FAULT TOLERANCE FOR DIGITAL SYSTEMS

Herbert Hecht
SoHaR Incorporated

Abstract

Fault tolerance is an essential methodology for digital systems, particularly for those that serve applications where failure has safety implications or where interruption of operations imposes serious financial penalties. There is no single fault tolerance technique that suits or is optimal in all circumstances. A taxonomy of fault tolerance techniques is presented and branches and leaves of this taxonomy are described in terms of areas of applicability, effectiveness of fault tolerance, and cost of implementation. Gaps in coverage and deficiencies of an individual technique can be overcome by employing a hierarchical structure of fault tolerance provisions, also referred to as defense-in-depth. The large selection of techniques that have been described and the continuing improvements provided by studies in the field support an encouraging outlook.

1. Introduction

Practically all digital systems include some fault tolerance provisions but in spite of this failures of digital systems are still a frequent occurrence. These incidents can be due to

- Design or implementation deficiencies of the fault tolerance provisions
- Unprotected portions of the fault tolerance provisions themselves
- Anomalies encountered in operation that fall outside the class of tolerated faults (lack of coverage)

The need to study and deal with these problems has been recognized for almost 50 years. The first International Symposium on Fault Tolerant Systems was held in 1971 at the Jet Propulsion Laboratory in Pasadena, California. As the venue indicates, much of the interest is fault tolerant computing stemmed from the need for computers on long duration space missions. Concerns and achievements during the early years of these investigations are documented in a volume commemorating the Silver Jubilee of that first symposium[1].

Today the need for fault tolerant computing is found throughout our daily environment: at the grocery store cash register, in electronic systems of our cars, and in the flight control systems of airliners to name just a few examples. To aid the understanding of the

field the next heading introduces a taxonomy of fault tolerance provisions, and subsequent sections explore the major branches of this taxonomy. Increasingly it is not only required that fault tolerance measures be effective but also that they be cost effective. For that reason a brief discussion of the economics of fault tolerance is presented at the end.

2. Taxonomy of Fault Tolerance Provisions

A baseline taxonomy of fault tolerance provisions is shown in Figure 1. The taxonomy is titled *baseline* because no claim is made that it is complete at present and certainly not for the future; it is expandable along branches and at the leaves.

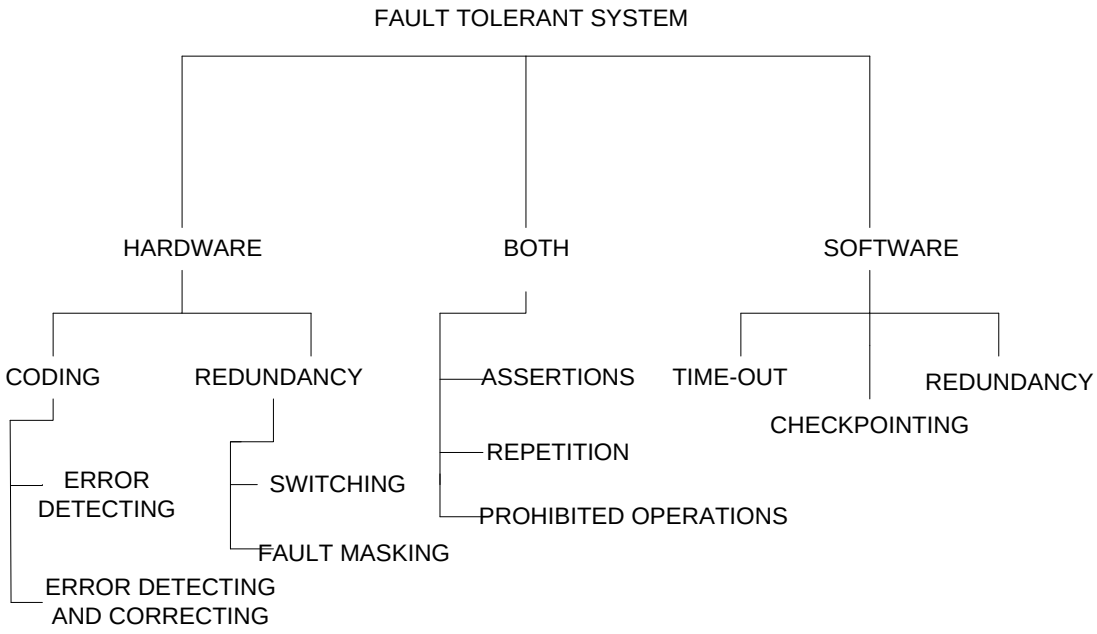

Figure 1 Baseline Taxonomy

The branch labels refer to the location of the fault, not to the means for recovery which in most cases involves both hardware and software elements. Each entry in the figure represents a vulnerability of not responding in the required manner when challenged by unusual failure or environmental conditions.

3. Hardware

3.1. Coding

The hardware branch has two subdivisions – coding and redundancy. Coding is a highly efficient means of error detection that is applicable only to digital equipment and addresses failure mechanisms some of which are specific to digital equipment such as upsets of memory bits as a result of background radiation. Coding is also widely used for detection and correction of errors in communication channels and input/output functions of digital components[2]. An example of error detecting coding is the addition of a parity bit to a group of information bits such that the sum of bits is either even or odd (as defined in a given environment). If the information bits are 01011100 and odd parity is used, a 1 bit will be appended which will make the sum of bits (sometimes called the cross-sum) 5, an odd number. If any one of the resulting nine bits is changed, the cross-sum will become even which signals that an error has occurred. Recovery from a detected error may involve repetition (re-transmission), access to a different data location, or use of an alternate means for completing a given function.

Error correcting codes function by assigning parity bits to groups of data bits such that the pattern of parity failures uniquely identifies the altered bit. An example of a 7,4 Hamming code is shown in Table 1. The numbers 7 and 4 stand, respectively for the total number of bits used (7) and the number of data bits covered (4). Richard Hamming was one of the pioneers in research on error correcting codes[3].

Table 1. Operation of 7,4 Hamming Code.

Bit Position	p1	p2	d1	p4	d2	d3	d4
p1	x		x		x		x
p2		x	x			x	x
p4				x	x	x	x

If only p1 indicates parity failure it must be in p1 itself. If p1 and p2 indicate parity failure it must be d1. If p1, p2 and p4 indicate parity failure it must be d4. As the number of data bits covered by the code is increased the number of required parity bits increases much more slowly. Fifteen total bits can cover up to 11 data bits. But note that this type of code covers single bit failures only and as the number of covered bits increases so does the probability of multiple bit failures. Block codes and burst error detecting codes[4] are more efficient than linear codes, such as the Hamming code, in specialized threat environments.

Codes are logic constructs and do not have failure modes in the usual meaning of that term. Codes fail when the extent of failures exceeds their capabilities. It is therefore important to know the failure characteristics of the medium (semiconductor, communication spectrum) to which the code is being applied and to select a code of an appropriate type and coverage. Within their application range codes are generally the most economical means of achieving fault tolerance.

3.2. Redundancy

Redundancy is widely employed in safety critical computer applications, such as aircraft flight controls, in electronic communication systems, and in commercial environments where interruption of a service can cause substantial losses, such as in financial services, electronic casino games and check-out lines. A number of concerns are common to all uses of redundant digital devices

a. Is the functioning of the back-up monitored during normal operation?

b. Will the hardware and software necessary for the fault tolerance provisions function in disturbed environments that may be associated with the failure of the primary device?

c. Will the transition to a back-up configuration be visible to the operators?

d. Will all data be recovered?

e. Will the transition to a back-up configuration change the operating environment?

The significance of the last item is illustrated by the loss of Air France flight 447 from Rio de Janeiro to Paris in 2009. The primary flight control system disconnected because of icing of the airspeed sensors and the pilots had to use a back-up fly-by-wire control system with much reduced capabilities and in particular without angle-of-attack protection which was a major feature of the primary system. They exceeded the allowable angle of attack, leading to a high altitude stall from which they could not recover.

3.2. A. Switching Redundancy

The general structure of a fault tolerant system using switching redundancy is shown in Figure 2. Two channels, A and B, furnish equivalent outputs. The channels are usually identical but they need not to be so. Advantages and disadvantages of non-identical channels are discussed later. The system information flow is shown by the solid lines. The monitoring data flow is shown by the heavy broken lines. The monitors operate on the output of each channel. In the simplest case they just look for data changes to determine that the channel has not failed completely. More typically they operate with assertions that look for data patterns appropriate to the current operation being carried out by the channel. When either monitor detects a failure it communicates with the monitor logic that then causes a switchover to the other channel. When the monitor logic receives simultaneous switching requests from both channels it can be programmed to alert higher level fault tolerance provisions or it may ignore the requests because they may be caused by faulty assertions. Sometimes the monitor logic is combined with the individual monitors. A frequent variant of this scheme is to monitor the fault tolerant output rather than the individual channel outputs. In this case, the monitoring may take place at a system output (such as motor speed) rather than computer output. If the failure is in the motor or its power supply, switching to the alternate computer channel will not fix the problem. No form of output monitoring can furnish information about the health of the initially inactive channel and that information must be obtained by separate means.

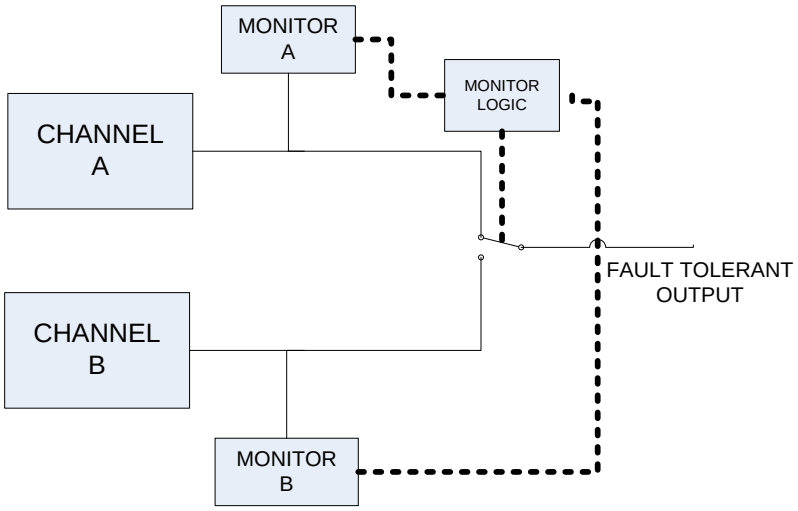

Figure 2 Redundancy with switching

Comparing the outputs of channels as shown in Figure 3 covers a broad range of failures and makes it unnecessary to define the parameters to be monitored. The disadvantage is that failure of comparison only indicates that the channels differ but does not provide any information about which channel has failed.

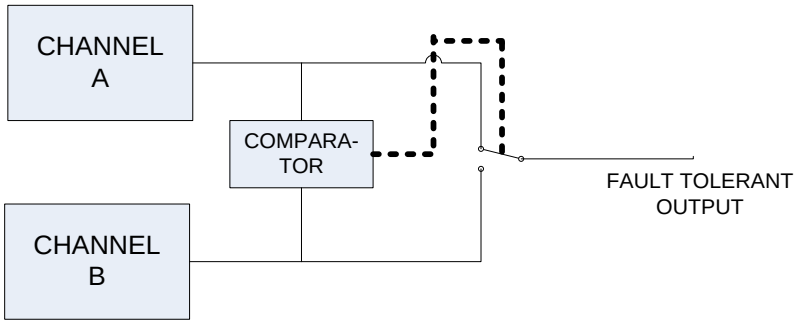

Figure 3 Failure detection by comparison

This deficiency can be overcome by combining comparison with output monitoring. If the channels disagree and the output is within the normal range it may be concluded that the inactive channel has failed. If the output monitor indicates an abnormal condition that indicates failure of the active channel and switching to the other channel is initiated. Another approach to dealing with the ambiguity of failure identification is in the pair-and-spare configuration shown in Figure 4.

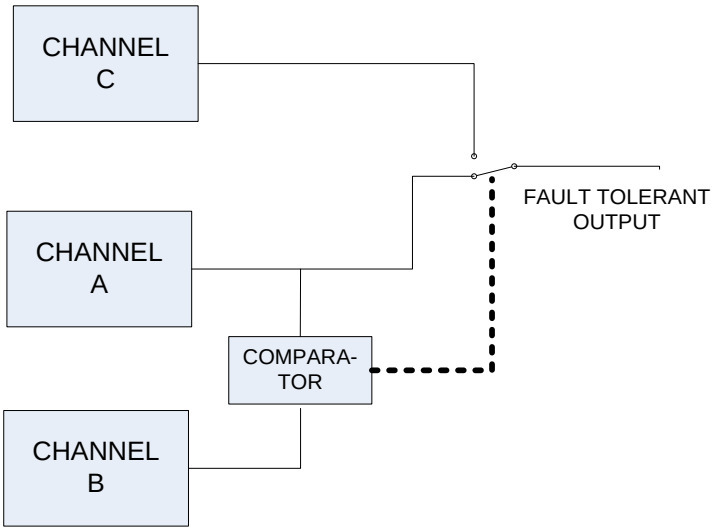

Figure 4 Pair-and-spare

When the comparison fails both channels (A and B) are made inactive and channel C is switched in. To be useful in safety-critical application this approach requires assurance that channel C is in an operational state while it is inactive. If this latter condition is met the pair-and-spare architecture is attractive because it provides broad failure detection and a simple switching arrangement.

All of the configurations mentioned here can be expanded to more than two channels. In particular, the pair-and-spare and be expanded to a pair-and-spare-pair by adding a channel D that is compared to C similar to the A-B comparison. Adding more channels increases the cost and maintenance requirements while offering only modest benefits in reliability. It may be considered where immediate replacement of a failed channel is difficult such as in remote locations or on board satellites.

3.2. B. Fault masking redundancy.

The simplest form of fault masking redundancy is represented by two dc power supplies feeding a common load as shown in Figure 5. In the fault-free state the two supplies share the load. If supply A fails, B becomes the sole power source (assuming that it is designed to handle the full load). The diodes prevent internal short circuit failures from affecting the output to the load. This arrangement is called fault masking because the load is not made aware that a failure has occurred.

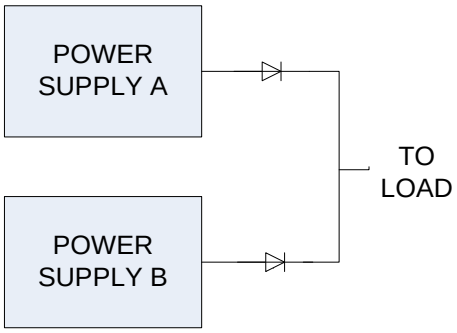

Figure 5 Fault masking power supplies

A more general fault masking architecture, also referred to a triple modular redundancy or TMR, is shown in Figure 6.

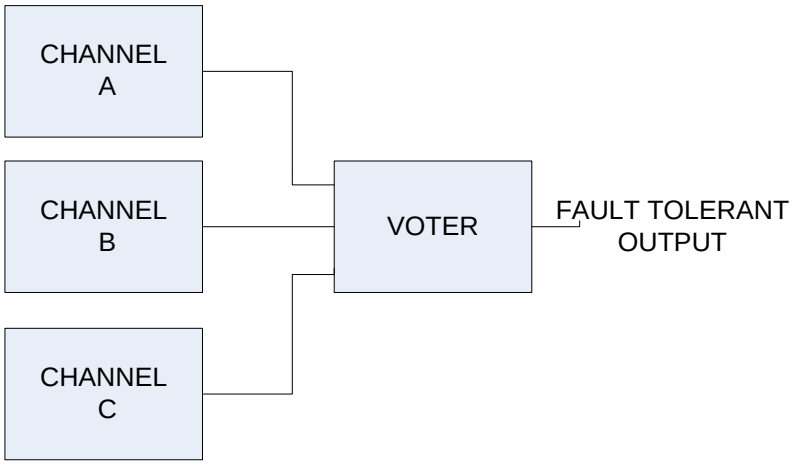

Figure 6 Generic fault masking architecture

The voter can compare the output of any two of the channels and if they agree pass this result as the fault tolerant output. If they do not agree, another two outputs are compared and if they do not agree then a third pair. If there is a single faulty channel there must be one pair of outputs that agree. When there is a failure the execution of this logic can be time consuming but it is usually tolerable in fault-free conditions. TMR is a fault masking architecture because the output is not aware of the occurrence of a failure. In practice it is desirable to know when a failure has occurred in order to replace the faulty channel as soon as possible; the required detection capability is easily incorporated into the voter. The voter is multiple orders of magnitude simpler than the individual channels and it can be constructed using a robust semiconductor technology. Therefore its failure probability is frequently neglected. Where this is not acceptable a fault tolerant voter can be used. One implementation is dual rail logic in which the channel outputs and their digital complements are submitted to separate voters, one representing the negative logic of the other[5].

3.2.C Fractional Redundancy

When one spare element can take the place of one of several active elements of the same type it is called fractional redundancy. Another term in wide use is k-out-of-n redundancy. The omnipresent example is that we carry one spare tire that can replace any of the four mounted (active) tires. Fractional redundancy is highly cost effective and is therefore widely used. In electronic systems typical applications of fractional redundancy are in registers, memory arrays, and I/O units.

The switching required for the replacement of a failed element with a spare can be implemented by conventional means. But preparing the replacement element for service may present a problem. If the failed element contained program or static data the replacement can be loaded from bulk memory. But if the memory contained data that had not been backed up the service may be temporarily degraded, e. g., until a filter reacquires past performance data. Fractional redundancy is therefore avoided where back-up data is not available and where a temporary degradation of service cannot be tolerated.

While the typical use of fractional redundancy employs switching some uses with fault-masking methodology can also be found. On of these is in fault masking power supplies of the type shown in Figure 5. The weight of power supplies is roughly proportional to their output power. Where weight must be minimized, e. g., in space vehicles, it is economical to provide the required power with two half-sized supplies and add an additional half-sized unit. In this way full power can be supplied after any one failure and the weight is only (roughly) 1.5 times that of a full-sized power supply.

Another example of fault-masking using fractional redundancy is in inertial instrument clusters for spacecraft and aircraft. The conventional arrangement has three instruments along orthogonal axes. A redundant set therefore requires six instruments but it is possible to arrange four instruments non-orthogonally (typically along the axes of a tetrahedron) such that orientation and acceleration can still be computed after failure of any one instrument is the cluster. The analysis and computation for recovering orthogonal values is non-trivial and has been the subject of graduate courses in computer science and navigation. The initial stimulus for these studies came from the space environment but the technique has also been employed in current aircraft navigation systems to avoid the cost of either duplicate sensors or grounding an aircraft until a replacement can be installed and calibrated.

3.3.D Partitioning for redundancy

The systems of concern are composed of several components and it is possible to implement redundancy either at the system or at the component level. The advantage of system level redundancy is that a single failure detection and replacement mechanism suffices. Advantages of component level redundancy are:

- Possibilities of applying fractional redundancy
- Smaller bulk of the replacement elements
- Higher reliability for the same amount of installed equipment

The disadvantage is the greater complexity of fault identification and recovery.
The concept can be understood with an example of a computer that consists of an electronic unit and a disk drive. If the computer is made redundant as a unit, all service is lost after a second failure. The alternative is to make the electronic unit redundant and the disk drive redundant. The system will remain operational after a first failure in one of the electronic units and a second failure in one of the disk drives but each potential failure must be separately monitored.

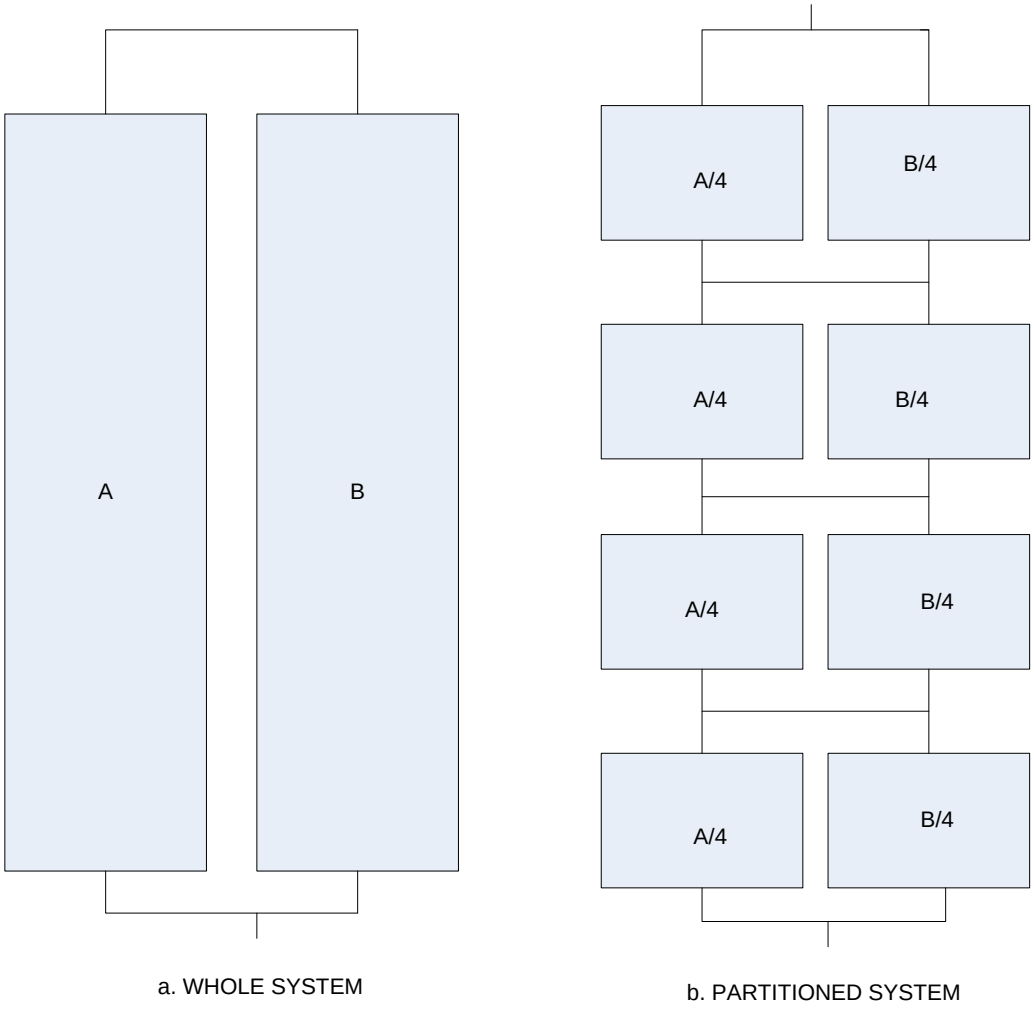

Figure 7 Whole and partitioned redundancy

Figure 7 shows two hypothetical configurations for a quantitative analysis of the reliability benefits of partitioning. In part (a) of the figure the two blocks A and B each represent identical whole systems. In part (b) each system is divided into four components each of which accounts for one-quarter of the failure rate of the system. Neglecting the failure probability of the fault detection and switching provisions, the failure probability of the whole system

$$F_W = f^2$$

where f represents the failure probability of either block (A or B). Using the same notation the failure probability of the partitioned system is

$$F_P = 4 \times (f/4)^2 = f^2/4 = F_W/4.$$

The failure rate advantage of the configuration in part (b) of the figure increases greatly as the failure probability f increases as shown in Figure 8. Assuming that the failure probability is a linear function of time this relationship can be interpreted as partitioning being very advantageous for environments that require long periods of operation without maintenance. It is not surprising that partitioning has been widely practiced for space missions.

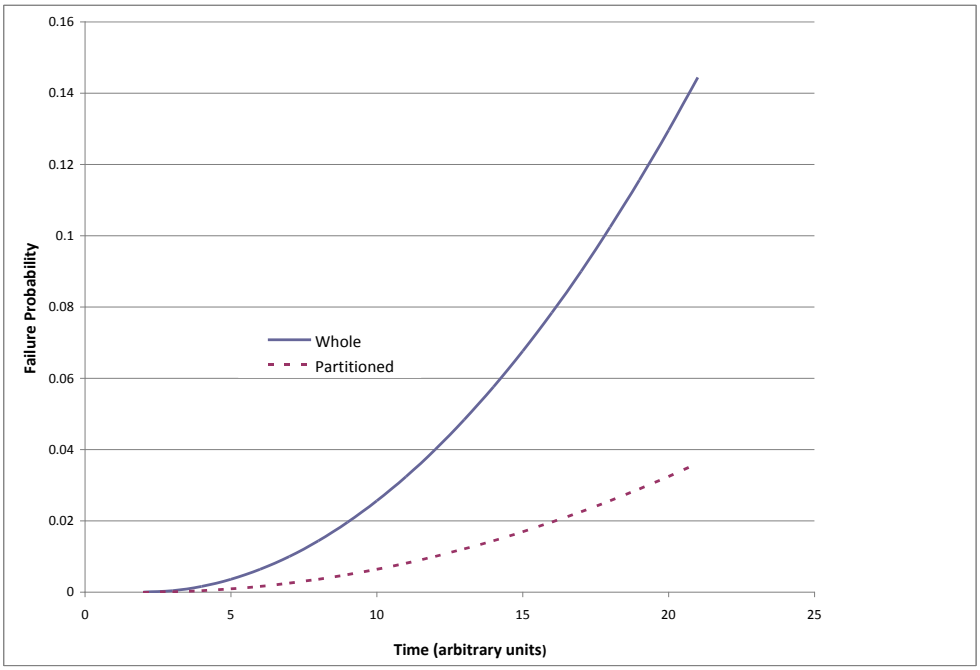

Figure 8 Effect of partitioning on failure probability

3.3.E Redundancy with diverse elements

In the previous discussion of redundancy it has been assumed that failures are due to random causes, and this is largely true for electronic components. But even a small deviation from this assumption can cause correlated failures to reduce the benefits of redundancy. Correlated failures can be due to

- Design deficiencies
- Manufacturing defects in a particular batch of product
- Sensitivity to failure due to the environment (heat, cold, humidity, vibration)
- Wearout, particularly in power semiconductors, or electromigration

The effect of non-random failures on the reliability of a redundant configuration is shown in Figure 9. The curves are constructed from the relation

$$F = 2 \times f \times (1 - RAND) + RAND \times f^2$$

Where F is the failure probability of the redundant structure, f is the failure probability of one of the redundant elements, and $RAND$ is the fraction of failures due to random causes.

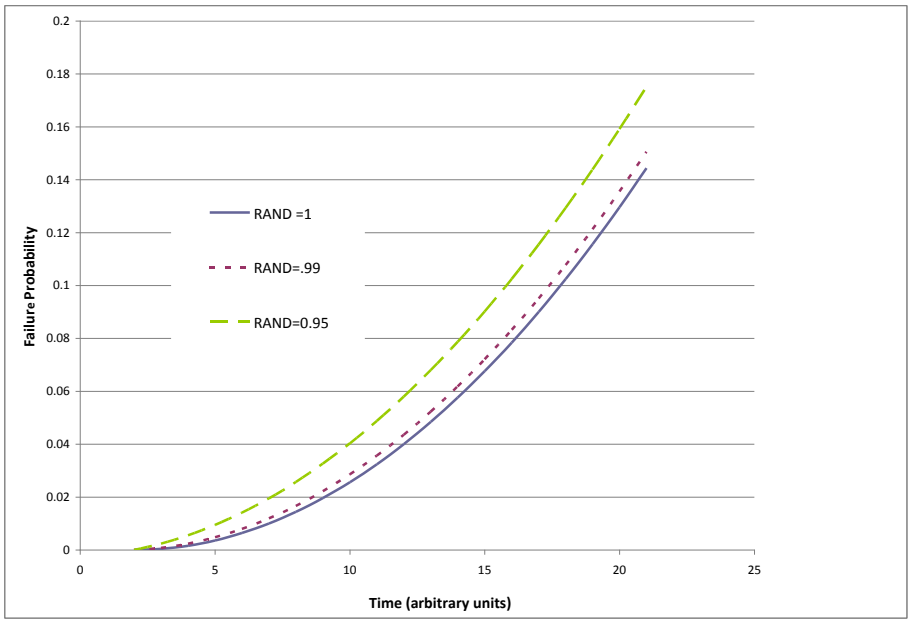

Figure 9 Effect of non-random failures

The likelihood of correlated failures can be greatly reduced by use of diverse elements but this incurs a high cost due to the need for separate

- specifications and design
- test procedures and test fixtures
- spares and maintenance provisions
- training of operating and maintenance personnel

Thus it is employed only in situations where there is particular concern about the possibility of common cause failures. This can be the case where the environment is not completely understood (e. g., deep sea and space), particularly where human safety is at stake. As seen in Figure 9 the effect of non-random failures increases with time. The need for employing diverse elements can therefore be reduced by facilitating timely replacement of failed elements which resets the time axis to zero.

4. **Both (Hardware and Software)**

This heading covers fault tolerance techniques that can be used to protect against faults due to hardware or software elements. The techniques themselves depend mostly on software.

4.1. Assertions

The general form of an assertion is

If <condition> then <action 1>
Else <action 2>

The *condition* can be an expected or desirable state in which case *action 1* is to permit program execution along the normal path and *action 2* tranfers to a recovery routine. The *condition* can also represent an undesirable state or a state that would be encountered only in case of a failure, in which case *action 1* transfers to a recovery routine whereas *action 2* permits normal execution of the program.

If the coverage of an assertion is deficient, leaving some failure conditions undetected, this is generally referred to as a type 1 error. The design of assertions must also avoid entering a recovery path when there is no failure which is generally referred to as a type 2 error. A frequent cause of the latter is "noise", an accidental data state that resembles a failure condition. This can be avoided by repeated sampling of the source of the data, a practice that is routinely employed where "noise" is encountered.

Protection against type 1 errors includes thorough understanding of the environment so that all states that represent the results of a failure are covered by the *condition*. Another protection is a hierarchy of assertions such that a faulty state that was not detected at a low level is detected (usually by assertions targeted at system states) at a higher level.

Because of their low resource requirements and wide applicability assertions are a front line tool of fault tolerance.

4.2. Repetition

"If at first you don't succeed, try again…" That is at the core of the use of repetition as a fault tolerance mechanism. Repetition is here employed as a sequential operation; parallel operation falls under redundancy as previously discussed. Repetition may be invoked as part of the recovery action of an assertion, in response to an error detected by a code or as a result of a hardware error detection mechanism in the computer. It may also be employed routinely for verification purposes, e. g., in a noisy communication environment.

Repetition by itself is effective in overcoming errors caused by transient conditions, such as a dip in the power supply, collision of data streams, or an overload at a computer

element. Repetition can also confirm that an anomalous condition is not due to a transient and requires the invocation of other fault tolerance provisions. Design decisions pertinent to repetition are discussed under rollback below. As with assertions, repetition is widely applicable and requires practically no hardware investment.

4.3. Prohibited operations

Some instructions (primarily in machine and assembler languages) are not needed for normal operations. When these instructions are accessed it indicates that an attempt has been made to execute a prohibited operation, and this can be used to cause an alarm to be raised. Similarly, an attempt to access a non-existing memory location can be considered as a prohibited operation and cause a similar alarm. At that point another fault tolerance mechanism can be invoked. Repetition is frequently used to see whether the cause or condition is temporary.

5. Software

This heading covers the detection of and recovery from faults that originate in programs or data.

5.1. Time-out

Some software errors, particularly when combined with unusual data values, cause loops to be executed indefinitely or route the program to a location that does not yield a response. These conditions can be detected by setting a count-down timer with the expected maximum time for completion of the segment. When the segment completes within the expected time the count-down timer is de-activated. If the timer is allowed to run to zero an alarm is raised. Similar time-out provisions can be applied if an operator or I/O response is expected within a prescribed time.

The response to a time-out can include

- Repetition as an initial measure and if necessary followed by others mentioned here
- Entering a back-up routine for accomplishing the same system goal as the aborted operation
- Entering a back-up routine to accomplish a degraded system goal
- Alerting the operator to decide on the response.

5.2. Checkpointing and rollback

A checkpoint is a step in a computer program at which a snapshot of data and computer states is obtained so that the program can be properly restarted at this point at a later time[6]. Rollback is the process of restarting at the checkpoint if the segment does not

terminate with an acceptable result. Checkpointing and rollback was implicitly used in previous headings, particularly in assertions and repetition.

One of the key design decisions is the length of the rollback segment. Selecting a long segment minimizes the switching overhead (the checkpointing and the evaluation at the end) but it leaves a possibly faulty result in the computer for a long time. Where the latter condition is not acceptable the higher overhead associated with a shorter segment length has to be tolerated.

5.3. Software redundancy

Sometimes a software defect that causes failure of a program on one computer (due to temporary conditions that are discussed under the heading of Repetition) may permit the same program running on a different computer to terminate successfully. Thus running the same program on several computers offers some protection from faulty software. But comprehensive fault tolerance can only be achieved by use of diverse software[7]. This incurs the penalties listed above for Redundancy with Diverse Elements (except that spares provisioning does not apply to software) and in addition it usually involves an increase in response time as will be described below.

The full benefits of diversity can only be achieved if programs employ a different approach from the ground up, use different design standards and are written in a different computer language. Where these rules are followed it is highly unlikely that all versions will terminate at the same time. The project manager has several choices

1. Accept the first program to terminate and depend on assertions to prevent unsafe actions. Further checks can be carried out in background mode when all versions have terminated.
2. Wait until all versions have terminated and proceed normally only if the results agree. In case of disagreements use voting (if more than two versions are used) or assertions to select the correct output.
3. Accept the output of the first program to terminate if it is within a narrow range of the previously accepted output; if not, wait as above.

6. Economics of Fault Tolerance

A key concept in the economic evaluation of dependability measures is that the incremental cost of an improvement in reliability should not exceed the expected benefit due to the reduction in outages. The following explores how this applies to fault tolerance in digital systems, first in formulating the incremental cost and then in estimating the expected benefit. At the outset let it be recognized that the data that are usually available seldom permit identification of an optimum fault tolerance provision but they may prevent gross mismatches between cost and benefits.

The cost of additional hardware and the associated installation is generally known at the outset. Every installed item will require power, maintenance, and environmental conditioning. In mobile applications the weight of the added equipment and its support may also have to be cost factors. In addition, all fault tolerance provisions will require

- Documentation
- Training
- Configuration control
- Monitoring of their performance

The benefit of the fault tolerance measure is the reduction in the expected cost of failure. Formally it can be computed for a given failure mode as

$$C_f = f \times Q_f$$

where C_f is the expected cost for that failure mode, f is the failure probability for that mode, and Q_f represents the total resources needed to deal with the failure, given that it occurs. Where a fault tolerance mechanism protects against more than one failure mode the expected cost of failure is the summation of the above expression over all covered failure modes.

The resources required to deal with a failure, designated Q_f above, include at least the following

- Consequence of outage
    - Loss of service
    - Damages (spoiled product, customer dissatisfaction)
    - Safety impact (actual or potential injuries, long term health effects)
    - Environmental impact (spillage, overflow, gas release)
- Cost of restoring the failed function
- Cost of failure reporting (e. g., to a licensing agency); this can be significant particularly in the medical field.

The first bullet has been expanded because it is frequently the governing cost element in assessing the cost of failure. The safety and environmental consequences may be rare but their cost can be huge and this causes responsible management to spend more for fault tolerance than would be justified by the formal evaluation.

7. Conclusions

Fault tolerance is an essential methodology for digital systems, particularly for those that serve applications where failure has safety implications or where interruption of operations imposes serious financial penalties. The preceding has shown that there is no

single fault tolerance technique that serves in all circumstances. Key factors in selecting suitable techniques include

- The type and frequency of faults to be tolerated
- The consequences associated with each type of fault
- The available resources
- Familiarity of the designer and project management with a given fault tolerance technique

Once a tentative fault tolerant design has been adopted it should be subjected to review. Gaps in coverage and deficiencies of an individual technique can be overcome by employing a hierarchical structure of fault tolerance provisions, also referred to as defense-in-depth. The large selection of techniques that have been described and the continuing improvements provided by studies in the field support an encouraging outlook.

References

[1] IEEE Computer Society, *FTCS-25, Highlights from 25 Years,* June 1995, Pasadena, California

[2] Fiala, Mueller et al., Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing, *Supercomputing Conference 2012 (SC12),* November 2012, Salt Lake City UT.

[3] David J. C. McKay *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, September 2003.

[4] Moon, Todd K. Error Correction Coding: Mathematical Methods and Algorithms. Hoboken, NJ: Wiley-Interscience, 2005

[5] Krishnamurthey, R. K. and L. R. Carley, Exploring the design space of mixed swing quadrail for low power digital circuits, *IEEE Transactions on VLSI Systems,* December 1997, pp. 388-400

[6] Yibei Ling, Jie Mi, Xiaola Lin: A Variational Calculus Approach to Optimal Checkpoint Placement. IEEE Trans. Computers 50(7): 699-708 (2001)

[7] Van den Brand, Mark and Friso Groote, "Software Engineering: Redundancy is Key", www.win.tue.nl/~jfg/articles/RedundancyIsKey.pdf, accessed 26 May 2015