

Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems

Joanne Bechta Dugan, Senior Member IEEE

Duke University, Durham

Salvatore J. Bavuso

NASA Langley Research Center, Hampton

Mark A. Boyd, Member IEEE

NASA Ames Research Center, Moffett Field

Key Words — Fault tree, Spare, Dynamic redundancy, Fault-tolerant computer system, HARP

Reader Aids —

Purpose: Describe advanced fault-tree models

Special math needed for explanations: Fault trees and Markov chains

Special math needed to use results: None

Results useful to: Reliability analysts

Abstract — Reliability analysis of fault-tolerant computer systems for critical applications is complicated by several factors. Systems designed to achieve high levels of reliability frequently employ high levels of redundancy, dynamic redundancy management, and complex fault & error recovery techniques. This paper describes dynamic fault-tree modeling techniques for handling these difficulties. Three advanced fault-tolerant computer systems are described, viz, a fault-tolerant parallel processor, a mission avionics system, and a fault-tolerant hypercube. Fault-tree models for their analysis are presented. HARP (Hybrid Automated Reliability Predictor) is a software package developed at Duke University and NASA Langley Research Center that can solve those fault-tree models.

1. INTRODUCTION

Several goals for the reliability analysis of fault-tolerant computer systems for critical applications are:

- predicting the reliability of the system for a specified mission time,
- facilitate tradeoff studies for various fault-tolerance techniques,
- compare alternative architectures for a system still in the design phase.

Even if a system exists only as a rough sketch on paper, techniques can analyze parametric sensitivity in order to determine which factors have the strongest impact on the reliability of the system.

Fault trees are frequently used for reliability analysis of critical systems. Fault-tree models are well accepted and solution methods are well known, but exact synthesis & analysis of fault trees with many basic events is often expensive. Several important types of dynamic behavior in advanced fault-tolerant systems cannot be adequately captured in a standard fault-tree

model, eg, fault & error recovery, sequence-dependent failures, and the use of spares. Markov models are an alternative that is flexible enough to model nearly any such dynamic system. Tools and techniques exist for solving even very large Markov models. However, the construction of a Markov model for any but the simplest system is tedious and error prone.

Definitions

Fault. An undesirable change in a hardware component of the system, eg, a short circuit between two leads, or an open-circuited transistor junction.

Error. The manifestation of a fault in the: a) information that is processed by the computing system, or b) internal system state.

Failure. An unacceptable deviation from the anticipated delivered-service.

Cold spare. An element that neither degrades nor fails while it is a spare.

Hot spare. An element whose degradation & failure behavior while it is a spare is the same as while it is active.

Warm spare. An element whose degradation & failure behavior lies between a cold spare and a hot spare. □

Acronyms

FEHM fault & error handling model

FORM fault occurrence & repair model (coverage model)

HARP hybrid automated reliability predictor □

Fault-Tree Gate Symbols

[These symbols are for newly-introduced gates]

CSP cold spare

FDEP functional dependency

SEQ sequence enforcing □

To exploit the relative advantages of both fault trees and Markov models, while avoiding many of the shortcomings, we define a model that is flexible enough to capture the dynamic aspects of the system, but which is (almost) as easy to use as a usual fault-tree. The model construction and solution are facilitated by the new model in the following three major ways; they are defined and demonstrated via example in this paper.

1. *Behavioral decomposition* is used to define models separately for system structure and fault & error recovery. Using behavioral decomposition, the model is composed of fault occurrence & repair (FORM) and fault & error handling (FEHM) submodels [1]. FORM contains information about the structure of the system and the fault arrival process. FEHM allows for the modeling of permanent, intermittent, and transients, and models the on-line recovery procedure necessary for each type.

2. The fault-tree model of system structure is internally and automatically converted to a Markov model [2], to which is added the fault & error recovery information. All occurrences of basic events that leave the system operational are enumerated; each combination becomes a state in the Markov chain. The advantage of allowing a fault-tree description of the system is that the modeler need not perform the tedious task of determining the Markov chain representation of a system that can be described as a fault tree. Very often, a relatively simple fault-tree can give rise to a very large and complicated state space in the corresponding Markov chain. The modeler can use the parsimony of the fault-tree representation of the system to generate the state space of the Markov chain automatically, and then adjust the Markov chain as needed.

3. Several additional gates are introduced into the fault-tree model to capture dynamic behavior [3]. These new gates are described in the next section, and their use is demonstrated in the examples. □

These techniques have been implemented in HARP (Hybrid Automated Reliability Predictor) [4,5], a software package for the analysis of advanced fault-tolerant systems, developed by NASA Langley Research Center and Duke University. The models in this paper are all solved using HARP.

Another type of dynamic behavior, not addressed in this paper, is *phased missions* in which the configuration of components required for system operation changes during the operation. Several authors have proposed methods for analyzing phased-mission reliability; these methods include fault trees and Markov models [6 - 9].

Assumptions

1. Faults are modeled as random, statistically independent events.
2. The fault hazard rate (failure rate) of a component is a constant.
3. Mission lengths are relatively short, so that the probability of more than a few component failures is low.
4. The system is not repairable while in use. □

Systems which violate these assumptions can be handled by more sophisticated techniques which fall outside the scope of this paper.

2. DYNAMIC FAULT-TREE GATES

A major disadvantage of traditional fault-tree analysis is the inability of usual fault-tree models to capture sequence dependencies in the system and still allow an analytic solution. As an example of a sequence-dependent failure, consider a system with one active component and one standby spare connected with a switch controller [10]. If the switch controller fails after the active unit fails (and thus the standby is already in use), then the system can continue operation. However, if the switch controller fails before the active unit fails, then the standby unit cannot be switched into active operation and the system fails when the ac-

tive unit fails. Thus, the failure criteria depend not only on the combinations of events, but also on their sequence.

Systems with various sequence dependencies are usually Markov modeled. If, instead of using usual fault-tree solution methods, the fault tree is converted to a Markov chain for solution, the expressive power of a fault tree can be expanded by allowing some sequence dependencies to be modeled [11]. There are several kinds of sequence dependencies in fault-tolerant systems. This section identifies several such dependencies, and defines specific gates to express them in fault-tree models. Subsequent sections demonstrate the use of these gate types.

2.1 Functional-Dependency Gate

Assumption

1. The system is configured such that the occurrence of some *trigger* event causes other dependent components to become inaccessible or unusable. Later faults in the dependent components do not further affect the system and are not considered. □

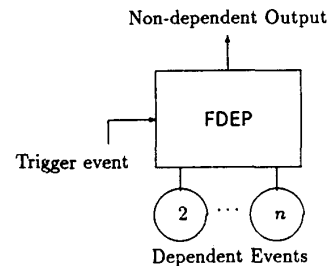


Figure 1. Functional-Dependency Gate

A *functional dependency* gate (see figure 1) has:

- 1 trigger-input (either a basic event or the output of another gate in the tree),
- a non-dependent output (reflecting the status of the trigger event),
- one or more dependent basic events. □

The dependent basic events are functionally dependent on the trigger event. When the trigger event occurs, the dependent basic events are forced to occur. In the Markov-chain generation, when a state is generated in which the trigger event is satisfied, all the associated dependent events are marked as having occurred. The separate occurrence of any of the dependent basic events has no effect on the trigger event.

Example

Communication is achieved through some network interface elements, where the failure of the network element isolates the connected components. The failure of the network element is the trigger event and the connected components are the dependent events.

2.2 Cold-Spare Gate

Systems with cold spares cannot be modeled exactly using usual fault-tree techniques because the system failure criteria cannot be expressed in terms of combinations of basic events, all using the same time frame.

We address this fault-tree deficiency by introducing a CSP gate (see figure 2), with one primary input and one or more alternate inputs. All inputs are basic events. The primary input is the one that is initially powered on, and the alternate input(s) specify the (initially unpowered) components that are used as replacements for the primary unit. The CSP gate has one output which becomes true after all the input events occur.

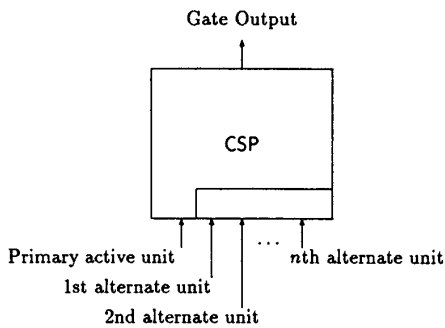


Figure 2. Cold-Spare Gate

The CSP gate can be used if spare units are shared between active processors. Then the basic event representing the cold spare has inputs to more than one CSP gate. The spare is available only to one of the CSP gates, depending on which of the primary units fails first. The FTTP example below illustrates the use of the CSP gate when one spare unit is available for 3 active processors.

The conversion of the fault tree to a Markov chain allows the consideration of cold spares. In a state where the primary unit is operational, the cold spares do not fail. However, once the primary unit has failed, then the first alternate unit can fail. After the first alternate fails, the remaining alternates are allowed to fail, one at a time in the order specified, until the spares are exhausted. The possibility of being unable to reconfigure correctly the spare unit into operation is captured in the (separately specified) coverage model.

The functional dependency gate and the CSP gate can interact in an interesting way. Let the spare units functionally depend on some other (otherwise unrelated) component. The occurrence of the trigger event can render one or more of the spares unusable, even if they have not been switched into active operation. Then, if the primary unit fails, the spares are unavailable to replace it. This is the only case where a cold spare can “fail” while it is in the spare state.

2.3 Priority-AND Gate

The priority-AND gate is: an AND gate plus the condition that the events must occur in a specific order [18]. The

priority-AND gate (figure 3) has 2 inputs, *A* and *B*. The output of the gate is true if:

- both *A* & *B* have occurred, and
- *A* occurred before *B*.

If both events have not occurred, or if *B* occurred before *A* then the gate does not fire. To represent the behavior that *A* occurs before *B* which occurs before *C*, the priority-AND gates can be cascaded as shown in figure 4.

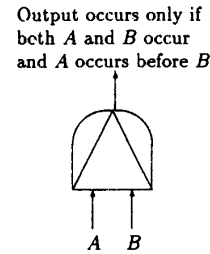


Figure 3. Priority-AND Gate

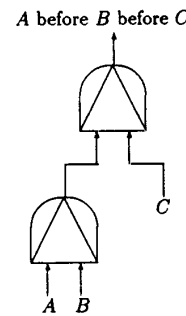


Figure 4. Cascading Priority-AND Gates

2.4 Sequence-Enforcing Gate

The *sequence-enforcing* gate (figure 5) forces events to occur in a particular order. The input events are constrained to occur in the left-to-right order in which they appear under the gate (ie, the leftmost event must occur before the event on its immediate right which must occur before the event on its immediate right is allowed to occur, etc.). The sequence enforcing gate can be contrasted with the priority-AND gate in that the priority-AND gate *detects* whether events occur in a particular order (the events can occur in any order), whereas the sequence enforcing gate *allows* the events to occur only in a specified order.

In the generation of a Markov chain from a fault tree containing a sequence-enforcing gate, the states that represent any other ordering than that permitted by the sequence enforcing gate are never generated. A later section shows an interesting application of the sequence-enforcing gate to model pooled spares.

3. FAULT-TOLERANT PARALLEL PROCESSOR

Acronyms

FTPP fault-tolerant parallel processor
 NE network element
 PE processing element

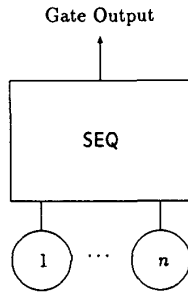


Figure 5. Sequence-Enforcing Gate

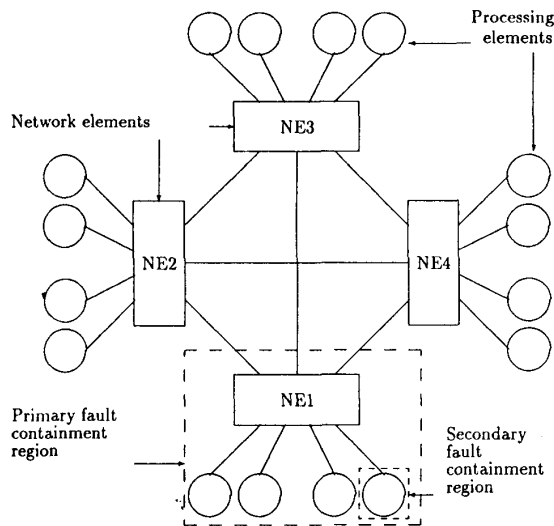


Figure 6. An Instance of the Fault-Tolerant Parallel Processor

3.1 System Description

We consider several models of the FTTP (Fault-tolerant Parallel Processor) [12,13] cluster to compare various configurations of triads with spares. Figure 6 is an instance of an FTTP cluster that consists of 16 processing elements (PEs), with 4 connected to each of 4 network elements (NEs). The NEs are fully connected. In the clusters modeled here, the 16 PEs are logically connected to form 4 triads, each with one spare. We investigate 3 triad/spare configurations, #1 & #2 with hot spares, and #3 with cold spares:

1. Hot spares. There is one PE spare for each triad and all spares are attached to the same NE.
2. Hot spares. There is one spare on each NE and the spare PEs can substitute for any failed PE attached to the same NE.
3. Cold spares. Otherwise the same as #1. □

□ The PEs in all three configurations functionally depend on the NE to which they are connected. If a NE fails permanently, the PEs connected to it are then considered failed.

3.2 Failure Criteria & Parameters

For all models, a triad fails when it has fewer than 2 active members; the system fails if any triad fails. For the purpose of illustration, the constant failure rates are specified as:

PE	$1.1 \times 10^{-4}/\text{hour}$
NE	$0.17 \times 10^{-4}/\text{hour}$

3.3 Fault Recovery

Assumptions for FTTP Example

[see section 3.5]

1. Error detection and recovery & reconfiguration from faults in PEs always succeed, but are not instantaneous. The system fails if a second (*near-coincident*) fault occurs in any other component (PE or NE) during attempted recovery from a first fault in a PE.
2. Half of the PE faults are transient, and can be recovered from without discarding the affected PE. The remainder of faults are permanent. The recovery hazard rate (from transient faults) is constant at 1/3.6 seconds. Coverage of NE faults always succeeds, and is instantaneous.

3.4 Fault-Tree Models

3.4.1 Configuration #1

Notation

T_{ij} member i of triad j
 TS_j spare for triad j .

Configuration #1 (figure 7) divides the active elements of a triad among NE1, NE2, NE3, and uses the PEs on NE4 as spares. The PEs that are in the same relative position on the first 3 NEs form a triad, and the PE in the same relative position on NE4 serves as a hot spare for the triad.

The fault-tree model for configuration #1 (figure 8) uses 4 functional-dependency gates (FDEP) to reflect the dependence of the PEs on the NEs. The FDEP gates are not explicitly connected to the other gates in the tree, since the reliability requirements (all 4 triads must be operational) do not explicitly mention the NEs. Figure 8 shows four 3/4 gates connected to the top OR gate, one 3/4 gate for each triad. A triad fails when only 1 member PE remains (3 of the 4 PEs in the triad have failed).

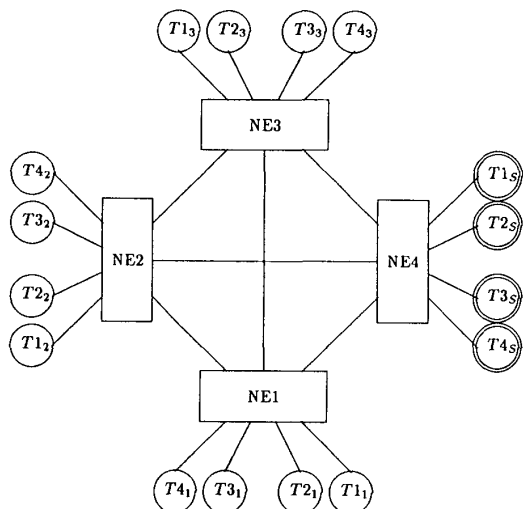


Figure 7. Configuration #1 [One spare per triad]

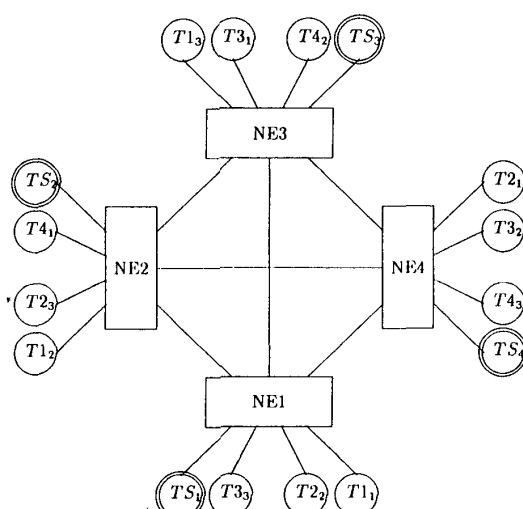


Figure 9. Configuration #2 [One spare per NE]

3.4.2 Configuration #2

Configuration #2 is an FTTP cluster with hot spares distributed across the NEs instead of grouped on the same NE (figure 9). The spare PE on each NE can substitute for any failed PE connected to the same NE. That is, TS_1 is a PE that can substitute for a failed PE connected to NE1.

The fault-tree model of this system (figure 10) is more complex than the one in section 3.4.1. The functional-dependency gates FDEP again reflect the dependence of the PEs on the NEs. A triad failure is again attributed to losing the majority of operational member PEs, but it is more difficult to describe the failure of a member of the triad. A member of the triad fails if:

- it and its spare fail, or
- its spare is not available when needed.

The spare is not available if some other PE on the same NE fails and uses the spare before it is needed by the first PE. For example, in figure 10 the leftmost OR gate that inputs into the leftmost 2/3 gate represents the failure of member #1 of triad #1, which fails if:

- both $T1_1$ & TS_1 fail, or
- TS_1 is being used because another failure has already occurred when $T1_1$ fails. TS_1 is already in use when $T1_1$ fails if either $T2_2$ or $T3_3$ (the other 2 active PEs on the same NE) have failed before $T1_1$ does. This condition is reflected in the Priority-AND gate that inputs to the same OR gate. □

There is a similar structure of AND and Priority-AND gates to represent the failure of the other members of the triads. □

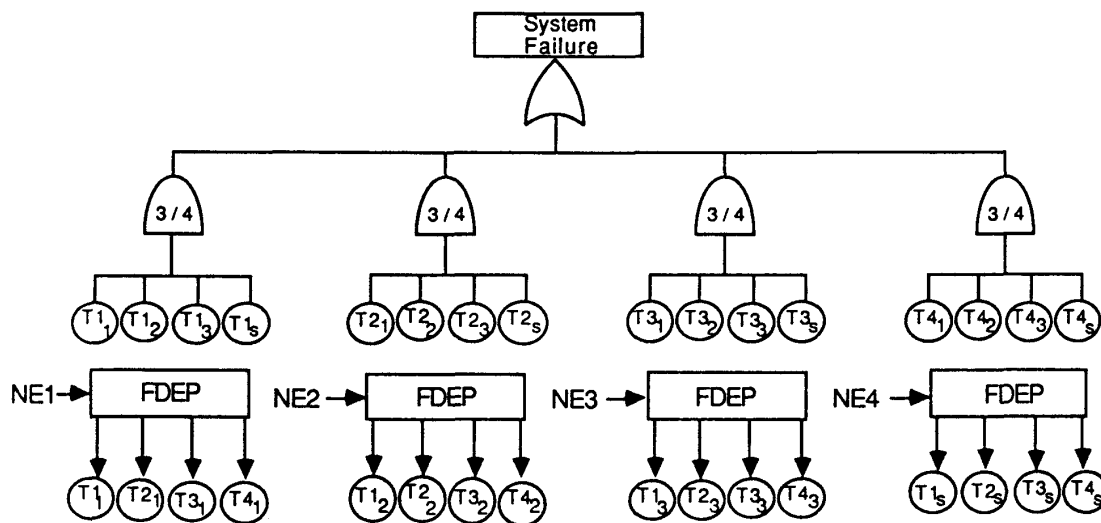


Figure 8. Fault-Tree Model for Configuration #1

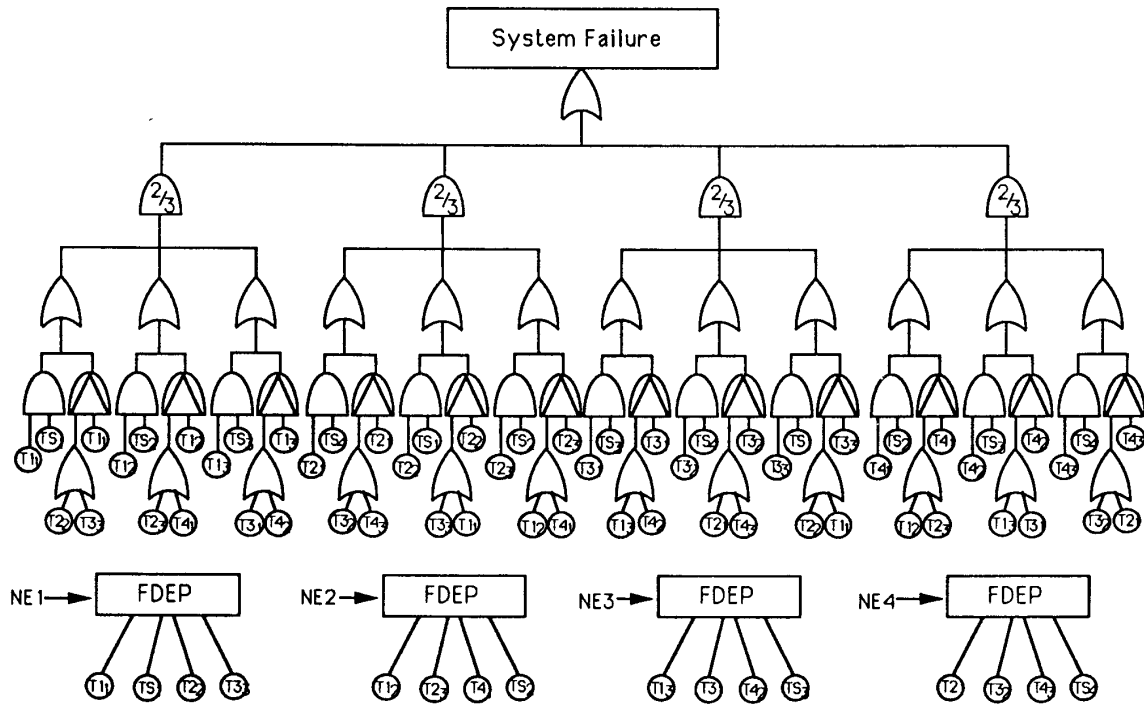


Figure 10. Fault-Tree Model for Configuration #2

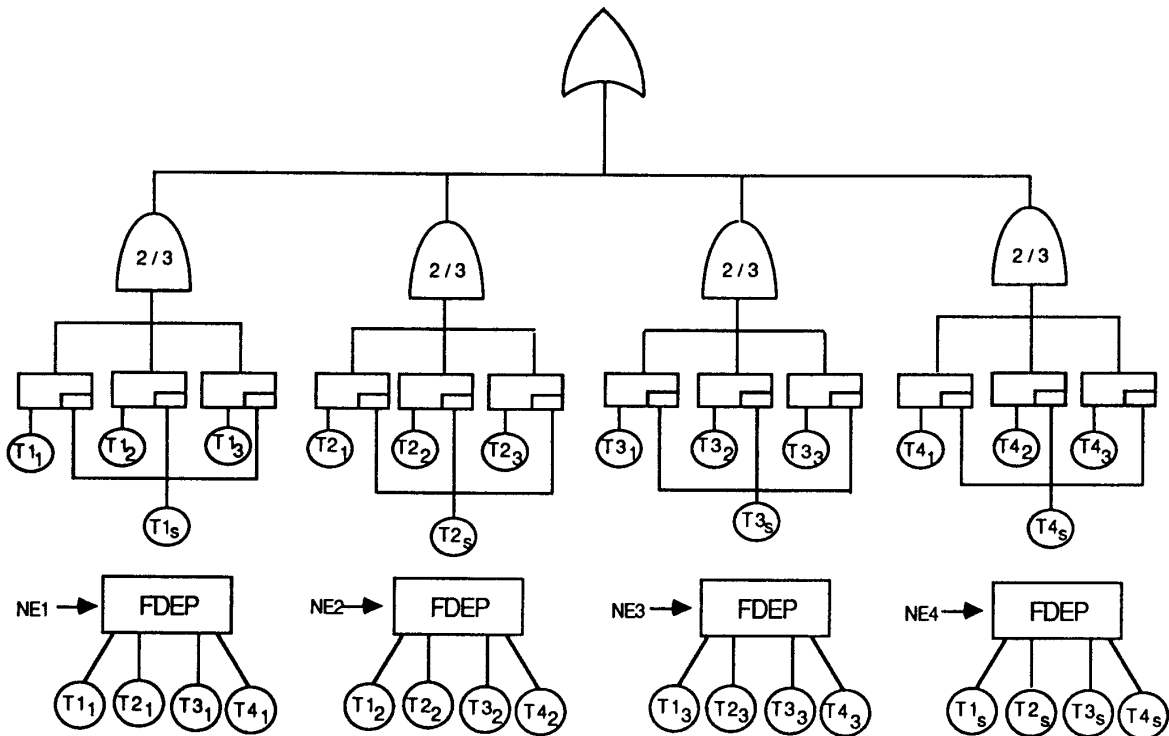


Figure 11. Fault-Tree model for Configuration #3
[One CSP per triad]

3.4.3 Configuration #3

Configuration #3 is used to investigate the effect on reliability of keeping the spare PEs unpowered until needed. The FTTP configuration modeled in this section is the same as configuration #1 (figure 7) except that the spare PEs are cold rather than hot. There is one spare PE for each triad, and all spare PEs are connected to the same NE. The fault-tree model for this system (figure 11) uses the CSP gate. There is 1 CSP gate for each member of each triad, where the initially active members of the triad are used as the primary inputs. The basic event representing the PE which is a cold-spare is connected to all 3 CSP gates since it can substitute for any of the PEs. Once the cold-spare PE has been activated to replace a failed triad-member it is unavailable to cover any further failures in the triad.

3.5 Results

This section presents the results for the models of the 3 FTTP configurations for a mission length of 10 hours. Table 1 compares the unreliability of the 3 configurations. We solved a truncated model (described in more detail later in this section) which produces bounds on the unreliability from a partial solution of the model. Table 1 shows the bounds on the unreliability, and the best case (optimistic) estimate of the probabilities of exhaustion of NEs (exh NE), exhaustion of PEs (exh PE) and near coincident faults (NCF).

Configuration #2 (hot spares are distributed across the NEs) not only required a more complicated fault tree for analysis, but also was appreciably less reliable than configuration #1. In configuration #2, 2 NE failures (alone) can kill the system, since 2 NE failures removes 2 members from at least 1 triad. For example, if NE1 & NE2 both fail, then T_1 & T_2 are both disabled, and no spare is available to replace them (because of the functional dependencies). The solution of the model for configuration #2 shows that the predominant cause of failure is the exhaustion of NEs. In configuration #1, the loss of 2 NEs (alone) does not cause any triad to fail, even though it can render all the spare PEs unusable.

In configuration #3, the spare PEs remained unpowered until needed, resulting in a modest decrease in unreliability. Since near-coincident faults contributed more highly to the system unreliability, the effect of keeping the PEs unpowered was not as important as might be anticipated.

For all 3 models, the Markov chain was truncated after considering 2 or 3 component failures, and so a pair of bounds on the actual reliability were generated. The bounds were tight enough after considering only 2 component failures for configuration #2, but we needed to consider a larger model for the other two configurations. The reason that the bounds were tighter for configuration #2 is that there were an appreciable number of failure states encountered when only considering 2 component failures. In configurations #1 & #3 there were not many failure states with only 2 failed components. Unfortunately, the number of states in a Markov chain increases exponentially with the number of component failures considered, so the increase in accuracy is accompanied by a large increase in solution times. Table 2 compares the results obtained from the

TABLE 1
Results of the Solution of All 3 FTTP Models
[Body of table gives probabilities (10^{-8})]

Configuration	#2: Hot spare/NE	#1: Hot spare/triad	#3: CSP/triad
(Best case) unreliability	20.7	4.06	2.64
(Worst case) unreliability	41.7	4.07	2.66
(Best case) exh. NE	17.4	0.135	0.104
(Best case) exh. PE	0.327	0.910	0.705
(Best case) NCF	3.02	3.02	1.83

TABLE 2
Comparison of Accuracy and Model Size
[Probability (10^{-8})]

Configuration	#2: Hot spare/NE	#1: Hot spare/triad	#3: CSP/triad
<i>Truncated at 2 component failures</i>			
(Best case) unreliability	20.7	4.06	2.63
(Worst case) unreliability	41.7	24.2	13.2
Number of states	201	123	225
Number of transitions	877	581	817
Runtime (CPU seconds)	138	99	99
<i>Truncated at 3 component failures</i>			
(Best case) unreliability		4.06	2.64
(Worst case) unreliability	<i>analysis not necessary for this example</i>	4.07	2.66
Number of states		961	2307
Number of transitions		5469	9777
Runtime (CPU seconds)		2653	5055

smaller model (truncated after 2 component failures) and the larger model (truncated after 3 component failures), as well as the size of the models and the run time for the complete generation and solution of the model on a DECstation 3100.

4. A MISSION AVIONICS SYSTEM

Acronyms

ASID Advanced System Integration Demonstration
MAS mission avionics system
VMS vehicle management system

4.1 System Description

The ASID project was the first large effort in the development of the PAVE PILLAR architecture for advanced tactical fighters. The Boeing Military Airplane Company was one of five contractors who designed implementations of the PAVE PILLAR project. A unique feature of the Boeing implementation [14] is the use of dual processor pairs wherever a single processor is required. This processor-pair uses comparison monitoring to achieve very high levels of error detection. For

critical functions, high levels of reliability are assured by using redundant processor-pairs in duplex or triplex mode. We analyze the reliability of the critical functions of the mission avionics subsystem of the ASID.

There are several critical functions within the MAS. The loss of any of these functions causes the system to fail. These critical functions include:

- the VMS,
- the crew station: control & display functions,
- mission & system management,
- local path generation,
- scene & obstacle following.

The VMS provides: a) airframe control, including flight and propulsion control, and b) utility system management & control. The crew station displays information to the pilot, contains mechanisms for pilot control actions, and manages crew station activity. The mission & system management allocates resources for real-time control functions.

Figure 12 is a block diagram of the architecture of the critical MAS. One processing unit each is required for: a) the crew station, b) local path generation, and c) mission & system management. Each of these processing units is supplied with a hot spare to take over control if the primary processor detects

an error. Each of the processing units is a pair of tightly coupled processors — to maximize the probability of error detection and minimize latency. Although there are 4 active processors for each of these functions, we treat the processor-pairs as a single processing unit, since they are not used independently. When a mismatch of results is detected, both members of the processing pair are removed from the system. Figure 12 thus shows that there are 2 processing units for these functions — the primary unit, and a hot spare.

The “scene & obstacle following” and VMS both require more functionality than one processing unit can provide, and thus each use 2 processing units. The “scene & obstacle following” processing units are also replicated, providing a hot spare. The VMS is triplicated, thus providing 2 hot spares.

In addition to the hot spares, 2 additional pools of spares are provided, each containing 2 spare processing units. Pool#1 covers the first 2 processor failures in the subsystems other than the VMS; pool#2 covers the first 2 processor failures in the VMS.

The subsystems are connected via 2 triplicated bus systems: #1 is a data bus, #2 is the mission-management bus. The replicated memory-system is connected to the data bus. The VMS has an additional triplicated bus, the vehicle-management bus.

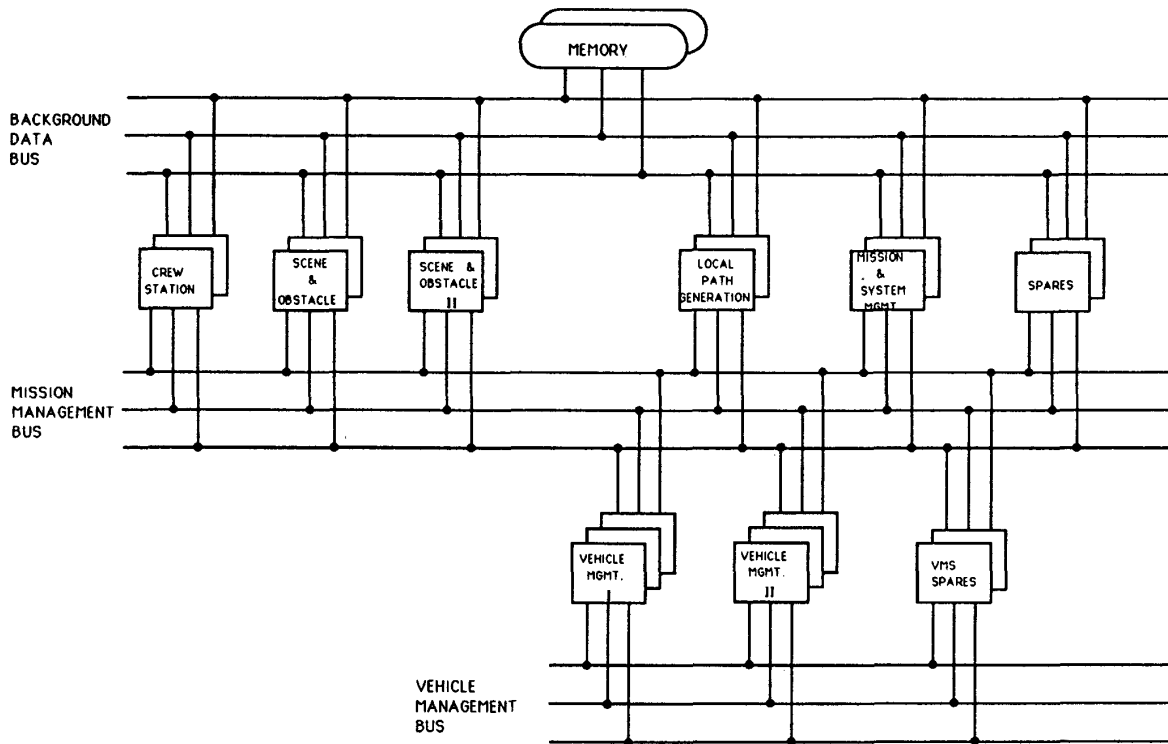


Figure 12. Block Diagram of Mission Avionics System Architecture

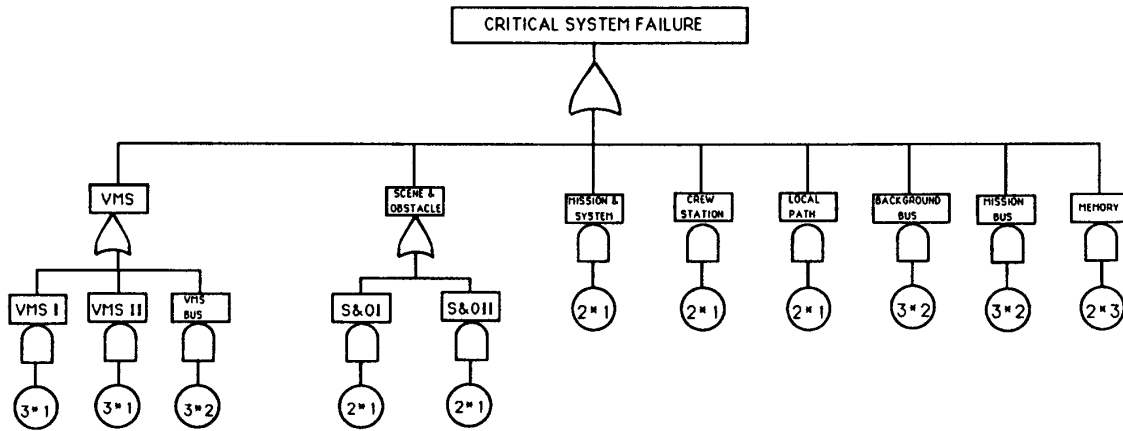


Figure 13. Fault-Tree Model of Mission Avionics System [No pooled spares]

4.2 Failure Criteria & Parameters

The system fails if:

- any of the functions cannot be performed, or
- both of the 2 memories fail, or
- all 3 of any one type of bus fail.

The following constant failure rates (10^{-6} /hour) were used.

processor pairs	25
buses	2.5
memories	1.0

4.3 Error & Fault Recovery

Error detection is perfect (because of the processing pairs) but it takes between 0.5 second and 5 seconds (uniformly distributed) for recovery to occur. If a second, near-coincident fault occurs during this interval, the system fails.

4.4 Fault-Tree Model

The fault-tree model of the MAS is complicated by the presence of the pooled spares. For ease of exposition, we first present a fault-tree model that ignores the pooled spares. We then describe the methodology for modeling pooled spares via a fault tree with sequence-dependency gates, by way of a simple example. Finally, we define the full fault-tree model of the MAS including the pooled spares.

4.4.1 Fault tree with no pooled spares

Figure 13 is the fault-tree model of the MAS with no pooled spares. This fault tree shows that the system fails if:

- any of the critical functions fail, or
- either of the bus systems fail, or
- both memories fail.

There are 3 types of components: processing units (type 1), buses (type 2) and memories (type 3). The crew station, for example, uses 2 components of type 1, so its basic event is labeled $2*1$. The memory system uses 2 memories and is thus labeled $2*3$, while the mission-management bus system uses 3 buses and is labeled $3*2$.

4.4.2 Modeling Pooled Spares

Before we add the pooled spares to the MAS fault-tree model, consider a simple system with two duplexes and 2 pooled spares. This simple system represents the “scene & obstacle following”. Figure 14 shows the fault-tree model of a 2 duplex system; figure 15 shows the equivalent Markov chain. This equivalent Markov chain is determined automatically by HARP.

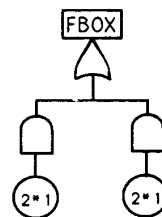


Figure 14. Fault-Tree Model of a 2-Duplex System

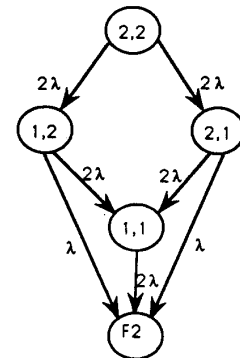


Figure 15. Markov-Chain Model of a 2-Duplex System

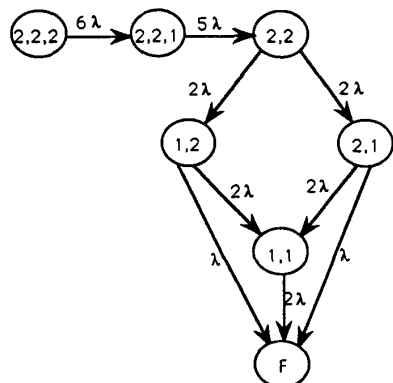


Figure 16. Markov-Chain Model of a 2-Duplex System with 2 Pooled Spares

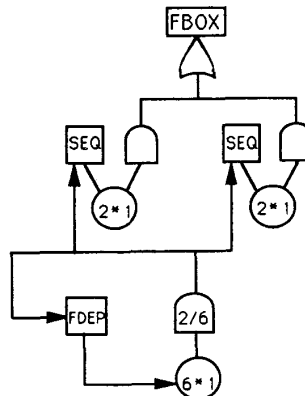


Figure 17. Fault-Tree Model of a 2-Duplex System with 2 Pooled Spares

Next, consider the desired Markov chain representation of the same 2-duplex system with the addition of 2 pooled spares (figure 16). The 2 pooled spares cause 2 states to be added to the front of the Markov chain. These 2 states represent the first 2 failures in the system, which deplete the spares. After the first 2 failures, 2 functioning duplexes remain, and the rest of the Markov chain in figure 16 is identical to that in figure 15.

We can use the fault tree in figure 17 to represent the 2-duplex system with 2 pooled spares. In figure 17, the combination of a) the 2/6 gate (which fires after the first 2 of 6 failures), and b) the FDEP gate, creates a Markov chain that models the first 2 failures of 6 components. After the first 2

failures, the FDEP gate stops any more of the 6 components from failing. The two SEQ gates in figure 17 do not allow the two basic events labeled with 1*2 to begin to fail until after the 2/6 gate has fired. After the 2/6 gate has fired, then the rest of the fault tree (identical to the one in figure 14) can occur as usual. In this rather unusual use of the FDEP & SEQ gates, the outputs of these gates are not connected to the top event in the fault tree because the condition associated with the outputs are not explicitly needed to determine system failure. This combination of FDEP & SEQ gates can be used in a more general setting to tie multiple Markov chains together.

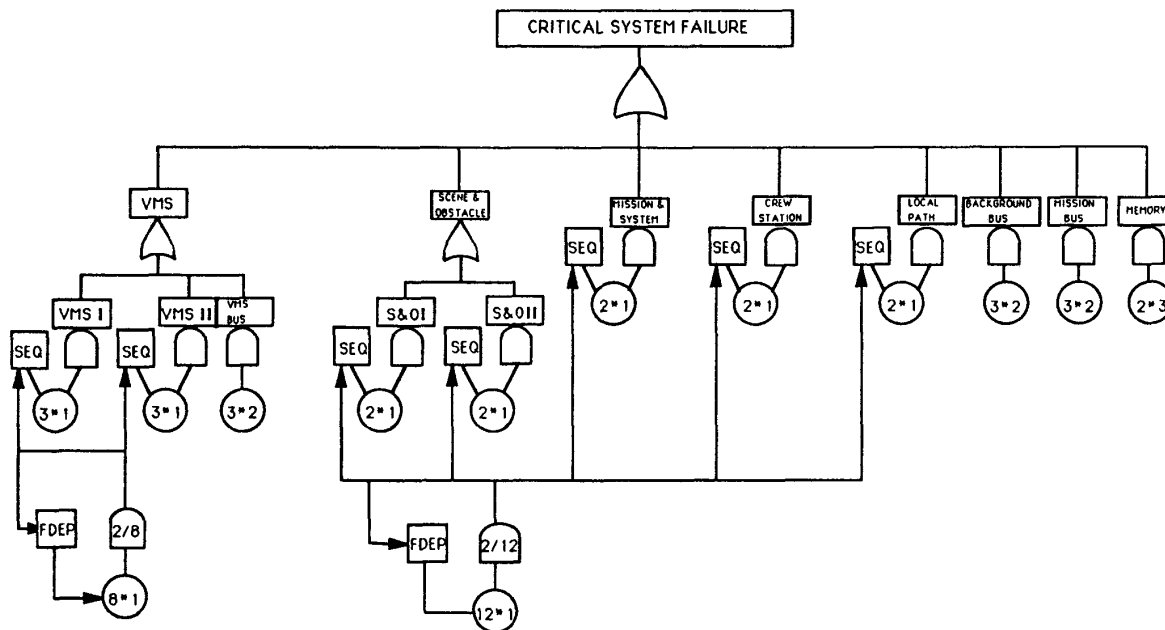


Figure 18. Full Fault-Tree Model of Mission Avionics System

4.4.3 Full Model and Results

Figure 18 is the full fault-tree model of the MAS including the pools of spares. The leftmost FDEP & SEQ gates show the 2 spares for the VMS while those to the right represent the other 2 spares.

Because of the sequence-dependency gates, this fault tree cannot be solved by usual combinatorial methods, but rather must be converted to a Markov chain for solution. HARP performs this conversion automatically, and produces a truncated Markov chain with 479 states and 2517 transitions. The Markov model is truncated after considering 5 component failures. Instead of producing an exact solution of the model, bounds that encompass the reliability of the full model are produced. For a 200 hour interval, the unreliability lies between 1.138×10^{-7} and 1.146×10^{-7} .

5. 3 FAULT-TOLERANT HYPERCUBE ARCHITECTURES

We next model three fault-tolerant hypercube architectures. All three contain 8 processing nodes connected in a hypercube of dimension 3. All three consist of 2 fault-tolerant modules with each module containing 4 processing nodes. The three architectures differ in —

- the ways that spare nodes are incorporated into the fault-tolerant modules,
- the way that messages are routed between processing nodes,
- the architecture of the individual processing nodes.

5.1 System Description

5.1.1 Architecture 1

Architecture 1 is based on the hierarchical approach to sparing proposed by Rennels [15] and is depicted in figure 19. It consists of 2 fault-tolerant modules of processing nodes. Each module contains 4 processing nodes and one hot spare node. The spare is connected by a port to each of the 4 active processors in the module.

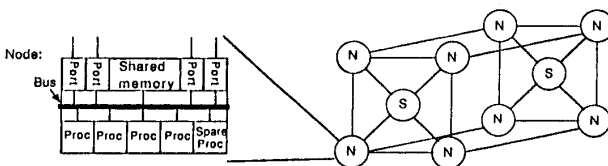


Figure 19. Architecture 1 [15]

The processing nodes themselves are comprised of 5 individual processors (4 active processors and a spare) which communicate over a bus and share a memory module. The memory module contains spare bit-planes and spare chips within a bit plane. The processing node is connected to its neighboring nodes in the hypercube by 4 ports. Three ports communicate across

the 3 dimensions of the hypercube, and port#4 communicates with the spare processing node of the module. Messages can be routed through the spare within the module in the event that a direct connection between two active nodes has failed — even if the spare has not been used to replace any failed processing nodes within the module. The spare therefore provides redundancy for connections as well as for processing nodes.

5.1.2 Architecture 2

Architecture 2, also depicted in figure 19, is identical to Architecture 1 except that the ports within each processing node are replaced by hyperswitch ports [16]. The hyperswitch allows an adaptive routing method to avoid failed or congested links within the hypercube. It permits any 2 nodes of the hypercube to communicate as long as there exists any nonfailed path between them anywhere throughout the hypercube. Architecture 2 also allows messages to be routed through the spare.

5.1.3 Architecture 3

Acronym

DMA direct memory access

Architecture 3 [17], depicted in figure 20, again configures processing nodes into 2 fault-tolerant modules (each containing 4 active processing nodes and one spare). It differs from Architectures 1 & 2 in several important ways:

- The inter-node connections are mediated by decoupling switches rather than being direct connections between ports of neighboring nodes.
- The hypercube connectivity and the switching of spares online and failed nodes offline is performed using these decoupling switches [17]. The switches are intended to be comparatively simple devices. One consequence of using the switches to control access to the spare nodes is that the spares cannot provide redundancy for links as was possible for architectures 1 and 2.

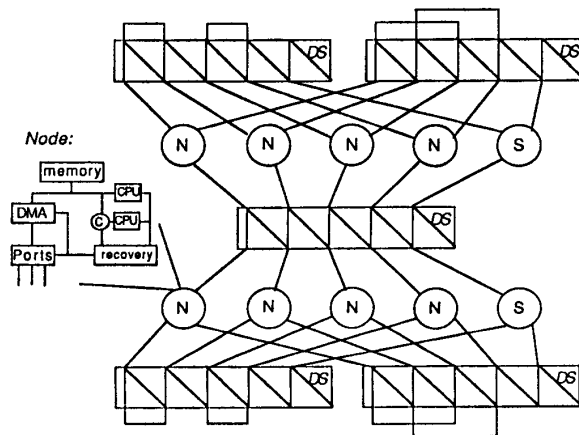


Figure 20. Architecture 3 [17]

- The processing nodes of the hypercube are much simpler and contain processors that are much less powerful than those of Architectures 1 and 2. Each processing node consists of 2 processors which perform identical computations in parallel. The output is compared to detect faults. A recovery module is responsible for fault handling upon the detection of a processor fault. The node may either declare itself failed or attempt a reconfiguration to a simplex configuration upon detection of such a processor fault. Both processors have access to a single memory module and a DMA module.
- Each processing node communicates with the outside world through three ports, each of which connects the node to its neighbor across one dimension of the hypercube. □

For this discussion we examine only the processing nodes of the various candidate architectures in isolation from the ensemble. The processing nodes of each architecture themselves can be configured in a variety of ways. The configuration can affect the reliability and power consumption of the node, which can in turn affect the ensemble reliability of the hypercube multiprocessor. The analysis in [2] considers the complete architectures.

5.2 Failure Criteria & Parameters

The processor nodes for Architectures 1 & 2 are identical, so their failure criteria are closely related. The difference between them is due to the message-routing scheme in each architecture. A processor node for the two architectures fails if:

- the memory fails, or
- the bus fails, or
- 2 out of the 5 processors fail (recovery from the first processor failure is achieved by switching in the spare to take the failed processor's place), or
- the node is disconnected from the other processing nodes in the hypercube. □

The events that cause a node to be disconnected differ for the two architectures.

The routing algorithm used for Architecture 1 allows only one path between each pair of nodes in the hypercube. However, since the spare processing node in each of the 2 fault-tolerant modules can relay messages within the module when a direct connection between 2 nodes in the module is not possible, it takes the failure of 2 of the 4 ports in a processing node to disconnect the node. In Architecture 2, a hyperswitch is used instead of the single path routing algorithm, so that all 4 ports in a node must fail in order to disconnect the node.

A processing node for Architecture 3 fails if:

- the memory fails, or
- the DMA module fails, or
- both processors fail, or
- any of its 3 ports fail (since the single path routing algorithm is used for this architecture). □

The component failure rates (10^{-6} /hour) for all three architectures are:

Active processor (Architectures 1 and 2) ..	1.990
Active processor (Architecture 3)	0.2306
Warm spare processor(Architecture 2)	1.0
Shared Memory (Architectures 1 and 2)...	0.3477
Memory (Architecture 3)	0.1147
DMA module (Architecture 3)	0.3477
Intra-node bus(Architectures 1 and 2)	0.1147
Hyperswitch and I/O port (all architectures)	0.3477

5.3 Fault Recovery

Assumptions for the Hypercube Example

- A processor error can be detected, the underlying fault located, and the spare successfully switched-in to replace the failed processor 95% of the time. The time required to do all of this is uniformly distributed between 0.9 seconds and 1.1 seconds. The remaining 5% of the time, the recovery does not succeed, leading to node failure.
- Detection & deactivation of a faulty port is successful 98% percent of the time. The time required for this is exponentially distributed with a mean of 0.1 sec. The remaining 2% of the time, a port error is not successfully detected, leading to node failure.
- No transient restoration is attempted, ie, all faults are considered to be permanent.

5.4 Fault-Tree Models

5.4.1 Hot spares

Figures 21 & 22 model the processing nodes in Architectures 1 & 2 when the spare processor in the node is a hot spare. The fault trees differ only in the modeling of port failures:

- Architecture 1 fails when 2 of the 4 ports fail (hence the 2/4 gate),
- Architecture 2 doesn't fail until all 4 ports have failed (hence the AND gate).

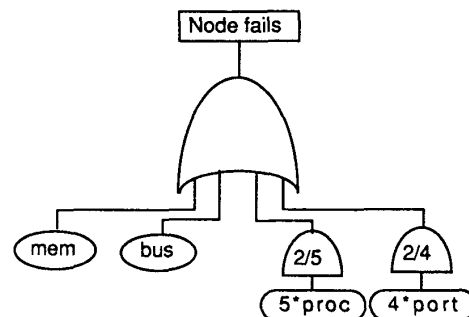


Figure 21. Fault-Tree Model of Architecture 1 Processing Node with Hot Spares

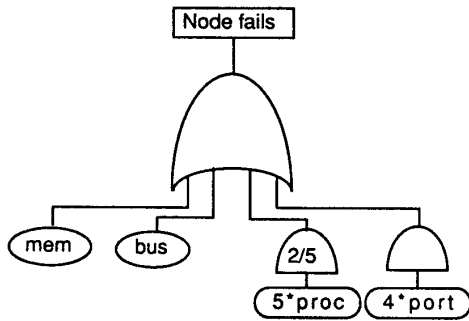


Figure 22. Fault-Tree Model of Architecture 2 Processing Node with Hot Spares

Figure 23 is a fault-tree model for the processing nodes of Architecture 3.

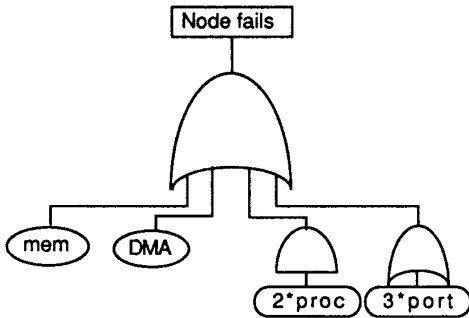


Figure 23. Fault-Tree Model of Architecture 3 Processing Node with Hot Spares

5.4.2 Cold spares

Power consumption by a multiprocessor with spare nodes can be reduced by having the spares unpowered until they are needed to replace a failed active processor; such spares are modeled here as *cold spares*. In HARP this type of configuration is modeled using the CSP gate, as depicted in figure 24 by a fault tree for Architecture 2. The CSP gate ensures that the spare processor does not fail until one of the 4 active processors fails. The 2/5 gate in parallel with the CSP gate maintains the requirement that 2 processor failures cause the node to fail. Such a configuration not only reduces power consumption, but enhances the reliability of the processing node.

5.4.3 Warm spares

Instead of being unpowered, the spare may be partially powered; such spares are modeled here as *warm spares*. The warm spare is modeled in HARP using the SEQ gate as shown in figure 25 for Architecture 2. In this example two basic event nodes appearing as inputs to the OR gate whose output feeds into the functional-dependency gate are used to represent the 4 active processors and spare before any processor failures. Upon the first

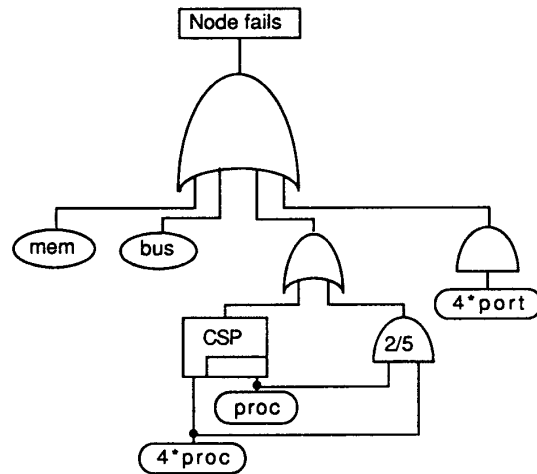


Figure 24. Fault-Tree Model of Architecture 2 Processing Node [Cold spares]

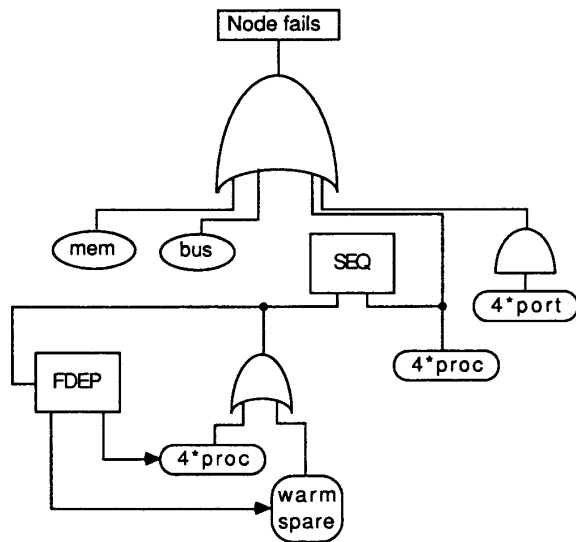


Figure 25. Fault-Tree Model of Architecture 2 Processing Node [Warm spares]

failure of a processor (either active or spare), these two basic events are *turned off*, as far as the fault tree is concerned, by the FDEP gate. The 4 remaining processors, now all active, are represented by the 4*processor basic event which appears as the rightmost input to the sequence-dependency gate. This basic event had been *turned off* prior to the first processor failure by the sequence-enforcing gate. After the first processor failure, the leftmost input to the sequence-enforcing gate is turned on, which *turns on* the basic event that is its rightmost input (ie, the processors of this basic event are now permitted to fail). Because this basic event is also an input to the top OR gate of the fault tree, a subsequent failure of any of the 4 processors causes the node to fail, again maintaining the requirement that failure of 2 of the 5 processors cause node failure.

5.5 Results

Figure 26 compares the 10-year unreliabilities of the processing nodes of each architecture, assuming all of them use hot spares. The unreliability of Architecture-3 processing nodes is much lower than those for Architectures 1 & 2, reflecting that: a) the reliability of individual Architecture-3 processors is much greater than that of the others, and b) there are only 2 that can fail instead of 5. As anticipated, the unreliability for Architecture-2 nodes is slightly better than the unreliability for Architecture-1 nodes because of the number of port failures required to cause a node failure.

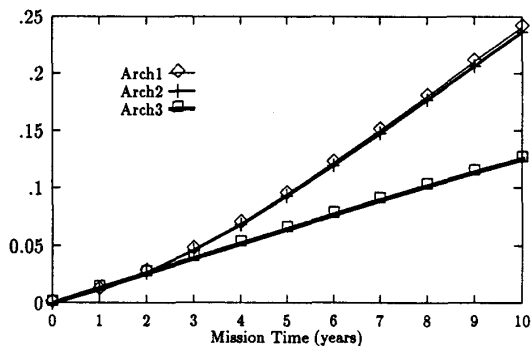


Figure 26. Comparison of Node Unreliabilities of All Architectures [Hot spares]

Figure 27 shows the 10-year unreliabilities for Architecture-2 processing nodes using hot, warm, and cold spares. In general, the reliability increases from configuration to configuration in that order. This is to be anticipated since the failure rate of the spare during its inactive period decreases in that order.

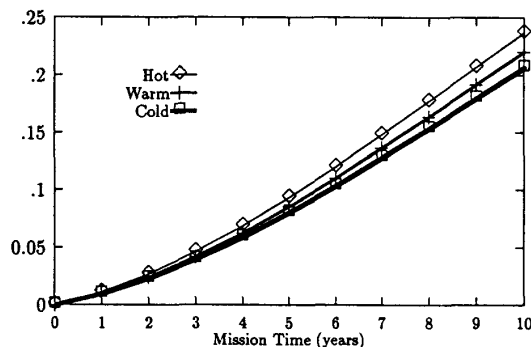


Figure 27. Unreliability of Architecture-2 Processing Nodes [Various types of spares]

ACKNOWLEDGMENT

This work was supported in part by US NASA Ames Consortium Agreement NCA2-617 and by US NASA Langley grant NAG-1-70.

REFERENCES

- [1] J. B. Dugan, K. S. Trivedi, "Coverage modeling for dependability analysis of fault-tolerant systems", *IEEE Trans. Computers*, vol 38, 1989 Jun, pp 775-787.
- [2] M. A. Boyd, J. Tuazon, "Fault tree models for fault tolerant hypercube multiprocessors", *Proc. Reliability & Maintainability Symp*, 1991 Jan, pp 610-614.
- [3] J. B. Dugan, S. J. Bavuso, M. Boyd, "Fault trees and sequence dependencies", *Proc. Reliability & Maintainability Symp*, 1990 Jan, pp 286-293.
- [4] J. B. Dugan, K. S. Trivedi, M. K. Smotherman, R. M. Geist, "The hybrid automated reliability predictor", *AIAA J. Guidance, Control & Dynamics*, vol 9, 1986 May-Jun, pp 319-331.
- [5] S. J. Bavuso, J. B. Dugan, K. S. Trivedi, E. M. Rothmann, et al, "Analysis of typical fault-tolerant architectures using HARP", *IEEE Trans. Reliability*, vol R-36, 1987 Jun, pp 176-185.
- [6] J. D. Esary, H. Ziehms, "Reliability analysis of phased missions", *Reliability and Fault Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety Assessment*, (Barlow, Fussell, Singpurwalla, eds), 1975, pp 213-236; SIAM.
- [7] B. Moret, M. Thomason, "Boolean difference techniques for time-sequence and common-cause analysis of fault trees", *IEEE Trans. Reliability*, vol R-33, 1984 Dec, pp 399-405.
- [8] M. K. Smotherman, K. Zemoudeh, "A non-homogeneous Markov model for phased mission reliability analysis", *IEEE Trans. Reliability*, vol 38, 1989 Dec, pp 585-590.
- [9] J. B. Dugan, "Fault trees and imperfect coverage", *IEEE Trans Reliability*, vol 38, 1989 Jun, pp 177-185.
- [10] E. J. Henley, H. Kumamoto, *Reliability Engineering and Risk Assessment*, 1981; Prentice-Hall.
- [11] M. A. Boyd, *Dynamic Fault Tree Models: Techniques for Analysis of Advanced Fault Tolerant Computer Systems*, PhD Thesis, 1991 Apr; Dept of Computer Science, Duke University.
- [12] R. E. Harper, J. H. Lala, J. J. Deyst, "Fault tolerant parallel processor architecture overview", *Proc. 18th Symp. Fault Tolerant Computing*, 1988, pp 252-257.
- [13] R. E. Harper, "Reliability analysis of parallel processing systems", *Proc. 8th Digital Avionics Systems Conf*, 1988; pp 213-219.
- [14] S. W. Behnen, W. A. Whitehouse, R. J. Farrell, F. M. Leahy, et al, "Advanced system integration demonstrations (ASID) system definition", Tech. Rep, 1984; USAF Wright Aeronautical Laboratories.
- [15] D. A. Rennels, "On implementing fault-tolerance in binary hypercubes", *Proc. IEEE Int'l Symp. Fault-Tolerant Computing, FTCS-16*, 1986 Jul, pp 344-349.
- [16] E. Chow, J. Peterson, H. Madan, "Hyperswitch network for the hypercube computer", *Digest 13th Symp. Computer Architecture*, 1988 May, pp 90-99.
- [17] S. C. Chau, A. Liestman, "Proposal for a fault-tolerant binary hypercube architecture", *Proc. IEEE Int'l Symp. Fault-Tolerant Computing, FTCS-19*, 1989 Jun, pp 323-330.
- [18] J. B. Fussell, E. F. Aber, R. G. Rahl, "On the quantitative analysis of priority-AND failure logic", *IEEE Trans. Reliability*, vol R-25, 1976 Dec, pp 324-326.

AUTHORS

Dr. Joanne Bechta Dugan; Department of Computer Science; Duke University; Durham, North Carolina 27706 USA.

Joanne Bechta Dugan: For biography see *IEEE Trans. Reliability*, vol 40, 1991 Apr, p 55.

Salvatore J. Bavuso; MS 478; NASA Langley Research Center; Hampton, Virginia 23665 USA.

Salvatore J. Bavuso is a senior researcher at NASA Langley Research Center in Hampton, Virginia. He received the BS in Mathematics from Florida State University in 1964 and the MS in Applied Mathematics from North Carolina State University at Raleigh in 1971. He has been instrumental in the development of advanced reliability modeling technology for over a decade and is the NASA project manager for the HARP and CARE III programs.

Mark A. Boyd; MS 269-3; NASA Ames Research Center; Moffett Field, California 94035 USA.

Mark A. Boyd was born in Pennsylvania in 1957. He was awarded a BA in Chemistry from Duke University in 1979, an MA in Computer Science and a PhD in Computer Science from Duke University in 1986 & 1991. He is a Computer Engineer in the Information Sciences Division at NASA Ames Research Center. His research interests include mathematical modeling of fault tolerant computing systems and parallel processing. He is a member of the IEEE and the ACM.

Manuscript TR91-056 received 1991 April 10; revised 1991 November 15.

IEEE Log Number 06064

◀TR▶

Annual Reliability and Maintainability Symposium

The P. K. McElroy Award for Best Paper

was bestowed on Charles H. Stapper, John A. Fifield, and Howard L. Kalter for their paper "High-Reliability Fault-Tolerant 16-Mbit Memory Chip" that was given at the 1991 Symposium in Orlando. For more information, see the *gold* section of your copy of the 1992 *Proceedings*.

Each year the Symposium presents The P. K. McElroy Award for the *best paper* at the previous Symposium. The Award consists of a plaque and a \$1000 honorarium. There are two criteria for *best paper*:

- The content of the written paper is lucid, excellent, and important to the theory and/or practice of R&M engineering.
- The verbal presentation of the paper at the Symposium is likewise lucid and excellent.

P. K. McElroy was an intensely practical person. Papers that receive the Award must be able to make a difference to R&M engineers and/or managers. It is not enough that the content be competent and important; that competence and importance must be readily obvious in both the written and verbal presentations.

Before the Symposium, the content of each written paper is examined by the Program Committee for technical excellence and clarity of exposition. The best of the papers are chosen and referred to a select group of past General Chair'n of the Symposium. Each person in that group listens to each presentation, and that group chooses the *best paper* to receive the P. K. McElroy Award.