A Simple Scheme for A Fault Tolerant DADO Machine[1]

Salvatore J. Stolfo
Department of Computer Science
Columbia University
New York, NY 10027
January 30, 1985

CUCS-214-85

[1]This research has been supported in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0165, the New York State Science and Technology Foundation under contract NYSSTFCAT(84)-15, as well as grants from Intel, Digital Equipment, Hewlett-Packard, Valid Logic Systems, AT&T Bell Laboratories and International Business Machines corporations. We gratefully acknowledge their support.

List of Figures

Binary tree multiprocessors, such as DADO, have many favorable advantages for hardware implementation. For example, binary trees are planar requiring linear area (VLSI implementations require area which is proportional to the number of processing elements) and are not pin-limited (off chip connections remain constant as device dimensions scale down and more processors are implemented on the chip). One often cited problem for binary trees, however, is that trees are not fault tolerant. In this brief note, we detail a simple method which guarantees operation of a binary tree machine after two successive faults, as well as a 50% chance of proper operation after a third successive fault. This scheme requires no extraordinary engineering changes and very simple software to support proper operation of the machine. The binary tree organization is thus maintained.

We first consider the types of faults which may occur in a typical multiprocessor system. The primary system components are:

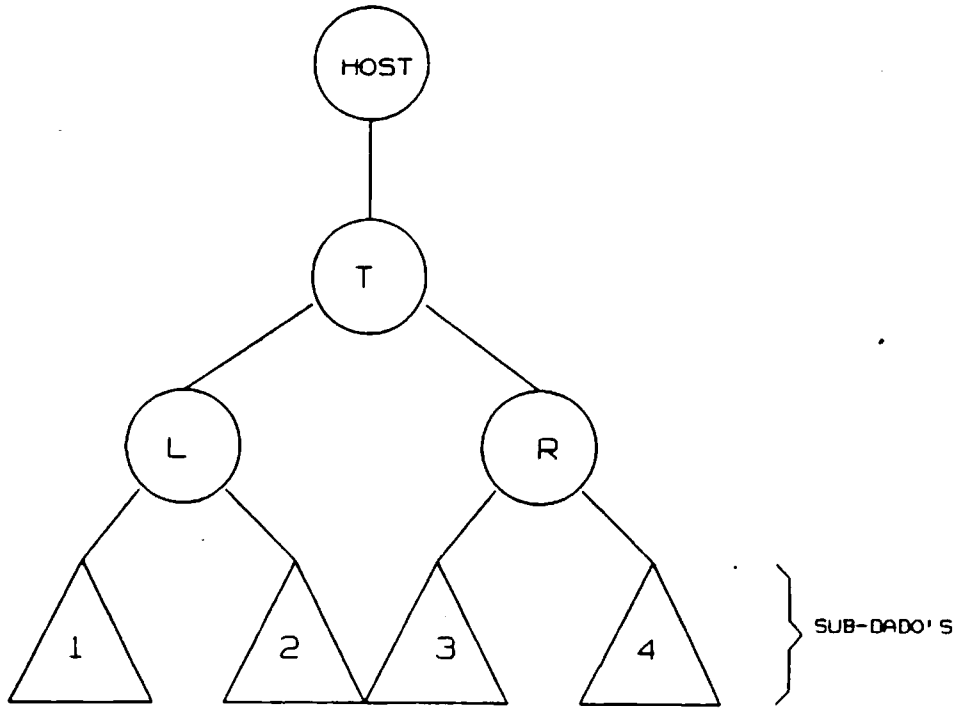
- printed wires on boards
- wires connecting two locations occupying space (non-printed wires)
- connectors between boards and connectors between IC's and boards
- and finally, IC's (packaged chips).

In modern technology, printed wires generally do not fail. Thus, it is safe for us to ignore this source of faulty operation. The scheme we shall describe, however, also covers this possibility. Non-printed wires are extremely vulnerable to vibration and physical abuse and are thus frequent sources of faults in a large-scale system. We note with interest that binary tree machines, and in particular DADO, employ only printed wires and no non-printed wires. (This is not the case for "butterfly type" machines, for example.)

We thus focus on connectors and IC's. Board connectors can be easily engineered for fault free operation for any system. However, connectors between IC's and boards (or solder joints) pose a more serious threat. If such connectors fail, the IC will not function. It is safe, therefore, to consider IC failure and connector failure as potential generators of the same sorts of faults. Thus, in the remainder of this note, we shall only refer to IC failure which includes the case for IC connectors or solder joint failures. The most common and frequent IC failure is simple parity errors on memory accesses. This can be handled with conventional error correcting methods and is assumed to be present. By IC failure we mean complete and irrevocable non-operation of a packaged circuit. By way of summary, we shall consider IC failures only for the DADO machine. We now introduce our scheme for fault tolerant operation. For pedagogical reasons we assume no hardware support for fault detection.

Suppose we are executing some parallel computation on a DADO machine of size N. (For DADO2, N=1023.) The essence of our scheme is to replicate the computation four times in a DADO machine of size 4N + 3. The three additional nodes form the uppermost part of the tree (nodes T, L and R in Figure, which act as arbitrators guaranteeing agreement between the four identical and concurrent processes in the four sub-DADO trees.

Figure 1: Quadruplicated Computation.



Nodes L and R work concurrently to guarantee agreement between their descendant subtrees. T guarantees their agreement and helps to isolate faults if they arise. An example will help to elucidate our scheme.

Suppose a fault occurs in sub-$DADO_2$. Thus, the results communicated to L will undoubtedly differ at some point in the computation. Immediately upon noticing this discrepancy, L notifies T by setting a pin which T reads continually. At the same time R has noticed no faults of its own and communicates valid results to T. T proceeds to transmit R's result to the external host, and also to L.

Node L, using this value supplied by R (via T), verifies that sub-$DADO_2$ has failed. Subsequent operation of L simply passes through values from sub-$DADO_1$ directly to T. Sub-$DADO_2$ can now either work independently to isolate its own fault for direct manual repair, or remain disconnected for the remainder of the operation of the machine. Notice the computation has continued and the host has received valid results.

We are now left with three independent and concurrent valid computations in sub-DADO's 1, 3 and 4. If another fault occurs in one of these sub-DADO'S, the other two computations will remain valid and isolate the faulty subtree in the same manner as explicated in our example. Thus, T is guaranteed to respond with valid data through two faults.

At this point we have two valid and concurrent computations remaining. If a third fault occurs (and we assume no hardware support for fault detection), T can choose randomly from the two remaining subtrees with a statistical probability of success 50% of the time.

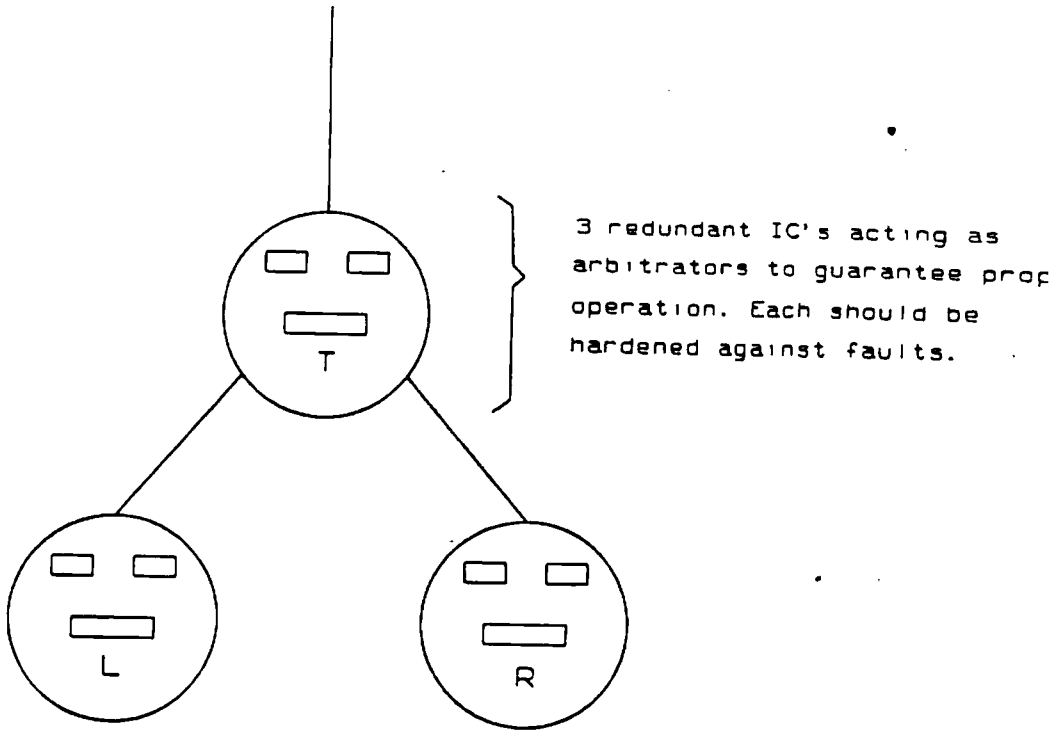
Several potential problems immediately come to mind. What if two faults occur in two sub-DADO's concurrently? In this extremely unlikely case, two valid sub-DADO's can be used by the three arbitrators T, L and R to isolate the two faulty sub-DADO's. This assumes the two failing sub-DADO's will respond differently. Indeed, the chances that precisely the same fault will occur in both, forcing both to respond with the same data at the same instance in time, is astronomically small. Thus, two concurrent faults can be handled provided two valid computations remain.

What if T, L or R fail? This is the key problem. This scheme works only if T, L and R can be "hardened" against their own faults. In this case an arbitration scheme for each node can be implemented by hardware redundancy, as is done in conventional fault tolerant systems offered by such companies as SYNAPSE, STRATUS and TANDEM. Duplication or triplication of circuitry with arbitration is the most commonly used approach.

The important point to note is that hardware redundancy and arbitration are needed only for three nodes T, L and R, and not for each of the thousands of nodes in the entire system.

It is interesting to note that T, L and R are performing simple functions and do not require a full PE implementation. That is to say, T, L and R are exactly the same simple circuit which can be implemented on a single IC Three such IC's can guarantee proper operation for each of T, L and R. Indeed, rather than 3 IC's, a single IC with three redundant circuits can do the same job while minimizing IC to board connections, which may fail. (DADO2's present design employs 5 IC's in each PE.) Thus, fault tolerance can be guaranteed by nine hardened chips, as illustrated in Figure1. We note that no extraordinary software systems are required to implement this scheme. No new "smart" kernel software is needed at each DADO PE. Indeed, board designs need not change as well. The three hardened PE's can be implemented on DADO's backplane, itself a printed-circuit board. This scheme is thus extremely simple, requiring a modest amount of engineering. The possibility exists as well of devising methods for repairing faults in one of the sub-DADO's based solely on copying the state of the valid sibling to the faulty subtree

Figure 2: Fault tolerance requires 9 hardened chips.



Our final question: what does it cost? This is perhaps the most interesting benefit of this approach. The scheme requires only a factor of 4 in hardware while maintaining performance. Note no non-printed wires are introduced with a modest engineering change to the backplane. In addition, application software and supporting system software operating in the sub-DADO trees need not be changed.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION COLUMBIA UNIVERSITY 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION NAVELEX
6c. ADDRESS (City, State, and ZIP Code) 450 Computer Science Building Columbia University New York, NY 10027	7b. ADDRESS (City, State, and ZIP Code) 2511 Jefferson Davis Highway Arlington, VA 22202
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. N00039-84-C-0165 / TASK NO. 2 / WORK UNIT ACCESSION NO
11. TITLE (Include Security Classification) A Simple Scheme for a Fault Tolerant DADO Machine	
12. PERSONAL AUTHOR(S) Stolfo, S. J.	
13a. TYPE OF REPORT SPECIAL 13b. TIME COVERED FROM 9/84 TO 12/84	14. DATE OF REPORT (Year, Month, Day) 1985, January 30 15. PAGE COUNT 4
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) DADO, Production Systems, Parallel Computer, Fifth Generation, Logic Programming, AI, LISP.
19. ABSTRACT (Continue on reverse if necessary and identify by block number)	
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Salvatore J. Stolfo	22b. TELEPHONE (Include Area Code) (212) 280-8111 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE