TABLE V
MEAN TIME FOR ERROR HANDLING (s)

	EARLY	LATE	COMBINED
Critical	38.0	46.2	40.4
Essential	4.8	4.2	4.6
Non-Essential	13.8	40.0	26.4

TABLE VI
AVERAGE FAILURE RATE (FAILURE/1000 h)

Error type	EARLY	LATE
Control	2.2	0.8
Timing	4.5	2.6
I/O and data management	21.0	1.1
Storage management	9.9	2.0
Storage exceptions	19.9	3.4
Programming exceptions	6.3	1.2
All	63.8	11.1

TABLE VII
FAULT TOLERANCE

Error Type	All Jobs	Critical Jobs
Control	0.80	0.50
Timing	0.90	0.00
I/O and data management	0.42	0.24
Storage management	0.89	0.77
Storage exceptions	0.63	0.16
Programming exceptions	0.69	0.26
All	0.88	0.43

methodology has been developed with the MVS system as a basis, the approach can easily be adapted to other systems with recovery features. The reliability measures proposed can be used in conjunction with other design goals for evaluating new systems.

Specifically, the correspondence proposes and estimates concise measures for recovery management and fault tolerance in MVS. Since MVS is a widely used complex system, it is important to quantify the effectiveness of its recovery management.

Storage management and storage exceptions were found to be the major error categories. This was related to the criticality of the storage management activity on the MVS system. The measurements showed that the system fault tolerance almost doubles when recovery routines are provided, in comparison to the case where no recovery routines are available. The system recovery routines are most effective in handling storage management problems (an important feature of MVS). However, even when recovery routines are provided, there is almost a 50 percent chance of system failure when critical system jobs are involved. Thus, there is still considerable scope for improvement. Timing, I/O and data management, and exceptions are the main problem areas. Timing errors in MVS, it is felt, are best dealt with through improved error detection. Here, both prechecking and postchecking of data structures might help. In the other cases, more robust recovery routines and better handling of hardware errors could produce a substantial improvement. Some caution is advised in applying the statistical figures calculated here to other situations. It is suggested that other systems be measured and analyzed in this manner so that a wide spectrum of practical results on operational software is available.

ACKNOWLEDGMENT

The authors thank Prof. E. J. McCluskey for extensive discussions during the period of this study; L. Rivers and L. Ong at CIT for providing valuable insight into the MVS system; and D. Rossetti, D. Lu, and A. Mahmood for their careful reading of an early draft of this paper.

REFERENCES

[1] M. A. Auslander, D. C. Larkin, and A. L. Scherr, "The evolution of the MVS operating system," *IBM J. Res. Develop.*, vol. 25, no. 5, Sept. 1981.
[2] X. Castillo and D. P. Siewiorek, "A workload dependent software reliability prediction model," in *Proc. 12th Int. Symp. on Fault-Tolerant Computing*, Santa Monica, CA, June 1982.
[3] A. Endres, "An analysis of errors and their causes in systems programs," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 140–149, June 1975.
[4] R. L. Glass, "Persistent software errors," *IEEE Trans. Software Eng.*, vol. SE-7, pp. 162–168, Mar. 1981.
[5] P. A. Hamilton and J. D. Musa, "Measuring reliability of computation center software," in *Proc. 3rd Int. Conf. Software Eng.*, Atlanta, GA, May 1978, pp. 29–36.
[6] H. Hecht, "Current issues in fault tolerant software," in *Proc. COMPSAC 80*, Chicago, IL, Nov. 1980, pp. 603–607.
[7] R. K. Iyer, S. E. Butner, and E. J. McCluskey, "A statistical failure/load relationship: Results of a multi-computer study," *IEEE Trans. Comput.*, vol. C-31, pp. 697–706, July 1982.
[8] R. K. Iyer and D. J. Rossetti, "A statistical load dependency model for CPU errors at SLAC," in *Proc. 12th Int. Symp. on Fault-Tolerant Computing*, Santa Monica, CA, June 1982.
[9] F. D. Maxwell, "The determination of measures of software reliability," The Aerospace Corp., El Segundo, CA, Rep. NASA-CR-158960, Dec. 1978.
[10] J. Musa, "The measurement and management of software reliability," *Proc. IEEE*, vol. 68, pp. 1131–1143, Sept. 1980.
[11] D. J. Rossetti and R. K. Iyer, "Software related failures on the IBM 3081: A relationship with system utilization," in *Proc. COMPSAC 82*, Chicago, IL, Nov. 1982.
[12] T. A. Thayer, M. Lipow, and E. C. Nelson, *Software Reliability: Study of Large Project Reality* (TRW Series of Software Technology, Vol. 2). Amsterdam: North-Holland, 1978.
[13] P. Velardi and R. K. Iyer, "A study of software failures and recovery in the MVS operating system," Stanford Univ., Stanford, CA, CRC Tech. Rep. 83-7, July 1983.

Fault Tolerance in Binary Tree Architectures

C. S. RAGHAVENDRA, A. AVIŽIENIS, AND M. D. ERCEGOVAC

Abstract — **Binary tree network architectures are applicable in the design of hierarchical computing systems and in specialized high-performance computers. In this correspondence, the reliability and fault tolerance issues in binary tree architecture with spares are considered. Two different fault-tolerance mechanisms are described and studied, namely: 1) scheme with spares; and 2) scheme with performance degradation. Reliability analysis and estimation of the fault-tolerant binary tree structures are performed using the interactive ARIES 82 program. The discussion is restricted to the topological level, and certain extensions of the schemes are also discussed.**

Manuscript received August 8, 1983; revised December 13, 1983. This work was supported by ONR Contract N00014-79-C-0866.

C. S. Raghavendra is with the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089.

A. Avižienis and M. D. Ercegovac are with the Department of Computer Science, University of California, Los Angeles, CA 90024.

Index Terms **—Binary tree, fault tolerance, network architecture, performance degradation, reliability modeling, spare processors.**

I. Introduction

New ideas in programming styles such as data flow computing and functional programming are currently under investigation [2], [4], [8]. Several machine designs have been proposed based on these styles [10], [11]; most consist of a large number of interconnected processing elements. The processing elements cooperate in the execution of programs by exchanging information. Both the fault tolerance and performance of the machine therefore depend on the capabilities of the network used for interprocessor communication [1].

In this correspondence, we consider the fault-tolerance issues in a binary tree architecture. In a binary tree architecture there is a certain probability that a node or a link can fail. The tree structure is physically static; if any node or link fails the tree structure no longer exists. Also, processors may not be able to communicate with each other. For many applications it is important to maintain the binary tree structure during execution of a task. An example is that of executing functional programs. In other applications it is necessary to maintain communication paths among pairs of nodes at all times. Redundant nodes and links can be employed for providing fault tolerance against node and link failures.

Fault-tolerance issues in tree architectures have been studied by several researchers [5], [7], [9]. In [7] a graph model is introduced for fault-tolerant computing systems, and design procedures for k-fault-tolerant systems are presented. In particular, design of optimal 1-fault-tolerant tree structures are discussed. The fault-tolerant system is designed such that it requires the minimum number of spare nodes and redundant links for tolerating any single failures in the system. However, the 1-fault-tolerant system cannot tolerate multiple failures. Another study [9] also considers design of optimal k-fault-tolerant tree structures. It extends the work of [7], and studies procedures for design of optimal fault-tolerant systems for a class of hierarchical tree structures. In these studies it is assumed that there are mechanisms for fault diagnosis, reconfiguration, and recovery.

The fault-tolerance schemes to be discussed in this correspondence provide protection against any single failure in each level of the binary tree structure or one failure per k nodes. Therefore, the system can tolerate multiple failures as long as they are in different levels of the tree. In particular, we discuss two different fault-tolerance schemes; the first scheme uses one spare node per level, with redundant links for protection against any failure in that level. The second scheme provides protection against failures by degrading performance. Detailed reliability analysis and estimation are performed using the ARIES 82 program [12].

II. Scheme with Spares

We describe a fault-tolerant scheme with redundant nodes and links to tolerate any single failure in a binary tree structure. The logical structure of the redundant binary tree is shown in Fig. 1. There is a spare node for each level in the tree (shown by circle within a circle) and there are redundant links as indicated by dashed lines in Fig. 1. With this arrangement any single failure in a level can be tolerated. When there is a failure, reconfiguration is done to maintain the logical structure of the binary tree. This scheme tolerates several failures if they are in different levels of the tree. If more protection against failures is needed, additional spare nodes can be used at lower levels of tree where the number of nodes increases rapidly.

In this redundant binary tree structure each node has between 3 and 5 extra links to provide redundant connections to other nodes. The reconfiguration upon node failure involves selecting a spare node and appropriate links for communication. In order to reduce the complexity of additional interconnections in the system with spares, two decoupling networks (DN_A and DN_B) can be used as shown in Fig. 2. These networks are simple and can be easily decomposed and integrated with the processors so that the modularity of the system implementation with a spare processor per level remains approximately the same as the modularity of the non-

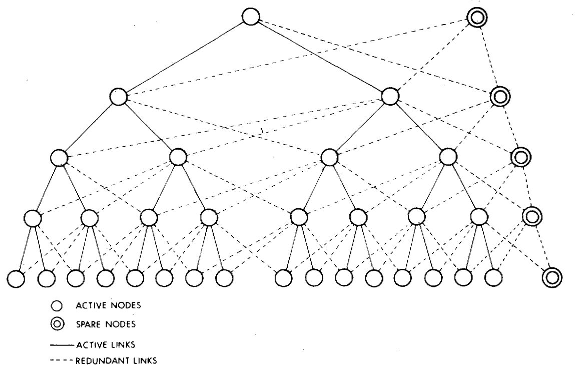

Fig. 1. Five-level tree structure with spares.

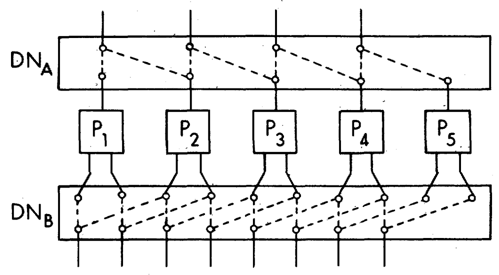

Fig. 2. Organization of level 2 subsystem using decoupling networks.

redundant tree structures [6]. When a processor fails, all links of processors to the right of it are readjusted to right neighbors as indicated in Fig. 3. The hardware reconfiguration is performed by setting switches in the decoupling networks by a separate host processor. Control information for switch setting is obtained locally at levels where processor failures occurred. The use of a decoupling network (to improve reliability of Illiac IV) has been considered before [3].

We can think of this redundant binary tree system of processors as a special purpose processor attached to a host computer. Specifically, this tree system might be used to execute a functional programming language. The host can perform such tasks as compiling and initializing the tree system. When a failure is detected by any node it signals the host which can perform reconfiguration and recovery of the executing program. The rollback and recovery procedures used can be similar to the ones described in [10]. The effectiveness of fault detection and recovery is accounted for by a coverage factor which is used in the reliability computations below.

A. Reliability Estimation for the Scheme with Spares

The redundant binary tree structure with spares can be considered as a fault-tolerant system consisting of a series of homogeneous subsystems S_i, $i = 1, 2, \cdots, N$. The ith level of the tree becomes a homogeneous subsystem with 2^i active nodes and a spare node. We analyze this fault-tolerant system to derive an expression for the reliability of the system in terms of the node reliabilities. For this analysis we consider only node failures. We make the assumption that the probability of link failures is negligibly small compared to that of node failures. Further, any link failure can be treated as failure of one of the two nodes that are connected by this link. With a failure rate λ, the reliability of a single node is $R = e^{-\lambda t}$. In a tree structure with n levels, there are $2^n - 1$ nodes and we assume that all of them must be working in a nonredundant system. Therefore, the reliability of nonredundant binary tree is

$$R_{nr} = R^{2^n - 1}.$$

With the coverage factor c, which is the conditional probability of successful recovery after a failure has been detected, the system reliability for redundant binary tree with one spare per level can be easily derived as

$$R_{sys} = \prod_{k=0}^{n-1} [R^{2^k+1} + R^{2^k}(1 - R) + 2^k c R^{2^k}(1 - R)].$$

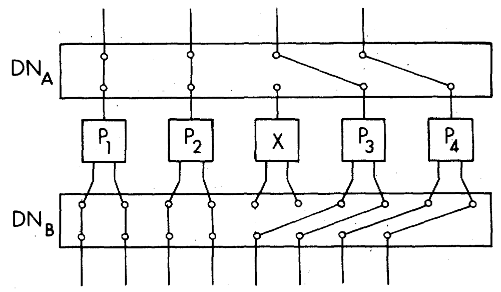

Fig. 3. Reconfiguration of level 2 subsystem after processor 3 fails.

TABLE I
RELIABILITY OF 10-LEVEL SYSTEM ($\lambda = \mu = 1$)

		System Reliability with Coverage							
		$c = 1$		$c = 0.99$		$c = 0.98$		$c = 0.95$	
t	R(non-red)	Rel.	RIF	Rel.	RIF	Rel.	RIF	Rel.	RIF
0	1	1	—	1	—	1	—	1	—
0.00005	0.9501	0.9996	124.75	0.9991	55.444	0.9985	33.266	0.9970	16.633
0.0001	0.9028	0.9983	57.176	0.9973	36.000	0.9963	26.270	0.9933	14.507
0.0002	0.8148	0.9933	27.671	0.9915	21.788	0.9896	17.808	0.9839	11.503
0.0003	0.7357	0.9856	18.354	0.9828	15.366	0.9801	13.281	0.9719	9.406
0.0004	0.6642	0.9751	13.486	0.9716	11.824	0.9681	10.527	0.9576	7.920
0.0005	0.5996	0.9624	10.649	0.9581	9.556	0.9539	8.685	0.9414	6.832
0.001	0.3595	0.8718	4.996	0.8650	4.744	0.8583	4.520	0.8383	3.961

This simplifies to

$$R_{sys} = R^{2^{n-1}} \prod_{k=0}^{n-1} [(2^k c + 1) - 2^k cR].$$

As an example we consider a 10-level system for reliability computation using the ARIES 82 interactive reliability evaluation tool [12]. In Table I we present the system reliability R_{sys} and reliability improvement factor (RIF) for a typical value of node failure rate and different values of coverage. RIF is defined as follows:

$$\text{RIF} = \frac{1 - R_{nr}}{1 - R_{sys}}.$$

The units are normalized, such that the product of units of failure rate and time is 1, i.e., λt is unity. Typically, failure rates are per million hours and time will be in millions of hours. The effect of coverage factor on system reliability is plotted in Fig. 4.

The mean time to first failure (MTFF) of this 10-level redundant binary tree with spares is 0.003072×10^6 h and is much higher than 0.0009775×10^6 h, which is the corresponding figure for the 10-level nonredundant binary tree.

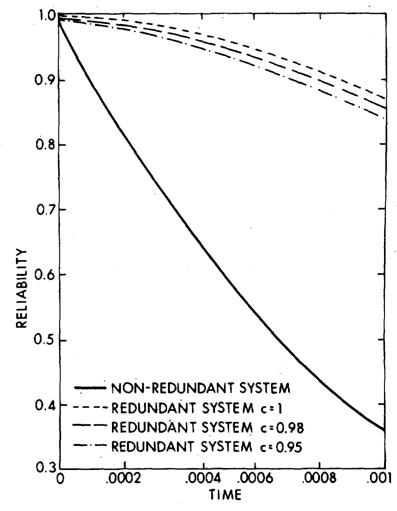

Fig. 4. System reliability with different coverages.

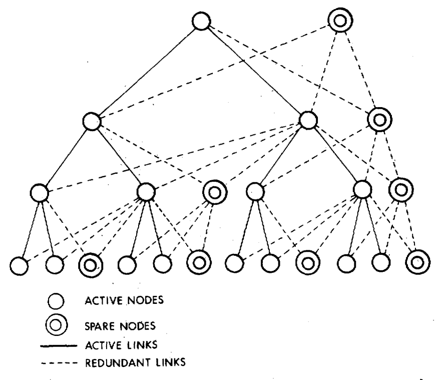

Fig. 5. Extended scheme with spares.

B. Extensions to the Scheme with Spares

The fault-tolerance scheme described in the previous section provides protection against any single failure in each level of the binary tree. The scheme of providing one spare per level gives good protection when there are relatively few levels (six or less). In a larger tree the lower levels have large numbers of nodes, and therefore one spare for the entire level does not provide good balance in the reliability of different levels. Clearly, more spares are required at lower levels to improve overall system reliability over a broader range of time. The scheme with spares can be extended by increasing the number of spares as the nodes per level of tree increase. The technique is to provide 1 spare for every $k = 2^j$ nodes, for some value of j. There is a variety of possible arrangements depending on the value of j. The extended fault-tolerance scheme with most spares will be that where there is a spare node for every pair of nodes (which is an extreme case) is presented in Fig. 5. We use the name *simplex* for a nonredundant binary tree, *level sparing* for a redundant binary tree with one spare per level, and *modular sparing* for a redundant binary tree with one spare per group of k nodes. Their corresponding system reliabilities are denoted by RSimp(a), RLev(a), and RMod(a, b), respectively, where a is the number of levels in the binary tree and b is the number of nodes in a group.

One can consider the problem of finding the best possible distribution of spare modules with a fixed number of spares. For example, in a 10-level tree we could get higher system reliability by distributing more spares at lower levels than the scheme of 1 spare per level. The effect of moving the spares from top levels to bottom levels is to improve the system reliability over a longer range of time, i.e., a flattening of the reliability curve. The results of this study on a 10-level tree when RLev(10) $\geq$ 0.95 indicates that the best possible distribution is no spares for levels 0–2, 1 spare for levels 4–9, and 4 spares for level 10.

Next, we consider the problem of incorporating redundancy systematically in a large binary tree. The two main criteria we use are: 1) any single failure in the system should be tolerated; this requires at least one spare per level of tree; and 2) distribution of redundant modules in a uniform manner for higher overall reliability. We consider a 20-level binary tree with about 10 percent redundancy. A 20-level binary tree architecture has been proposed for executing reduction language programs [11]. The approach is to provide 1 spare up to a certain level which has 2^k nodes and then provide 1 spare for every 2^k nodes. This would double the number of spares per level after the kth level. We computed system reliabilities for four different redundant binary trees, namely, a) 1 spare per level RLev(20); b) 1 spare for levels 0–6 and 1 spare per 64 nodes RMod(20, 64); c) 1 spare for levels 0–4 and 1 spare per 16 nodes RMod(20, 16); d) 1 spare for levels 0–3 and 1 spare per 8 nodes RMod(20, 8). The amounts of redundancy in Mod(20, 64), Mod(20, 16), and Mod(20, 8) are about 1.6, 6.2, and 12.5 percents, respectively. In order to get accurate reliability figures, unlimited precision arithmetic was used on a VAX 11/750 machine. The reliability calculations of these four systems along with nonredundant system are presented in Table II.

The failure rates of nodes are assumed to be 1 per million hours of operation; this is a typical figure for microprocessors. Then the time scale in Table II is 0.1 h to 500 h. The reliability of the nonredundant 20-level tree is less than 0.9 after about 6 min of operation and is close to 0 after about 10 h. The redundant system with 1 spare per level has reliability less than 0.9 after about 50 min and drops to 0 after 20 h. It is clear from Table II that with modest redundancies of 1–6 percent, the system reliability of 20-level binary tree can be improved to 0.9 up to 50–100 h, and greater than 0.9999 up to 1 h. The reliability improvement factors of these redundant systems with respect to nonredundant systems are on the order of 10 for Lev(20), 10^5 for Mod(20, 64), and 10^6 for Mod(20, 8). This approach of providing 1 spare per level for a few top levels and providing 1 spare for every 2^k nodes thereafter is quite attractive as the resulting redundant binary tree is uniform and nicely partitioned. The size of the decoupling networks (the hardcore of the system) will be relatively small, and hence can be designed to be highly reliable.

III. Scheme with Performance Degradation

In this scheme, the tree system operates with a degradation in performance when there is a node failure. The redundancy employed is one spare node for the root and extra links from each node as indicated in Fig. 6. When a node fails, its neighbor will take over its computation and thus will have to do more work. This kind of fault-tolerance scheme is more suitable in systems where the communication is quite robust and the nodes are fairly powerful computers performing independent tasks.

In this redundant structure, each node acts as a spare to its neighbor. Upon a node failure, its neighbor performs additional tasks, probably with twice the loading. A link failure is treated as equivalent to a node failure. Reconfiguration and recovery involves signaling the neighbor nodes to perform additional tasks and using redundant spare links for communication.

In this scheme, failures of one out of any two nodes can be tolerated. Multiple node failures can be tolerated provided they are nonadjacent. The system performance will be severely degraded and possibly destroyed if two adjacent nodes fail; however, the probability of such an event is small when independent failures are assumed.

A. Reliability Estimation for the Scheme with Performance Degradation

Now we consider the reliability analysis and estimation of the scheme where only one spare node is used at the root level and each node acts as active spare to its neighbor. In an n-level system there will be in all 2^{n-1} pairs of nodes, and all pairs must be working. Each pair can tolerate a failure without total system failure. The reliability expression for this system is given by

$$R_{sys} = [R^2 + 2cR(1 - R)]^{2^{n-1}}$$

where $R = e^{-\lambda t}$.

An evaluation similar to Table I has been performed using the ARIES 82 program for a 10-level fault-tolerant system with performance degradation. The results obtained for system reliability and RIF with various coverage factors are presented in Table III.

In order to perform the sensitivity analysis with respect to coverage, we compute the system reliability with different values of coverage and draw a plot as before. Considering the RIF values from Table III, we can clearly observe that the effect of coverage on system reliability is quite significant.

In the scheme with performance degradation we are using only one spare computer, and trade time for hardware failures. By comparing the results obtained for this scheme and the scheme with spares one can see that there is significant difference. Here, when a processor fails its neighboring processor takes over its tasks. This processor has to do more work which is reflected by decrease in

TABLE II
System Reliabilities with Different Redundancies ($\lambda = \mu = 1$)

Percent Red. Time	RSimp(20) 0	RLev(20) 10^{-5}	RMod(20, 64) 1.6	RMod(20, 16) 6.25	RMod(20, 8) 12.5
0.0000001	0.900453	0.998222	0.9999996592	0.9999999109	0.9999999528
0.0000002	0.810815	0.993103	0.9999986369	0.9999996435	0.9999998113
0.0000005	0.591977	0.960790	0.9999914805	0.9999977718	0.9999988204
0.000001	0.350437	0.866962	0.9999659233	0.9999910872	0.9999952814
0.000002	0.122806	0.622388	0.9998637061	0.9999643498	0.9999811260
0.000005	0.005285	0.132693	0.9991485778	0.9997772147	0.9998820455
0.00001	$2.79e^{-5}$	0.004378	0.9965993879	0.9991092054	0.9995282788
0.00002	e^{-10}	e^{-6}	0.9864725533	0.9964419706	0.9981145568
0.00005	e^{-23}	e^{-18}	0.9184993621	0.9779761533	0.9882761278
0.0001	0	0	0.7122481754	0.9148171779	0.9539355206
0.0002	0	0	0.2588418605	0.7006593626	0.8281756225
0.0005	0	0	0.0002385752	0.1090396017	0.3084127581

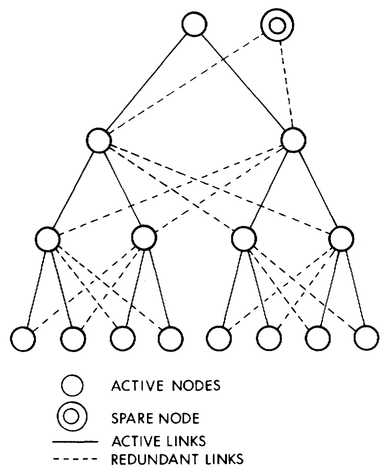

Fig. 6. Scheme with performance degradation.

TABLE III
Reliability and RIF of 10-Level System with Performance Degradation ($\lambda = \mu = 1$)

	Coverage = 1		Coverage = 0.99		Coverage = 0.98		Coverage = 0.95	
t	Rel.	RIF	Rel.	RIF	Rel.	RIF	Rel.	RIF
0.00005	0.999998	38986	0.9995	97.25	0.9990	48.69	0.9974	19.51
0.0001	0.999995	18986	0.9989	94.51	0.9979	47.39	0.9949	19.01
0.0002	0.999979	9044.8	0.9979	89.65	0.9959	45.09	0.9898	18.15
0.0003	0.999954	5737.5	0.9969	84.93	0.9938	42.85	0.9847	17.30
0.0004	0.999918	4100.9	0.9958	80.59	0.9918	40.77	0.9797	16.51
0.0005	0.999872	3129.9	0.9947	76.55	0.9897	38.85	0.9746	15.77
0.001	0.999489	1252.5	0.9893	59.98	0.9792	30.88	0.9497	12.73
0.01	0.950570	20.23	0.8593	7.11	0.7768	4.48	0.5738	2.34

throughput. The amount of degradation in performance depends on the level at which failures occur. It is easy to observe that failures close to the root node causes larger loss of performance than failures close to leaf nodes. Therefore, a good solution would be to use spares and performance degradation at the top few levels, and to use only performance degradation at exclusively at lower levels, i.e., near leaf nodes. If it is required that the logical binary tree should be maintained throughout computation then we have to use the scheme with spares.

IV. Discussion

We have examined the fault-tolerance issues in binary tree network architectures. Two different fault-tolerance schemes were developed and discussed in detail. Reliability analysis and estimation of both types of fault-tolerant binary trees were performed and results were discussed. The scheme with spares may be suitable for systems built with large numbers of inexpensive processors. Further, the scheme with spares can be extended for added protection by using more spares in tree levels near the leaves. The scheme with performance degradation may be more suitable for multiprocessor systems consisting of mini- or mainframe computers, where performance can be traded for cost.

The results show that systematic sparing and degradation are readily applicable to binary tree architectures. Combinations of both techniques are also possible and offer some attractive intermediate solutions. The ARIES 82 reliability program has been used to derive quantitative estimates of reliability improvement. The improvements are significant; it is especially encouraging to note that major improvements are attainable with realistic coverage values of 0.98 or less.

References

[1] A. Avižienis, "Fault tolerance: The survival attribute of digital systems," *Proc. IEEE,* vol. 66, pp. 1109–1125, Oct. 1978.

[2] J. Backus, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs" (1977 ACM Turning Award Lecture), *Commun. Ass. Comput. Mach.,* vol. 21, pp. 613–641, Aug. 1978.

[3] I. A. Baqai and T. Lang, "Reliability aspects of the Illiac IV computer," in *Proc. Parallel Processing Conf.,* 1976, pp. 123–131.

[4] J. B. Dennis, "The varieties of data flow computers," in *Proc. 1st Int. Conf. on Distrib. Comput. Syst.,* Oct. 1979, pp. 430–439.

[5] A. Despain and D. Patterson, "X-tree: A structured multiprocessor computer architecture," in *Proc. 5th Symp. on Comput. Arch.,* Apr. 1978, pp. 144–151.

[6] M. D. Ercegovac, T. Lang, and C. S. Raghavendra, "Design and use of decoupling networks in tree structures," in preparation.

[7] J. P. Hayes, "A graph model for fault-tolerant computing systems," *IEEE Trans. Comput.,* vol. C-25, pp. 875–884, Sept. 1976.

[8] R. M. Keller, G. Lindstrom, and S. S. Patil, "A loosely coupled applicative multi-processing system," in *Proc. AFIPS Conf. 1979 NCC,* June 1979, pp. 613–622.

[9] C. L. Kwan and S. Toida, "Optimal fault-tolerant realizations of hierarchical tree systems," in *Proc. 1981 Int. Symp. Fault-Tolerant Computing,* Portland, ME, June 1981, pp. 176–178.

[10] C. K. C. Leung and J. B. Dennis, "Design of a fault tolerant packet communication computer architecture," in *Proc. 1980 Int. Symp. Fault Tolerant Computing,* Kyoto, Japan, Oct. 1980, pp. 328–335.

[11] G. A. Mago, "A network of microcomputers to execute reduction languages," *Int. J. Comput. Informat. Sci.,* vol. 8, no. 5, 6, pp. 349–385, 435–465.

[12] S. V. Makam, A. Avižienis, and G. Grusas, "ARIES 82 users' guide," Dep. Comput. Sci., Univ. California, Los Angeles, Tech. Rep. CSD-820830, Aug. 1982.

Modified Berger Codes for Detection of Unidirectional Errors

HAO DONG

Abstract—**Modified Berger codes are defined in this correspondence. They are less expensive than the ordinary Berger codes in terms of the number of check bits and the cost of checkers. As a tradeoff, their error detection ability is slightly lower, although these codes can detect most unidirectional errors.**

Index Terms—**Berger codes, self-checking circuits, unidirectional errors.**

Manuscript received July 29, 1983; revised December 8, 1983. This work was supported in part by a research fellowship from the Education Ministry of the People's Republic of China and by the National Science Foundation under Grant MCS-7904864.

The author was with the Center for Reliable Computing, Computer Systems Laboratory, Departments of Electrical and Computer Science, Stanford University, Stanford, CA 94305. He is now with the Harbin Shipping Engineering Institute, Harbin, Herlongjiang, People's Republic of China.

I. Introduction

It is seen that some physical defects in LSI or VLSI circuits tend to generate *unidirectional* errors. There are several classes of codes, such as *m*-out-of-*n* codes, Berger codes, and two-rail codes, that can be used to detect unidirectional errors. It has also been proven that low-cost *AN* codes and inverse residue codes with group length $m + 1$ can detect all unidirectional errors of weight less than or equal to m [6], [7].

Berger codes have been proved to be the optimal separable codes that detect any unidirectional error [3], [4]. However, in [4], the author pointed out that one could not make a Berger code detect unidirectional errors of weight less than or equal to m by simply cutting down the number of the check bits, where m is an integer less than the number of information bits in a codeword. In this correspondence, we will define *modified Berger codes* (MB codes) so that these codes will detect all the unidirectional errors of weight less than or equal to m. Then we will estimate the actual error detection ability of these codes. Totally self-checking checkers for MB codes are also described.

II. Definition

A codeword of a Berger code has two parts: information D and check symbol C. Suppose D has I bits and C has k bits. Let $I1$ and $I0$ be the number of 1's and the number of 0's, respectively, in the I information bits. The check symbol of a codeword is the binary number of $I0$, or the complement (bit by bit) of the binary number of $I1$. That is,

$$C = I0 \quad \text{or} \quad C = (2^k - 1) - I1 .$$

We have

$$k = [\log_2(I + 1)]$$

where $[a]$ is the least integer greater than or equal to a. If $I = 2^k - 1$ then using $I0$ or $I1$ will result in the same code, and the code is called the maximal length Berger code [1].

Assume now all the erroneous bits are within the I information bits. If we use $I0$ modulo $(m + 1)$ or $I1$ modulo $(m + 1)$ $(1 < m < I)$ as the check symbol, denoted by $C1$, then all the unidirectional errors of weight less than or equal to m will be detected by this code because no such error could change one codeword to another. In this case $J = [\log_2(m + 1)]$ bits are needed for the check symbol $C1$. Let Pk $(k = 0, 1, \cdots,)$ be the subset of codewords in which every codeword has an $I1 = k$. The column $C1$ of Table I shows an example of such a code.

A problem arises from the fact that the errors may also change the check bits. For example, an error may change a codeword in $P1$ to a codeword in $P0$ with only 4 erroneous bits (including the three check bits). As the number J usually is very small ($[\log_2(m + 1)]$), it is reasonable to use a second-level code to detect any error in the check bits. We may use any of the codes mentioned in the beginning of this paper to encode the check symbol $C1$ with another check symbol $C2$. These codes with check symbol $C1$ and $C2$ are called modified Berger codes in this correspondence and the maximum weight of errors detected by an MB code is denoted by m. Table I shows an example of MB codes with $m = 7$. The check symbols $C1$ and $C2$ in Table I form a two-rail code. In MB codes because any unidirectional error in the check bits is detected by the second coding, either $I0$ modulo $(m + 1)$ or $I1$ modulo $(m + 1)$ can be used directly as the check symbol $C1$. It is clear that the MB code in Table I (with check symbols $C1$ and $C2$) can detect any unidirectional error of weight less than or equal to 7, and that this error detection ability is effective regardless of the number of information bits in the code.