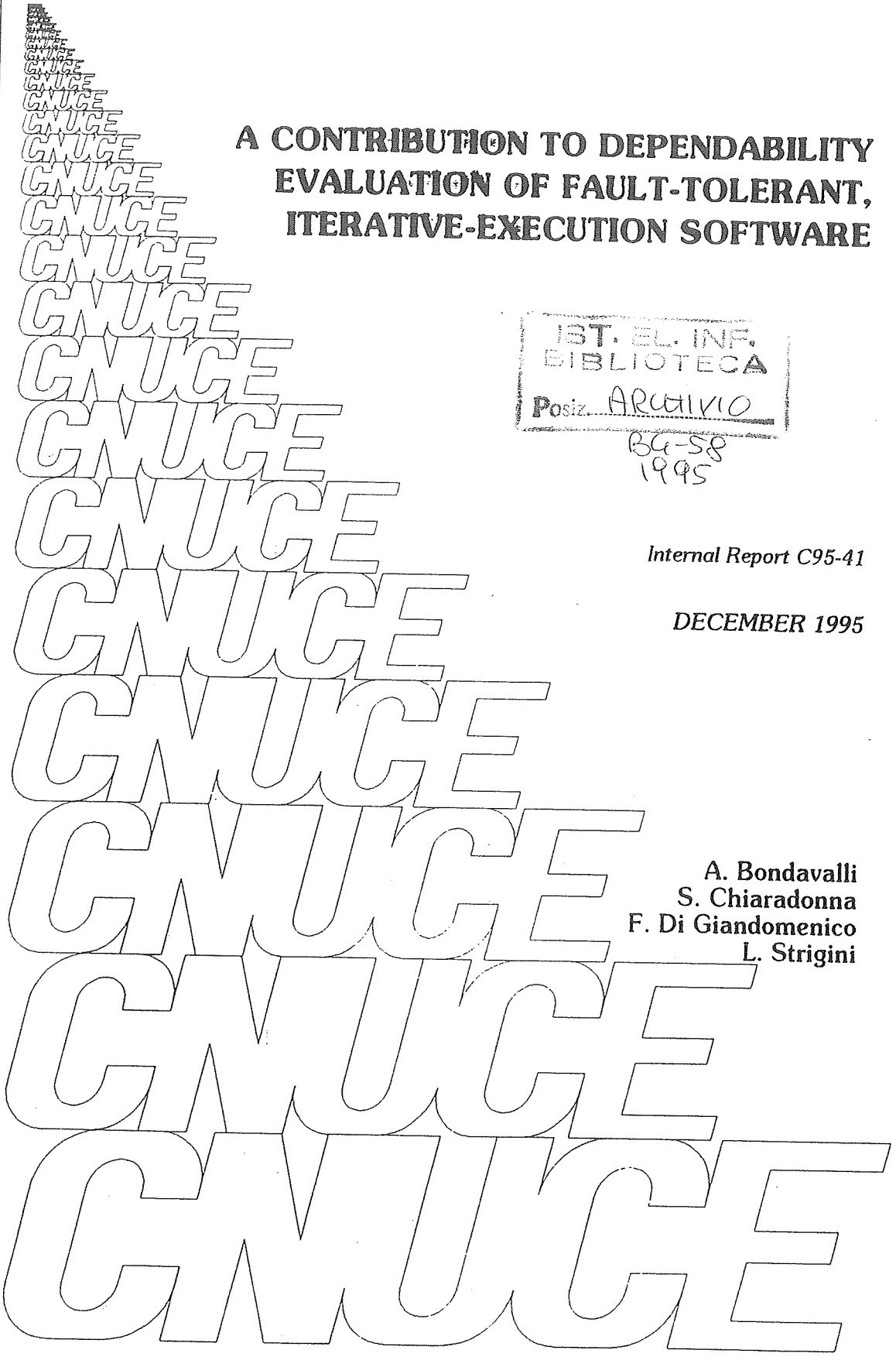
B4-58



A CONTRIBUTION TO DEPENDABILITY EVALUATION OF FAULT-TOLERANT, ITERATIVE-EXECUTION SOFTWARE

IST. EL. INF.
BIBLIOTECA
Posiz. ARCHIVIO
B4-58
1995

Internal Report C95-41

DECEMBER 1995

A. Bondavalli
S. Chiaradonna
F. Di Giandomenico
L. Strigini

A Contribution to Dependability Evaluation of Fault-Tolerant, Iterative Execution Software

Andrea Bondavalli-, Silvano Chiaradonna-, Felicita Di Giandomenico- and Lorenzo Strigini-

1 CNUCE-CNR, Via S. Maria 36, 56126 Pisa, Italy
2 1EI-CNR, Via S. Maria 46, 56126 Pisa, Italy
3 Centre for Software Reliability, City University, Northampton Square, London ECI V OHB, UK

Abstract

We consider the dependability of fault tolerant software executed iteratively, as e.g. in process control applications. We first recall the models usually adopted for evaluating the probability of mission survival (reliability at a certain time) and performability, and show the results obtained by applying these models to the adaptive scheme for software fault-tolerance SCOP, "Self-Configuring Optimal Programming" and to the more popular schemes, recovery blocks and multiple version programming. Then we explore the consequences on dependability figures of two characteristics of iterative software: a) system failure must be defined in terms of the behaviour of the software over successive iterations, because the controlled system can usually tolerate short bursts of errors of the control software; b) the probabilistic correlation between successive executions of the software is an important factor in determining the failure behaviour of the software. Positive correlation is to be expected for various reasons, not least the fact that the input values representing physical variables of the controlled system evolve along a "trajectory" in the input space of the software. We present models accounting for these characteristics and evaluate the effects of different distributions of the correlation between successive executions of the software and the sensitivity of the dependability figures to our model parameters.

In control systems, the inputs often follow a trajectory of small steps in the input space, representing points in the state space of the controlled object. Experiments have shown contiguous failure regions in the program input space, i.e., connected subsets of the input space such that all the individual points in them cause the program to fail. Theoretical justifications have also been advanced for claiming that this clustering of failures is generally to be expected [5]. This implies that the inputs which originate failures of the software are very rarely isolated events but more likely grouped in *clusters* [1], [6]. A software variant would produce bursts of errors when the input trajectory intersects a "fault region" for that variant. So, an error at one execution would be an indication of a high probability (i.e., higher than the marginal probability for that variant) of an error at the next execution.

Causes for positive correlation can be found in other types of systems as well: periods of peak load in time-shared computers or in communication links could lead, through unusual timing conditions, to a high probability of errors in all the executions that take place during the peak. Last, issues of imperfect recovery (state corruption) and interactions with hardware faults further complicate the problem. One consequence of this likely positive correlation is that the mission survival probabilities derived from the independence assumption are probably very pessimistic. To show this, we can point out that, with independence, and if we call p_f the probability of failure at an execution, the probability of not failing over a mission of n executions is $(1-p_f)^n$; but, if the failures at all executions were completely correlated (either no execution fails or all fail), the same probability would become $(1-p_f)$, no matter how long the mission is. If we also consider, in our model of system failure, bursts of software failures to be more dangerous than single failures, it is clear that predicting the distribution of bursts is trivial with the independence assumption, but obviously optimistic: in reality, once a first failure happens, a burst is much more likely to follow than predicted by the independence assumption.

3 "White box" models with independence between successive executions

In this section, we describe the "white box" modelling approach, as applied to the SCOP scheme with three variants, and recall the solutions of similar models for the NVP and RB schemes.

3.1 Basic assumptions

The assumptions we make in this model are quite similar to those of several other authors [2], [20], [28], [29]. These assumptions include:

1. the behaviour of each variant at an iteration is statistically independent of its behaviour at other iterations;
2. all the outcomes (of an execution) which include errors of subcomponents (variants and adjudicators) can be subdivided into two classes: those where there are *undetectable* errors (some variants fail together and produce consistent results, an adjudicator fails and decides that an incorrect result from a variant is correct); and those where only *detectable* errors occur; in this latter case, the errors of the subcomponents are statistically independent events;
3. "compensation" among errors never happens: e.g., if a majority of erroneous results exists, it never happens that the adjudicator errs by choosing a correct result instead;
4. a certain degree of symmetry is assumed in the probabilistic behaviour of the system: the probabilities of most events are assumed invariant under permutation among the variants; this has obvious mathematical convenience;
5. the adjudicator is specified to see correct results as consistent (that is, a failure by the adjudicator to see them as consistent is considered as an error of the adjudicator);
6. the watchdog timer is assumed never to fail.

In modelling RB and NVP, we have made slight changes, with respect to [2] and [28], to the definitions of the events considered and the assumptions made. However, the results obtained are comparable for all practical purposes, if parameters with similar meanings are given similar values. The detailed differences are discussed in [10]. In short, they are mostly due to our not using the notion of *fault* (a bug in the software, as opposed to the *error*, or incorrect behaviour caused by the bug) in defining the events of interest.

3.2 *The model for SCOP*

A redundant component based on the SCOP scheme with three variants includes:

- three functionally equivalent but independently developed programs (variants);
- an adjudicator which determines a consensus result from the results delivered by the variants. We assume as a delivery condition a 2-out-of-3 majority;
- a watchdog timer which detects violations of the timing constraint (executions exceeding the maximum allowed duration).

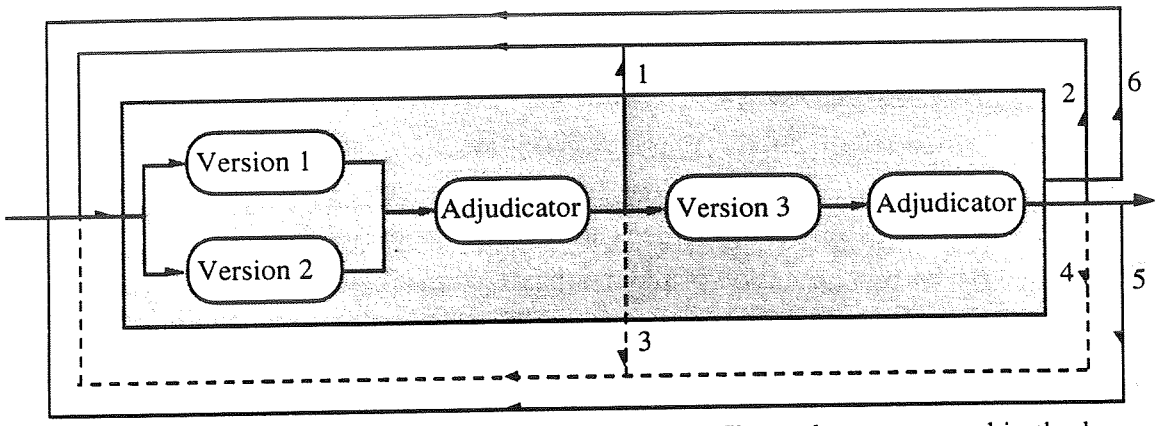


Figure 1. Possible execution paths in SCOP operation. The paths represented in the lower half of the picture imply value failure (undetected for the paths drawn as dashed lines)

Figure 1 shows the operation of the SCOP scheme. Each iteration is divided in two phases. In the first phase, variant 1 and variant 2 begin to execute at the same time. After both have completed their executions, the adjudicator compares their results. If they are consistent, it accepts them. Otherwise, the second phase begins, variant 3 executes, and then the adjudicator decides on the basis of all three results, seeking a 2-out-of-3 majority. The paths in Figure 1 correspond to the different possible outcomes:

(1): at the end of the first phase there exists a majority representing a correct computation and the output is a correct result;

(2): at the end of the first phase the result is rejected, at the end of the second phase there exists a majority representing a correct computation and the output is a correct result;

(3): at the end of the first phase an erroneous result is accepted (undetected failure);

(4): at the end of the first phase the result is rejected, at the end of the second phase an erroneous result is accepted (undetected failure);

(5): at the end of the second phase the result is rejected (detected failure);

(6): the duration of the redundant execution exceeds a specified limit (the real-time constraint) and the execution is aborted (detected failure).

3.2.1 Dependability submodel

The relevant events defined on the outcomes of one execution of the SCOP component and the notations for their probabilities are as illustrated in Table 1. The assumption of no compensation between errors has allowed us to reduce the event space to be considered.

Error Types (Events)	Probabilities
3 variants err with consistent results	q_{3v}
2 variants err with consistent results (the 3rd result is inconsistent with them, and may be correct or erroneous)	q_{2v}
The adjudicator errs and terminates the execution with phase 1, selecting an erroneous, non-majority result	q_{vd1}
The adjudicator errs and terminates the execution with phase 2, selecting an erroneous, non-majority result	q_{vd2}
A variant errs, conditioned on none of the above events happening (i.e., there are one or more detected errors; their statistical independence is assumed)	q_{iv}
The adjudicator errs, at the end of either phase 1 or phase 2, by not recognising a majority (hence causing a detected failure), conditioned on the existence of a majority	q_d

Table 1. Error types and notation for SCOP

The detailed model of one execution of the redundant component, without considering the operation of the watchdog timer, is shown in Figure 2. Table 2 shows the definitions of the states.

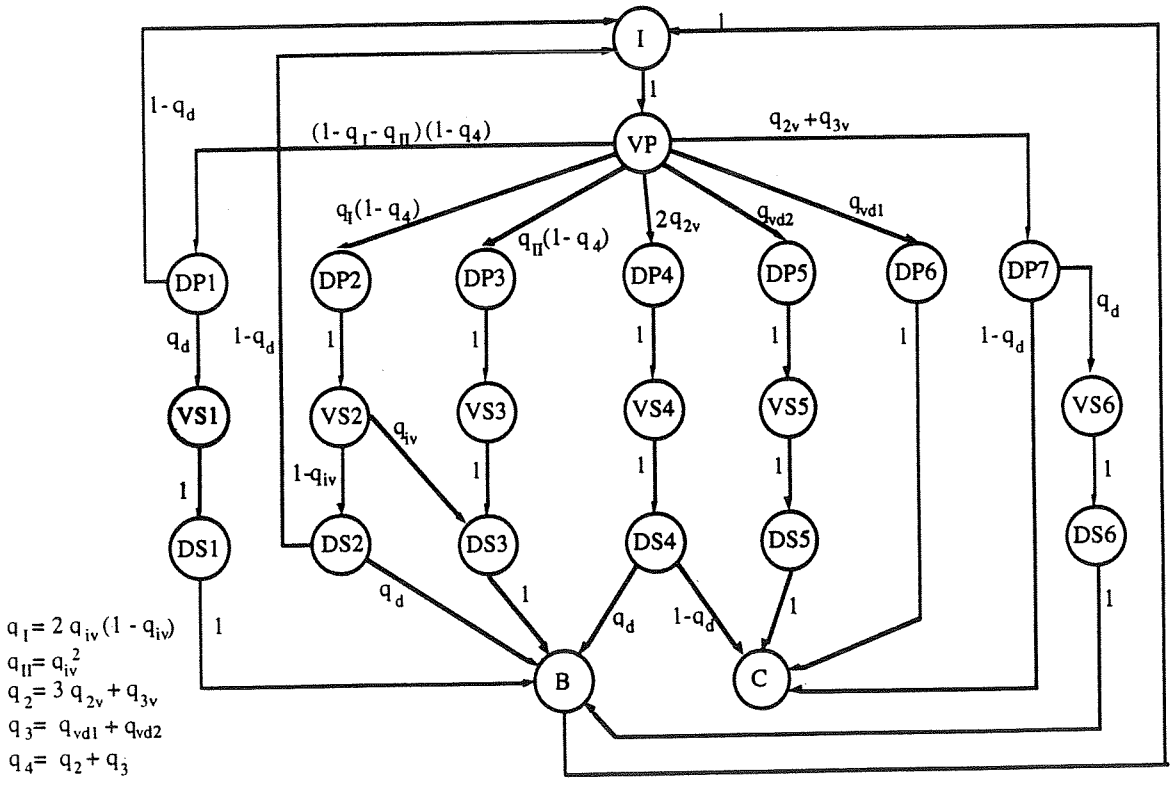


Figure 2. The dependability submodel for SCOP

States	Definition
I	initial state of an iteration
VP	execution of two variants in the first phase
$\{DP_i \mid i \in \{1,2,3,4,5,6,7\}\}$	execution of adjudicator after VP
$\{VS_i \mid i \in \{1,2,3,4,5,6\}\}$	execution of one variant in the second phase
$\{DS_i \mid i \in \{1,2,3,4,5,6\}\}$	execution of adjudicator after VS_i
B	detected failure (caused by a detected value error)
C	undetected failure (caused by an undetected value error)

Table 2. Definition of states for the SCOP dependability model

The graph is somewhat complex, in order to represent clearly all the possible paths of execution, showing how certain executions terminate with the first phase, while others go on with the execution of the third variant and a new adjudication.

Most of our parameters are the unconditional probabilities of sets of outcomes of the whole redundant execution (including the executions of both the variants and the adjudicator): hence, some of the arcs exiting VP are labelled with these probabilities, and are followed by arcs, as e.g. from DP_5 to VS_5, labelled with a probability equal to 1. We briefly describe (out of numerical sequence to simplify the explanation) the meanings of the arcs from VP.

DP_7 at the end of phase 1, variants 1 and 2 are both erroneous and in agreement (this includes the case of a consistent error among all 3 variants, an event which has a clear physical meaning, though it can only be observed if the adjudicator fails to recognise the agreement in phase 1);

DP_5 variants 1 and 2 are correct and thus in agreement, variant 3 fails, and the adjudicator fails in such a way as not to recognise the agreement in phase 1, and to choose the result of variant 3 as a majority;

DP_6 one among variants 1 and 2 fails, the other does not, but the adjudicator fails to notice the disagreement and chooses the wrong result as correct;

DP_4 at the end of phase 1 there is no majority (either one variant is in error, or both are, but with inconsistent results), and then variant 3 also errs, forming an erroneous majority with either variant 1 or variant 2. Neither DP_5 nor DP_6 occurs. This leads to either an undetected or a detected failure, depending on whether the adjudicator recognises this majority or fails to recognise it;

DP_3 none of the above events occurs, and variants 1 and 2 produce inconsistent, erroneous results: no majority exists; the adjudicator recognises the lack of a majority;

DP_2 none of the above events occurs; one among variants 1 and 2 produces an erroneous result; depending on whether variant 3 produces a correct result, phase 2 terminates with a correct majority (DS_2) or not (DS_3);

DP_1 none of the above events occurs; variants 1 and 2 are correct.

In states DP_1, DS_2, DS_4, DP_7 a majority exists. The adjudicator may fail to recognise it, with probability q_d, and produce a detected failure. It has been plausibly assumed that if the adjudicator fails in this fashion at the end of phase 1, it will consistently fail at the end of phase 2: hence the probabilities equal to 1 on the arcs downstream of DS_1 and DS_6. To simplify the expression of the solution, we define a set of intermediate parameters as shown in the bottom left corner of Figure 2. We call the probabilities of an undetected and of a detected failure, without the watchdog timer (that is, due solely to the *values* of the results of the subcomponents), p_{cv} and p_{bv}, respectively. From the state transition diagram, it follows that:

$p_{cv} = q_3 + q_2(1-q_d)$ and

$p_{bv} = q_2 q_d + q_{II}(1-q_4) + q_I(1-q_4)(q_{iv} + (1-q_{iv})q_d) + (1-q_I-q_{II})(1-q_4)q_d =$

$= (1-q_4)q_d + (3q_{iv}^2(1-q_{iv}) + q_{iv}^3)(1-q_4)(1-q_d) + q_2 q_d$

3.2.2 Timing submodel

The model described in 3.2.1 does not account for the possibility of a time-out. In principle, all the events defined on this model may have a non-null intersection with the event "the execution exceeds its allotted time and is aborted by the watchdog timer". As we call this event a subset of "detected failure" (i.e., from the system point of view, "benign failure"), the implication is that considering time-outs increases (or, in the limit, leaves unaltered) the probability of benign failures and decreases the probabilities of successes and catastrophic failures.

To estimate this probability experimentally, one would run statistical testing on the software, and estimate the tail of the distribution of execution times. One could also measure execution times of the individual subcomponents, but would then need some assumption about the correlations between these distributions and between them and value failures. If one considered dynamic scheduling, as in [11], [26-29], the distribution of execution times would also be needed for predicting how many executions take place in a given time. We will show the effects of this different approach in Section 4.2.

A simple, though unrealistic model ([26], [28], [29]) assumes exponentially, independently distributed execution times for the subcomponents. In addition, the duration of an execution (without the watchdog timer) is assumed to be independent of its producing a value error or not. We applied this model to SCOP in [11], and will use its solution here as an example. To quantify the effect of time-outs, [28] considers the event "time-out" as a subset of the event "the execution produces a correct value". We, on the contrary, use the above-mentioned assumption of independence between the execution times of the subcomponents and their error behaviour, that is, time-outs have the same relative frequency for executions that would produce a correct value as for those producing a detected or undetected value failure.

3.2.3 Probability of completing a mission and performability

Let p_s, p_b, and $p_c=1-p_s-p_b$ be respectively the probabilities of success, benign failure and catastrophic failure per execution. Then, the probability of completing a mission consisting of a number n of iterations is that of a series of n executions without catastrophic failure, $(1 - p_c)^n$.

The value $E[M_n]$ of the expected total reward is: $E[M_n] = n \frac{p_s}{p_s + p_b} (1 - p_c)^n$ which is the product of the probability of completing a mission without a catastrophic failure $(1 - p_c)^n$ and the expected number of successes in n iterations without a catastrophic failures $n \frac{p_s}{p_s + p_b}$. So, when static scheduling is adopted, the probability of completing the mission has a very strong influence on the performability measure; the differences between different schemes are driven only by the probability of completing a mission (the number of executions being constant) while the reward structure influences only the slope of the performability functions.

3.3 Models for NVP and RB

We do not describe the detailed models we use [10]. The definitions of the relevant events considered are listed in the following tables. The expressions for p_{bv} and p_{cv} for NVP only differ from those obtained for SCOP in having $3q_{vd}$ instead of q_3. When evaluating these expressions, we have rather arbitrarily considered both q_{vd2} and q_{vd1} as similar events of "common-mode failures among the adjudicator and one variant", and accordingly assigned them the probabilities q_{vd} and $2q_{vd}$, respectively: with the values we have later assigned to these parameters, this arbitrary assignment has a negligible effect on the results.

Error Types	Probabilities
3 variants err with consistent results	q_{3v}
2 variants err with consistent results (the 3rd result is inconsistent with them, and may be correct or erroneous)	q_{2v}
The adjudicator errs by selecting an erroneous, non-majority result	q_{vd}
A variant errs, conditioned on none of the above events happening (i.e., there are one or more *detected* errors; their statistical independence is assumed)	q_{iv}
The adjudicator errs by not recognising a majority (hence causing a detected failure), conditioned on the existence of a majority	q_d

Table 3. Error types and notation for NVP

Error Types	Probabilities
The secondary variant errs and the adjudicator accepts its result, conditional on the secondary being executed	q_{sa}
Common mode error of P and AT, or P, S and AT (primary variant errs and the adjudicator accepts its result)	q_{pa}, q_{psa}
Common mode error of P and S (primary and secondary variant err with consistent results)	q_{ps}
Detectable error of the primary or secondary alternate (assumed independent)	q_p, q_s
Error of the acceptance test AT causing it to reject a result, given the result is correct	q_a

Table 4. Error types and notation for RB

	RB	NVP
probability of undetected failure without watchdog timer	$p_p\, q_a\, q_{sa} + q_p\, q_{sa} + q_{pa} + q_{psa}$ **with** $p_p = 1-q_p-q_{ps}-q_{pa}-q_{psa}$	$3\, q_{vd} + q_2\,(1 - q_d)$
probability of detected failure without watchdog timer	$p_p\, q_a\, (p_s + q_s) + q_p\, p_s\, q_a + q_p\, q_s + q_{ps}$ **with** $p_s = 1-q_s-q_{sa}$	$(1 - q_1)\,(1 - q_2)\, q_d + q_1\,(1 - q_2) + q_2\, q_d - 3\, q_{vd}\,[(1 - q_1)\, q_d + q_1]$ **with** $q_1 = 3\, q_{iv}^2\,(1 - q_{iv}) + q_{iv}^3$

Table 5. Solutions of the NVP and RB models (for one execution)

The NVP and SCOP schemes behave in exactly the same manner with regard to failures of the variants: a SCOP execution scheme using 2-out-of-3 majority guarantees that, with a correct adjudicator, exactly the same outcome will be produced as that produced by the same variants when organised in an NVP scheme. The differences may lay in the error behaviour of the adjudicator, and the different probabilities of the outcomes involving such errors. These probabilities are exceedingly difficult to estimate, but in the next section we plausibly assume them to be low compared to those of one or two variants failing. As for the timing of executions, for NVP the model has to represent the fact that adjudication only takes place after the slowest variant has terminated, but no second phase of execution is ever needed. In the case of RB, only the primary variant executes in the first phase, so there is no waiting for a slower variant before running the acceptance test, but a second phase may follow, as in SCOP. These differences imply that the same subcomponents, when combined in different schemes, would give different probabilities of time-out.

4 Results of the "white box" model with independence between successive executions

4.1 Solution of the basic model

We now show the results obtained from the models described above. Our choice of parameters is, by necessity, arbitrary, but it will suffice to show the implications of these models and especially of assuming independence between successive executions.

Table 6 lists the values of the parameters for the "white box" dependability submodel (without timing), used to produce the plots of the solutions of the models (Figures 3 and 4).

RB	NVP	SCOP
$q_p = q_s = q_a$: from 0 to $5\ 10^{-5}$	q_{iv} : from 0 to $5\ 10^{-5}$	q_{iv} : from 0 to $5\ 10^{-5}$
$q_{psa} = 10^{-10}$	$q_{3v} = 10^{-10}$	$q_{3v} = 10^{-10}$
$q_{ps} = 10^2\ q_p^2$	$q_{2v} = 10^2\ q_{iv}^2$	$q_{2v} = 10^2\ q_{iv}^2$
$q_{pa} = 10^2\ q_p^2$	$q_{vd} = 10^{-10}$	$q_{vd1} = 2\ 10^{-10}$
$q_{sa} = 10^2\ q_p^2$		$q_{vd2} = 10^{-10}$
	$q_d = 10^{-9}$	$q_d = 10^{-9}$

Table 6. Values of the dependability parameters used for Figures 3 and 4

Our choice of parameter values reflects some plausible assumptions. We set q_{2v} proportional to q_{iv} so that, in our plots, moving towards the right along the horizontal axis will represent increasing probabilities of value failures of the variants (with a particular, fixed relationship between the probabilities of the different kinds of failure). In defining the proportion between benign failures (a function of q_{iv}) and catastrophic failures (a function of q_{2v}) we have plausibly chosen a probability of coincident failure of two variants, q_{2v}, markedly higher than would be implied by an assumption of independent failures. The probabilities of coincident errors of two or three subcomponents are higher than those of independent errors of the corresponding subcomponents (reflecting a positive correlation among subcomponent failures). The adjudicator (acceptance test for RB) has a much lower error probability than the variants in an NVP or SCOP system, and a comparable probability for RB.

The probability of time-out, p_{bt}, has been determined using the exponential model mentioned in Section 3.2.2, with the values shown in Table 7 for the timing parameters. For RB and SCOP, it is assumed that the variant chosen to be only executed after an error is detected is the slowest one: p_{bt} thus increases with the probability of value errors. NVP has a higher, constant value of p_{bt} because it unconditionally executes the slowest of the variants. Table 8 shows the effects of varying q_{iv} on p_{bt} and on μ, the average duration of an execution.

RB	$\lambda_p = \lambda_a = 1/2, \lambda_s = 1/7$ $\tau = 25$ msecs
NVP / SCOP	$\lambda_1 = \lambda_2 = 1/2, \lambda_3 = 1/7, \lambda_d = 5$ $\tau = 25$ msecs

Table 7. Values of the timing parameters ($msec^{-1}$) used for Figures 3 and 4

	q_{iv}	0	$5\ 10^{-5}$
SCOP	p_{bt}	$8.281\ 10^{-6}$	$1.317\ 10^{-5}$
	μ	3.2000	3.2007
NVP	p_{bt}	0.029	0.029
	μ	7.7613	7.7613
RB	p_{bt}	$5.031\ 10^{-5}$	$5.802\ 10^{-5}$
	μ	3.9999	4.0007

Table 8. Probability of time-out, p_{bt}, and average execution times (msec), for the extremes of the range of q_{iv} in Figures 3 and 4

In all the figures, the mission duration is 10^6 iterations, a reasonable order of magnitude for, e.g., a workday in non-continuous process factory operation, or in office work, and a flight for civil avionics (with an execution rate of a few tens of iterations per second).

With these parameter values, Figure 3 shows, for each scheme, the probability of mission failure, i.e., of having at least one undetected error in a mission. As shown, this choice of parameter ranges extends to unrealistically low probabilities of success of a mission. The fact is that, due to the assumed independence between successive iterations, these are approximately exponential curves. The curves cross the vertical axis at a value greater than 0 (which is not apparent in our plots), representing the failures due to coincident errors of three variants and/or of the adjudication. The difference between the curves for NVP and SCOP is only due to their different values of p_{bt}, and would decrease if the execution times of the fastest and of the slowest variant were less different.

Figure 4 shows the performability measures. As explained in Section 3.2.3, these amount to the product of the probability of surviving the mission times the expected number of successes per completed mission.

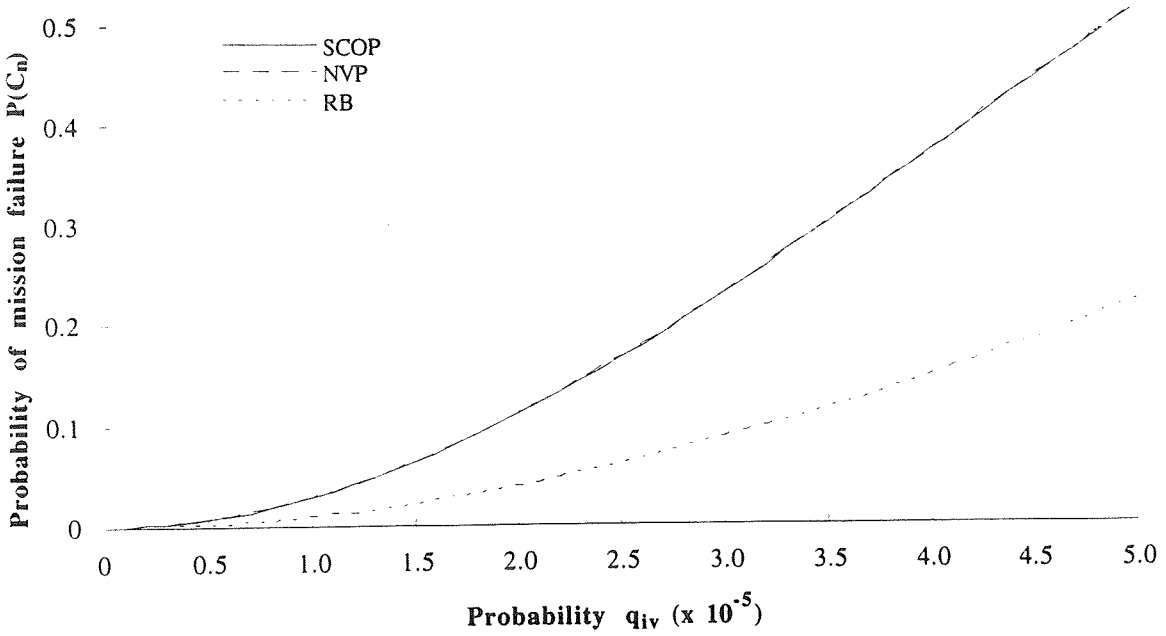


Figure 3. Probability of mission failure for RB, NVP and SCOP, in the case of independence between successive executions, as the failure probability per execution increases, with q_{2v} proportional to q_{iv}

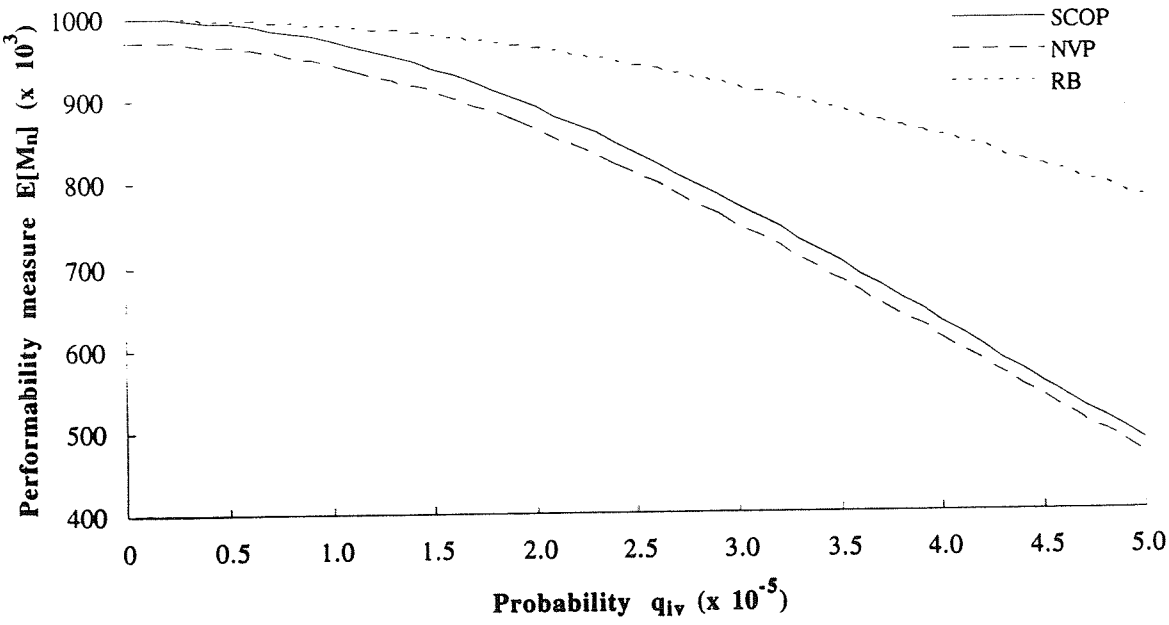


Figure 4. Performability comparison of RB, NVP and SCOP, in the case of independence between successive executions, as the failure probability per execution increases, with q_{2v} proportional to q_{iv}

4.2 *Effects of dynamic scheduling on performability*

We show, by way of comparison, how measures would change if one assumed, as in [11], [28], [29], that each iteration is triggered by the termination of the previous one and measured performability over a mission of fixed duration.

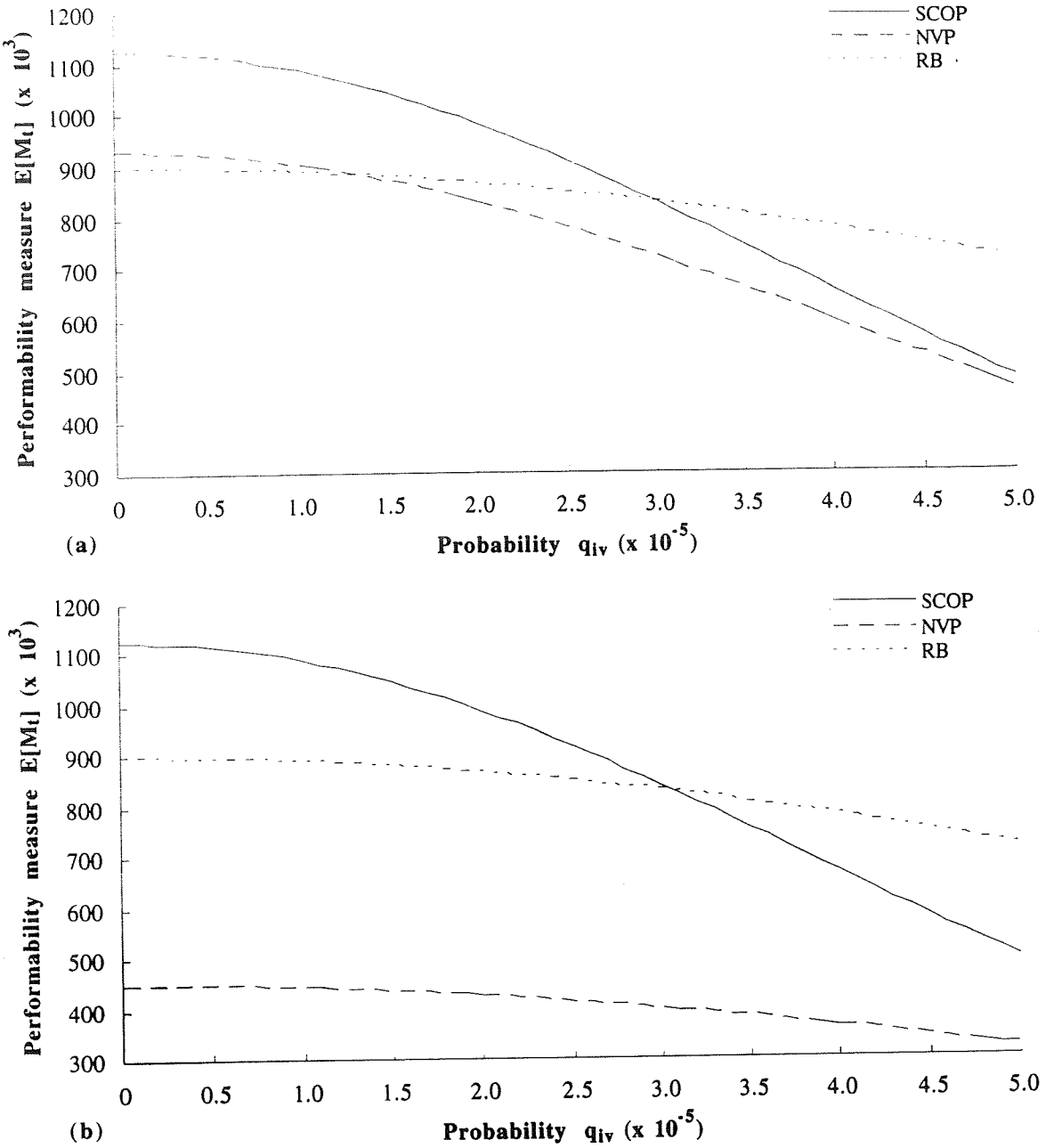


Figure 5. Performability comparisons of RB, NVP and SCOP with dynamic scheduling, as the failure probability per execution increases with q_{2v} proportional to q_{iv}. The execution rates of the variants are equal in (a) and strongly different in (b), about one tenth of those in Table 7

The plots in Figure 5 show this measure (which we call $E[M_t]$ as opposed to $E[M_n]$, measured over a fixed number of iterations), which we derived in [11]. The dynamic scheduling implies that many more executions can be packed in the same duration. To show numbers of executions (and hence performability values) in the order of those in Figure 4, we have made the execution rates of the variants one tenth of those used for Figures 3 and 4. The plots appear

very different from those of Figure 4, because different schemes imply different expected numbers of executions per mission. RB has an advantage over SCOP, and SCOP over NVP, as in SCOP the adjudication must always wait for the second-slowest variant and in NVP for the slowest one. However, the adjudication has been assumed to be faster for SCOP and NVP than for RB, and this explains the high performability shown for SCOP for abscissae close to zero. The number of executions is also affected by the fact that an execution may, in RB and SCOP, last for two phases instead of one; but this may only affect the small fraction of executions where at least one error takes place, so that the number of executions per mission can be considered practically constant, for a given scheme, once the distributions of execution times for the subcomponents are assigned. Towards the left in these figures, as the probability of undetected failure approaches zero, the utility of a mission tends to the mean number of executions in a mission, decreased by the fraction of detected failures. As one moves to the right, the probability of undetected failures, and hence missions with zero utility, increases. SCOP and RB, packing more executions in the same mission time, suffer more than NVP.

The differences among Figures 5.a and 5.b are explained by considering that differences in the mean execution times of the variants increase the performance disadvantage of NVP with respect to SCOP and RB. With our parameters, while the number of executions per mission is maximum in SCOP, which explains SCOP having the highest $E[M_t]$ for the lower values of the abscissae, the slope of the curves is smallest for RB, as its probability of undetected failure per execution is roughly one third of that of the others. An interesting consideration is that in the left-hand part of these plots, SCOP yields the best performability values, while its probability of surviving a mission is the worst.

4.3 Effects of different reward models

These performability measures are based on a specific, very simple reward model which of course has limited applicability. We illustrate in this section the effects of choosing different reward models, in the context of static scheduling, limiting ourselves to *additive* models of the form:

$$M_n = \begin{cases} \text{"no. of successes"} \cdot G_S - \text{"no. of detected failures"} \cdot C_B, & \text{if mission success} \\ \text{"no. of successes"} \cdot G_S - \text{"no. of detected failures"} \cdot C_B - C_C, & \text{if mission failure} \end{cases}$$

where G_S, C_B and C_C represent the gain associated to a successful iteration, the cost associated to a detected failure and the cost associated to the mission failure respectively. Of course, depending on the application context, much more complex reward structures may apply, in which the pattern of failures over time is also a factor. In addition, in most practical contexts, many categories of mission failures with different costs would be considered (as well as multiple categories of benign failures).

With the above reward structure, the expected reward $E[M_n]$ over a mission of n iterations is:

$$E[M_n] = (p_s G_S - p_b C_B - p_c C_C)\frac{1-(1-p_c)^n}{p_c}$$

Figure 6 shows the plots of the performability for RB and SCOP for three different sets of values of the parameters G_S, C_B and C_C, as indicated on the left corner of the figure. For example, with reference to the scenarios briefly described in Section 2.2, case a), where C_B and C_C are zero, would model the space probe application (after an initial investment, only the gain from successes is considered). Case b) and case c) could model a continuous production process (e.g. of a chemical), where each iteration adds some small amount of product and a mission failure means damaging machinery. In case b), benign failures are taken as delaying production, without damage, and in case c) they do damage some amount of product. The abscissae represent, as in Figures 3 and 4, q_{iv} under the hypothesis that q_{2v} is a function of q_{iv}, thus representing a combined measure for inconsistent and consistent failures. For case c), the performability curves start low at the abscissa $q_{iv}=0$, because of the cost of benign failures due to time-outs and adjudication failures; this is more apparent for RB which has a higher probability of time-outs than SCOP.

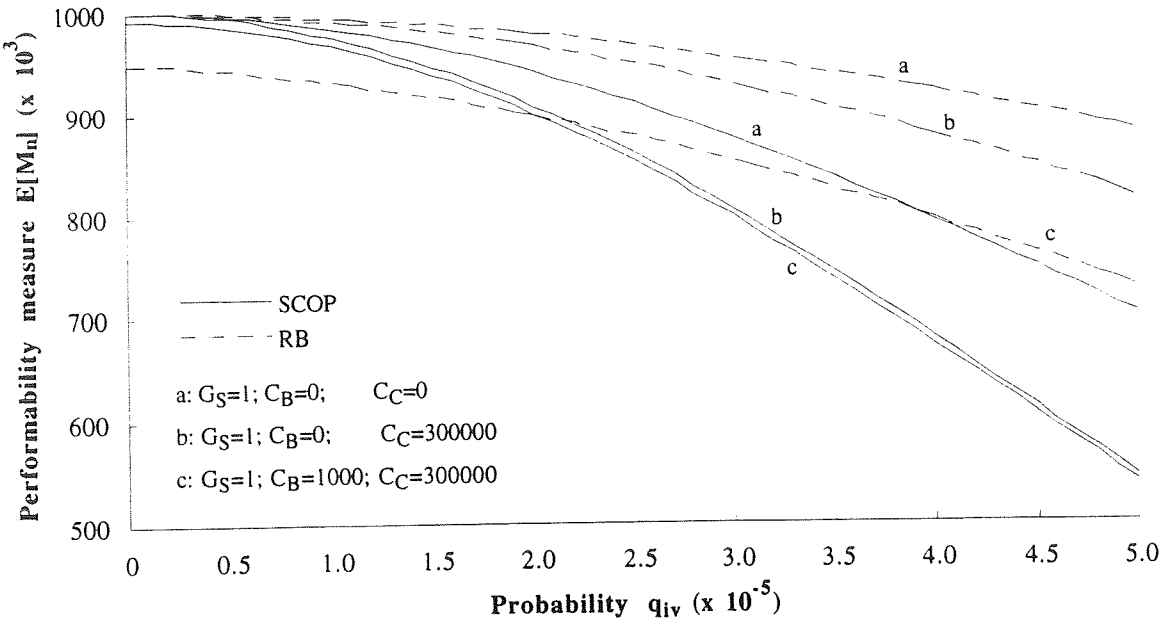


Figure 6. Performability comparison of RB and SCOP with a different reward model, in the case of independence between successive executions, as the failure probability per execution increases with q_{2v} proportional to q_{iv}

Of course, this selection of parameter values is still just a sample of the reward structures that may apply in different real-world situations: the figure shows how heavily the reward structure chosen affects the relative "goodness" of the different schemes.

5 Modelling sequences of executions with a "black box" model of the fault-tolerant component

5.1 Correlated executions and new failure models

In this section, we drop many details of the "white box" model, but attempt to model more realistically the behaviour of a software component over a sequence of executions. The fault-tolerant component is now modelled as a black box, with probabilities p_S, p_b, and $p_c=1-p_S-p_b$ of, respectively, success, benign failure and catastrophic failure per execution. These (marginal) probabilities are constant during the mission, but the probability of an outcome at a specific execution *conditional* on the outcome of the previous execution is now allowed to differ from the corresponding marginal probability. For instance, we allow:

$p_S \neq$ p(success at the i-th execution | benign failure at the (i-1)-th execution).

This models the fact that, in reality, the inputs to the software evolve along a *trajectory* through the input space, and this trajectory "crosses" the failure regions: once it has entered a failure region (i.e., the software has failed once), another failure is more likely to happen than it would be after a failure-free iteration. Of course, the likely number of failures in a series depends on the size of the failure region (measured along the trajectory), and the "speed" of the trajectory (i.e., the statistics of the distance between two subsequent inputs). In the rest of this discussion, we shall not consider these two factors individually, but only their combined effect, i.e., the distribution of the number of iterations during one crossing of a failure region. This distribution would in practice be either measured experimentally (in which case, however, the experiment would possibly allow one to measure directly the dependability figures of interest), or guessed by the evaluator using knowledge about the input space and the "speed" with which a trajectory crosses its different regions. For the rest of this discussion, our choice

of only considering the number of steps inside a failure region can be simply interpreted in the sense that we use the length of a step in the trajectory (distance between two inputs) as the unit of measurement for distances in the inputs space. Of course, reasoning in terms of numbers of steps is also applicable for applications for which an intuitively sound definition of a distance function over the input space, or of a trajectory through it, are not available.

In addition, in this section we will consider the possibility that long bursts of benign failures (as may be expected with a high positive correlation between successive executions) may cause mission failure. We will model this by the assumption that, although the controlled system can survive an individual benign failure of the control computer, any series of n_c or more benign failures in a row (where the parameter n_c is a characteristic of the controlled system representing its "*robustness*") will cause the mission to fail. This is a common characteristic of continuous-control systems. In certain cases, the controlled system has enough physical inertia that an individual erroneous output from the control system will not cause the controlled system to move into a prohibited state. For such systems, the probability of an isolated failure with catastrophic consequences is zero. In other cases, although some of the failures of the control system are immediately catastrophic, others (e.g., those where the control system internally detects its own failure and outputs a safe value to the controlled system) are not, and only if they are repeated the resulting lack of active control may cause the controlled system to drift into a dangerous state. Of course, in either case the assumption that any sequence of up to n_c-1 failures will be tolerated, and all longer sequences will be catastrophic, is still a simplification of reality, yet more realistic than assuming that a controlled system can tolerate any arbitrary series of benign failures.

To demonstrate the effects of different sets of modelling assumptions, we first show the effects of assuming that repeated benign failures may cause a catastrophic failure, while still assuming independent outcomes for successive iterations, and then we abandon the independence assumption as well.

5.2 "Black box" model

We derived our model of correlation between successive iterations from those in [12], [30]. We define two kinds of failure events for a software component:

i) point failure: which happens when the input trajectory enters a failure region,

ii) serial failure: a number of consecutive failures, happening with probability 1 after the occurrence of a point failure, i.e., after the input trajectory enters a failure region. The number of serial failures subsequent to any point failure is a random variable, and we can choose its distribution to represent what we believe to be the stochastic characteristics of the input trajectories and the failure regions.

In [12], this model was applied to the primary variant in a recovery block scheme. Correlation between the successive failures of the alternate variant is not considered, since at the first (point) failure of the alternate the whole scheme fails and execution stops. From these modelling assumptions a simple Markov chain with discrete time is developed allowing an analytical evaluation of the reliability (MTTF) of the RB. [30] analysed the different forms of correlation of the RB structure, including correlation among the different alternates and among alternates and the acceptance test on the same inputs. To model the correlation between successive inputs, [30] used the same assumptions as [12], including the same event set, and used a SRN (Stochastic Reward Nets) model to evaluate the effects of input correlation on the MTTF.

With respect to this model, we introduce two variations:

- any failure of the software component (either "point" or "serial") may be either benign or catastrophic (immediate mission failure) for the controlled system;
- furthermore, all sequences of at least n_c benign failures produce a mission failure for the controlled system.

The outcomes of an individual iteration may thus be: i) *success*, i.e., the delivery of a correct result, ii) a *benign failure* of the program, i.e., an output that is not correct but does not, by itself, cause the entire mission to fail, or iii) a *catastrophic failure*, i.e., an output that causes

the immediate failure of the entire mission. The statistics of sequences of these events are modelled on the basis of the considerations below.

Failure Regions: Experiments [1], [5] and considerations about software structure indicate that failure points (input values on which the software fails) are grouped in continuous failure regions. "A priori", we assume that the probability that an input belongs to a failure region is the same for every input; one should assume a different distribution, when modelling a specific system, if it were known, for instance, that some parts of the input space are more prone to failures than others. Once the trajectory enters a failure region, though, the probability that the next input will cause a failure (i.e., that it will still belong to the failure region) increases. In [5] some two-dimensional views of fault regions (blob defects) are shown for a specific program, and a number of factors affecting the shapes of the faults were identified. Since there is no evidence for choosing particular shapes on a general basis, we have chosen shapes which were simple to model, and evaluated bounds which do not depend on the specific assumptions which are difficult to verify.

Input Sequence: The inputs form a "trajectory", that is, every input value is close to the previous one. We have a random or deterministic walk trajectory, with a step length that must be small compared to the size of the input space (if the step length becomes comparable to the size -in each dimension- of the input space then, as shown in [6], we obtain a uniform distribution of the inputs and independence between successive inputs). Many different types of trajectories may be considered. Examples are: 1) the next input is obtained from the previous one by modifying the values on each dimension of a random small quantity; 2) (a subcase of 1) a "forward-biased" trajectory: passing from one input to the next the direction may only change slightly; 3) (a subcase of 2) a trajectory of points on a straight line, at a random, small distance from each other.

Our model is appropriate for a situation in which the following assumptions are true:

1) a single success before the n_c-th consecutive failure will bring the system back to a stable state, i.e. the memory of the previous failure sequence is immediately lost;
2) the trajectory of the input sequence is "forward-biased": passing from one input to the next the direction may vary with a small angle;
3) the failure regions are convex.

The main purpose of these assumptions is to simplify the modelling without restricting too much the class of applications that can be modelled. Actually, many control applications (e.g., radar systems or navigation systems) show "forward-biased" trajectories. Assumptions 2) and 3) constrain us to trajectories that, once they have left a failure region, are unlikely to re-enter it soon. They thus allow us to consider as a constant the probability of entering a failure region since, after leaving a certain failure region, a) there is a low probability of re-entering the failure region just left within a small number of iterations and b) after an appropriate number of iterations, the probability of re-entering that failure region is equal to the probability of entering any other failure region.

We point out that we are modelling only one of the possible causes of correlation between successive failures, i.e., continuous failure regions in the input space. There are other possible causes, e.g., the fact that an error at one execution is likely to corrupt the state of the software and make an error more likely at each subsequent execution. This effect could, in some cases, be satisfactorily represented by our models: the internal state variables can be considered as inputs, and it is plausible that at least some of their values evolve along continuous trajectories. However, further study is needed to determine which models are sufficiently general (or flexible) to represent the effects of the different causes of correlation.

5.3 Mission failure from repeated benign failures (with independence between successive iterations)

We now assume that, although the controlled system can survive an individual benign failure of the control computer, any series of n_c or more benign failures in a row will cause the mission to fail.

Probability of completing a mission. Our new assumption of course decreases (all model parameters being equal) the probability of surviving a mission. A reasonably tight test of whether the probability of a mission failure due to a series of benign failures, $p_{c\text{-serial}}$, can be neglected can be obtained as follows. An upper bound on $p_{c\text{-serial}}$ is the probability that a series of iterations without catastrophic failure is followed by a success and then n_c benign failures:

$$p_{c\text{-serial}} \leq \sum_{i=0}^{n-n_c-1} (1-p_c)^i p_s p_b{}^{n_c} = p_s p_b{}^{n_c} \frac{1-(1-p_c)^{n-n_c}}{p_c}$$

The total probability of mission failure is larger than $(1-(1-p_c)^n)$, the probability of mission failure if series of benign failures are of no concern. If the upper bound on $p_{c\text{-serial}}$ is negligible in comparison with this lower bound on the probability of mission failure, then it is legitimate to neglect $p_{c\text{-serial}}$ in computing the latter.

Performability. With our hypothesised reward structure, the only effect of the increased probability of mission failure is that a smaller proportion of the missions will be completed. Thus, the expected performability will be decreased by the same amount as the probability of completing a mission.

5.4 Correlation between successive iterations, allowing mission failures from repeated benign failures

Abandoning the independence assumption, we now consider that a benign failure at an iteration of the program makes it more likely than otherwise that the program will fail at the next iteration as well. The simplest model of behaviour exhibiting this property is the three-state Markov chain in Figure 7, in which $p_{BB}>p_B$ and $p_{BS}<p_S$. We underscore that this is the simplest model of correlated failures, but not a realistic one for any one software product.

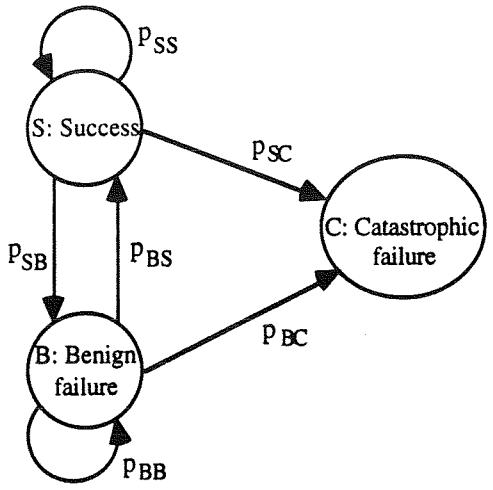


Figure 7. The model for the iterative execution of a system with correlation

Still, even this very simple model allows one to appreciate the changes with respect to assuming $p_{BB}=p_{SB}$, $p_{BS}=p_{SS}$, $p_{SC}=p_{BC}$ (i.e., independent outcomes among successive iterations, conditional on not having reached mission failure): the system has a higher marginal probability of being in the "benign failure" state, and is thus less dependable. The value of the expected total reward will diminish due to the increased number of benign failures, while the completion of missions may be affected via two mechanisms. The first is an increased probability of mission failure due to a series of benign failures. The second mechanism is an increased probability of the next iteration producing a catastrophic failure if the last iteration produced a failure, albeit benign. This (modelled by setting $p_{BC}>p_{SC}$) looks like a realistic assumption in many cases: for instance, one may assume that a benign failure implies that the program has entered a region of its input space where failure in general is especially likely (as

implied by the assumption of positive correlation), and that a fixed proportion of such failures happens to be immediately catastrophic[1].

A plausible model that accounts for the robustness factor of the system n_c is that in Figure 8. After an iteration with success (state S), the program has a probability p_{sb} that the next execution will produce a benign failure (i.e., that the input trajectory will enter a failure region). However, to model the correlation between successive failures, we assume that, once in a failure region, the probabilities of staying there for one, two or more iterations are given by the parameters p_1, p_2, etc. The parameter p_{nn} designates the probability of staying for n_c iteration or more, i.e., long enough to cause mission failure: $p_{nn} = 1 - \sum_{i=1}^{n_c-1} p_i$.

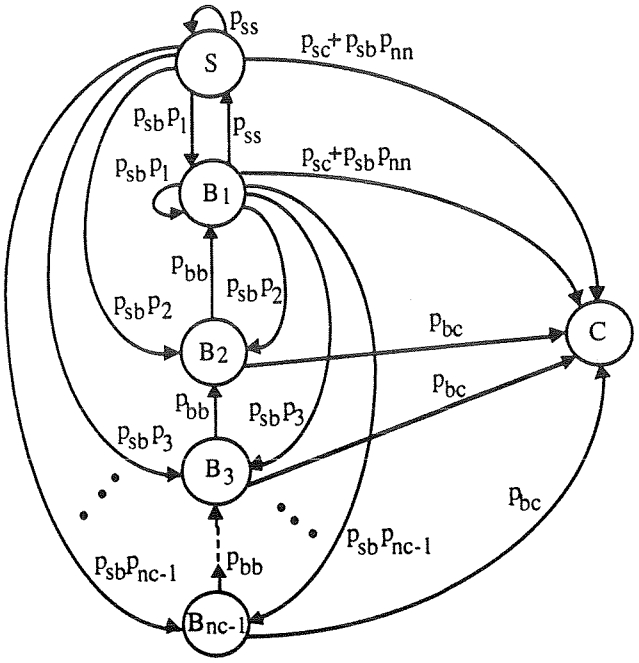


Figure 8. The model for iterative executions with failure clustering

So, for instance, with probability $p_{sb}p_2$ the program enters a failure region, represented by state B_2 in our model, from which, unless a catastrophic failure occurs (arc from B_2 to C labelled p_{bc}) it will be compelled to move to B_1, after which it exits the failure region. This explains the role of the states B_1, B_2, etc., all designating a benign failure of the last execution. If the sequence of failures were longer than n_c-1, a mission failure would occur. This is described by the term $p_{sb}p_{nn}$ in the expression of the probability of the transition from S to C. By choosing the values of the parameters p_i, we assign a distribution function to the variable "number of consecutive failures", conditional on being inside a failure region. Parameters p_{sc} and p_{bc} represent, respectively, the probability of catastrophic failure from state S and from any of the states B_i. Notice that once in B_1 (the last benign failure in the crossing of a failure region), the program may move to another success, or it may enter another failure region: this is modelled by the series of downward arcs issuing from B_1, labelled $p_{sb}p_1$, $p_{sb}p_2$, $p_{sb}p_3$, etc. The probabilities on these arcs are the same as those on the downward arcs issuing from S on the left: the probabilities of the trajectory entering a new failure region is independent of how long ago it left another failure region. This seems appropriate for representing a situation of sparse,

---

[1] However, there are other realistic scenarios: for instance, there may be a controlled system where most erroneous control signals are immediately catastrophic, but the control system is engineered to detect its own internal errors and then issue a safe output and reset itself to a known state from which the program is likely to proceed correctly. Then, one may well assume that most benign failures are due to this mechanism, and are very likely to be followed by a success: $p_{BC} < p_{SC}$ and, indeed, $p_{BB} < p_{BS}$. We will not consider this scenario any further.

small failure regions. Notice that this model allows one to represent positive correlation between successive executions, and independence can be modelled as a limiting case, with $p_1 = 1$ and $p_i = 0$ for all $i>1$. Then, only state B_1 would be reached after a benign failure (B_2,..., B_{nc-1} could never be reached) and from B_1 the same behaviour as from state S is allowed.

State C models the failure of a mission due either to crossing a failure region and staying there for at least n_c iterations, or to a catastrophic failure. A third cause for mission failure exists, i.e., crossing two or more failure regions without interruption, staying in each one for less than n_c failures but so that the total number of consecutive benign failures exceeds n_c-1. This mechanism is not represented by a state in our chain, but rather by a trajectory which, after entering one of the B_i states, and stepping up all the way to B_1, takes one of the downward arcs from B_1 back to one of the B_i, and does so one or more times until it has spent n_c iterations in the set of the B_i states. In solving the model, we neglect this third cause of mission failure: the error thus introduced is negligible, with the ranges of parameter values used here [7]. Hence, in this discussion, the random variable "duration of a failure burst" is practically equivalent to "length of stay in a failure region".

Probability of completing a mission. Under these conditions, the positive correlation between successive iterations will affect the probability of completing a mission both via the probability of having n_c consecutive benign failures and via the longer stays in failure regions, where we assume an increased risk of catastrophic failure. We can use again the expression:

$$P(\text{mission success}) = \prod_{i=1}^{n} P(X_i \neq C \mid X_{i-1} \neq C)$$

where each term in the product now has the form:

$$P(X_i \neq C \mid X_{i-1} \neq C) = (1 - p_{sc}) + (p_{sc} - p_{bc})p\left(X_{i-1} = B_2 \text{ or..or } X_{i-1} = B_{n_c-1} \mid X_{i-1} \neq C\right) +$$
$$-p_{nn}p_{sb}\left(1 - p\left(X_{i-1} = B_2 \text{ or..or } X_{i-1} = B_{n_c-1} \mid X_{i-1} \neq C\right)\right)$$

We note that, in the right-hand side of this equation, the first additive term would be alone with the independence assumption and if sequences of benign failures did not affect the probability of mission failures (i.e., if $p_{bc}=p_{sc}$ and $n_c=\infty$). The second term represents the contribution of the higher (or lower, as the case might be) probability of catastrophic failure after a benign failure, compared to that after a success. The third term represents the probability of mission failure due to n_c or more consecutive failures.

Performability. The expected total reward will be affected by:

1) the increased number of benign failures,
2) if $p_{bc}>p_{sc}$, the increased probability of mission failure due to a catastrophic failure, and
3) the probability of not completing a mission due to sequences of n_c or more failures.

The value of $E[M_n]$ is expressed by the following formula (derived in [7]):

$$E[M_n] = nP(\text{mission success}) - \frac{\sum_{h=1}^{n_c-1} h p_h p_{bb}^{h-1}}{\sum_{h=1}^{n_c-1} p_h p_{bb}^{h-1}} *$$

$$* \sum_{i=1}^{n} i * \sum_{1 \le j_1,\ldots,j_i \le n_c-1} \sum_{\substack{0 \le h \le j_i - 1 \\ j = j_1 + \ldots + j_i \le n+h}} \binom{n-(j-i-h)}{i} p_{sb}^{\,i} (1-p_{bc})^{j-i-h} (1-p_{sc}-p_{sb})^{n-(j-h)} \prod_{k=1}^{i} p_{j_k}$$

An approximated expression, which is easier to compute and gives a lower bound on $E[M_n]$ is:

$$E[M_n] = nP(\text{mission success}) - \frac{\sum_{h=1}^{n_c-1} h p_h p_{bb}^{h-1}}{\sum_{h=1}^{n_c-1} p_h p_{bb}^{h-1}} (n_c - 1) \sum_{i=1}^{n} i \binom{n}{i} (p_{sb}(1-p_{nn}))^i (1-p_{sc}-p_{sb})^{n-i}$$

6 Effects of different distributions of the length of stay in failure regions

The distributions of i) the duration (in steps) of the crossing of a failure region by a trajectory, ii) the number of steps between two failure regions, determine the dependability figures for a mission of the fault-tolerant component. By assuming failure regions which are small and far apart, we have avoided the need to model the time between encountering them with any precision. The distribution of the stay in the failure regions must be guessed at, by making plausible assumptions derived from one's knowledge of the type of software, and previous experience with similar software. Of course, this cannot produce a precise prediction, but the spread of results obtained with plausible assumptions will be a useful indication.

A useful consideration is that, whatever the chosen distribution function, its effect on dependability is determined by two parameters: its mean (which determines the duration of higher exposure to catastrophic failures), and the probability that a sequence of benign failures is equal or longer than the critical threshold n_c. A good estimate of these parameters, even without the knowledge of the complete distribution, should yield a satisfactory prediction. We shall therefore concentrate on these two factors (while keeping all others constant) in the following reliability and performability evaluation, and will show the effect of choosing distribution functions belonging to different families but with the same values of one or the other of these two parameters. The families of distributions used in the figures include some common distributions from the literature, an example of a distribution derived from assumptions about the microscopic behaviour of the software, and two limiting distributions, which can be shown to provide an upper and a lower bound on the dependability figures obtainable for given values of the two parameters. In detail, these distributions are:

- geometric distribution, defined as $p_i = (1-q)q^{i-1}$, $i \geq 1$, for $q \in (0,1)$ and $p_1 = 1$, $p_i = 0$, $i \geq 2$ for $q=0$;
- modified negative binomial, defined as $p_i = \binom{i+r-2}{r-1}(1-q)^r q^{i-1}$, $i \geq 1$, for $q \in (0,1)$ and $p_1 = 1$, $p_i = 0$, $i \geq 2$ for $q=0$;
- Poisson, defined as $p_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!}$, $i \geq 1$;
- an ad-hoc distribution, for a hypothetical application where the following knowledge has been obtained: i) the input space is a discrete two-dimensional (Cartesian) space; ii) the shape of failure regions is that of a square with its sides parallel to the axes of the input space; iii) the input trajectory is a directed straight line crossing the square region failures vertically, horizontally or diagonally (details are in [7], [8]);
- two limiting classes of distribution functions, representing the two extreme behaviours of an input trajectory for given values of n_c and p_{nn}:

  g defined such that $\sum_{i=1}^{n_c-2} g(i) = 0$, with $g(n_c-1) = 1-p_{nn}$ and $\sum_{i>n_c-1} g(i) = p_{nn}$. The input trajectory, if it enters a failure region, stays in it for at least (n_c - 1) iterations;

  f defined such that $f(1) = 1-p_{nn}$, with $\sum_{i=2}^{n_c-1} f(i) = 0$, and $\sum_{i>n_c-1} f(i) = p_{nn}$. After entering a failure region, the trajectory may either exit immediately (after one benign failure) or stay in it for at least n_c iterations.

We show the results for the probability of mission failure $P(C_n)$ and for the performability $E[M_n]$ with the distributions described above for the length of stays in failure regions. The other parameters (shown in Table 9) are set consistently with the analysis of the fault tolerant components in Section 4.

Parameters and their values	
$p_{sb} = 8\ 10^{-6}$	$p_{bc} = 10^{-4}$ (notice that
$p_{sc} = 8\ 10^{-9}$	$p_{bb} = 1 - p_{bc}$ $p_{bc} >> p_{sc}$)
$p_{ss} = 1 - p_{sb} - p_{sc}$	$n_c = 10$
	$n = 10^6$

Table 9. Parameter values used in the numerical evaluation

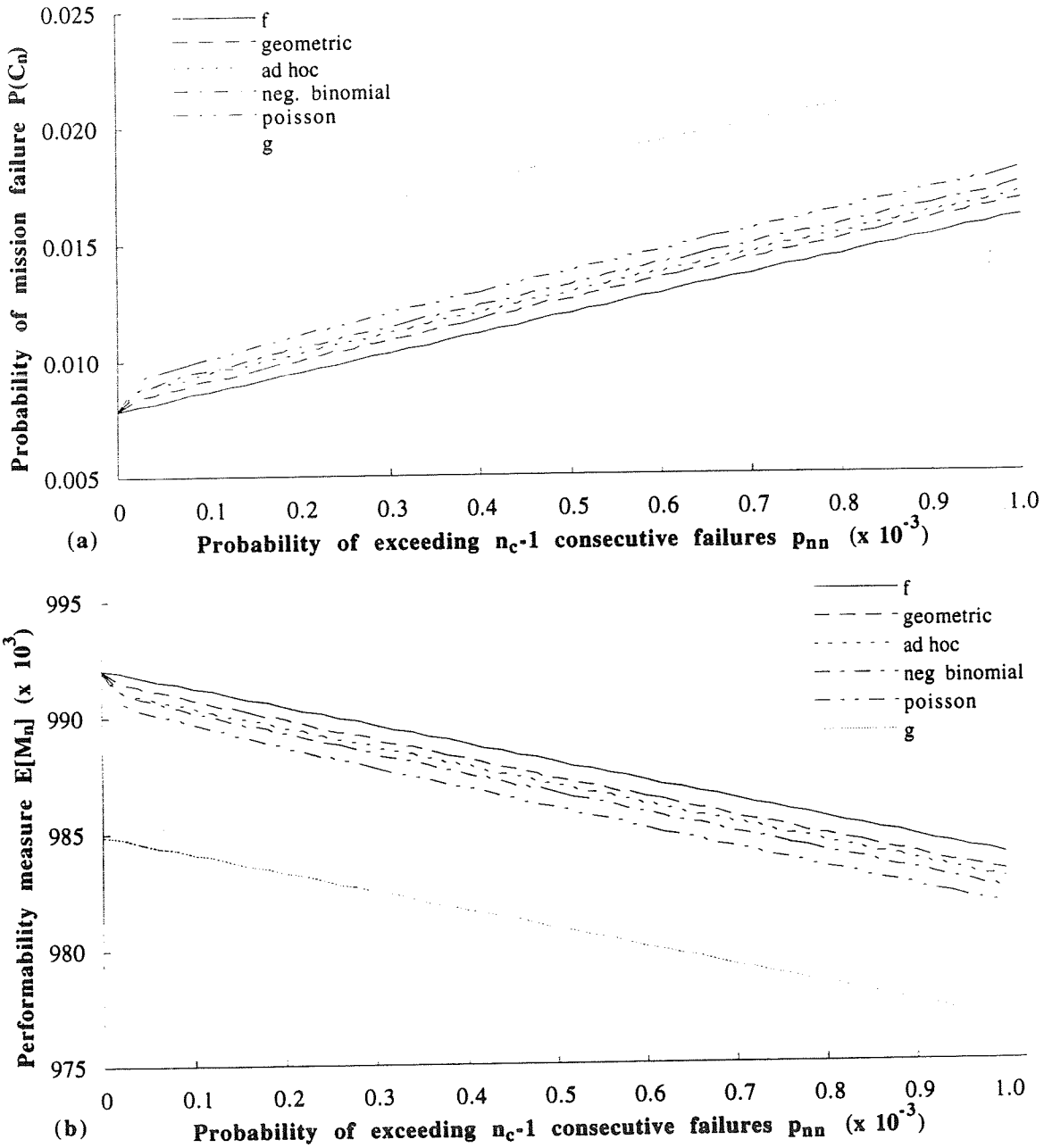


Figure 9. Probability of mission failure (a) and performability (b) as a function of p_{nn} for different distributions of the duration of failure bursts

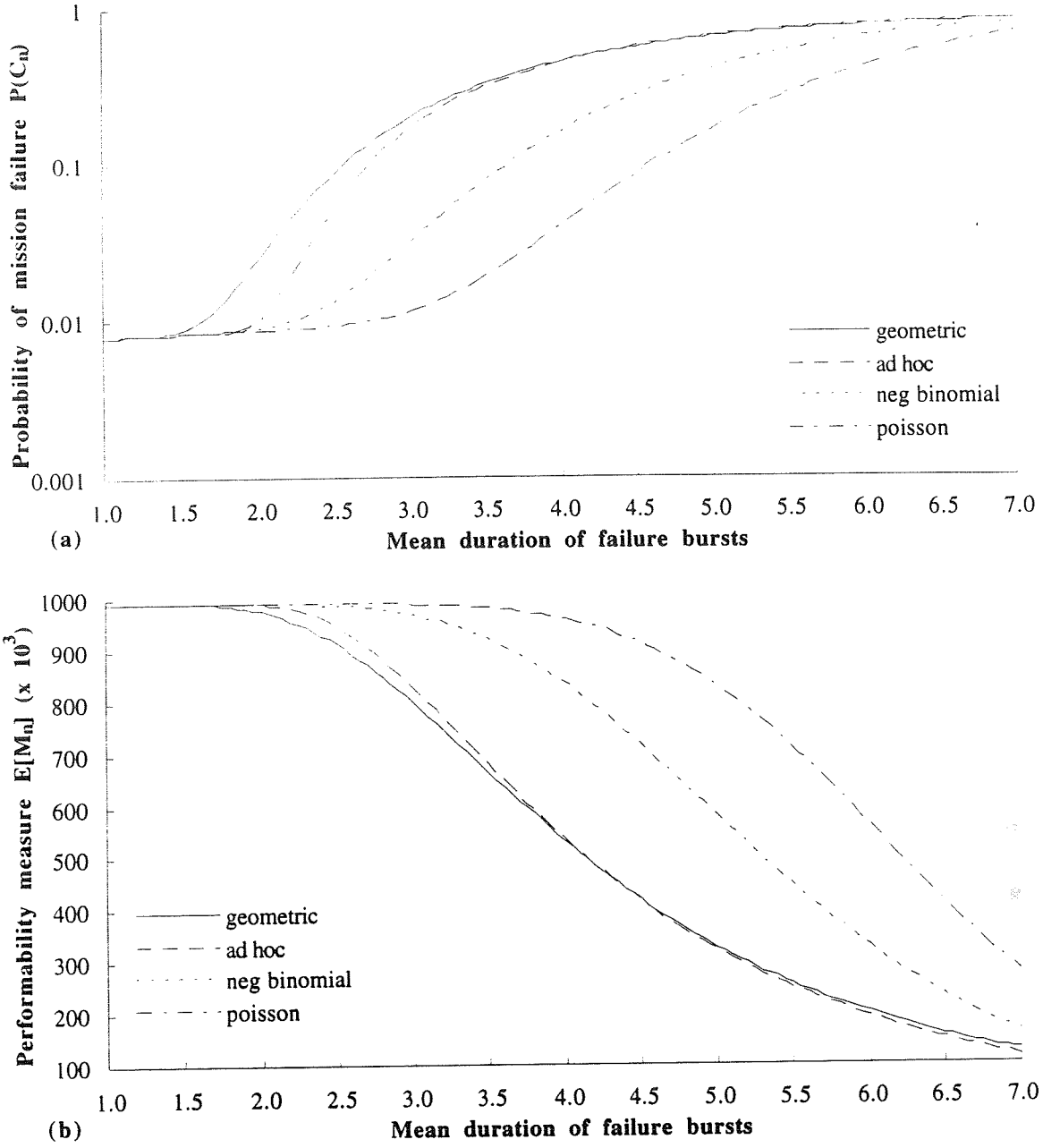


Figure 10. Probability of mission failure (a) and performability (b) as a function of the mean duration of failure bursts

Our dependability measures are plotted in Figures 9.a and 9.b as functions of p_{nn}, in the range $[0, 10^{-3}]$. The non-zero value of $P(C_n)$ for $p_{nn}=0$ is due to the catastrophic failures. In the case of independence, $p_{nn}=1.85\ 10^{-46}$, indistinguishable from 0 in our plots, and $P(C_n)=0.00786828$.

A few remarks about Figure 9 follow:

1) *f* shows better figures than *g* because we set $p_{bc} > p_{sc}$. Moreover, increasing n_c would increase the difference between their probabilities of mission failure;

2) as noted above, the curves for the four plausible distributions are all enclosed between those of *f* and *g*; their distance from *f* depends on the mean duration of stay in the failure regions (and on the difference $p_{bc} - p_{sc}$);

3) the value of p_{sb} determines the slope of the curves; increasing p_{sb} increases the probability of entering a failure region and hence the probability of mission failure;

4) f and g give bounds on the probability of mission failure and the performability, allowing simple tests on the viability of a specific design, given only an estimate of p_{nn}.

Figures 10.a and 10.b show our dependability measures as functions of the mean stay in a failure region. We observe that, for a given mean, distributions with higher variance cause worse behaviour. The range of parameter values shown extends to unrealistic situations: with such values, our plots show that, to obtain probabilities of mission failure smaller than 10^{-1}, the mean stay in failure regions must be limited to 2.5-4.5 steps.

7 How correlation between successive iterations affects dependability predictions

We will now bring together the results of the previous sections to see how non-zero correlation between successive iterations would alter the predictions obtained with the independence assumption from the "white box" model of Section 3. In principle, the parameters of the "black box" model could be derived from hypotheses on the microscopic behaviour of the fault-tolerant component, e.g., the correlation between the failure of variant A at iteration i and the failure of variant B at execution j. However, we do not know how to select such hypotheses to be plausible for any specific fault-tolerant component, and we limit ourselves to showing the effect of correlation between successive executions of the whole fault-tolerant component, assuming one knows the parameters describing it.

Our purpose is to show how, "everything else being equal", varying degrees of correlation between failures in successive iterations of the fault-tolerant component affect dependability predictions. Our "white box" and "black box" models have many interdependent parameters, and we have to decide what is to "remain equal". We choose to assign the marginal probabilities (per iteration) of success p_s, benign failure p_b and catastrophic failure p_c, which one could in theory derive from the "white box" models in Section 3. These "marginal" probabilities have a clear intuitive meaning: for instance, p_b is the probability that the given software will suffer a benign failure on one input chosen at random from the operational distribution of inputs. For a given triplet of values for p_s, p_b and p_c, we will show how the dependability measures we are interested in vary, depending on: i) the way failures cluster in time, that is, the correlation among failures over time, and ii) the value of n_c.

More precisely, we can separate four factors which affect the mission-related dependability measures:

- the locations of the failure points in the input space, or, in other words, the positions and shapes of the failure regions. This is a characteristic of the program code, independent of how the program is used (at least if we only consider programs with a deterministic behaviour);
- the subdivision of the failure points into those causing benign failures and those causing catastrophic failures. This may vary with the characteristics of the controlled system, but we will assume that in our case it never changes. With fault-tolerant software, a natural conservative assumption is that the only benign failures happen when the software detects and signals its own internal errors, and in this case the subdivision is indeed a characteristic of the code only;
- the "input profile", i.e. the probabilities of the different possible input trajectories during operation, which determine the timing of failures and the duration of failure bursts. This probability distribution is certainly a function of the "usage environment" of the controlled system, e.g. for a ship's auto pilot it will be a function of which routes the ships sails, how often and with which pattern of speeds and headings; for an air conditioning system for buildings, it will vary with the climate and exposure of the building where it is installed. Once one has chosen the set of possible installations and usage modes for one's software, the probability distribution of trajectories is known, in theory: it can be simulated in practice by testing the software with a simulator of the

controlled system. Clearly, the evolution of each individual trajectory is affected by the outputs of the control software itself. While the software behaves correctly, these control outputs can reasonably be considered a function of its specification only. When the software fails, it is the details of the faults that determine which control outputs it produces;

- the robustness of the controlled system, which decides how long a burst of benign failures it can tolerate, i.e., our n_c parameter.

To separate the effects of these factors, we initially model the behaviour of the program during an infinite succession of missions by the Markov chain in Figure 11.

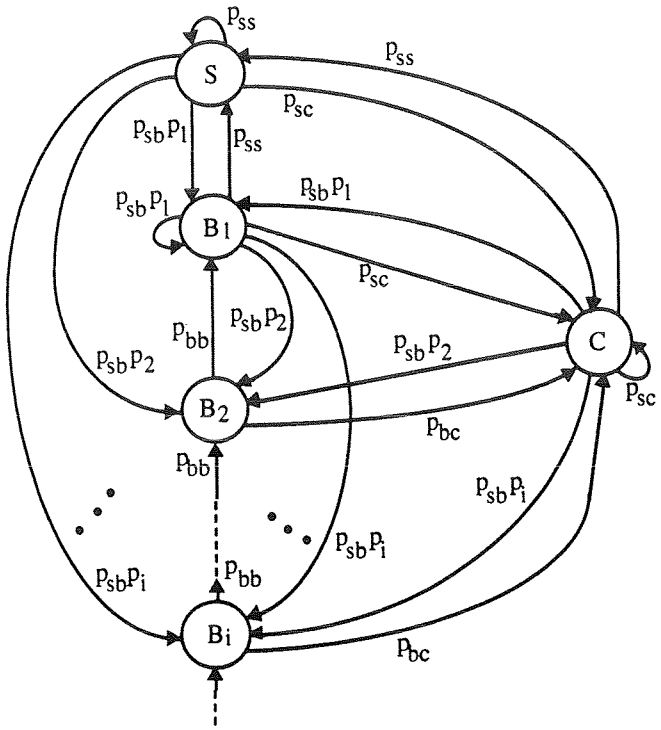


Figure 11. The model for the behaviour of the system during an infinite succession of missions

This model, like that in Figure 8, allows one to represent independence between successive executions by setting $p_1 = 1$ and $p_i = 0$ for all $i>1$, but differs from that model in two ways:

1. repeated benign failures cannot cause a catastrophic failure (only "pointwise" catastrophic failures are possible). This model thus describes the effects of the program code and of its usage environment, but not of the robustness of the controlled system, described by the parameter n_c;
2. state C is not an absorbing state: we have added transitions from state C, all with the same probabilities as the transitions from S to the same destination states. This represents the fact that a catastrophic failure terminates a mission, and the next mission starts as after a success. A mission which terminates in state S is followed by another which starts in state S, so this transition is indistinguishable from any other transition from S to S. As our marginal probabilities, p_s, p_b and p_c, can be seen as the average probabilities of being in the states S, $B=\{B_i\}$, or C over infinitely many missions, they are the steady-state probabilities of the different states in this model[2].

---

2 Actually, this model does not exactly represent the fact that missions last for a fixed number of iterations, after which the system is reset and restarts from the S state. In the model, instead, once the

Notice that to derive these marginal probabilities via testing one would need to test the software along a realistic distribution of trajectories (e.g. using a simulator of the controlled system), interrupting the execution at each catastrophic failure. However, with that kind of testing, one would be better advised to attempt and measure directly the dependability measures of interest, rather than the parameters of this (unavoidably unrealistic) model. The purpose of the discussion that follows is to show the interplay of the different probabilistic parameters characterising the behaviour of our software.

The standard method for the steady-state solution of our Markov chain [31] leads, with the transition rates indicated in Figure 11, to the following system of linear equations:

$$\begin{cases} (p_s + p_{b1} + p_c)p_{ss} = p_s \\ (p_s + p_{b1} + p_c)p_{sb}p_i + p_{bi+1}(1 - p_{bc}) = p_{bi}, i = 1,2,3,... \\ (p_s + p_{b1} + p_c)p_{sc} + (p_b - p_{b1})p_{bc} = p_c \\ p_s + p_b + p_c = 1, \;\; p_b = \sum_{i=1}^{\infty} p_{bi} \end{cases}$$

In these equations, we only know the marginal probabilities p_s, p_b, p_c, and, obviously, cannot derive the transition probabilities which are our unknowns, and which we need for computing our dependability measures. Knowing that the system tends to be in a certain state a certain fraction of the time does not tell us how often it moves into and out of that state. This latter information describes correlation over time, i.e., what we wish to model. These equations allow solutions with multiple degrees of freedom: "correlation over time", that is, "the way failures group into bursts", cannot be fully described by one number. To visualise the effects of correlation, we now choose special assumptions under which it *is* described by one number. We choose to assume that:

i) $p_{bc} = k\, p_{sc}$, with $k>1$ on the basis of the discussion in Section 5.4;

ii) the duration of a failure burst has a geometric distribution with parameter q (so that all the probabilities p_i can be derived as a function of q), as defined in Section 6. In our model q=0 represents independence, while q approaching 1 represents the maximum positive correlation, i.e. the case in which a trajectory, after entering a failure region, will remain in it forever.

Under these assumptions, we solve the system above and obtain all the transition probabilities as functions of the steady state probabilities, the parameter k and the parameter of the geometric distribution, q. These expressions are listed in Table 10.

If we now apply these transition probabilities to the model of Figure 11, we can solve it for missions that last n executions unless they are interrupted. We can also take account of the fact that sequences of n_c or more benign failures cause a mission to fail, by plugging our transition probabilities back into the model of Figure 8 (the parameter p_{nn} in Figure 8 must be set to $1 - (p_1+p_2+...+p_{n_c-1})$)[3].

---

system has entered a state in the set B_i, it will (with non-zero probability) go through i successive benign failures before returning to S. If we include an infinite number of benign failure states B_i, this could be a serious problem indeed. In practice, no significant discrepancy should arise if i) even if infinite B_i states exist, the probabilities p_i dwindle to negligible values when i exceeds some threshold, and ii) this threshold is much smaller than the number of iterations per mission. This latter condition ensures that sequences of benign failures which are interrupted by the end of the mission - the only sequences that cause discrepancies between the model and reality - are a small fraction of the total number of sequences of benign failures. All this is predicated on our assumption that transition probabilities are time-invariant: there is no "especially dangerous" phase in a mission. This assumption would need to change, e.g., to consider hardware failures over a long mission (more likely towards the end than at the beginning).

3 This procedure will in general introduce an error: given a program, with its failure regions, and given a distribution of input trajectories, changing the value of n_c changes the subset of trajectories which end prematurely with a mission failure, and may thus change the parameters of the model. A direct method for finding mission-related dependability measures is to use the model of Figure 8, with the correct value of n_c and the values of all the transition probabilities in the actual missions, including those

p_{ss}	$\frac{p_s}{1-p_b q}$
p_{sb}	$\frac{p_b((1-q)(1-p_b q)+kq((1-q)p_b+p_c))}{(1-p_b q)(1+(k-1)p_b q)}$
p_{sc}	$\frac{p_c}{1+(k-1)p_b q}$
p_{bc}	$\frac{kp_c}{1+(k-1)p_b q}$
p_{bb}	$1-\frac{kp_c}{1+(k-1)p_b q}$

Table 10. Expressions for the transition probabilities of the model in Figure 11

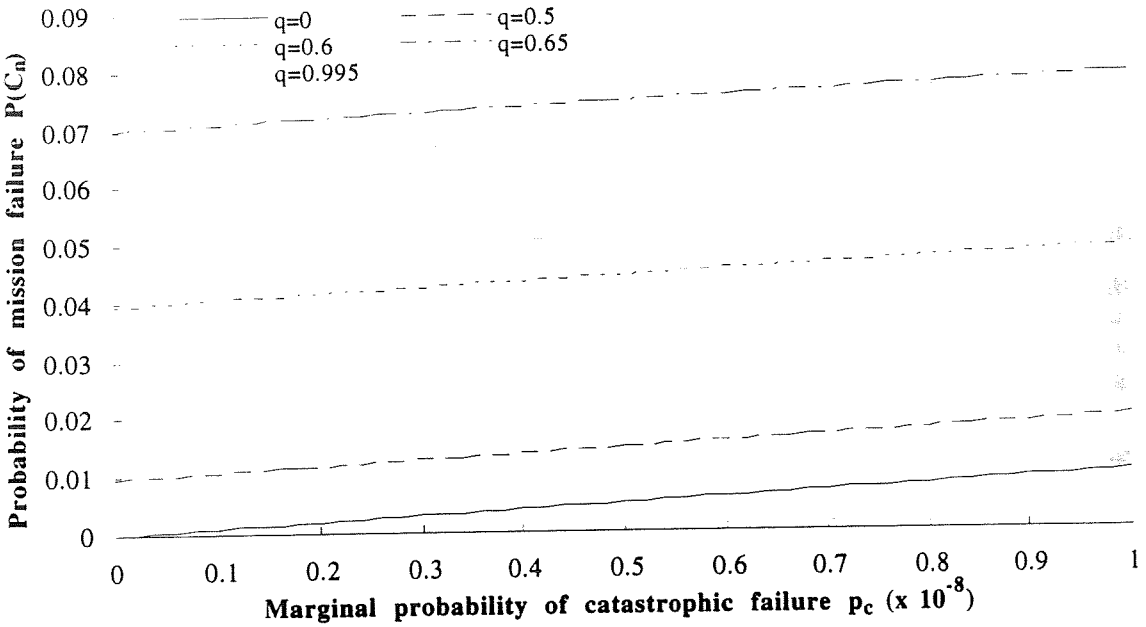


Figure 12. Probability of mission failure, as a function of p_c, for different values of the parameter q of the geometric distribution of the lengths of bursts of benign failures

Figure 12 shows the curves obtained in such a way, i.e. the probability of mission failure, as a function of the marginal probability p_c, for a geometric distribution of the lengths of bursts of benign failures, and for different values of the parameter q of the geometric distribution. We have set $p_s = 105\ p_b$, i.e. $p_b = (1-p_c)/(10^5+1)$ and $p_s = [(1-p_c)10^5]/(10^5+1)$, k=100 (i.e., $p_{bc}=100\ p_{sc}$) and $n_c=10$.

We can now compare the predictions of this model, for this special case of geometric distribution of the lengths of bursts of benign failures, with those of the independence-based model in Section 3. We consider the SCOP fault-tolerant component, and assume again (as we did for Figures 3 to 6) that q_{2v} is proportional to q_{iv}, the failure probability per execution of the

---

which end with n_c benign failures. From that model one can also derive the marginal probabilities of the different states, and thus trace plots like those in our Figure 12. This would certainly be a correct procedure, but very unlikely to be feasible. We only intend to give one example of how changes in correlation over time may affect these measures, and we have thus preferred the procedure we have described, which shows the effects of the different factors separately in its different steps.

Last, we show in Figure 16 how the probability of mission failure varies as a function of the robustness of the controlled system, described by the parameter n_c. Here, the value of q_{iv} is set to 10^{-5} and three different values of q are considered. Given a distribution of the length of failure bursts, increasing n_c reduces p_{nn}, the probability of n_c or more benign failures in a row, which would cause mission failure. Again, decreasing values of q, approaching 0, reduce the probability of mission failure. In our setting, in the case of independence (q=0), the effects of sequences of failures have negligible influence on the probability of mission failure.

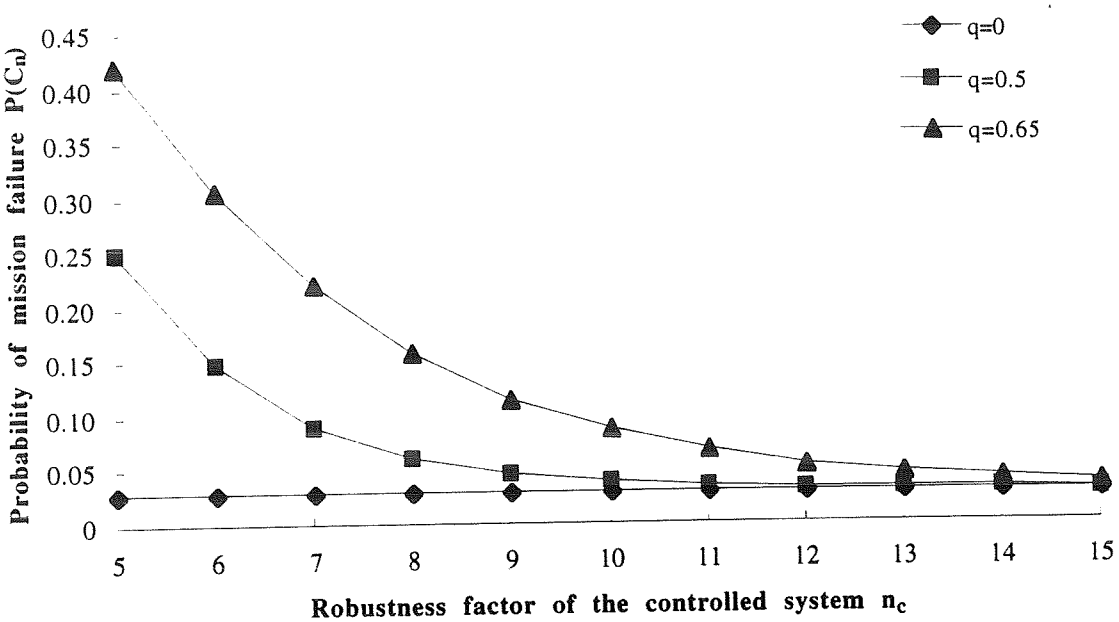


Figure 16. Probability of mission failure for a fault-tolerant component built using SCOP, as a function of n_c, for different values of correlation between iterations

8 Conclusions

Evaluating the dependability levels that can be obtained with alternative design solutions is obviously desirable. In the modelling of design schemes with software fault tolerance, and limited to software with iterative execution (e.g., process control software), we have improved on the previous literature by studying the effects of both:

1. the tendency of failures to appear in bursts (longer than independence between successive iterations would imply) , and
2. the ability of a controlled system to tolerate short bursts of failures.

With all "structural" models of software dependability, including ours, estimating the parameter values is difficult, so that the usefulness of the models is mostly in showing the relative importance of different phenomena which the designer may hope to control, rather than in producing reliable predictions about a system. Modelling more complex phenomena (as we did, compared to previous authors) implies having to estimate more parameters: the more complete models (like ours) are therefore more difficult to use for practical guidance than the less complete ones. Their advantage is in showing whether and when the less complete models may give misleading predictions. In particular, we have shown how, if we model the phenomena of failure bursts and the resilience of the controlled system to benign failures, the results may vary considerably from those obtained with simpler models.

Our analysis has shown that the dependability figures are a complex function of several phenomena and that the same values of dependability are obtained by different combinations of parameters. These indications are useful for designers who believe that they can control some of the phenomena described by our models. For instance, when additional effort directed at

improving one of these parameters seems to be of doubtful utility, a designer can attempt to improve other parameters: if one does not know how to improve the probability of failure per execution, it may be worthwhile to spend some effort in avoiding (through error detection and recovery) long bursts of "benign" failures. It may appear convenient for a designer to accept a higher probability of failure per execution in return for a lower probability of long bursts of failures; or to pursue the latter so as to be able to build less resilience (a lower n_c) into the controlled system.

We must underscore that the specific model of "bursty" failures that we chose (i.e., our modelling assumptions) has the advantage of "plausibility", not of proven realism. In other words, it is based on sensible considerations about programs in general, but it has still to be checked how well it represents the behaviour of any real program. In any case, one should not consider such a model to have been experimentally validated until it had been tested on quite many individual, real programs of the type of interest. Even more strongly we must caution the reader against considering the restrictive assumptions we used in the second part of Section 7, for the sake of an easier visualisation of results, as applicable to software in general. The plots shown there visualise how varying the correlation among iterations, even for constant probabilities of the outcomes of one iteration, changes the dependability measures of interest, under a special set of circumstances. However, we showed that more degrees of freedom are possible in the behaviour of a system, which may cause wider variations than those we showed. A designer wishing to estimate bounds on the behaviour of a specific system should consider the effects of varying all parameters without arbitrary restrictions.

One use of modelling is in showing bounds, rather than precise predictions, on measures of interest. A useful contribution of this paper is our specification of the probability distribution functions f and g for the length of stay in a failure region, or duration of failure bursts (in Section 6), which give upper and lower bounds on the dependability measures obtainable with a given probability of non-catastrophic failure bursts.

Even when designers have an intuition of how they could improve the values of individual parameters, they are usually unable to estimate them with precision or design for a certain required value: no software designer knows how to put just enough effort in designing a program so that its reliability is approximately as required, but not too much more. However, an interesting special case for the use of our models would be that of a program whose input space and use characteristics are so structured that we can devise a testing strategy to ensure an upper bound on (or to estimate the distribution of) the sizes of failure regions and hence, given some knowledge about the dynamics of the controlled system, on the duration of failure bursts.

Although our intention was to improve the evaluation of fault-tolerant software, the "black box" models we discussed in Sections 5 and 7 clearly apply to all software with iterative executions.

A limit of our analysis is that it does not explicitly trace the effects of the statistics, measured across sequences of executions, of the detailed internal events inside a fault-tolerant component. These effects are clearly important: for instance, if one knew whether a failure of one version at one iteration should be interpreted as implying that another version is more (or less) likely than usual to fail at the next execution, i.e., if one had more knowledge about the overlapping of failure regions for different versions, one could map their implications on the sequences of failures of the fault-tolerant component seen as a "black box". This would allow an extension of models like those of Eckhardt and Lee [14] and of Littlewood and Miller [19], describing the effects of a varying degree of "problem difficulty" over the input space and between different software variants, to the case of repeated executions. However, we have not found a mathematical description of these phenomena with parameters intuitive enough to allow a useful analysis.

References

[1] P. E. Amman and J. C. Knight, "Data Diversity: An Approach to Software Fault Tolerance," *IEEE Transactions on Computer*, vol. C-37, pp. 418-425, April 1988.

[2] J. Arlat, K. Kanoun and J. C. Laprie, "Dependability Modelling and Evaluation of Software Fault-Tolerant Systems," *IEEE Transactions on Computers*, vol. C-39, pp. 504-512, April 1990.

[3] A. Avizienis and L. Chen, "On the Implementation of N-Version Programming for Software Fault Tolerance During Program Execution," *Proc. 1st Int. Conf. on Computing Software Applications (COMPASAC-77)*, 1977, pp. 149-155.

[4] A. Avizienis and J. P. J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," *IEEE Computer*, vol. 17, pp. 67-80, August 1984.

[5] P. G. Bishop, "The Variation of Software Survival Time for Different Operational Input Profiles (or why you Can Wait a long Time for a big Bug to Fail)," *Proc. 23th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-23)*, Toulouse, France, 1993, pp. 98-107.

[6] P. G. Bishop and F. D. Pullen, "PODS Revisited - A Study of Software Failure Behaviour," *Proc. 18th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, Tokyo, Japan, 1988, pp. 1-8.

[7] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico and L. Strigini, "Modelling Correlation Among Successive Inputs in Software Dependability Analyses," CNUCE-CNR, Pisa, Italy, Technical Report C94-20, 1994.

[8] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico and L. Strigini, "Dependability Models for Iterative Software Considering Correlation between Successive Inputs," *Proc. IEEE Int. Symp. on Computer Performance and Dependability (IPDS'95)*, Erlangen, Germany, 1995, pp. 13-21.

[9] S. S. Brilliant, J. C. Knight and N. G. Leveson, "Analysis of Faults in an N-Version Software Experiment," *IEEE Transactions on Software Engineering*, vol. SE-16, pp. 238-247, February 1990.

[10] S. Chiaradonna, A. Bondavalli and L. Strigini, "Comparative Performability Evaluation of RB, NVP and SCOP," CNUCE-CNR, Pisa, Italy, Technical Report C94-02, 1994.

[11] S. Chiaradonna, A. Bondavalli and L. Strigini, "On Performability Modeling and Evaluation of Software Fault Tolerance Structures," *Proc. 1st European Dependable Computing Conference (EDCC-1)*, Berlin, Germany, 1994, pp. 97-114.

[12] A. Csenski, "Recovery Block Reliability Analysis with Failure Clustering," *Preprints Int. Working Conf. on Dependable Computing for Critical Applications (DCCA-1)*, Santa Barbara, California, 1989, pp. 33-42.

[13] F. Di Giandomenico and L. Strigini, "Adjudicators for Diverse Redundant Components," *Proc. 9th IEEE Symp. on Reliable Distributed Systems (SRDS-9)*, Huntsville, Alabama, 1990, pp. 114-123.

[14] D. E. Eckhardt and L. D. Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 1511-1517, December 1985.

[15] J. Gray, "Why do Computers Stop and What Can be Done About it?," *Proc. 5th Symposium on Reliability in Distributed Software and Database Systems*, Los Angeles, California, 1986, pp. 3-12.

[16] IEEE-TR, "Special Issue on Fault-Tolerant Sofware," *IEEE Transactions on Reliability*, vol. R-42, pp. 177-258, June 1993.

[17] J. C. Knight and N. G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming," *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 96-109, January 1986.

[18] J. C. Laprie, J. Arlat, C. Beounes and K. Kanoun, "Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures," *IEEE Computer*, vol. 23, pp. 39-51, July 1990.

[19] B. Littlewood and D. R. Miller, "Conceptual Modelling of Coincident Failures in Multiversion Software," *IEEE Transactions on Software Engineering*, vol. SE-15, pp. 1596-1614, December 1989.

[20] M. R. Lyu and Y. He, "Improving the N-Version Programming Process Through the Evolution of a Design Paradigm," *IEEE Transaction on Reliability, Special Issue on Fault-Tolerant Software*, vol. R-42, pp. 179-189, June 1993.

[21] J. F. Meyer, "On Evaluating the Performability of Degradable Computing Systems," *IEEE Transactions on Computers*, vol. C-29, pp. 720-731, August 1980.

[22] V. F. Nicola and A. Goyal, "Modeling of Correlated Failures and Community Error Recovery in Multiversion Software," *IEEE Transactions on Software Engineering*, vol. SE-16, pp. 350-359, March 1990.

[23] B. Randell, "System Structure for Software Fault Tolerance," *IEEE Transactions on Software Engineering*, vol. SE-1, pp. 220-232, June 1975.

[24] L. Strigini, "Software Fault Tolerance," PDCS ESPRIT Basic Research Action, Technical Report 23, July 1990.

[25] G. F. Sullivan and G. M. Masson, "Using Certification Trails to Achieve Software Fault Tolerance," *20th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-20)*, Newcastle-upon-Tyne, U.K., 1990, pp. 423-431.

[26] A. T. Tai, "Performability Concepts and Modeling Techniques for Real-Time Software," Ph. D. thesis, UCLA, Computer Science Dept., Los Angeles, 1991.

[27] A. T. Tai, "Performability-Driven Adaptive Fault Tolerance," *Proc. 24th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-24)*, Austin, Texas, 1994, pp. 176-185.

[28] A. T. Tai, A. Avizienis and J. F. Meyer, "Evaluation of Fault Tolerant Software: a Performability Modeling Approach," in *Dependable Computing for Critical Applications 3*, C. E. Landwehr, B. Randell and L. Simoncini (Eds.), Springer-Verlag, 1993, pp. 113-135.

[29] A. T. Tai, A. Avizienis and J. F. Meyer, "Performability Enhancement of Fault-Tolerant Software," *IEEE Transactions on Reliability, Special Issue on Fault-Tolerant Sofware*, vol. R-42, pp. 227-237, June 1993.

[30] L. A. Tomek, J. K. Muppala and K. S. Trivedi, "Modeling Correlation in Software Recovery Blocks," *IEEE Transactions on Software Engineering*, vol. SE-19, pp. 1071-1085, November 1993.

[31] K.S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, London: Prentice-Hall, 1982.

[32] J. Xu, A. Bondavalli and F. Di Giandomenico, "Dynamic Adjustment of Dependability and Efficiency in Fault-Tolerant Software," in *Predictably Dependable Computing Systems*, B. Randell, J. C. Laprie, H. Kopetz and B. Littlewood (Eds.), Springer-Verlag, 1995, pp. 155-172.