Journal of the KSQC
Vol.21, No.1, June 1993

1993년 6월 韓國品質管理學會誌 제21권 제1호

# A study on the Design Techniques and Analysis of Fault-Tolerant Computers

Jai-Rip Cho*

## ABSTRACTS

The art of designing and analyzing fault-tolerant computers is surveyed with special emphasis on problems of analyzing the behavior of computers that have autonomous repair capability. The survey covers the following topics : (1) general issues in computer reliability, (2) fault-tolerance state relations and requirements, (3) computational hierarchy, (4 fault characteristics, (5) fault diagnosis, (6) fault-tolerance schemes for logic network and machines, (7) fault-coverage effects, and (8) fault-tree analysis of coverage.

This paper does not include techniques for verifying nonredundant hardware or system software designs or for verifying the correctness of application programs.

## 1. Introduction

Considering their complexity, modern computers and remarkably reliable. Nevertheless, the application in which extreme reliability is essential are growing in number and variety.

* Department of Industrial Engineering, Kyung-Hee University

The reliability required by these applications cannot be achieved merely by diligent application of standard computer engineering practice, i.e., designing conservatively and using quality components, with minimal redundancy. Thus, in recent years are being designed with some from of built-in reliability enhancement. The general object of this enhancement is to enable the computer to tolerate faulty components. Some schemes for fault tolerant through some form of redundancy date from the earliest days of the modern computer era. Simple forms of these and more recent schemes appear in current machines but many difficult thoretical and problems remain to be solved that will require more complex uses of redundancy.

Not the least of these problems is the analysis of the effectiveness of proposed fault-tolerance mechanisms. A computer with a high order of reliability enhancement may exhibit extremely complex behavior under fault conditions. Even for nonredundant computers, it may be extremely difficult to determine whether some program function of interest will be a given hardware malfunction.

In this survey of fault-tolerant computer design and analysis, we will discuss the nature of faulty behavior in computers, and describe a variety of approaches to fault tolerance. We then discuss the deficiencies of current analysis techniques and indicate some open problems of reliability analysis.

We do not survey work on program correctness. From the viewpoint of someone who sees the computer as a component in, says, a power plant, this may seem to be a vital omission. We offer two justifications for this omission. First, failures in programs and in machines are different in kind. Program failures are the result of human design errors, while machine faults are departures from designed behavior that result from an uncontrollable physical event. Second, the technologies for coping with these two kinds of faults are quite different, and it would be impractical to survey both technologies in the present context.

The distinction between application and computer-system functions is convenient, but it is by no means immutable. At the highest system levels, there is a recognized need for systems that are "rugged" with respect to computational errors regardless of their origin [ 3 ]. At the lowest system levels, hardware design mistakes may be extremely difficult to discover prior to production and installation. Some of these mistakes can cause operational errors that are indistinguishable from those induced by transient physical faults.

There is a very active body of research on program correctness [ 23 ]. It is hoped that in the future the results of this research will be factored into a general reliability model for computer systems. Modern computer engineering attempts to deal with harware and software in a unified way, at least up to the executive level, but the integration of application programs, together with hardware and executive system software, into a single reliability model is beyond the present art. Therefore, for this survey, we defind the problem of computer reliability to be the assurance that application programs are provided with computing facilities that perform their originally specified functions when physical malfunctions occur.

The survey will cover the following topics : (1) general issues in computer reliability. (2) Fault-tolerance state relations and requirements. (3) computational hierarchy. (4) Faullt characteristics. (5)Fault diagnosis. (6) Fault-tolerance schemes for logic networks and machines. (7)Fault-coverage effects. (8) Fault-tree analysis of coverage. Several other surveys of fault-tolerant computing are available [ 21 ].

## 2. General Issues in Computer Reliability

Much the same language used in discussing the reliability of general physical systems can be applied to computer system. Thus, a computer may be described as a complex interconnection of distinct components, subject to a variety of fault types, and capable of achieving some level of fault tolerance by means of various forms of redundancy. There are, however, some extraordinary and even unique aspects that lend a special character to problems of computer reliability.

Computers contain some conventional electronic and mechanical components that are subject to straightforward reliability analysis, but the major components(logic elements) have troublesome reliability features. For example, logic components are unhealable. Furthermore, in many instances, their reliability cannot effectively be ascertained. So many logic components are used in a system(from $10^2$ to $10^5$ devices) that their reliability must be at least as good as $10^{-6}$ failures per hour to ensure any useful service at all. It is not hard to test them to this level, but testing to $10^{-1}$ becomes very expensive in the number of components tested and the duration of the test. That level of experience is available for mature technologies, but the demand for ever faster and cheaper components drives manufacturers to use new logic technologies, for which such experience is not available. Consequently, component test experience is seldom adequate to justify estimates of component reliability better than $10^{-6}$ to $10^{-7}$ failures per hour, even though the true reliability may be much better.

In many electronic systems, the components are far less reliable than the wires and connectors that join them. This is not the case with computers. Semiconductor fabrication is standarized and highly automated, but the interconnections are highly varied and require much hand work. They are also highly vnlnerable to accidental damage during maintenance. One of the main automations for using Large Scale Integration technology(LSI), beyond its low cost, is that by bringing more logic elements(gate) into a device the number of interdevice connections can be greatly reduced, thus further reducing system cost and increasing reliability.

Another characteristic of computers that is crucial for reliability devices from the vast number of discrete states and the character of the changes in state. Even excluing their

memory subsystems, computers may contain $10^2$ to $10^3$ to binary storage elements, allowing $2^{100}$ to $2^{1000}$ system states. The rate of the state change is high (on the order of $10^8$ bits / second) but even more significantly, there is no "inertia" involved in the state changes, as there is in physical systems. Thus, a fault within the state variables that control program sequence could throw a computer from one program state (step) into a wholly unrelated program state from which, in general, there would be no recovery.

Computer malfunctions are highly varied in character. A system may fail due to single or multiple component failures, and the multiple failures may or may not be independent. A component may fail permanently, or it may suffer a transient malfunction due to an environmental cause. Some of these environmental causes (e. g., stray single coupling) may result from particular configurations of data that may be very rare and difficult to reproduce, and furthermore may be indistinguishable from random physical disturbances.

Many different strategies for accommodating faults are employed by computer designers, although most systems provide no explicit accommodation at all except for what may be programmed by the user. Some systems have built-in elements that detect errors automatically or that may be used to run diagnostic programs. A few system, real and paper, have built-in capability for automatic correction of errors due to a malfunction. In some cases, the correction is structural and passive, as in a series or parallel connection of resistors or diodes. In others, the correction is achieved through a dynamic process of error detection, diagnosis, correction, and recovery. This process must be carried out by equipment that is itself subject to failure. The system thus tolerates faults through a kind of (possibly imperfect) self-knowledge. Some automatic control systems may be considered to have some limited "self-knowledge", but uniquely among fault-tolerant systems, the self-knowledge in a fault-tolerant computer can include any computable function (a class enormously large than the class of passive or switch-redundant functions) and consequently has potentially much greater power for self-repair.

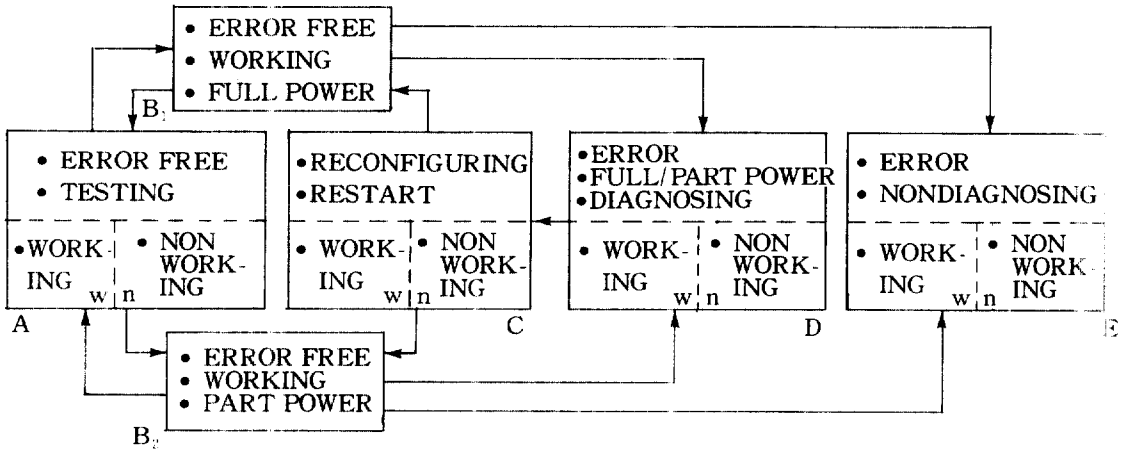
Dynamic fault tolerance is the most powerful kind of self-repair, while it is very difficult to analyze, it is essential for some applications. The review emphasizes this mode of achieving reliable computing.

## 3. Fault-Tolerance State Relations

This section presents a general model of fault-tolerant computers. The model establishes terminology that is used later for a more detailed discussion of computer reliability in a general and dynamic fault-tolerance in particular. It also discusses the ranges of quantitative reliability requirements in various applications.

Figure 1 represents the reliability states and state transitions of dynamic fault-tolerance

computer.




| | | | |
|---|---|---|---|
| 1. Erroneous Results | BE or BD | 4. Unavailability | BA or $BD_n$ $C_n$ or $BE_n$ |
| 2. Late Results | BDCB | 5. Checkout | BA |
| 3. Degradation of Computing Power | $B_1DCB_2$ | | |

Fig. 1. Fault-Tolerance State Relations

Various reliability characteristics may be identified by particular trajectories through the state diagram, e.g.,

| Reliability Characteristic | State Sequence |
|---|---|
| Erroneous results | (B, E) or (B, D) |
| Late results | (B, D, C, B) |
| Degradation of computing power | ($B_1$, D, C, $B_2$) |
| Unavailability | (B, A) or (B, $D_n$, $C_n$) or (B, $E_n$) |
| Checkout | (B, A) |

These characteristics may be quantified appropriately as probabilities or time durations.

A unique quality of computer reliability is the enormous quantitative range of requirements placed on these characteristics described pertain to the computer itself. When the computer is used in a control system, it may be important that the system be fail safe or fail soft with respect to computer failure. In fail-safe behavior, some system states are not to occur under any fault condition, perhaps at the price of losing some valuable computational service. In fail-soft behavior, a fault may degrade service but eliminate it completely.

## 4. Computational Hierarchy

A major reason for the success of computer technology has been the practice of organizing computing processes hierarchically. This practice has been intensified in recent years. Table 1 lists the levels of modern computing process hierarchy [15]. For each level, a characterization is given of the consequences of a malfunction, in terms appropriate to the process at that level. The ordering of the levels(especially near the top)is not unique.

Table 1. Hierarchy of Computational Processes

| Computational Level | Consequence of Faults |
|---|---|
| Human Control | Improper data or control input, inadequate maintenance |
| Fault-handling Executive | Inadequate or slow fault detection, recovery, repair |
| Multi-program Executive | Faulty program sequence and linkage, incorrect resource allocation, incorrect input-output, loss of security, interprogram interference, loss of data |
| Application Program / Data | Erroneous computation, blockage, loss of data |
| Instructions / Addresses / Operands | Erroneous results, wrong program state |
| Processors / Memory System / External Interface | Stuck, erroneous or free-running computing functions |
| Logic Modules / Memory Modules | Errors in complex logic and memory functions |
| Gates / Memory sensing and Selection | Errors in elementary logic and memory functions |
| Power Supply / Timing Circuits | Permanent and transient errors in gate and memory signals |

For example in some systems, Application Programs belong more naturally above the Executive. We note that : (1) A Fault at any level may be manifested as a higher level, and (2) A fault may be accommodated by some corrective process either at its own level or at a higher level.

The theory of computational hierarchy is the subject of several current investigations [17] An important problem is the formalization of computation as a hierarchy of abstract machines. Such formalization should facilitate the analysis and proof of the fault-tolerance behavior of complex systems.

## 5. Fault characteristics and Fault Diagnosis

Figure 2 displays the major fault mechanism as they are manifested at the logic element level of the computer hierarchy [9]. The description abstracts the effects from the actual

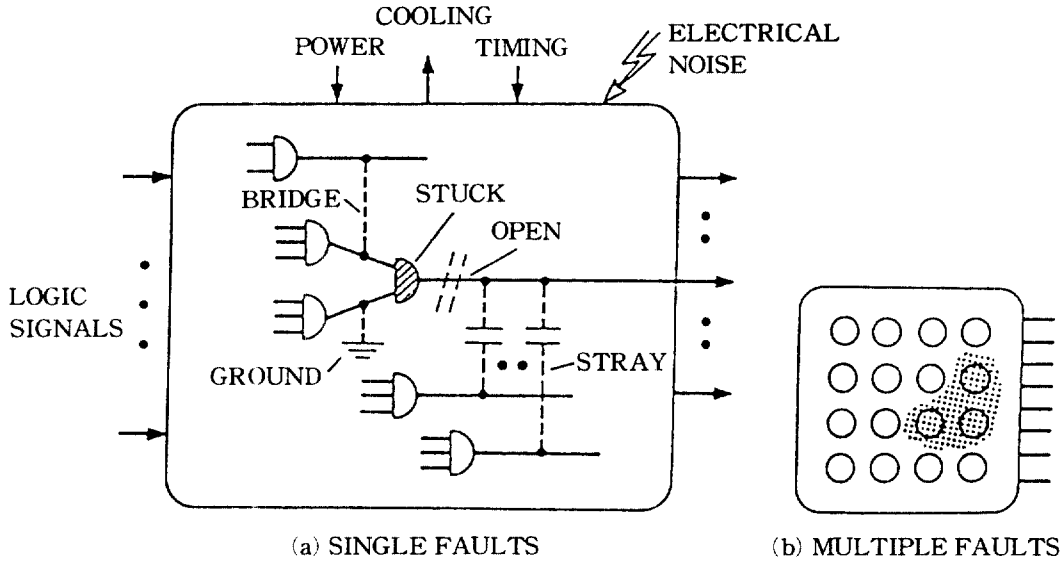



Fig. 2. Faults in Logic Networks

physical causes, which may include electromagnetic disturbances and physical breakdowns of various kinds, such as chemical contamination and mechanical breakage.

The faults (illustrated in Figure 2a) include open circuits, grounds, and direct or capacitive bridging faults in interconnections ; logic elements that are stuck to some state or that realize some logic function other than intended ; and logic errors induced externally by electrical noise, defective power supply, or defective thermal or mechanical control.

Method for analyzing faults in logic networks have been studied extensively [ 4 ]. The primary motivations have been (1) Acceptance testing of logic modules at manufacture, (2) diagnostic testing in support of system maintenance, and (3) automatic diagnosis of self-repairing computers.

A useful Classification for logic networks is by their capability for storing information. Combinational networks not store information except in distributed circuit capacitance during single transients. Sequential networks are designed to store information. In general, sequential networks have a state behavior such that the next state is a function of the present network input.

Initial work on this problem dealt only with the external behavior of a network as specified by its state table [ 11 ]. This approach is effective, independently of fault type, but only for small networks. For networks of practical interest, e. g., 20 state variables, the cost of generating complete test sets by behavior methods is prohibitive. Effective methods have been developed that exploit knowledge of the internal physical structure of the network,

given reasonable assumptions about the class of likely faults.

Successful methods have been developed that treat a sequential network as a spatially iterated combinational network [ 2 ].

Another difficult problem arises in the testing of networks that have been designed to mask single faults. It is desirable to determine if any fault has occurred so that one might replace the network in anticipation of a second, perhaps lethal, fault [ 6 ].

There are some theoretical results on the generation of minimal test sets for small, constrained combinational networks [ 24 ]. There are no techniques that can guarantee detection of all double faults in combinational networks or even all single faults in sequential networks, but practical, effective test generation methods exist for networks of up to 500 gates having arbitrary structure [ 22 ]. Given the present testing art, it seems essential that large networks be structured specially so as to facilitate testing [ 7 ].

In large low-redundancy systems, the down-time depends on the speed and precision of fault-location procedures. There is a great need for more powerful diagnostic programs [ 12 ].

## 6. Fault-Tolerance Schemes for Networks and Machines

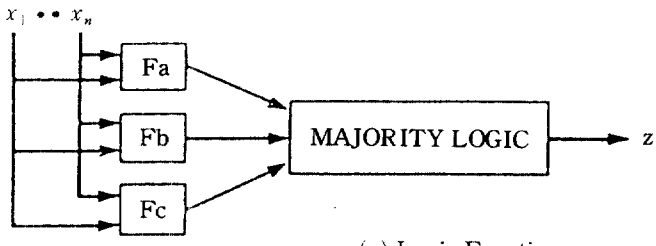
This section surveys some representatives of the large number of fault-tolerance schemes that have been developed for several computational levels. The schemes are presented in order of increasing complexity of dynamic behavior. The references are representative of an extensive literature on the reliability analysis of fault-tolerant computers.

### (1) Passive Fault Masking

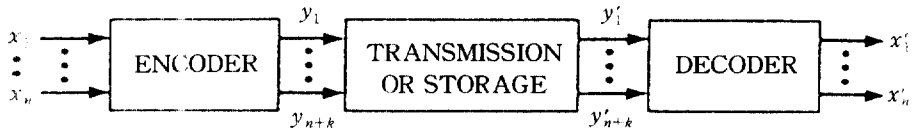
In the early, one of the pioneers of modern computing, John von Neumann, Proposed that arbitrary reliability might be realized from faulty logic units by using an ensemble of identical units to represent a single logical variable. The value of the variable is determined on the basis of majority vote. A fault in one of the units might be said to be "masked". Figure 3a shows a popular triple-redundancy scheme.

If the reliability of the voting itself be replicated [ 8 ]. In such a scheme, each logic module in the nonredundant network is replaced by a module with three inputs per logic variable and three nominally identical outputs per logic function. The redundant system will cost about four times as much as the nonredundant system. Symmetric faults(false zeros and false ones equally probable) in arbitrary logic functions cannot be masked with less than three-fold redundancy or detected with less than two-fold redundancy :

Two serious limitation of passive fault masking in large systems are (1) it makes very




(a) Logic Functions




(b) Transmission or Storage Functions

Fig. 3. Passive Network Fault Masking

inefficient use of redundant elements, e. g., a double fault in only a single module can cause systems failure, at which time all redundant modules become worthless, and (2) it greatly increases the number of interconnection, an important source of failures. A third serious limitation to this in modern technoligical realizations is its vulnerability to nonindependent multiple faults.

Figure 3b displays the use of error-detecting cause for fault tolerance in data transmission or data storage functions. Communication-type codes may be used but special codes have been developed to meet the need for high-speed, parallel encoding and decoding. The cost of such systems depends on the memory word length, but is less than 1.5 times the cost of a nonredundant system. This scheme currently is widely used in large computers. Fault-tolerant codes for arithmetic and logical operations have been studied, and are advocated by some. Their benefit is controversial, because of the small cost advantage and possible sensitivity to coherent multiple faults [ 18 ].

### (2) Acting Fault Masking

Figure 4 displays a scheme, which combines fault masking and a simple form of dynamic fault tolerance. Here, the masking rule is selective, according to fault experience.

At any moment, three of the functional modules are connected to a majority element. When a module fails, its disagreement with its companions is noted and causes its replacement by a spare. The system start with $N=2k+1$ spares and may continue to

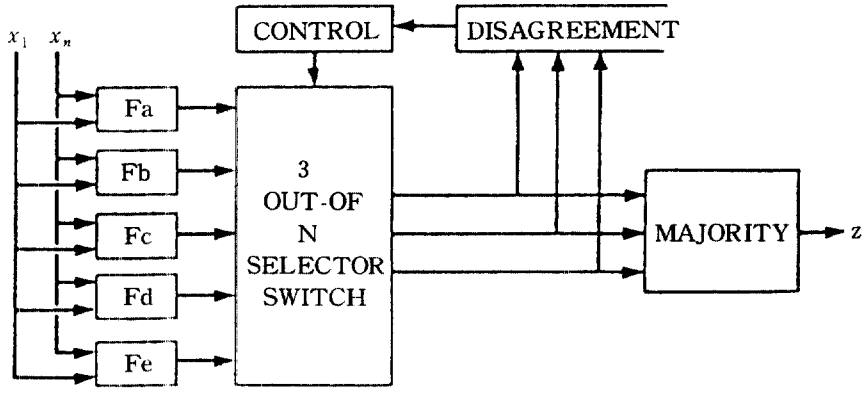



Fig. 4. Active Network Fault Masking

operate correctly after $2k-1$ failures, wheres a von Neumann scheme can survive only $k$ failures. Various implementations have been analyzed extensively [ 16 ].

### (3) Network Reconfiguration

Figure 5 displays a recently reported scheme for tolerating faults in an iterative array of memory modules [ 5 ]. The effective connections among the modules are reconfigured so as

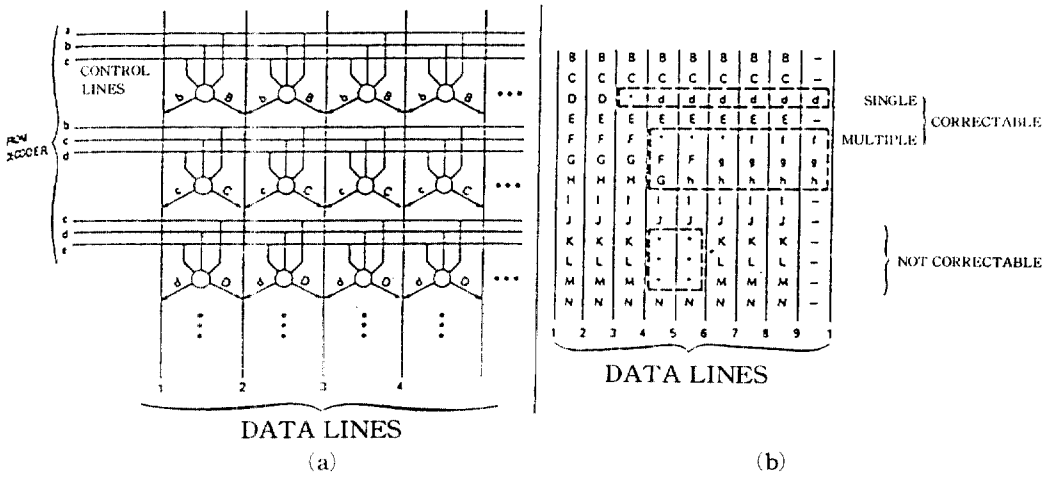



Fig. 5. Memory Network Reconfiguration

to bypass faulty modules. Figure 5b illustrates module output assignments that correct various fault patterns. Earlier similar schemes substitute whole rows or columns when any module in a row or column fails, but this scheme uses only one spare module for each fault experienced.

The cost for this economy is a more complex reconfiguration algorithm, and a reliability model which is more difficult to analyze than for row or column substitution.

### (4) Self-Repairing Computer Systems

The inefficient use of redundant resources, the heavy burden of decision circuits, and the increase of fault coherence in modern device technology have led designers to apply fault tolerance at a level of machine organization higher than logic networks.

About 20 years ago, several fault-tolerant computer systems were developed that were based on the protection of functional subsystems such as the memory subsystem and arithmetic unit.

The STAR computer, illustrated schematically in Figure 6, was designed for long unattended life in space without degradation [ 1 ].

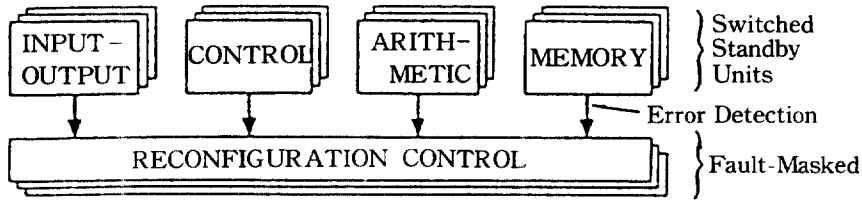



Fig. 6. Self-Repair Computer with Stand-by Module Substitution-Central Control

The system provides a set of spares for the standard subsystems, which are applied under the control of a special logic unit. Various codes are used to facilitate detection of an error condition, and special logic unit determines whether or not a spate should be switched to replace a suspected operational unit. The control unit itself is protected by network fault masking redundancy.

The Bell ESS-1 (Electronic Switching System) employs dual redundancy for its various subsystems [ 10 ]. In this scheme, illustrated in Figure 7, the outputs of a pair of identical units are compared continually. Disagreement initiates a diagnostic process to determine which unit is in error.

This process is automatic and very effective in location faults, but some human participation is required in order achieve satisfactory resolution of faults to within a small

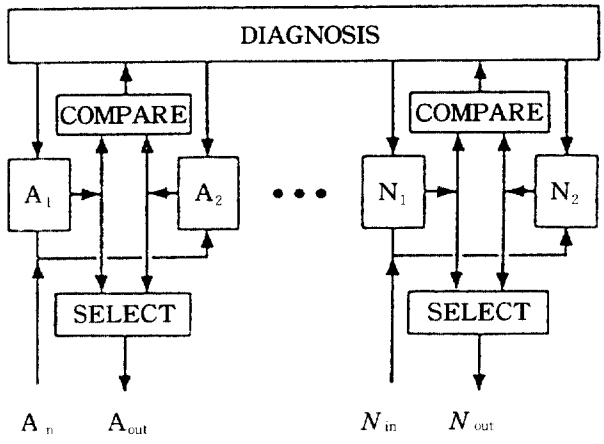



Fig. 7. Dual Redundant System

number of replaceable elements. The primary reliability objective is continuity of service, with minimal loss of existing subscriber connections.

### (5) Homogeneous Computer Systems

Using new digital technology, the cost of central processors has been so greatly reduced that it has become feasible to treat a processor as a basic units of reconfiguration.

This is the basis for several recent exercises in ultra-reliable computer architecture for aerospace applications and for the experimental, high-availability time sharing system.

One such scheme is the SIFT (Soft ware Implemented Fault Tolerance) design, illustrated in Figure 8. The major objectives of this design are : extremely reliable computation ($10^{-10}$ failures per hour), controlled degradation over a set of computations according to their priority rating, and easy of maintenance

The hardware consists of a set of conventional memory processor units, interconnected by a data bus. Various forms and degrees of redundancy may be realized by executive programs within each processor. An example of redundant computation is given in the " redundant validation " segment of Figure 8b. This is the scheme discussed in paragraph 6 (1) except that the connections are programmed. In the intended application, the computation may be preorganized as a set of task. These tasks are then assigned redundantly to the set of memory-processor units, as illustrated in the "configuration table" segment of Figure 8b.

The art of designing and certifying integrated hardware-software fault-tolerance systems and of analyzing their reliability is greatly in need of development. A major outstanding

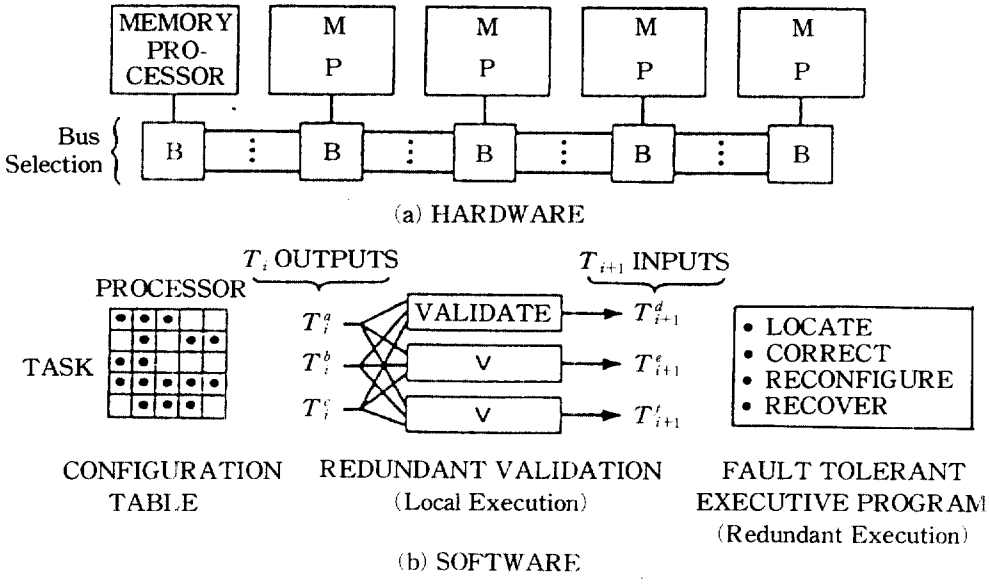



Fig. 8. Multiple Computer-Distributed Control-Software Implemented Fault Tolerance

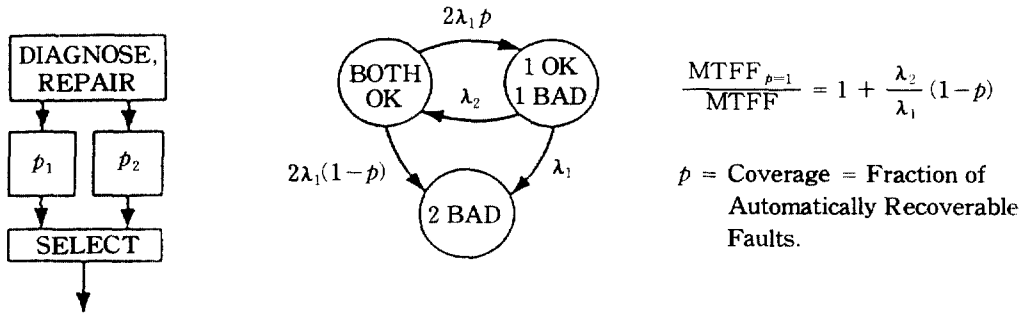
problem for fault-tolerant computer engineering is to develop realistic models of system-software and application program reliability, and combine them with hardware reliability models so as to provide a comprehensive model for the reliability of the computational function in general control system.

### (7) Coverage Effects

How good does built-in fault tolerance have to be? A naive view would hold that in as much as faults in modern computers are rare to begin with, a fault-tolerance mechanism of only moderate performance would be required in order for the system to realize the full potential of its redundancy. Bouricius et al. warned that this was a dangerous assumption when they introduced the concept of coverage, defined as that fraction of all possible faults in a self-repairable computer from which the computer can automatically [ 2 ].

Arnold, in a short but influential paper, quantified the role of coverage in several simple models of fault-tolerant system. His analysis of a dual-redundant system is illustrated in Figure 9. The system (Figure 9a) consists of two identical computing modules $p_1$, $p_2$, with common data input (not shown), subject to diagnosis and to repair.

Possible system states (Figure 9b) are "Both OK", "One OK, One Bad", and "Two Bad". The time to module failure and to module repair are assumed to be distributed exponentially. The mean-time-to-failure of each module is $1/\lambda_1$, the mean-time-to-repair of each module is




(a) DUAL SYSTEM (b) STATE TRANSITION RATES (c) MEAN TIME TO FIRST FAILURE

Fig. 9. Effect of Imperfect Coverage on Failure of Repairable System

$1/\lambda_2$, and the coverage is $p$.

Figure 9b shows the state transitions and their rates. From "Both OK", the fraction $p$ of the $2\lambda_1$ failure rate applies to transitions to the " One Ok, One Bad " state, since failures leading to this state are assumed to be recoverable. The remaining $(1-p)$ of the failure rate takes the system directly to the "Two Bad"(system failed) state. There are two exist transitions from the "One OK, One Bad" state. The first represents repair of the "Bad" module, which takes the system back to the "Both OK" state, at the rate $\lambda_2$. The second represents failure of the remaining good module, which takes the system to "Two Bad" at the rate $\lambda_1$.

The key issue is the relationship between $p$ and $\lambda_2$. Arnold derives the mean-time to first failure(MTFF) as $MTFF = \lambda_2/2\lambda_1^2 \times 1/1 + (\lambda_2/\lambda_1)(1-p)$.

For $(\lambda_2/\lambda_1)(1-p) > 1$, $1/MTFF \approx 2\lambda_1(1-p)$, *i. e.*, MTFF will be independent of $\lambda_2$ : that is, the system failure rate will be dominated by module failure and coverage, and not by recovery. Reasonable practical values might be : $1/\lambda_1 = 3$ months and $1/\lambda_2 = 2$ hours. Then $\lambda_1/\lambda_2 > 10^{-3}$, and for uncovered faults greater than one percent, MTFF will be insensitive to repair time.

Considering the enormous number of possible faults in a computer system, the requirement that fewer that one percent of them be uncovered by built-in fault-tolerance logic(itself subject to failure) is severe indeed. This analysis points to the essential role of coverage analysis in assessing the reliability fault-tolerant computer systems.

## 8. Reliability Analysis

When the state model of fault-tolerant systems discussed in paragraph 2 is considered, it is clear that there may be different reliability characteristics of interest, pertaining either to state occupancy or to state transitions. The objective of reliability analysis, then, is to estimate the probabilities of the various state transitions. In fact, existing work deals almost exclusively with the failure transitions.

Most studies of large fault-tolerant systems have used very simple models, borrowed from general engineering reliability practice [13]. This section will describe two recent developments that attempts to deal more realistically with multilevel and multimode systems.

### (1) CARE-2 and TASRA

CARE-2 [20] developed by the Raytheon Company after F. Mathur's CARE program [14], and TASRA, developed by the Battelle Institute, are programs for the evaluation of fault-tolerant system reliability on the basis of subsystem reliability estimates. CARE-2 is process oriented. It allows the analyst to describe the various phases of fault detection, fault isolation, and system recovery. TASRA is structure oriented. It allows the analyst to specify, for each subsystem in a fault-tolerant system, the logic of how failure modes of each input are combined to affect the failure modes of each subsystem output.

CARE-2 and TASRA allow the analyst to evaluate system fault coverage in terms of the probabilities of correct operation of relevant processes and subsystems at a relatively high system level. In principle, they could be extended to the most primitive fault events, but in a modern computer, they would be overcome by the complexity of detail. They offer no help in abstracting and classifying complex hardware-software interactions.

### (2) Fault-Tree Analysis of Coverage

As observed earlier, the number of possible fault in a modern computer is too great to be analyzed individually. The only hope is to classify them in a way that is related to the logical structure of the fault-tolerance process.

The fault tree appears to be a useful device for accomplishing this classification because it fits naturally into the hierarchical mode of describing computing processes. In this section, we illustrate a possible approach to the use of a fault tree to analyze coverage, and discuss some of the problems that will have to be solved to make it a useful tool.

The tree in Figure 10, constructed for a hypothetical self-repair computer, has an uncovered fault as its Top Event. The first level list classes of faults any of which lead to an uncovered fault, i. e., (1) An error undetected by the fault-tolerance mechanism. (2)

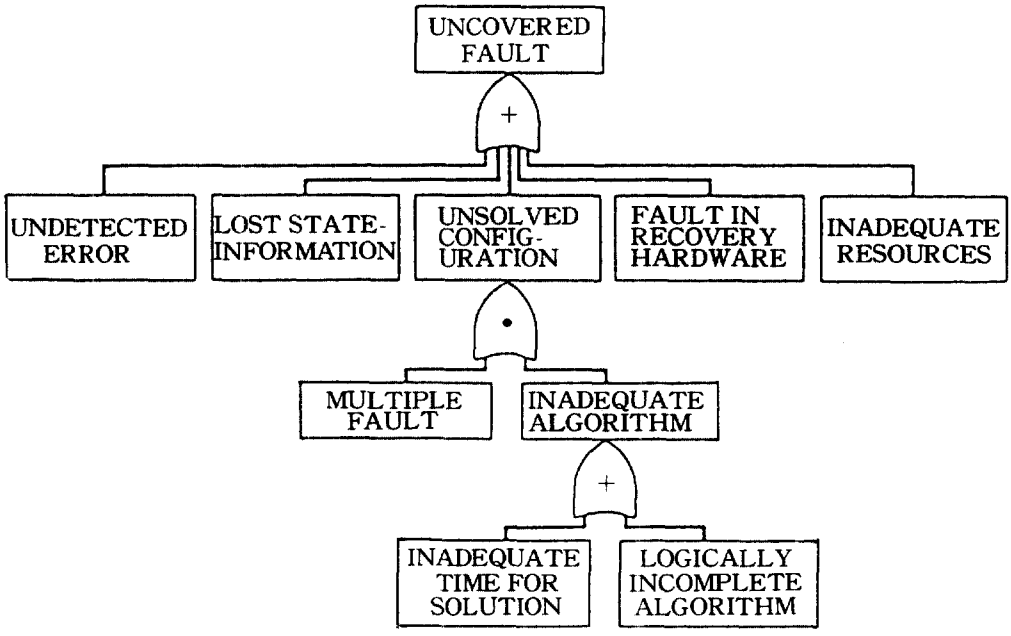



Fig. 10. Fault-Tree Analysis of Coverage

Information about the state of an application program that is lost during the error and recovery episode despite a possible successful recovery of computing capability. (3) A fault in the logical networks used by the recovery process. (4) Inadequate reserve computing resources. (5) A fault configuration that is unsolvable by the built-in fault-tolerance mechanism.

These categories each include a large number of faults, whose enumeration will require further decomposition. Figure 10 shows a decomposition for the "unsolvable configuration" fault type. The first level of this decomposition postulates the coincidence of a multiple-fault condition and a fault-tolerance algorithm that is somehow inadequate to that fault configuration (single-fault contributions to the Top Event probability assumed to be negligible).

The next level indicates ways in which the fault-tolerance algorithm may be inadequate : (1) The algorithm may simple not be powerful enough to solve the given fault problem, or (2) The algorithm may be too slow at solving the given configurations to meet a recovery deadline.

These conditions are still at a high level of abstraction, and require much further decomposition.

## 9. Conclusions

We have surveyed the major facets of computer reliability, with special emphasis on problems of analyzing the behavior of fault-tolerant computers with autonomous repair capability. For this survey, reliability is defind to errors in the basic(nonredundant) hardware or software designs, and in the application program. Within this scope the major immediate needs in fault-tolerant computer engineering are (1) more powerful method for analyzing the fault coverage of hardware-software self-repair processes, and (2) more effective algorithm for rapidly diagnosing multiple failures in large low-redundancy systems. Due to the great complexity of computer systems, it is clear that new analysis techniques must be accompanied by new methods for structuring systems for easy of analysis. The next major step will be to enlarge the scope of fault-tolerant computing to include application programs and other environmental factors.

## REFERENCES


1. Avizients, A. et al., (1971), "*The STAR (Self-Testing-and-Repairing) Computer : An Investigation of the Theory and Practice of Fault-Tolerant Computer Design*", IEEE Trans. on Computers, Vol. C-20, No. 11, pp. 256-270.
2. Bouricius, W. G., et al., (1971) "*Algorithms for Detection of faults in Logic Circuits*", IEEE Trans. on Computers, Vol. C-20, No. 11, pp. 250-258.
3. Fischler, M. A. and Firschein, O., (1974), "*A Fault-Tolerant Multiprocessor Architecture for Real-Time Control Applications*", Proc. of First Annual Symposium on Computer Architecture, University of Florida.
4. Friedman, A. D. and Menon, P. R., (1984), Fault Detection in Diagital Circuits, Prentice-Hall.
5. Goldberg, J. et al., (1974), "*An Organization for Highly Survivable Memory*", IEEE Trans. on Computers, Vol. C-23, No. 7, pp. 693-705.
6. Gray, F. G. and Meyer, J. F., (1971), "*Locatability of Faults in Combinational Networks*", IEEE Trans. on Computers, Vol. C-20, No. 11, pp. 1407-1410.
7. Hayes, J. P. and Friedman, A. D., (1974), "*Test Point Placement to Simplify Fault Detection*", IEEE Trans. on Computers, Vol. C-23, No. 7, pp. 727-746.
8. Jensen, P. A., (1964), "*The Reliability of Redundant Multiple-Line Networks*", IEEE Trans. Reliability, Vol. 13, pp. 23-33.
9. Kamal, S. and Page, C. V., (1974), "*Intermittent Faults : A Model and*

*Detection Procedure*". IEEE Trans. on Computers, Vol. C-23, No. 7, pp. 713-724.

10. Keister, W. et al., (1964) "*No. 1 ESS Issue*", Bell System Technical Journal, Vol. 43, No. 5, pts. 1 and 2, pp. 1831-2610.
11. Kime, C. R., (1966), "*An Organization for Checking Experiments on Sequential Circuits*", IEEE Trans. on Computers, Vol. C-15, No. 1, pp. 113-115.
12. Lawrenz, D. A., (1974), "*Diagnostic Test Design*", Digest of the Fourth Annual International Symposium on Fault-Tolerant Computing, IEEE No. 74, ch0864-9c, Urbana, Illinois.
13. Mathur, F. P., (1972), "*On Reliability Fault-Tolerant Digital Systems*", IEEE Trans. on Computers, Vol. C-20, No. 11, pp. 1376-1386.
14. Mathur, F. P., (1971), "*Care Program (Computer-Aided Reliability Estimation*", Technical Memo, No. 361-10 JP Laboratory, California.
15. Neumann, P. G., (1972), "*A Hierarchical Frameork for Fault-Tolerant Computing Systems*", IEEE Computer Society Conference.
16. Ogus, R. C., (1974) "*Fault-Tolerance of the Iterative Cell Array Switch for Hybrid Redundancy*", IEEE Trans. on Computers, Vol. C-23, No.7, pp. 667-678.
17. Parnas, D. L., (1974), "*On a Buzzword : Hierarchical Structure*", Information Processing 74, Vol.2, pp. 336-339.
18. Patterson, W. W. and Metze, G., (1984), "*A Fail-Safe Asynchronous Sequential Machine*", IEEE Trans. on Computers, Vol. C-23, No. 4, pp. 369-378.
19. Rao, T. R. N., (1984), Error Coding for Arithmetic Processors, Academic Press.
20. Rennels, D. A. and Avizienis, A., (1973), BMS : "*A Reliability Modeling System Self-Repairing Computers*", Digest of Third International Symposium on Fault-Tolerant Computing, California.
21. Short, R. A. and Goldberg, J., (1971), "*A Survery of Soviet Activities in the Design of Fault-Tolerant Digital Machines*," IEEE Trans. on Computers, Vol. C-20, No. 11, pp. 1337-1352.
22. Verma, J. P. et al., (1974), "*Automatic Test-Generation and Test-Verification of Digital Systems*", Proc. of the IEEE ACM 11th Design Automation Workshop, Denver, Colorado.
23. Waldinger, R. J. and Levitt, K. N., (1973), "*Reasoning about Programs*", Proc. ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages.
24. Yau, S. S. and Tang, Y. S., (1981), "*An Efficient Algorithm for Generating Complete Test Sets For Combinational Logic Circuits*", IEEE Trans. on Computers, Vol. C-20, No. 3, pp. 1245-1251.