data, varying the probability of an unrelated software fault had the major impact to the system reliability, while from the by-time data, varying the probability of a related fault had the largest impact. This could be due to the fact that the by-time data compares results in a finer granularity level, and is thus more sensitive to related faults among program versions.

3. In comparing the three different architectures, DRB performed better than NVP which in turn was better than NSCP. DRB also enjoyed the feature of relative insensitivity to time in its reliability function. DRB might perform worse than NVP to begin with, but in the long run it could become better.

4. The acceptance test in DRB had to be very reliable for (3) to remain true. If the acceptance test in DRB is as unreliable as its application versions, DRB loses its advantage to NVP and NSCP.

5. NSCP did not seem to perform very well in the reliability analysis. However, it is expected to gain more improvement and close the gap to the other two architectures if a safety analysis is performed.

Needless to say, more data points are wanted for the validation of our models and for more evidences of the advantages and disadvantages of the three fault-tolerant system architectures.

Acknowledgements

This work was partially funded by NASA AMES Research Center under grant number NCA2-617. The authors are grateful to Yu-Tao He and Stacy Doyle for their assistance. The models presented in this paper were solved using SHARPE [17].

References

[1] Algirdas Avižienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501, December 1985.

[2] P.G. Bishop, D.G. Esp, M. Barnes, P. Humphreys, G. Dahl, and J. Lahti. PODS - a project of diverse software. *IEEE Transactions on Software Engineering*, SE-12(9):929–940, September 1986.

[3] D. Briere and P. Traverse. Airbus A320/A330/A340 electrical flight controls: A family of fault-tolerant systems. In *Proceedings of the 23rd Symposium on Fault Tolerant Computing*, pages 616–623, 1993.

[4] Joanne Bechta Dugan and K. S. Trivedi. Coverage modeling for dependability analysis of fault-tolerant systems. *IEEE Transactions on Computers*, 38(6):775–787, 1989.

[5] Robert Geist and Kishor Trivedi. Reliability estimation of fault-tolerant systems: Tools and techniques. *IEEE Computer*, pages 52–61, July 1990.

[6] Gunnar Hagelin. ERICSSON safety system for railway control. In U. Voges, editor, *Software Diversity in Computerized Control Systems*, pages 11–21. Springer-Verlag, 1988.

[7] A. D. Hills. Digital fly-by-wire experience. In *Proceedings AGARD Lecture Series*, number 143, October 1985.

[8] Allen M. Johnson and Miroslaw Malek. Survey of software tools for evaluating reliability availability, and serviceability. *ACM Computing Surveys*, 20(4):227–269, December 1988.

[9] K.H. Kim and Howard O. Welch. Distributed execution of recovery blocks: An approach for uniform treatment of hardware and software faults in real-time applications. *IEEE Transactions on Computers*, 38(5):626–636, May 1989.

[10] Jaynarayan H. Lala and Linda S. Alger. Hardware and software fault tolerance: A unified architectural approach. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-18*, pages 240–245, June 1988.

[11] Jean-Claude Laprie. Dependability evaluation of software systems in operation. *IEEE Transactions on Software Engineering*, SE-10(6):701–714, November 1984.

[12] Jean-Claude Laprie, Jean Arlat, Christian Beounes, and Karama Kanoun. Definition and Analysis of Hardware- and Software- Fault-Tolerant Architectures. *IEEE Computer*, pages 39–51, July 1990.

[13] Jean-Claude Laprie and Karama Kanoun. X-ware reliability and availability modeling. *IEEE Transactions on Software Engineering*, pages 130–147, February, 1992.

[14] Michael R. Lyu and Yu-Tao He. Improving the N-version programming process through the evolution of a design paradigm. *IEEE Transactions on Reliability*, June 1993.

[15] C. V. Ramamoorthy, Y. Mok, F. Bastani, G. Chin, , and K. Suzuki. Application of a methodology for the development and validation of reliable process control software. *IEEE Transactions on Software Engineering*, SE-7(6):537–555, November 1981.

[16] Brian Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2):220–232, June 1975.

[17] R. Sahner and K. S. Trivedi. Reliability modeling using SHARPE. *IEEE Transactions on Reliability*, R-36(2):186–193, June 1987.

[18] R. Keith Scott, James W. Gault, and David F. McAllister. Fault-tolerant software reliability modeling. *IEEE Transactions on Software Engineering*, SE-13(5):582–592, May 1987.

[19] Kang G. Shin and Yann-Hang Lee. Evaluation of error recovery blocks used for co-operating processes. *IEEE Transactions on Software Engineering*, SE-10(6):692–700, November 1984.

[20] George. E. Stark. Dependability evaluation of integrated hardware/software systems. *IEEE Transactions on Reliability*, pages 440–444, October 1987.

[21] Pascal Traverse. Airbus and ATR system architecture and specification. In U. Voges, editor, *Software Diversity in Computerized Control Systems*, pages 95–104. Springer-Verlag, June 1986.

[22] Udo Voges. Use of diversity in experimental reactor safety systems. In U. Voges, editor, *Software Diversity in Computerized Control Systems*, pages 29–49. Springer-Verlag, 1988.

[23] L. J. Yount. Architectural solutions to safety problems of digital flight-critical systems for commercial transports. In *Proceedings AIAA/IEEE Digital Avionics Systems Conference*, pages 1–8, December 1984.