Littlewood, B., Bloomfield, R. E., Popov, P. T., Povyakalo, A. A. & Strigini, L. (2004). The impact of "difficulty" variation on the probability of coincident failure of diverse systems. Paper presented at the International Conference on Control and Instrumentation in Nuclear Installations, 2004, Liverpool, UK.

CITY UNIVERSITY LONDON

City Research Online

The impact of 'difficulty' variation on the probability of coincident failure of diverse systems

Peter Bishop, Robin Bloomfield, Bev Littlewood, Peter Popov, Andrey Povyakalo, Lorenzo Strigini

Adelard and Centre for Software Reliability, City University

Abstract

Redundancy and diversity have long been used as means to obtain high reliability in critical systems. Whilst it is easy to claim that, say, a 1-out-of-2 diverse system will be more reliable than each of the two channels, *assessing* the actual reliability of such systems can be difficult. Some years ago, new probability models were developed to address this problem in the case of diverse *software* systems. They depend upon a notion of variation of 'difficulty' – more precisely 'propensity to fail' – across the input space. These models show that independence of failures will occur only in very special circumstances, and so such independence cannot simply be assumed. They were later shown to apply to certain kinds of hardware systems. If we cannot claim independence of channel failures, the computation of system reliability is difficult, because complete knowledge of the difficulty function is needed. This is unlikely to be available for software. Instead, we are unlikely to know more than the *marginal pfd* (probability of failure on demand) of the software. In this paper we consider the case of a 1-out-of-2 system in which one channel contains software, and the other channel contains only hardware equipment. We show that a useful upper (i.e. conservative) bound can be obtained for the system *pfd* using only the unconditional *pfd* for software (together with information about the variation of hardware 'difficulty', which is likely to be known or estimatable).

1 Introduction

In earlier work (Eckhardt and Lee 1985; Hughes 1987; Littlewood and Miller 1989) the idea of difficulty variation has been introduced as an explanation for dependence in failure behaviour between diverse system components to be used in a fault tolerant system (e.g. a 1-out-of-2 system). The idea is a simple one. The 'difficulty' of a demand can be thought of intuitively as its propensity to induce failure in the system that has to handle it (more formally it is the probability of that system failing on the demand). If this difficulty varies across demands for each of the component sub-systems of a 1-out-of-2 system, interest centres upon the association between the two difficulty functions. When there is positive association here – roughly, what is very difficult for one version also tends to be very difficult for the other – then it is more likely that there will be positive association between their *failures*. In such a case, wrongly assuming independence of failures between the two versions will give an *optimistic* estimate of the reliability of the 1-out-of-2 system[1].

A conceptual achievement of these models is their establishment of, and explanation for, the inevitability of dependence of failure behaviour between versions. No longer is it possible to claim that the two diverse component systems of a 1-out-of-2 fault tolerant architecture will fail independently of one another, without making very strong claims: essentially that there is no variation of 'difficulty' for at least one of the channels. This means that the simple arithmetic of independence is not applicable for the computation of the *system* reliability as a function of the component reliabilities. Informally, it means that we need to know *how dependent* the version failures will be.

The problem of estimating the reliability of such a 1-out-of-2 system is thus hard. It seems to require a complete knowledge of the two difficulty functions. Whilst it seems feasible to obtain reasonable approximations of difficulty functions for hardware systems, for software things seem much more problematic. In fact, we are likely to know – or be able to estimate – the *marginal* probability of failure on demand of the software, but not know how the *pfd* varies across demands, or demand classes. In this paper we consider, for simplicity, a 1-out-of-2 system in which there is software in one channel only. We show how to obtain a pessimistic (but attainable) bound for the probability of failure on demand for such a system, which requires only the marginal *pfd* of the software to be known (together with the varying hardware *pfd*s across different demand classes). The basic ideas here can be generalised to more complex systems than the example we use in the paper.

2 Statement of problem

The example we shall use for illustration throughout the paper is a nuclear reactor protection system with two channels: the *X*-train and *Y*-train.

---

[1] It is possible to do better than independence if there is negative correlation between the difficulty functions.

Consider first the simple situation in which each channel is built of hardware alone, as in Fig 1. We shall assume that demands are of several different *types*. A demand of one type will typically have a different probability of failure from a demand of another type. In the case study that prompted this work, a demand type could be characterised by the equipment that was needed to function correctly for the demand to be successfully met. Some demand types, for example, required more such equipment than others, and thus might be considered in our informal terminology more 'difficult' – i.e. the chance of failure would be greater. *Within* a demand class, although demands are of the same type they will differ from one another in some respects: for example, the reactor state will be different, represented by the readings of sensors for temperature, pressure, etc. Nevertheless, all demands within a demand class have the same *pfd*. In the event that all demands of this type require exactly the same equipment to function without failure, this assumption may be reasonable.

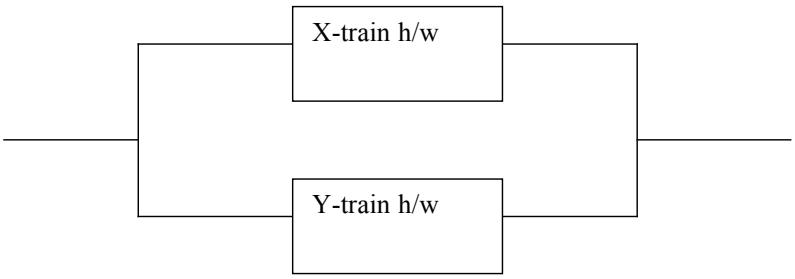

Figure 1

With these assumptions, we can see that the (marginal) probability of failure on demand of the 1-out-of-2 system, i.e. for a randomly chosen demand, is the probability of both *X*-train hardware and *Y*-train hardware failure:

$$pfd_{X_h Y_h} = \sum_i p_{hi}^X p_{hi}^Y f_i \qquad (1)$$

where p_{hi}^X is the probability of X-train hardware failure on demand type i, p_{hi}^Y is the probability of Y-train hardware failure on demand type i, and f_i is probability that a randomly chosen demand is of type i.

Clearly, the *pfd* is different from the result that would be obtained under an incorrect assumption of independence of failure between the two channels, which is

$$pfd_{X_h} . pfd_{Y_h} = \left(\sum_i p_{hi}^X f_i \right) \left(\sum_i p_{hi}^Y f_i \right)$$

The true result will exceed the incorrect result (based on the false independence assumption) so long as there is positive covariance between the *X*- and *Y*-train demand type *pfd*s, p_{hi}^X and p_{hi}^Y. This is similar to the result of Eckhardt and Lee (Eckhardt and Lee 1985) for software diversity. The positive covariance means that there is a tendency for large demand type *pfd*s in the *X*-train to be associated with large demand type *pfd*s in the *Y*-train: informally, if we see the *X*-train fail, we note

that the demand type was probably one with a large *pfd*, and that the *Y*-train *pfd* is also thus probably large, and therefore its probability of failure is greater than it would be unconditionally.

Note the assumption of conditional independence in (1), i.e. for a given demand type, failure of *X*-train hardware is independent of failure of *Y*-train hardware. This is reasonable if it can be assumed that there is constancy of *pfd* within a demand type for either train. Alternatively, if the *pfd* for a demand class in such expressions is an upper bound on the true *pfd* (varying over demands within the demand class), then we shall have a conservative bound from this expression.

The important point here is that for each demand type in the sum here, every demand within that class requires exactly the same minimal set of functioning hardware to satisfy the demand.[2]

We now consider the situation which is the subject of this paper, in which one channel has software: see Fig 2.

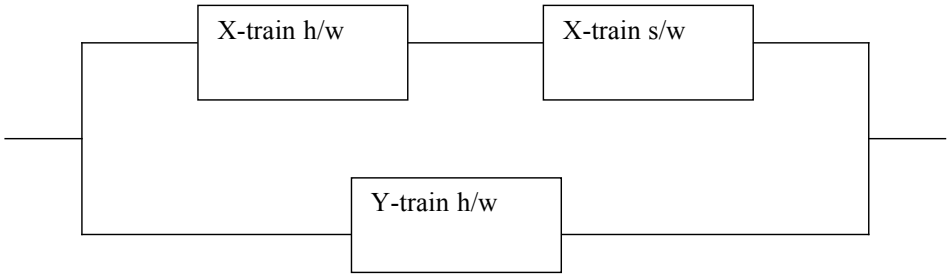

Figure 2

The probability of failure on demand is now:

$$pfd_{X_{h+s}Y_h} = \sum_i \left(p^X_{hi} + p^X_{si} - p^X_{hi,si}\right)p^Y_{hi} f_i \qquad (2)$$

where p^X_{si} is the probability of failure of the X-train *software* on a demand of type i, and $p^X_{hi,si}$ is the probability of simultaneous hardware and software failure on a demand of type i.

The practical difficulty we face now is that neither p^X_{si} nor $p^X_{hi,si}$ are likely to be known or estimatable. We can proceed conservatively by ignoring the probability of joint hardware/software failures, since

[2] There is a slight problem here if the demand types for channel *X* and those for channel *Y* do not exactly coincide. A demand type in the sum (1) can be defined as a set of demands that have the same *X*-train probability of failure for all the demands in the set, and the same *Y*-train probability of failure for all demands in the set. This will generally involve more demand types for the *system* than there are for each channel alone.

$$pfd_{X_{h+s}Y_h} \le \sum_i \left(p^X_{hi} + p^X_{si}\right)p^Y_{hi}f_i$$

$$= pfd_{X_hY_h} + \sum_i p^X_{si}\,p^Y_{hi}f_i \qquad (3)$$

The second term here can be thought of as the maximum error (underestimation) there could be in the probability of system *pfd* if we ignored the effect of software failures. Whilst it is unlikely that the effect of software failures on system *pfd* will be ignored completely in this way, it is more likely that *variation* of the software *pfd* across demands will be ignored, since this variation will be unknown.

If we were to assume that there is no variation in software *pfd*, we have in an obvious notation:

$$p^X_{si} \equiv pfd_{X_s}$$

which, when substituted into (3), gives

$$pfd_{X_{h+s}Y_h} \le pfd_{X_hY_h} + \sum_i p^X_{si}\,p^Y_{hi}f_i = pfd_{X_hY_h} + pfd_{X_s}.pfd_{Y_h} \qquad (4)$$

The maximum error that arises from ignoring software *pfd* variation is thus the difference between (3) and (4):

$$\sum_i p^X_{si}\,p^Y_{hi}f_i - pfd_{X_s}.pfd_{Y_h} \qquad (5)$$

Notice that this is zero if there is no variation in the *Y*-train hardware *pfd*: in this case the *X*-train software and the *Y*-train hardware fail independently. The error arising from incorrectly assuming the *X*-train software *pfd* does not vary is completely masked by the fact of there being no variation in the *Y*-train. This result was first noted in (Littlewood and Miller 1989): in general, if there is no difficulty variation in one channel, then the two channels fail independently, *regardless of what happens in the other channel*.

In the next section we show how to compute the worst value (5) can take.

3 Worst case error from ignoring software *pfd* variation

We need to find the worst case value of the first term in (5) for a *given* value of the marginal software *pfd*, $pfd_{X_s} = \sum_i p^X_{si} f_i$. In other words, we need to find which allocation of pfd_{X_s} among the different demand classes maximizes $\sum_i p^X_{si}\,p^Y_{hi}f_i$.

Now

$$\sum_i p^X_{si}\,p^Y_{hi}f_i = Cov\left(p^X_{si}, p^Y_{hi}\right) + E\left(p^X_{si}\right)E\left(p^Y_{hi}\right)$$

where $E\left(p_{si}^X\right)$ is simply the marginal probability on demand of the software, $pfd_{X_s} = \sum_i p_{si}^X f_i$, and $E\left(p_{hi}^Y\right)$ is the marginal probability of failure on demand of the *Y*-channel hardware, $pfd_{Y_h} = \sum_i p_{hi}^Y f_i$. If we keep these two probabilities constant, the maximum underestimate of $\sum_i p_{si}^X p_{hi}^Y f_i$ occurs when $Cov\left(p_{si}^X, p_{hi}^Y\right)$ takes its maximum value. It is easy to see that this occurs when we associate large values p_{si}^X with large values of p_{hi}^Y.

Informally, then, we proceed by allocating as much of pfd_{X_s} as we can to the demand class that has maximum *Y*-channel *hardware* pfd; we allocate as much of the remaining pfd_{X_s} to the demand class with the next largest *Y*-channel *hardware* pfd, and so on until we have 'used up' all of pfd_{X_s}.

Rather more precisely the procedure to find the worst case error in ignoring software *pfd* variation, (5), is as follows:

> *Denote by* i^* *the demand class that has maximum* Y*-train hardware* pfd, *i.e.*
>
> $p_{hi^*}^Y = \max\left\{p_{hi}^Y\right\}$.
>
> *If*
>
> $$pfd_{X_s} \le f_{i^*} \tag{6}$$
>
> *then the maximum possible value of* $\sum_i p_{si}^X p_{hi}^Y f_i$ *occurs when*
>
> $$p_{si^*}^X = \frac{pfd_{X_s}}{f_{i^*}};\ p_{si}^X = 0 \text{ for all other values of } i \tag{7}$$
>
> *and the maximum value of* $\sum_i p_{si}^X p_{hi}^Y f_i$ *is then*
>
> $$pfd_{X_s}.p_{hi^*}^Y. \tag{8}$$
>
> *In the event that the software* pfd *is too large, and (6) is violated, the result extends in an obvious way: as much as possible of the software* pfd *is assigned to the demand type with the largest* Y*-train hardware pfd, as here; as much as possible of the remaining software* pfd *is assigned to the demand type with the next largest* Y*-train hardware* pfd*; and so on until all the software* pfd *'has been used up'.*

We call this procedure 'bin-filling' as the demand 'bins' with the highest *Y*-train hardware failure probability are assigned a maximum software *pfd* ('filled up') until we run out of available software *pfd*.

The worst case error in estimation of system *pfd* from ignoring variation in software *X*-channel *pfd* is thus the value of the expression (5) using the maximum value of $\sum_i p_{si}^X p_{hi}^Y f_i$ computed as above.

A proof of this result is too long for the present paper but can be found at the following URL:

http://www.csr.city.ac.uk/people/andrey.povyakalo/BL_DISPO2_01_AppendixA.pdf

4 Discussion

The work by Eckhardt and Lee (and later work) introduced a new way of looking at the reasons for dependence between the failure behaviour of diverse versions. In these models, everything turns on what we have called 'difficulty variation' over demands. This earlier work gave novel insights into the reasons why claims for independence are rarely supportable. Unfortunately, it also introduced some serious difficulties for anyone wishing to exploit the models to estimate the actual probabilities of failure of real systems, since this requires estimation of these 'difficulty functions'.

In this paper we have looked at a particular system: a 1-out-of-2 system in which only one channel contains software. In the example that motivated this work – a protection system for a nuclear reactor – we were able to identify a small number of *demand types* (<20) for each of which a hardware *pfd* could be estimated. In fact these had been estimated as part of the wider safety case for the reactor. For software, on the other hand, only a marginal *pfd* could be estimated. Our aim, therefore, was to obtain a bound on the error in the estimate of the system *pfd* in the event that the variation of software *pfd* across demands were to be ignored – essentially by making the false assumption that the marginal software *pfd* applied equally to every demand class. Our main result here is such a bound, but it may not be the tightest available – essentially because of conservatively ignoring the probability of simultaneous hardware and software failures in expression (3): we hope to present tighter bounds in future work.

As we have found elsewhere whilst working on these models of diversity, these results are quite surprising and subtle: witness, for example, the pivotal role played by variation in *Y-train hardware pfd* when we take into account *X-train software* failures. We do not think that these results could have been obtained without the formal model of diversity, although we believe that they are intuitively plausible in retrospect.

Of course, if a conservative value of the software *pfd* were to be used over all demand classes (i.e. one that is not exceeded by the true *pfd* of any demand class), then the calculated system *pfd* would be conservative. In fact, it may be *very* conservative: our result points to a way of lessening this conservatism.

A small word of warning is appropriate at this stage. The results in Section 3 all depend upon some assumptions of *conditional* independence of failures: for example, independence between failures of the *X*-train (hardware *and* software), on the one hand, and the *Y*-train (hardware only) on the other, for each demand type *i*.

Essentially, this amounts to assuming an equivalence, or indifference, between demands that make up a demand type, i.e. assuming there is no *pfd* variation *within* a demand type (there is only variation *between* demand types and between the trains' hardware and the software). If these assumptions are not correct, the worst case errors given above will be too optimistic.[3]

Acknowledgements

This work was partially supported by the DISPO-2 (DIverse Software PrOject) Project, funded by British Energy Generation Ltd and BNFL Magnox Generation under the IMC (Industry Management Committee) Nuclear Research Programme under Contract No. PP/40030532, and by the project DIRC ('Interdisciplinary Research Collaboration in Dependability of Computer-Based Systems') funded by UK Engineering and Physical Sciences Research Council (EPSRC).

References

Eckhardt, D. E. and L. D. Lee (1985). "A Theoretical Basis of Multiversion Software Subject to Coincident Errors." IEEE Trans. on Software Engineering **11**: 1511-1517.

Hughes, R. P. (1987). "A new approach to common cause failure." Reliability Engineering **17**: 211-236.

Littlewood, B. and D. R. Miller (1989). "Conceptual Modelling of Coincident Failures in Multi-Version Software." IEEE Trans on Software Engineering **15**(12): 1596-1614.

[3] This observation is equally true of calculations of system reliability when only hardware failures are considered, as in (1). However, this should not be of serious concern for two reasons. In the first place, it may be more plausible for there to be little or no variation of *pfd* for hardware within a demand type, compared with software. Secondly, the usual conservatism in the estimation of the individual demand type *pfd*s may overcome the difficulty: if the common assumed value for each particular demand type is greater than the largest true *pfd* among the individual demands within that type, then conservatism is guaranteed in the overall calculation.