NASA Technical Memorandum **102747**

TUTORIAL

ADVANCED FAULT TREE APPLICATIONS USING HARP

Joanne Bechta Dugan
Duke University
Durham, North Carolina

Salvatore J. Bavuso
NASA Langley Research Center
Hampton, Virginia

Mark A. Boyd
Duke University
Durham, North Carolina

NOVEMBER 1993

National Aeronautics and
Space Administration

Langley Research Center
Hampton, VA 23681-0001

Tutorial: Advanced Fault Tree Applications using HARP

Contents

Abstract

Reliability analysis of fault tolerant computer systems for critical applications is complicated by several factors. In this tutorial, we discuss these modeling difficulties and describe and demonstrate dynamic fault tree modeling techniques for handling them. Several advanced fault tolerant computer systems are described, and fault tree models for their analysis are presented. HARP (the Hybrid Automated Reliability Predictor) is a software package developed at Duke University and NASA Langley Research Center that is capable of solving the fault tree models presented in this tutorial.

Part I
Background

1 Introduction

Fault tolerant computer systems for critical applications are characterized by several factors which complicate their analysis. Systems designed to achieve high levels of reliability frequently employ high levels of redundancy, dynamic redundancy management, and complex fault and error recovery techniques. In this tutorial we consider advanced fault tree modeling techniques to include these factors in the analysis of system reliability.

In this tutorial, we assume the following

- Faults occur randomly and are statistically independent.
- Lifetime distributions are exponential. Faults occur at a constant average rate, which is referred to as the failure rate of the component.
- Mission lengths are relatively short, so that the probability of more than a few failures is low.
- The systems are not repairable while in use.

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

There are several possible for the reliability analysis of fault tolerant computer systems for critical applications. In addition to predicting the reliability of the system for a specified mission time, these techniques can facilitate tradeoff analysis for various fault tolerant techniques, or can be used to compare alternative architectures for a system still in the design phase. Even if a system exists only as a rough sketch on paper, analysis techniques can be used to 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 analysis of fault trees with many basic events is often expensive, both in terms of developing the model and in solving the model once it is developed. Also, several important types of dynamic behavior in advanced fault tolerant systems cannot be adequately captured in a standard fault tree model. These dynamic behaviors include transient recovery, intermittent errors, and sequence dependency. Markov models present an alternative modeling technique that is flexible enough to model nearly any such dynamic system. Tools and techniques exist for the solution of even very large Markov models. However, the construction of a Markov model for any but the simplest system can be tedious and error prone.

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 standard fault tree. The model construction and solution process is facilitated by the new model in three major ways, which are defined and demonstrated via example in this tutorial.

- *Behavioral decomposition* is used to separately define models for system structure and fault recovery.
- Several additional gates are introduced into the fault tree model to capture dynamic behavior.
- The fault tree model of system structure is internally and automatically converted to a Markov model, to which is added the fault recovery information.

These techniques have been implemented in HARP (the Hybrid Automated Reliability Predictor), a software package for the analysis of advanced fault tolerant systems, developed by NASA Langley Research Center and Duke University.

The models exampled described are all solved using HARP. The techniques implemented in HARP are described in more detail in other publications. References [4, 14, 23, 2, 3] are general papers describing HARP. More details of the models presented here, as well as other models using HARP appear in [1, 6, 7, 15, 11, 12, 10]. Modeling the recovery process is covered in detail in [13].

2 Behavioral decomposition

A common approach to modeling complex systems consists of structurally dividing the system into smaller subsystems (e.g. processors, memory units, buses), analyzing

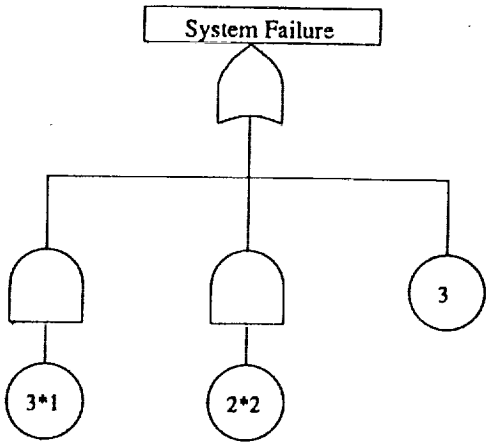

Figure 1: Fault tree model of example system

the dependability of the subsystems separately, and then combining the subsystem solutions to obtain the system solution. A system level analysis can then be effected by analyzing each subsystem separately and combining the results to obtain the final solution. This structural decomposition is allowed only if the subsystems' fault tolerant behaviors are mutually statistically independent.

An alternative to such a structural decomposition is *behavioral decomposition*. Generally, the time scale for the occurrence of faults and their associated errors is relatively long (i.e. weeks or months) while the time scale for recovery is relatively short (milliseconds). Behavioral decomposition exploits this time scale difference, by allowing an analyst to describe the two behavior types (occurrence and recovery) in separate models.

Using behavioral decomposition, the model is decomposed into fault-occurrence and repair (FORM) and fault and error handling (FEHM) submodels. The FORM contains information about the structure of the system and the fault arrival process. The FEHM (often called the *coverage* model) allows for the modeling of permanent, intermittent, and transient faults, and models the on-line recovery procedure necessary for each fault type. We describe this process of model construction by way of a simple three processor, two memory (3P2M) example system.

2.1 The fault occurrence and repair model (FORM)

We wish to model a computer consisting of three processors and two shared memories (3P2M) communicating over a shared bus. The system is operational as long as one processor can communicate with one of the memories. We describe the system structure model as a fault tree, as shown in figure 1, where the top event, System

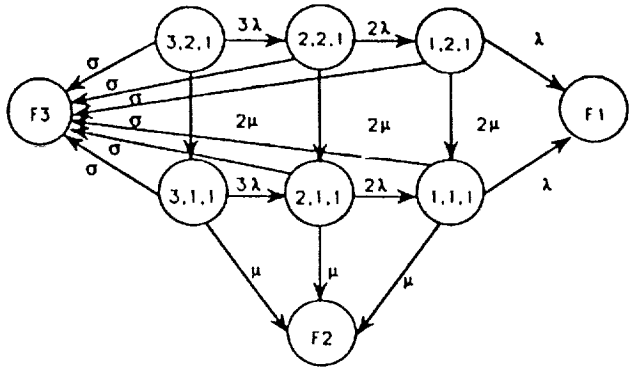

Figure 2: Markov chain model of example system

Failure is caused by bus failure OR all processors failing, OR both memories failing. The abbreviation for the combined basic event $i * j$ represents i statistically independent occurrences of component type j.

Figure 2 shows the (continuous time) Markov chain representation of the system whose fault tree is shown in figure 1. The states are labeled with an ordered triple, where element

1. denotes the number of operational processors,
2. denotes number of operational memories, and
3. denotes the state of the bus.

An arc between states (i, j, k) and $(i-1, j, k)$ is labeled with $i * \lambda$ (where λ is the failure rate of processors). Likewise, an arc between states (i, j, k) and $(i, j-1, k)$ is labeled with $j * \mu$ (where μ is the failure rate of memory units). The failure rate of the bus is σ.

F1 represents exhaustion of the processor cluster

F2 represents exhaustion of the memories, and

F3 represents failure of the bus.

The fault tree in figure 1 can be automatically converted to the Markov chain in figure 2. All possible 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 make adjustments to the Markov chain as needed.

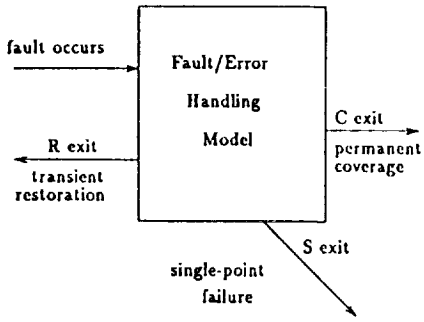

Figure 3: General structure of FEHM

2.2 The fault and error handling model (FEHM)

We next concentrate on modeling the detailed behavior of the system when a fault occurs. The general structure of a model that represents the recovery process that is initiated when a fault occurs is shown in figure 3. The entry point to the model signifies the occurrence of the fault, and the three exits signify three possible outcomes. The transient restoration exit (labeled R) represents the correct recognition of and recovery from a transient fault. A transient is usually caused by external or environmental factors, such as excessive heat or a "glitch" in the power line. It is generally believed that the vast majority of faults are transient. Successful recovery from a transient fault restores the system to a consistent state without discarding any components, for example by retrying an instruction or rolling back to a previous checkpoint. Reaching this exit successfully requires timely detection of an error produced by the fault, performance of an effective recovery procedure, and the swift disappearance of the fault (the cause of the error).

The permanent coverage exit (labeled C) denotes the determination of the permanent nature of the fault, and the successful isolation and removal of the faulty component. The single point failure exit (labeled S) is reached when a single fault causes the system to crash. This generally occurs if an undetected error propagates through the system, or if the faulty unit cannot be isolated and thus the system cannot be reconfigured.

2.2.1 Example memory FEHM

As an example of a FEHM for the memory subsystem of figure 1, a hypothetical recovery procedure is shown in figure 4.

- The memory uses an error correcting code, so a single-bit error is always detectable and correctable, and no reconfiguration is required. If 98% of all memory faults affect only a single bit, then the probability of reaching the R exit is $r = 0.98$.
- The 2% of faults that affect more than one memory bit are 95% detectable. When a multiple memory error is detected, the affected portion of memory is discarded, the memory mapping function is updated, and the needed information is reloaded from a previous checkpoint and updated to represent the current state of the system. Experimentation on a prototype system revealed that this recovery from the detected multiple memory errors works 85% of the time. Thus, the probability of reaching the C exit is the probability that a multiple fault occurs, is detected, and is recovered from $c = (0.02) \times (0.95) \times (0.85) = 0.01615$. The first two moments [1] of the time to perform this recovery have been determined by experiment to be 0.45 and 0.25 (time scale in seconds).
- There are two paths to the single point failure exit.
  1. The memory fault causes a single-point failure if a multiple-bit error is not detected (with probability 0.02×0.05),
  2. if a multiple-bit memory error is detected, but the attempted recovery is not successful. (with probability $0.02 \times 0.95 \times 0.15$)

  Thus, the probability of reaching the S exit is $s = (0.02) \times ((0.05) + (0.95) \times (0.15)) = 0.00385$.

2.2.2 Example processor FEHM

The recovery process for the faults that occur in the processor is more complex. A processor contains built-in test circuitry so that error checking occurs concurrently with instruction execution. If an error is detected, the instruction is retried immediately. Partial results are stored in case the retry is unsuccessful, so that the computation can be continued from some intermediate point (called a checkpoint). The process of continuing a computation from a previously saved checkpoint is called a rollback. In some cases the fault is such that the rollback is not successful, so the computation must start over after a system-level recovery procedure is invoked.

An example of a processor fault coverage model is shown in figure 5, and represents the following recovery procedure [20].

[1] If, in successive experiments, recovery times are T_1, T_2, ..., T_k, then the mean (first moment) is $\frac{1}{k}\sum_{j=1}^{k} T_j$ and the second moment is $\frac{1}{k}\sum_{j=1}^{k} T_j^2$.

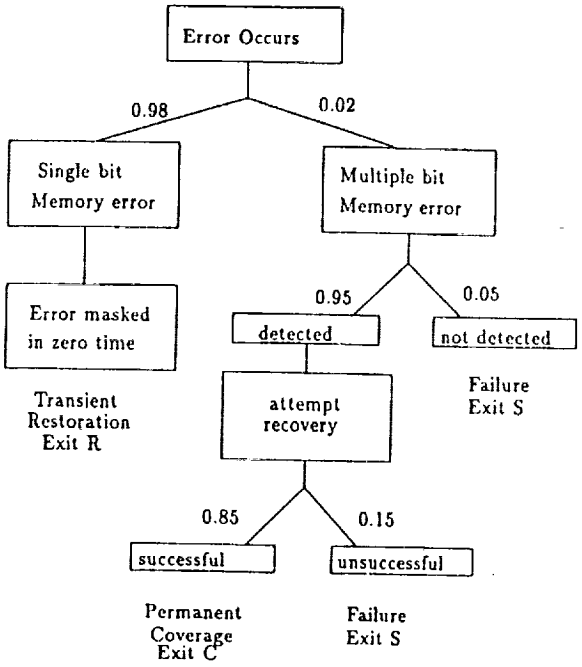

Figure 4: Recovery model for memory subsystem

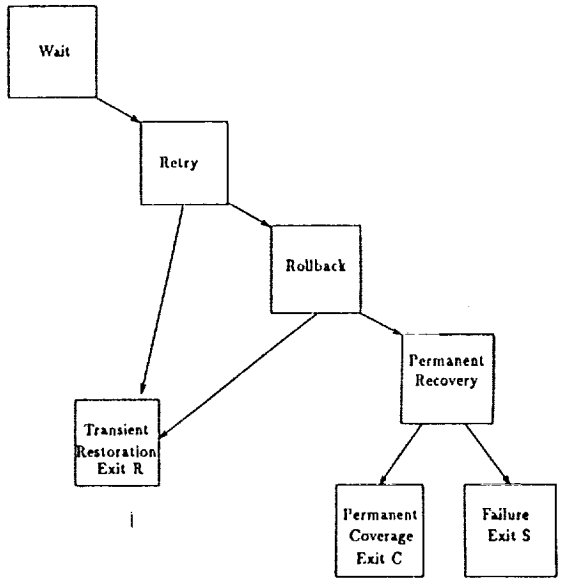

Figure 5: Recovery model for processor subsystem

Transient Recovery Procedure. Assume that the fault is transient, and begin a multi-step recovery procedure that continues as long as an error is detected. If an error persists after all three steps have been performed, then a permanent recovery procedure must be invoked.

Step 1. Wait for 0.1 second and do nothing. If the fault is transient it may disappear during this time, allowing rollback to succeed.

Step 2. Retry the current instruction several times, for as long as a half-second. The probability that the retry will be successful (i.e., no error is detected) is 0.5.

Step 3. If an error persists, perform a rollback to a previous checkpoint, followed by recomputation, taking 2 sec. total. The rollback succeeds in removing the error 80% of the time.

If the fault is transient, a transient recovery can be successful only if the fault has disappeared before the step begins. For the analysis of this example, assume that the lifetime of a transient fault is exponentially distributed with a mean lifetime of 0.25 seconds. Transients comprise 90% of all faults, while the other 10% are permanent.

Permanent recovery. If an error still persists after the rollback, it is assumed to be caused by a permanent fault, and a system level permanent fault recovery process is begun, to remove the offending processor from the set of active units and to reconfigure the system to continue without it. The permanent fault recovery process succeeds with probability 0.875.

The analysis of this coverage model consists of calculating the probability of system recovery (PSR_i) for each of the three steps of transient recovery, and for permanent recovery. This calculation entails determining two intermediate sets of quantities: the probability that the transient has gone before step i is reached, and the probability that step i is taken.

Transient Recovery Exit. The transient recovery exit is reached if the fault is transient, and any of the three steps is successful in achieving system recovery.

Step 1. Step 1 is taken with probability one immediately upon the occurrence of the fault. Step 1 performs no actual recovery, so the probability of successful system recovery at the end of step 1 is zero ($PSR_1 = 0$).

Step 2. System recovery from a transient error will occur in step 2 if

- the transient has disappeared during step 1 (with probability $1 - e^{-0.1/0.25} = 0.329$), and
- the retry is successful (with probability 0.5).

Thus, $PSR_2 = (0.329) \times (0.5) = 0.165$.

Step 3. System recovery from a transient will occur in step 3 if

- steps 1 and 2 are unsuccessful (with probability 1 - 0.165 = 0.835, and
- the transient has disappeared during step 1 or 2 (with probability $1 - e^{-(0.1+0.5)/0.25} = 0.909$), and
- the rollback is successful (with probability 0.8).

Thus, $PSR_3 = (0.835) \times (0.909) \times (0.8) = 0.607$.

The probability of transient recovery is then the sum of the probabilities associated with the three steps (0+0.165+0.607=0.772). The transient restoration exit is reached if the fault is transient (with probability 0.9) and if transient recovery is successful; thus, $r = 0.9 \times 0.772 = 0.695$.

Permanent Coverage Exit. There are two cases to consider for the analysis of the permanent recovery exit. The first case deals with the invocation of the permanent recovery procedure to handle a persistent transient fault; the second deals with recovery from a permanent fault.

Case 1. The permanent recovery process is initiated against a transient fault if the three steps of transient recovery have been unsuccessful. The probability associated with this case is then the product of

- the probability that the fault is transient (0.90);
- the probability that the three steps of transient recovery were not successful ($1 - 0.772 = 0.228$); and
- the probability that the permanent recovery process is successful (0.875)

Case 2. The permanent recovery process is successful against a permanent fault if

- the fault is permanent (with probability 0.10); and
- the permanent recovery process succeeds (with probability 0.875).

The probability of reaching the permanent coverage exit is then the sum of the probabilities associated with the two cases, thus $c = 0.179 + 0.0875 = 0.267$.

Single-Point Failure Exit. There are two cases to consider for the single-point failure exit.

Case 1. For a transient fault, the single-point failure is reached if the permanent recovery procedure is invoked, and fails to achieve system recovery. The probability associated with this scenario is $(0.228 \times (1 - 0.875) = 0.028)$. Multiplying by the probability that the fault is transient yields the probability for Case 1: $(0.9 \times 0.028 = 0.0252.)$

Case 2. For a permanent fault, the single-point failure exit is reached if the permanent recovery procedure is unsuccessful. Multiplying by the probability that the fault is permanent yields the probability for Case 2: $((1-0.875) \times (0.10) = 0.0125)$.

The probability of reaching the single-point failure exit is the sum of the probabilities associated with Cases 1 and 2, thus $s = (0.0252 + 0.0125) = 0.0377$.

2.3 Near-coincident faults

In highly reliable systems, such as those used for flight control, the probability of a second fault occurring while attempting recovery from a given fault cannot be ignored. The occurrence of a second, near-coincident fault (while attempting to handle a single fault) causes immediate system failure, if the second and first faults are critically coupled. The modeler must designate which sets of faults are critically coupled, or can assume either extreme: all faults are critically coupled or no faults are critically coupled. Once the set of critically coupled faults has been determined, the calculation of the probability of near-coincident faults is straightforward, given some measure of the time spent in a recovery model. In the 3P2M example, if a processor fails while a memory failure is being handled, or during the recovery from a fault in another processor, the system fails. If, however, a memory fails during a processor recovery, no immediate failure occurs. A bus failure would interfere with processor or memory recovery.

A fourth exit is then added to the FEHM model, representing the occurrence of a near-coincident fault before another exit is reached. Consequently, the probability of reaching one of the original three exits is reduced by a factor equaling the probability that an interfering near-coincident fault does not occur. This single-entry 4-exit model is then automatically inserted into the FORM model, as described in the following section.

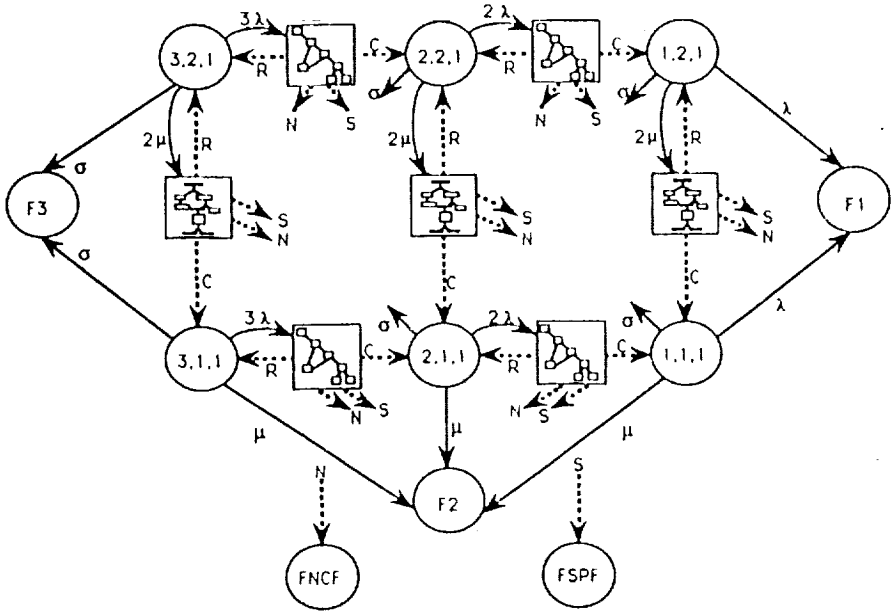

Figure 6: Combination of FEHM and FORM models

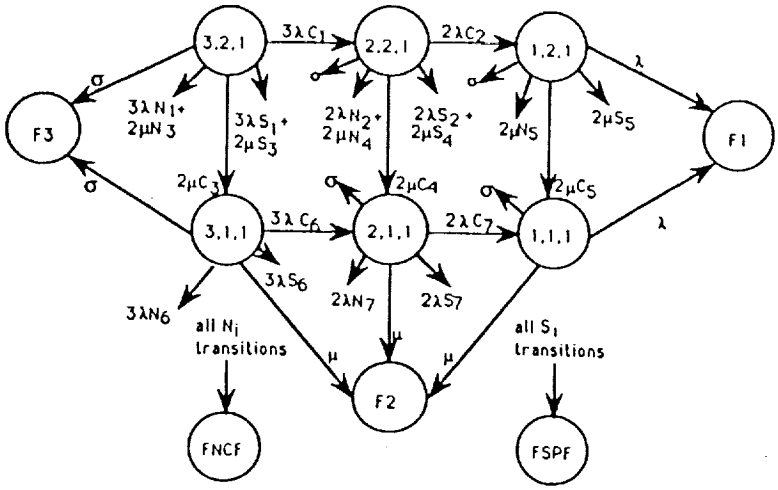

Figure 7: Reduction of combined FEHM and FORM models

Cause of Failure	Probability
Exhaustion of Processors	2.20×10^{-10}
Exhaustion of Memories	1.61×10^{-10}
Exhaustion of Buses	9.99×10^{-6}
Single Point Failure	3.53×10^{-5}
Near-Coincident Faults	4.49×10^{-10}
Total Unreliability	4.53×10^{-5}

Table 1: Solution of 3P2M example system

2.4 Combining FORM and FEHM models

Once the FORM and FEHM models are described, they are then combined. We demonstrate this process for the Markov chain in figure 2 which results from the fault tree in figure 1. For each failure of a redundant component, the appropriate FEHM model is invoked. That is, a FEHM model is inserted on each failure arc between operational states in the Markov chain, as shown in figure 6. In the 3P2M example, the FEHMs on the horizontal failure arcs are copies of the processor recovery model (figure 5), while the FEHMs on the vertical failure arcs are copies of the memory coverage model (figure 4). Two failure states are inserted:

- *FSPF* denoting the occurrence of a single-point failure, and
- *FNCF* denoting the occurrence of critically coupled near-coincident faults.

Each FEHM model is then solved for the probability of reaching each of its three exits, and the FEHM model is replaced by a branch point. The resulting Markov chain (see figure 7) is then solved for the reliability of the system, which is given by the probability that the system is not in any failure state.

Table 1 shows the results of the reliability analysis for a 10 hour mission of the 3P2M example. For this model, we assume that the failure rate of the processor $\lambda = 10^{-4}$, for the memory $\mu = 10^{-5}$ and for the bus $\sigma = 10^{-6}$. The largest contributor to the unreliability is single-point failure, that is, faults from which recovery is not successful.

3 Dynamic fault-tree gates

A major disadvantage of traditional fault tree analysis is the inability of standard 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 [19]. 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. Thus, the failure criteria depend not only on the combinations of events, but also on the sequence in which events occur.

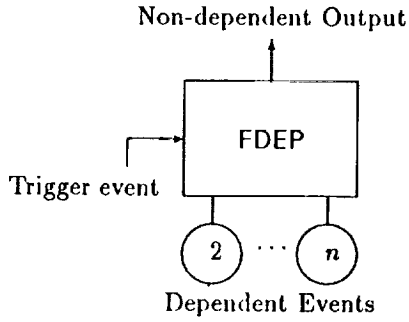

Figure 8: Functional dependency gate

Systems with various sequence dependencies are usually modeled with Markov models. If, instead of using standard 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 certain kinds of sequence dependencies to be modeled by defining special purpose gates to capture specific types of sequence dependent behaviors. There are several different kinds of sequence dependencies in fault tolerant systems. This section identifies several such dependencies, and defines specific gates to express these behaviors in fault tree models. Part II demonstrate the use of these gate types in several examples.

3.1 Functional dependency gate

Suppose that a system is configured such that the occurrence of some event (call it a *trigger* event) causes other dependent components to become inaccessible or unusable. In this case, later failures of the dependent components will not further affect the system and should not be considered. A *functional dependency* gate (see figure 8) has a single 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) and 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

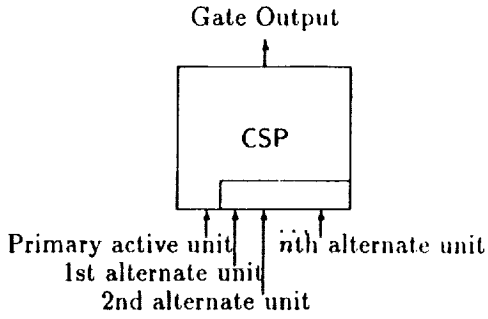

Figure 9: Cold spare gate

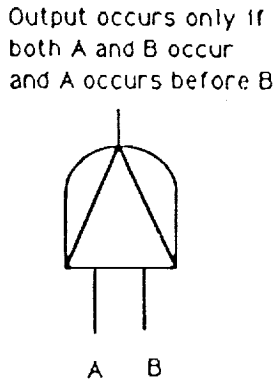

Figure 10: Priority-AND gate

occurred. The occurrence of any of the dependent basic events has no effect on the trigger event.

The functional dependency gate is useful where communication is achieved through some network interface elements, where the failure of the network element isolates the connected components. In this case, the failure of the network element is the trigger event and the connected components are the dependent events. Part II describes several applications of the functional dependency gate.

3.2 Cold spare gate

Consider a system that utilizes cold spares, that is, spare components that are unpowered, and thus do not fail before being used. Such systems cannot be modeled exactly using standard 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 *cold spare* gate (see figure 9), with one primary input and one or more alternate inputs. All inputs are basic events. The primary input is the one that is originally powered on, and the alternate input(s) specify the (initially unpowered) components that are used as replacements for the primary unit. The cold spare gate has one output which becomes true after all the input events occur.

The conversion of the fault tree to a Markov chain makes the consideration of cold spares possible. In a state where the primary unit is operational, the cold spares are not permitted to 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 cold spare gate can interact in an interesting way. Suppose that the spare units are functionally dependent 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 yet. Then, if the primary unit fails, the spares are unavailable to replace it. This is the one case where a spare can "fail" even while it is unpowered. Part II gives examples of the use of the cold spare gate.

3.3 Priority-AND gate

The priority-AND gate is logically equivalent to an AND gate, with the added condition that the events must occur in a specific order. The priority-AND gate (as shown in figure 10) has two inputs, A and B. The output of the gate is true if both A and B have occurred, and if 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 11.

3.4 Sequence enforcing gate

The *sequence enforcing gate* 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 (i.e., 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.). There may be any number of inputs (see figure 12), the first of which may be a (possibly replicated) basic event or the output of some other gate. All inputs other than the first are limited to being (possibly replicated) basic events. 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) where the sequence enforcing gate will only *allow* the events to occur in a specified

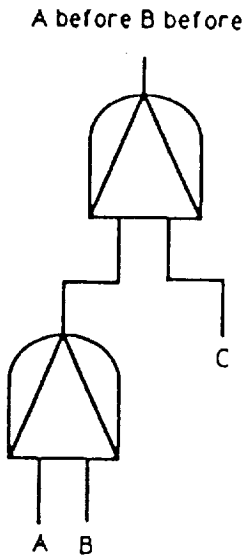

Figure 11: Cascading priority-AND gates

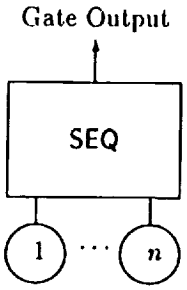

Figure 12: Sequence enforcing gate

order.

In the generation of a Markov chain from a fault tree containing a sequence enforcing gate, states that represent any other ordering than that specified by the sequence enforcing gate are never generated. In part 2 of this tutorial we will show an interesting application of the sequence enforcing gate to model pooled spares.

Part II
Examples

We study several examples of advanced fault tolerant systems, and develop fault tree models to analyze the reliability of these systems. The models are all solved with HARP, the Hybrid Automated Reliability Predictor, developed at NASA Langley Research Center and Duke University. The parameters used for these model and the details of the recovery mechanism are pure conjecture, and should not be interpreted as a factual representation of the parameters associated with the systems.

4 Cm*: a loosely-coupled distributed system

4.1 System description

An instance of the *Cm** system (shown in figure 13) consists of 2 clusters of processors and memories connected by links [22]. Each cluster consists of 4 local switch interface controllers (*S.locals*), each attached to one processor and one 12K memory module. Each processor has 4K of memory on board. The *K.map* is a cluster controller connecting the *S.locals*; the clusters are connected by intercluster communications (*L.inc*). A fault in the *K.map* renders the associated *S.locals* (and their connected processors and memories) inaccessible, while a fault in an *S.local* makes the processor and memory modules connected to it inaccessible.

The *Cm** system exhibits three characteristics that are typical of reliable distributed systems.

1. There are functional interdependencies which can make the development of the fault tree model difficult, for example, the dependence of the accessibility of the processors and memories on the state of the *S.locals*.

2. There are many potential system states: since there are 27 components, the system can be in any one of $2^{27} > 134$ million states, if any component can be in one of two states, functional and failed.

3. There are many failure modes: there are 5405 minimal cut sets for this system (a *cut set* is a set of components whose failure causes the system to fail).

4.2 Failure criteria

The system is considered operational as long as there are 3 processors that can communicate with 3 memories. As long as the *L.inc* is operational, these requirements can be satisfied by the components of both clusters. But, if the *L.inc* fails, the requirements must be met within one cluster.

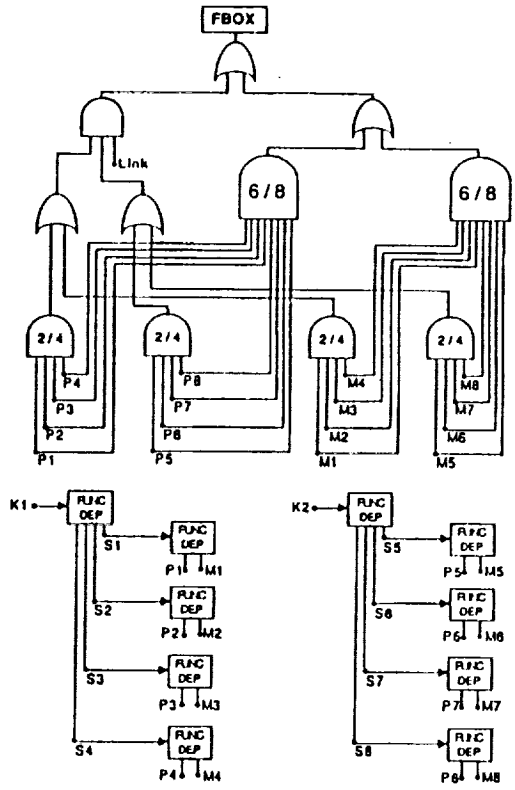

Figure 14: Fault tree model of *Cm** system

4.3 Fault tree model

The development of the fault tree model of the *Cm** system is simplified by the use of a functional dependency gate, to capture the interconnection dependencies. A fault tree model of the Cm* system is shown in figure 14. System failure (the top event) can be attributed to one of two causes which are shown as inputs into the uppermost OR gate. Failure occurs when either the *L.inc* fails and the requirements cannot be satisfied by a single cluster (the left input to the uppermost OR gate), or (independent of the state of the *L.inc*) there are an insufficient total number of processors or memories in both clusters. The output of an m/n gates is true when m of the n input events have occurred.

The functional dependencies of the *S.locals* on the *K.maps* and of the processors and memories on the associated *S.local* are captured in the functional dependency gates (FDEP) shown in figure 14. In this case, there were no explicit reliability requirements concerning the *K.maps* or *S.locals*, so the functional dependency gate is not explicitly connected to the top event in the fault

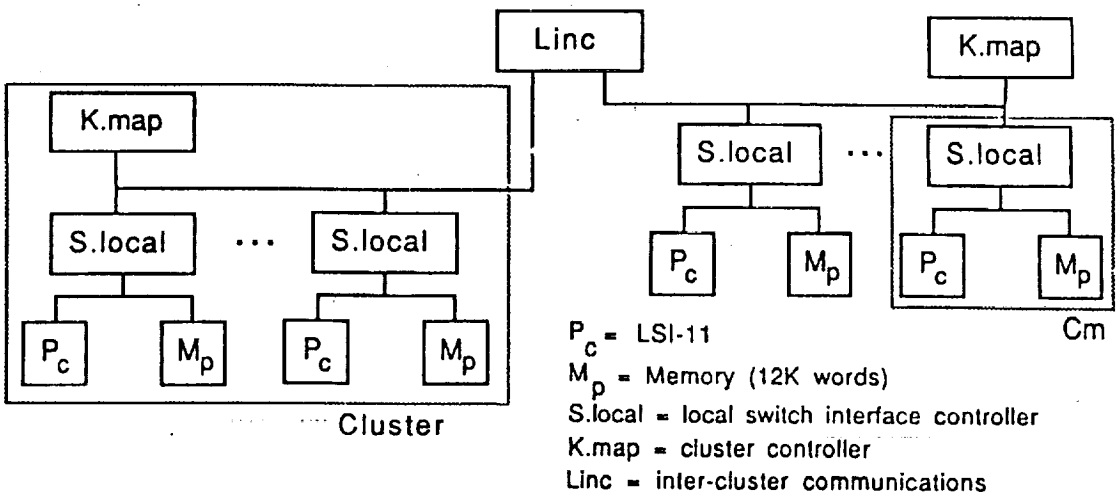

Figure 13: A diagram of the Cm* system.

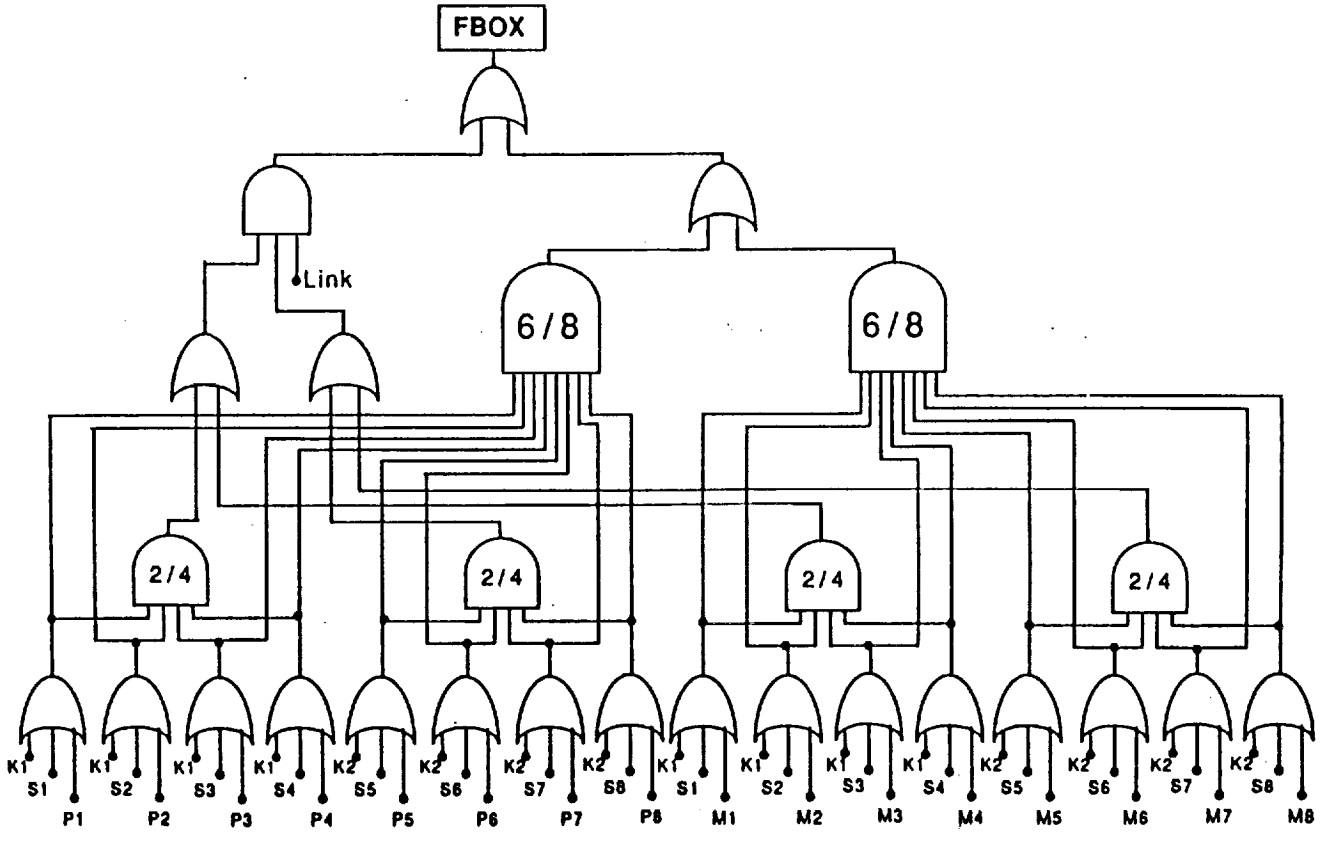

Figure 15: Fault tree model of *Cm** system without functional dependency gates

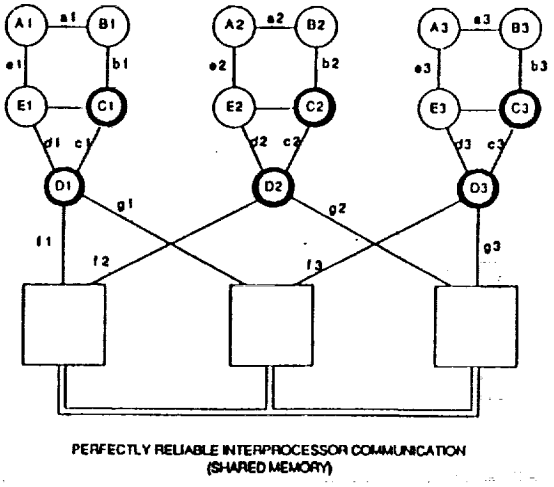

Figure 16: An AIPS I/O network used for example calculations

tree. In order to solve a fault tree model containing functional dependency gates via standard combinatorial solution methods, we need to convert the model to a strictly combinatorial one. To accomplish this conversion, the dependency gates can be replaced with OR gates in the following manner. For each occurrence of a dependent basic event, replace that basic event with a logical OR of the basic event and its trigger event. Thus in the *Cm** system, each basic event representing a processor failure is replaced by a logical OR of that processor event, its *S.local* and its *K.map*. Memory events are altered in a similar manner. The fault tree that results from replacing the functional dependency gates is shown in figure 15. The replacement of the functional dependency gates only produces a correct result if no FEIIM models are used, that is, if all faults are permanent and are instantaneously and perfectly covered.

5 AIPS: a system of fault-tolerant building blocks

5.1 System description

An example of the AIPS (Advanced Information Processing System) I/O network is shown in figure 16. The AIPS system, designed at the Charles Stark Draper Laboratory, is intended to provide fault-tolerant building blocks that can be used for a variety of real-time control applications [16]. The AIPS I/O network might be used in a flight control system, and consists of 3 rings, each of which contains 5 nodes. Three of the nodes on each ring (those labeled *A*, *B*, *E*) are connected to sensors and/or actuators. Each such device is triplicated, with one copy of each device connected to each ring, via a node in the same location (with the same letter label). The remaining two nodes, *C* and *D*, are termed *root nodes* because they provide the connections to the triplicated computers.

5.2 Failure criteria and parameters

The I/O network fails when

1. Nodes in the same location on two different rings either fail or become isolated from both root connections, OR
2. if 2 of the 3 computers fail or become disconnected from both rings, OR
3. when 2 of the three rings become disconnected from both computers.

As long as a node can communicate with one computer, it can communicate with all computers that are up because the computers are assumed to be connected by a perfectly reliable interconnection mechanism (such as shared memory). For the purpose of this analysis we consider only the I/O network and the computer connections, and not the possible failures of the devices (such as sensors and actuators) connected to the nodes. The failure parameters used for this analysis are

- Node failure rate: 6×10^{-6} per hour
- Link failure rate: 12×10^{-6} per hour
- Computer failure rate: 10^{-4} per hour

5.3 Fault recovery

Recovery from faults in nodes and links is assumed to be perfect and instantaneous. For the computers, however, more detailed coverage modeling is necessary. It is assumed that 85 percent of the faults that occur in the computer system are transient, with the remaining 15 percent being permanent or intermittent in nature. Recovery from computer faults is assumed to be perfect, but not instantaneous: the time to recover from a transient is 1 second, while the time to recover from a permanent or intermittent is uniformly distributed between 1 and 5 seconds. During the recovery interval, if a second, near-coincident fault occurs in either of the other computers, the recovery is interrupted, and system loss is conservatively assumed to occur.

5.4 Fault tree model

The fault tree model of the AIPS I/O network has 102 nodes, including 39 basic events, and is too large to be presented here as a whole. However, figure 17 is a sketch

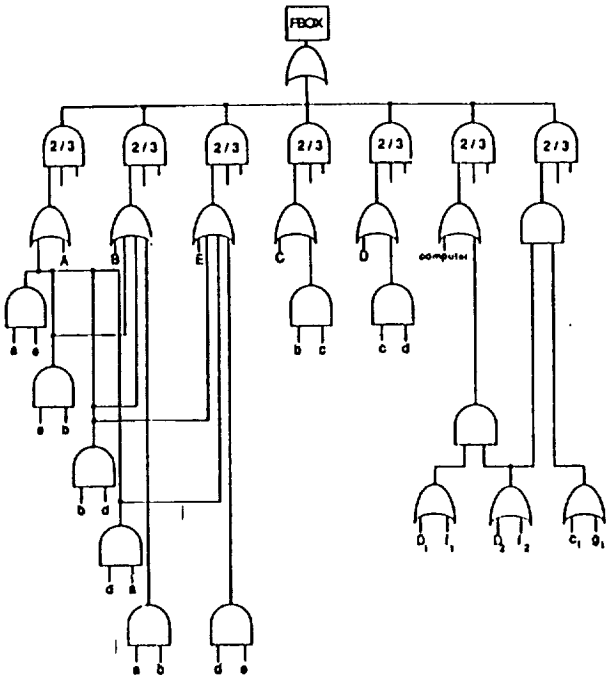

Figure 17: Fault tree model of AIPS I/O network

of the fault tree with some of the paths complete. The system fails when one of the seven triplicated subsystems fail (hence seven 2/3 gates are connected to the top OR gate), these being node groups A through E, the computers, and the root connections between the rings and the computers. A representative of each of the 7 subsystems is shown in detail; the other members of each triplicated subsystem are analogous. The results of the solution of this model appear in table 2.

5.5 Truncated fault tree

An interesting alternative to the development of the full fault tree model is the concept of a *truncated fault tree*. For the AIPS network (figure 16), the expansion of only 2 failure levels produced a reasonably accurate estimate of the system unreliability. For this case, we could have produced a similar result with a much simpler fault tree, one which explicitly defined only the 2 component failure combinations. Consider the fault tree representation of the AIPS network in figure 18. The top event of this tree is 2-component failure system loss, where the system loss is caused by losing 2 members of any triplicated subsystem. No combination of 2 link failures, or one link failure and one other component failure can lead to system failure, and so the link basic events do not input to any gates in the truncated fault tree. The presence of these *dangling basic events* (basic events that do not input to any gate

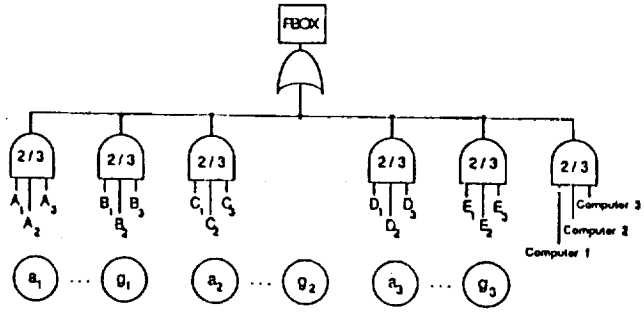

Figure 18: Truncated fault tree model of AIPS network

in the fault tree) can be used to bound the failure probability. If the dangling basic events are ignored then the solution of the fault tree gives an optimistic estimate of the unreliability of the system.

If we are using a strictly combinatorial solution method, we can use the dangling basic events to determine the upper bound on the unreliability by using a k-out-of-n gate. Connect all n basic events (those that are dangling as well as those that are not) to an 3-out-of-n gate (a gate that is activated on the third component failure), and OR its output with the top event of the tree. This is equivalent to assuming that the third component failure causes system failure.

If we need to include the effects of imperfect coverage in the model, we can use the dangling basic events in conjunction with the conversion of the fault tree to a Markov chain. As the Markov chain state space is expanded, all the basic events become part of the state definition. The resulting Markov chain can be used to produce bounds on the unreliability of the system from the solution of the truncated fault tree. It is not necessary in this case to add the m-out-of-n gate as was done with the strictly combinatorial solution. The basic events are simply left dangling. The presence of dangling basic events is crucial to the determination of correct bounds on the system unreliability.

The solution of the truncated Markov chain corresponding to the truncated fault tree of the AIPS system is shown in table 3. A comparison of the numbers in this table with those in table 2, shows that the truncated fault tree can give reasonable results. The time needed by a reliability analyst to determine a truncated fault tree is substantially less than the time required to derive a complete fault tree model of a system. Further, the combination of a truncated solution technique and a truncated fault tree can allow more faith to be placed in the model, since if there are missing failure combinations they may

Full Fault Tree Model Solution AIPS I/O Network Example System		
Truncation Level Size Of Truncated Model	1 Component Failure 42 states, 190 transitions	2 Component Failures 770 states, 5155 transitions
Lower Bound on Unreliability Upper Bound on Unreliability	0.125e-6 2.94e-6	0.126e-6 0.128e-6
Total Run Time	65 CPU seconds	1295 CPU seconds

Table 2: Solution of example AIPS system

Truncated Fault Tree Model Solution AIPS I/O Network Example System		
Truncation Level Size Of Truncated Model	1 Component Failure 42 states, 190 transitions	2 Component Failures 770 states, 4879 transitions
Lower Bound on Unreliability Upper Bound on Unreliability	0.126e-6 0.640e-6	0.1261e-6 0.1263e-6
Total Run Time	58 CPU seconds	1144 CPU seconds

Table 3: Solution of truncated fault tree model of AIPS system

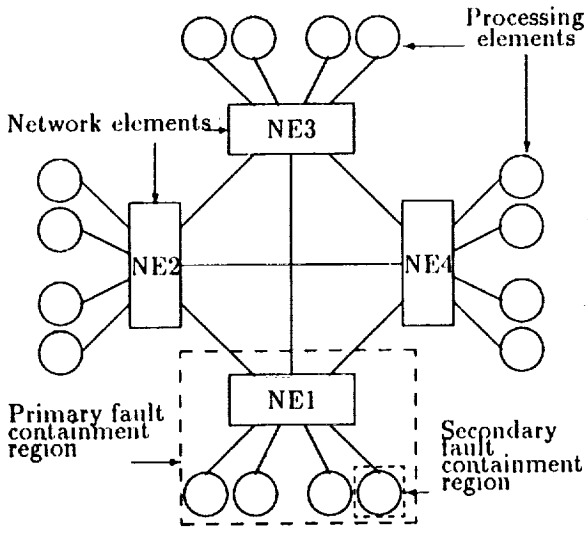

Figure 19: An instance of the fault tolerant parallel processor

be accounted for by the bounding technique.

6 FTPP: Fault tolerant parallel processor

6.1 System description

Next we consider several models of the FTPP (Fault Tolerant Parallel Processor) [18, 17] cluster, to compare various configurations of triads with spares. An instance of an FTPP cluster is shown in figure 19, and consists of 16 processing elements (PE), with 4 connected to each of 4 network elements (NE). The network elements are fully connected. In the clusters modeled here, the 16 processors are logically connected to form 4 triads, each with one spare. We investigate three triad/spare configurations, the first two with hot spares and the third with cold spares:

#1 utilizes hot spares; there is one spare for each triad and all spares are attached to the same network element.

 # 2 also uses hot spares; there is one spare on each network element and the spare PE can substitute for any failed PE attached to the same network element.

 # 3 is the same as #1, with all spares on the same NE, but in configuration #3 the spares are cold.

The processing elements in all three configurations functionally depend on the network element to which they are connected. If a network element experiences a permanent failure, the processing elements connected to it are then considered failed.

6.2 Failure criteria and parameters

For all models, a triad fails when it has fewer than 2 active components; the system fails if any triad fails. Failures occur at a constant rate of 1.1×10^{-4} per hour for processing elements, and 1.7×10^{-5} per hour for network elements.

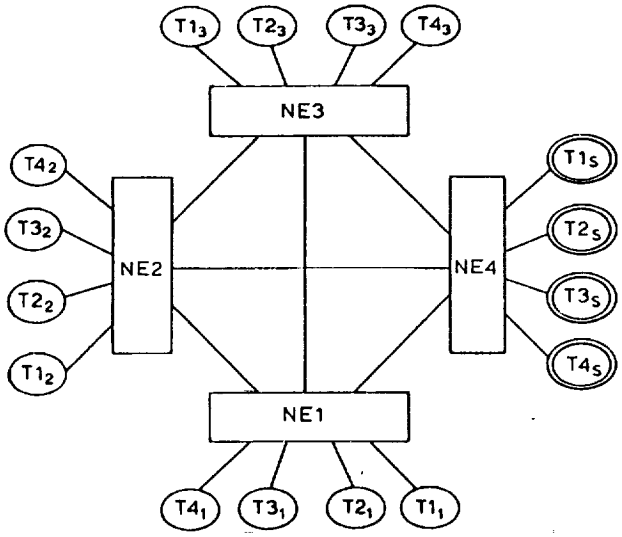

Figure 20: Configuration #1 with one spare per triad

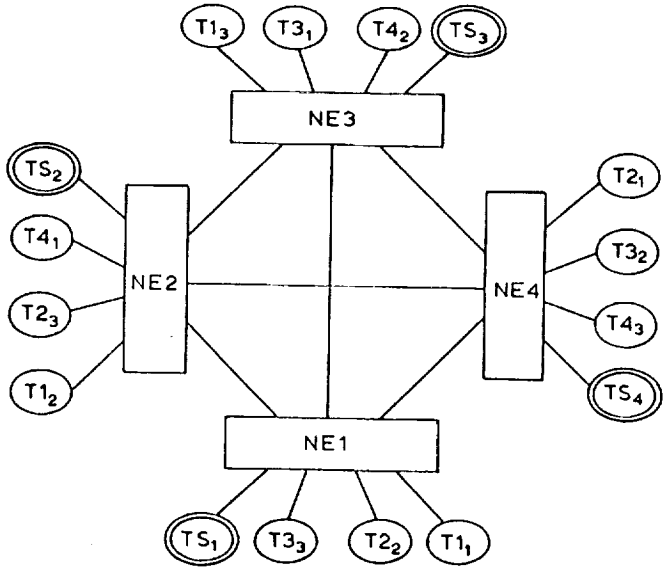

Figure 22: Configuration #2 with one spare per NE

6.3 Fault recovery

Recovery and reconfiguration from faults in processing elements are both perfect, but take a non-zero amount of time. If a second fault occurs in any other component during attempted recovery from a first fault, the system fails. Half of the faults that occur in the processing elements are transient, and can be recovered from without discarding the affected component. The remainder of faults are permanent. The time to recover is exponentially distributed with a mean of 3.6 seconds. Coverage of NE failures is both instantaneous and perfect.

6.4 Fault tree models

6.4.1 Configuration #1

Configuration #1 (shown in figure 20) divides the active elements of a triad among *NE1*, *NE2* and *NE3*, and uses the PE's on *NE4* as spares. The PE's that are in the same relative position on the first three network elements form a triad, and the PE in the same relative position on *NE4* serves as a hot (active) spare for the triad.

The fault tree model for configuration #1, shown in figure 21, uses four functional dependency gates (FDEP) to reflect the dependence of the processing elements on the network elements. 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 network elements. Figure 21 shows four 3/4 gates connected to the top OR gate, one 3/4 gate for each triad. A triad fails when only one element remains (3 of the 4 elements have failed).

6.4.2 Configuration #2

Configuration #2 is an FTPP cluster with hot spares distributed across the network elements instead of grouped on the same network element (see figure 22). The spare element on each network element can substitute for any failed PE connected to the same NE. That is, processing element TS_1 can substitute for a failed PE connected to *NE1*.

The fault tree model of this system is a bit more complex than the one presented in section 6.4.1, and is shown in figure 23. The functional dependency gates FDEP again reflect the dependence of the processing elements on the network elements. A triad failure is again attributed to losing the majority of operational elements, but it is more difficult to describe the failure of a member of the triad. A member of the triad is failed if it and its spare fail or if 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 23, the leftmost OR gate that inputs into the leftmost 2/3 gate represents the failure of the first member of the first triad. This member fails if both $T1_1$ (the first member of the first triad) and its spare (TS_1) fail, or if the spare is being used because another failure has already occurred when $T1_1$ fails. The spare will already be in use when $T1_1$ fails if either $T2_2$ or $T3_3$ (the other two active components 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.

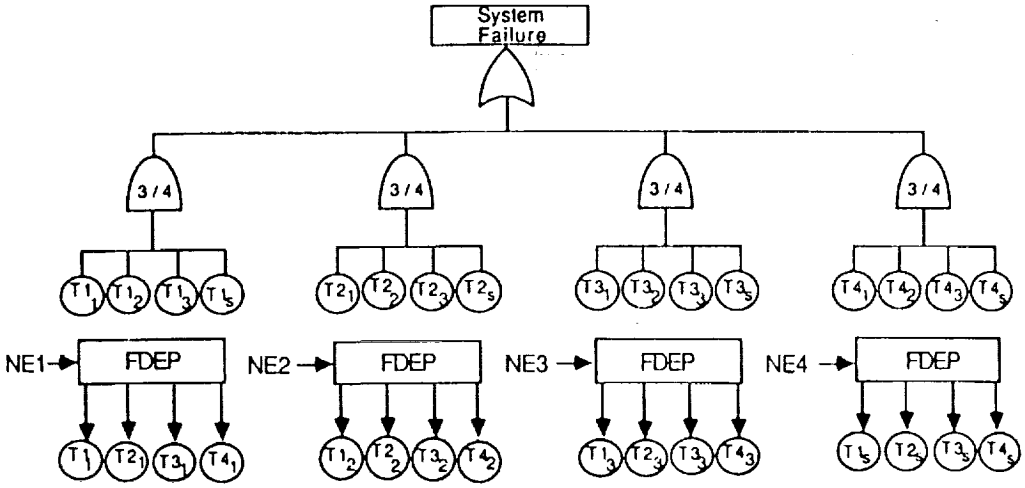

Figure 21: Fault tree model for configuration #1

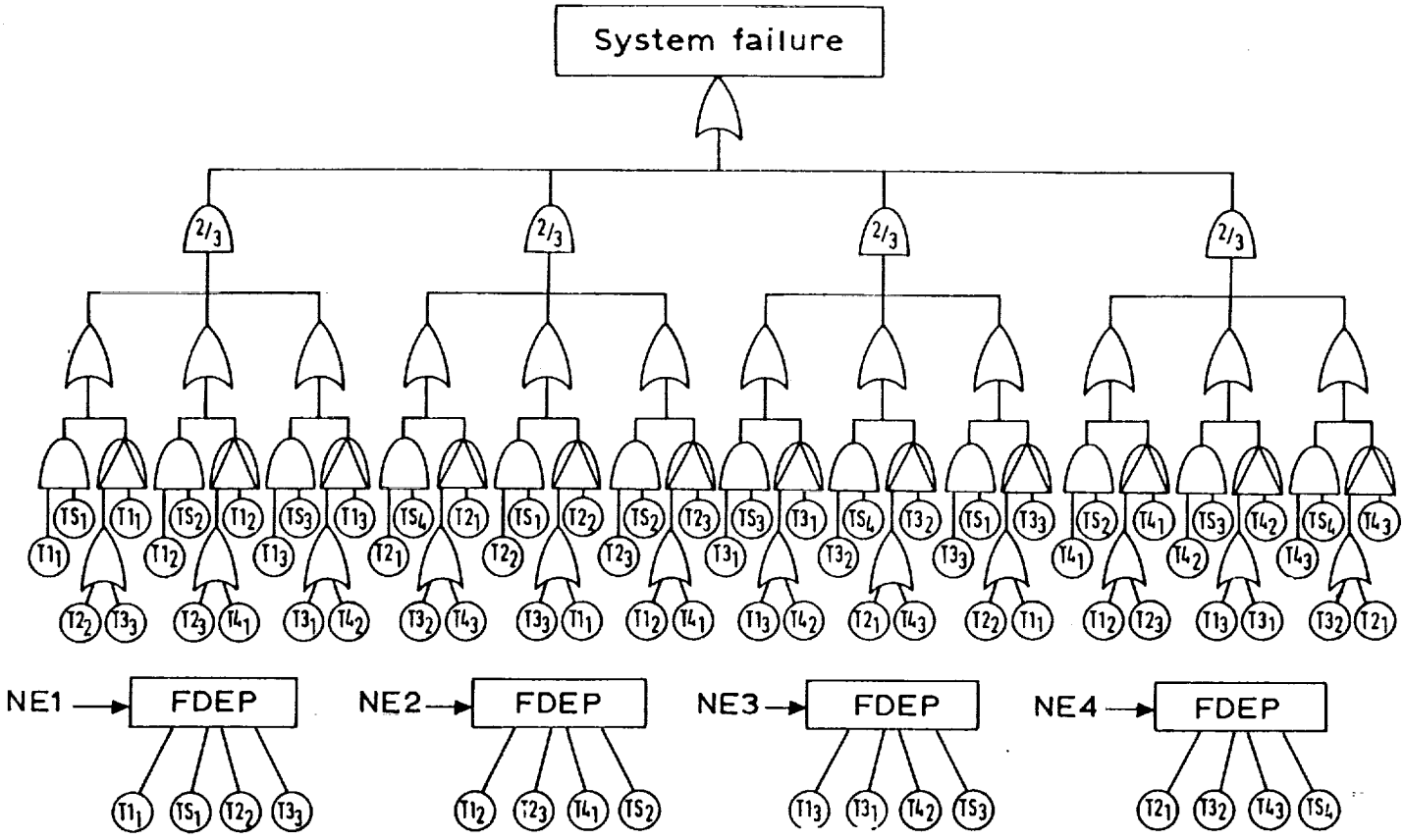

Figure 23: Fault tree model for configuration #2

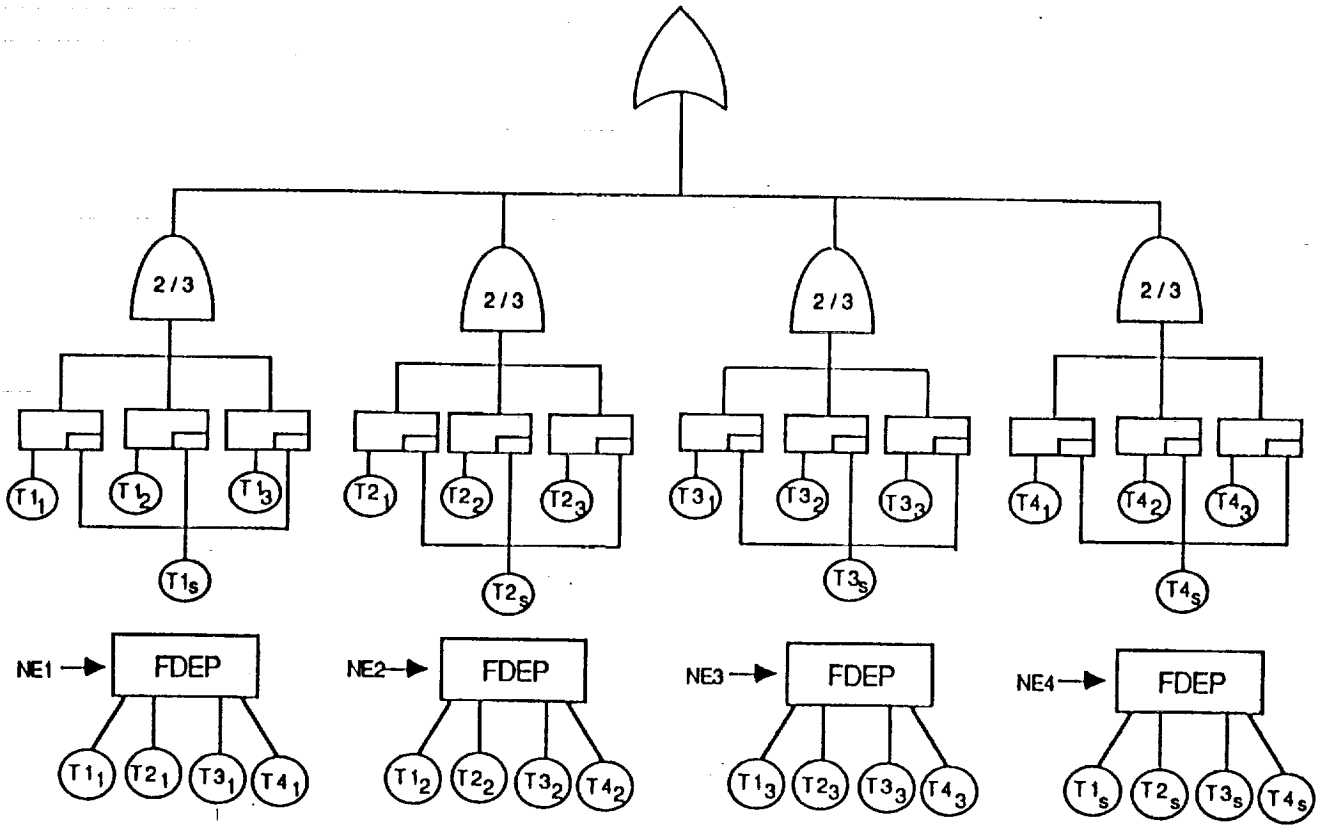

Figure 24: Fault tree model for configuration #3 with one COLD spare per triad

6.4.3 Configuration #3

The third configuration is used to investigate the effect on reliability of keeping the spares unpowered until needed. The FTPP configuration modeled in this section is the same as configuration #1 (figure 20) except that the spares are cold rather than hot. There is one spare for each triad, and all spares are connected to the same network element. The fault tree model for this system, shown in figure 24, uses the cold spare gate. There is one cold spare 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 cold spare PE is connected to all three cold spare gates since it can substitute for any of the elements.

6.5 Results

This section presents the results obtained from solution of the models of the three FTPP configurations for a mission length of 10 hours. Table 4 compares the reliability of the three 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 4, shows the bounds on the unreliability, and the best case (optimistic) estimate of the probabilities of exhaustion of network elements (exh NE), exhaustion of processing elements (exh PE) and near coincident failures (NCF).

Configuration #2 (that distributed the hot spares across the network elements) not only required a more complicated fault tree for analysis, but also was appreciably less reliable than configuration #1. In configuration #2, the failure of 2 network elements (alone) can kill the system, since the failure of 2 network elements removes 2 members from at least one triad. For example, if *NE1* and *NE2* both fail, then $T1_1$ and $T1_2$ 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 network elements. In configuration #1, the loss of 2 network elements (alone) does not cause any triad to fail, even though it can render all the spare elements unusable.

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

For all three models, the Markov chain was truncated after the consideration of 2 or 3 faults, and so a pair of bounds on the actual reliability were generated. The bounds were tight enough after only considering 2 faults for configuration #2, but we needed to consider a larger model for the other two cases. The reason that the bounds were tighter for configuration #2 is that there were a significant number of failure states encountered when only considering 2 component failures. In the #1 and #3 configurations, 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 5 compares the results obtained from the smaller model (truncated after 2 failures) and the larger model (truncated after 3 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.

7 ASID MAS: a mission avionics system

7.1 System Description

The ASID (Advanced System Integration Demonstration) project was the first large scale 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 [5] is the use of dual processor pairs wherever a single processor is required. This processor-pair uses comparison monitoring so as 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 part of the ASID system.

There are several critical functions within the mission avionics system (MAS). The loss of any of these functions causes the MAS to fail. These critical functions include the vehicle management subsystem (VMS), the crew station control and display functions, mission and systems management, local path generation, and scene and obstacle following functions. The vehicle management subsystem provides airframe control, including flight and propulsion control, as well as providing utility systems management and control. The crew station subsystem displays information to the pilot, contains mechanisms for pilot control actions, and manages crew station activity. The mission and systems management subsystem allocates resources for real time control functions.

Figure 25 is a block diagram of the architecture of the critical mission avionics system. One processing unit is required for the crew station functions, local path generation, and mission and system management. Each of these

Configuration	#2: Hot spare per NE	#1: Hot spare per triad	#3: Cold spare per triad
(Best Case) Unreliability	0.207×10^{-6}	0.406×10^{-7}	0.264×10^{-7}
(Worst Case) Unreliability	0.417×10^{-6}	0.407×10^{-7}	0.266×10^{-7}
(Best case) exh. NE	0.174×10^{-6}	0.135×10^{-8}	0.104×10^{-8}
(Best case) exh. PE	0.327×10^{-8}	0.910×10^{-8}	0.705×10^{-8}
(Best case) NCF	0.302×10^{-7}	0.302×10^{-7}	0.183×10^{-7}

Table 4: Results of the solution of all three FTPP models

Configuration	#2: Hot spare/NE	#1: Hot spare/triad	#3: Cold spare/triad
Truncated at 2 component failures			
(Best Case) Unreliability	0.207×10^{-6}	0.406×10^{-7}	0.263×10^{-7}
(Worst Case) Unreliability	0.417×10^{-6}	0.242×10^{-6}	0.132×10^{-6}
Number of states	201	123	225
Number of transitions	877	581	817
Runtime (CPU seconds)	138	99	99
Truncated at 3 component failures			
(Best Case) Unreliability		0.406×10^{-7}	0.264×10^{-7}
(Worst Case) Unreliability	*analysis not*	0.407×10^{-7}	0.266×10^{-7}
Number of states	*necessary for*	961	2307
Number of transitions	*this example*	5469	9777
Runtime (CPU seconds)		2653	5055

Table 5: Comparison of accuracy and model size

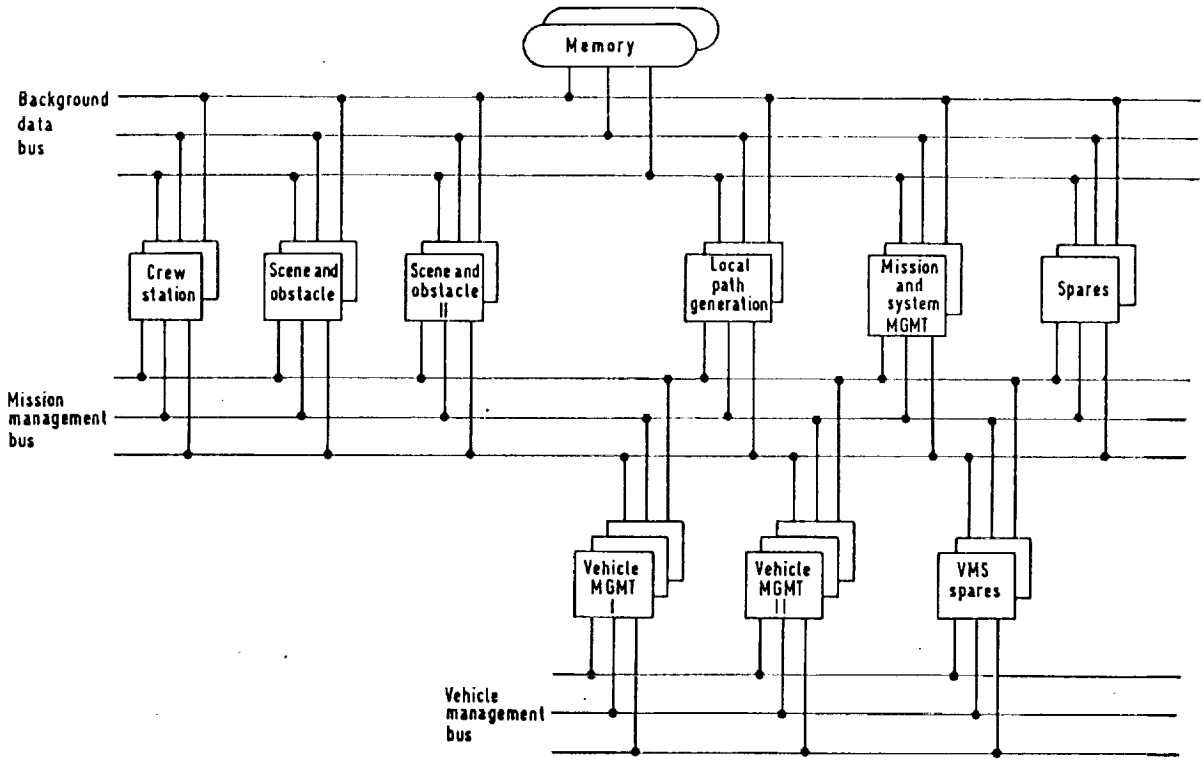

Figure 25: Block diagram of mission avionics system architecture

processing units is supplied with a hot spare backup to take over control if the primary processor should detect an error. Each of the processing units is really a pair of tightly coupled processors so as to maximize the probability of fault detection and minimize latency. Although there are really 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 25 thus shows that there are two processing units for these functions, where one is the primary unit and the other is a hot spare.

The scene and obstacle subsystem and the VMS both require more functionality than one processing unit can provide, and thus each use 2 processing units. The scene and obstacle processing units are also replicated, providing a hot spare backup. The VMS is triplicated, providing 2 hot spare backups.

In addition to the hot spare backups, 2 additional pools of spares are provided, each containing 2 spare processing units. The first pool can be used to cover the first 2 processor failures in the subsystems other than the VMS; the second pool covers the first 2 failures in the VMS.

The processing units are connected via 2 triplicated bus systems, the first being a data bus and the second being the mission management bus. The replicated memory is connected to the data bus. The VMS has an additional triplicated bus, the vehicle management bus.

7.2 Failure criteria and parameters

The MAS fails if any of the functions cannot be performed, or if both of the 2 memories fail, or if all 3 of any one type of bus fail. The following MTBF (mean time between failures) values, giving rise to the following failure rates, were used.

- processor pairs: 40,000 hours; failure rate: 2.5×10^{-5}
- buses: 400,000 hours; failure rate: 2.5×10^{-6}
- memories: 1,000,000 hours; failure rate: 1.0×10^{-6}

7.3 Fault recovery

Fault 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 failure occurs during this interval, we say that the system fails because of *near-coincident failures (NCF)*.

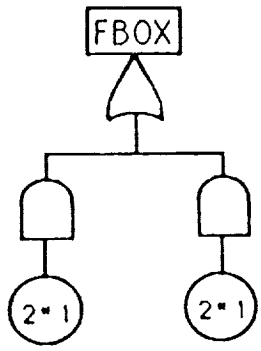

Figure 27: Fault tree model of a 2-duplex system

7.4 Fault tree model

The fault tree model of the mission avionics system 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 mission avionics subsystem including the pooled spares.

7.4.1 Fault tree with no pooled spares

The fault tree model of the MAS with no pooled spares is shown in figure 26. This fault tree shows that the MAS fails if any of the critical functions fail, or if either of the bus systems fail, or if both memories fail. There are 3 types of components in the example fault tree, 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 subsystem uses 2 memories and is thus labeled $2*3$, while the mission management bus subsystem uses three buses and is labeled $3*2$.

7.4.2 Modeling pooled spares

Before we add the pooled spares to the fault tree model of the MAS, consider a simple system with two duplexes and 2 pooled spares. The fault tree model of this 2 duplex system is shown in figure 27, while the equivalent Markov chain is shown in figure 28. This equivalent Markov chain is determined automatically by HARP.

Next, consider the desired Markov chain representation of the same 2-duplex system with the addition of 2 pooled spares (figure 29). 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 will deplete the spares. After the first 2 failures, 2 functioning

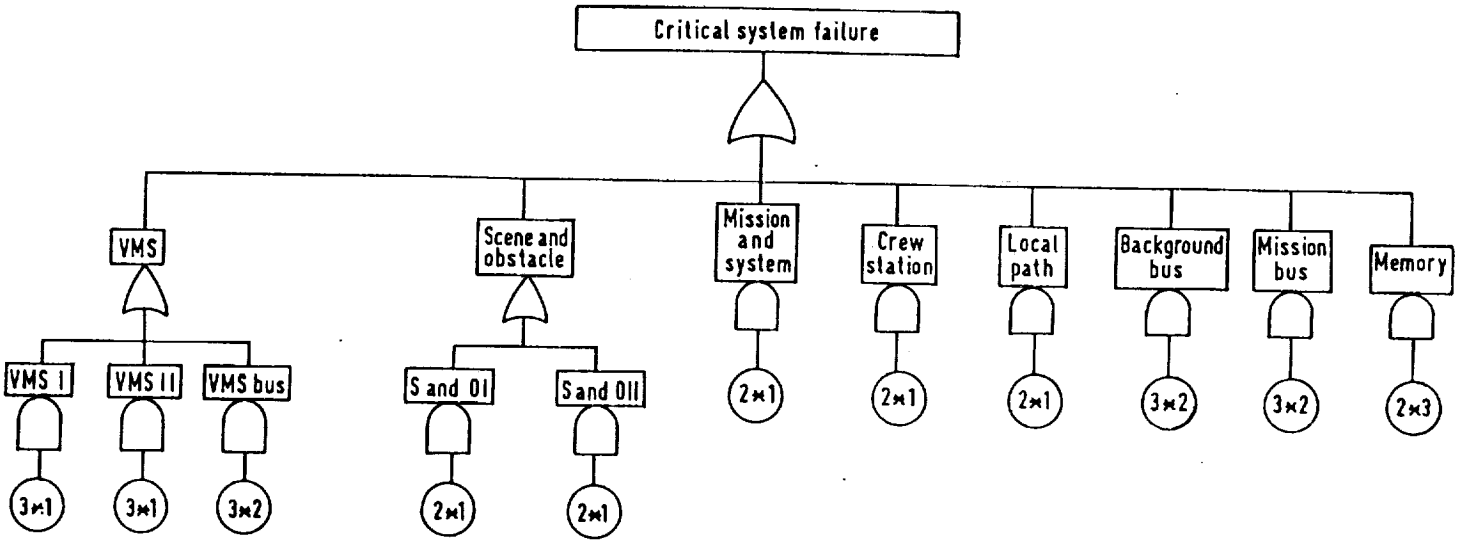

Figure 26: Fault tree model of MAS with no pooled spares

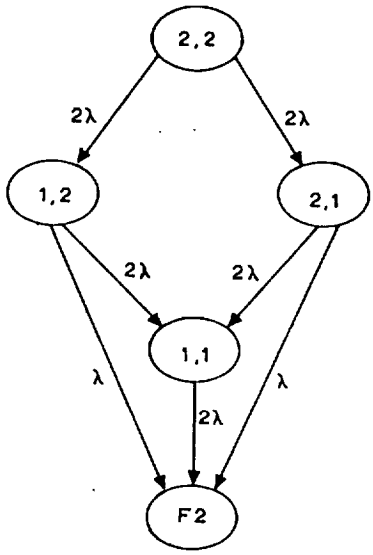

Figure 28: Markov chain model of a 2-duplex system

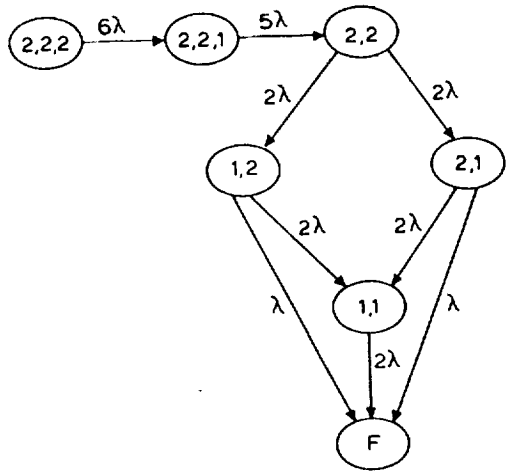

Figure 29: Markov chain model of a 2-duplex system with 2 pooled spares

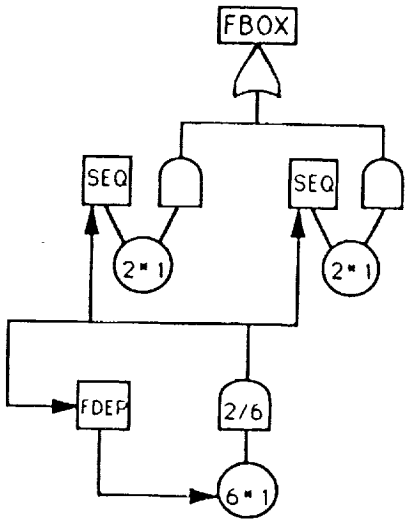

Figure 30: Fault tree model of a 2-duplex system with 2 pooled spares

duplexes remain, and the rest of the Markov chain in figure 29 is identical to that in figure 28.

We can use the fault tree shown in figure 30 to represent the 2-duplex system with 2 pooled spares. In figure 30, the combination of the 2/6 gate (which fires after the first 2 of 6 failures) and 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 30 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 (which is identical to the one in figure 27 can occur as usual. This combination of FDEP and SEQ gates can be used in a more general setting to tie multiple Markov chains together.

7.4.3 Full model and results

Figure 31 is the full fault tree model of the MAS, including the pools of spares. The leftmost FDEP and SEQ gates show the 2 spares for the vehicle management system, while those to the right represent the other 2 spares.

Because of the sequence dependency gates, this fault tree cannot be solved by standard 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 reliability estimate, 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}.

8 Three 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, in the way that messages are routed between processing nodes, and in the architecture of the individual processing nodes. The hypercube architectures are described in more detail in [7] and are discussed only briefly here.

8.1 System description

8.1.1 Architecture 1

Architecture 1 is based on the hierarchical approach to sparing proposed by Rennels[21] and is depicted in figure 32. It consists of 2 fault tolerant modules of processing nodes. Each module contains 4 processing nodes and one spare node. The spare is connected by a port to each of the 4 active processors in the module.

The processing nodes themselves are comprised of 5 individual processors (4 active processors and a spare) which communicate over an 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 three dimensions of the hypercube, and the fourth port communicates with the spare processing node of the module.

8.1.2 Architecture 2

Architecture 2, also depicted in figure 32, is identical to architecture 1 except that the ports within each processing node are replaced by hyperswitch ports[9]. 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.

8.1.3 Architecture 3

Architecture 3[8], depicted in figure 33, differs from architectures 1 and 2 in several important ways. Processing

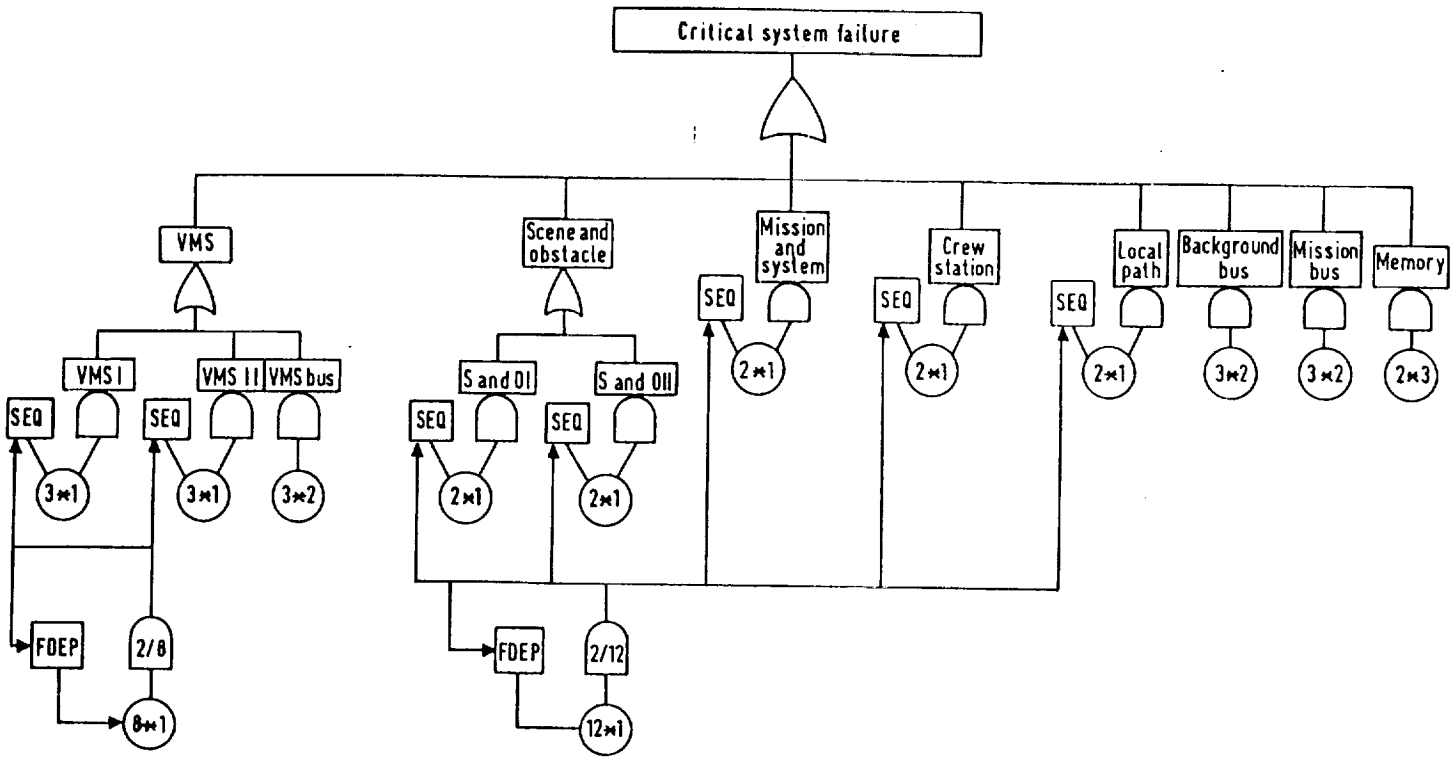

Figure 31. Full fault tree model of mission avionics system

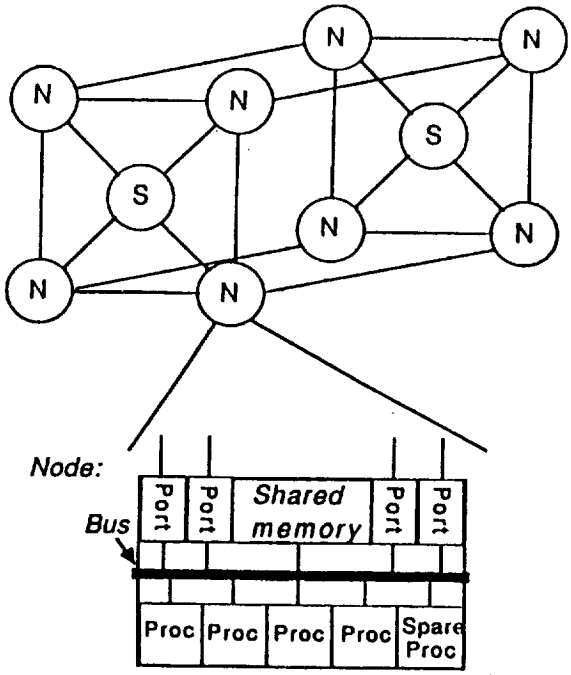

Figure 32: Architecture 1 (Rennels)

nodes are again configured into 2 fault tolerant modules (each containing 4 active processing nodes and one spare), however 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[8]. 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.

The processing nodes of the hypercube are much simpler and contain processors that are much less powerful that 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 (direct memory access) module. Finally, 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.

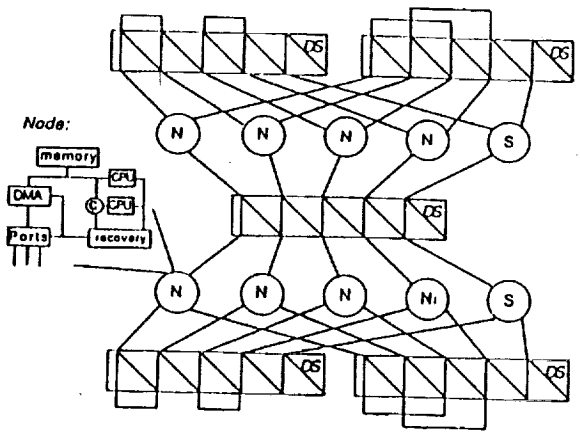

Figure 33: Architecture 3 (Chau and Liestman)

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 chosen can affect the reliability and power consumption of the node, which can in turn affect the overall ensemble reliability of the hypercube multiprocessor.

8.2 Failure criteria and parameters

The processor nodes for architectures 1 and 2 are identical, so their failure criteria are closely related. The difference between them is due to the message routing scheme employed by each architecture. A processor node for architectures 1 and 2 will fail if:

- the memory fails OR
- the bus fails OR
- 2 out of the 5 processors fail (the first processor failure is presumably recovered from 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. Since the spare processing node in each of the two 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 four 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 four 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 unit fails OR
- both processors fail OR,
- since the single path routing algorithm is used for this architecture, the node will fail if any of its 3 ports fail.

The component failure rates for all three architectures are listed below.

- Active processor (architectures 1 and 2): 1.990×10^{-6} per hour
- Active processor (architecture 3): 2.306×10^{-7} per hour
- Warm spare processor(architecture 2): 1.0×10^{-6} per hour
- Shared Memory (architectures 1 and 2): 3.477×10^{-7} per hour
- Memory (architecture 3): 1.147×10^{-7} per hour
- DMA module (architecture 3): 3.477×10^{-7} per hour
- Intra-node bus(architectures 1 and 2): 1.147×10^{-7} per hour
- Hyperswitch and I/O port (all architectures): 3.477×10^{-7} per hour

8.3 Fault recovery

The FEHM used for the processors assumes that a processor failure can be detected, located, and the spare successfully switched in to replace the failed processor 95% of the time, and that 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 reconfiguration attempt does not succeed, leading to node failure. The FEHM used for the ports assumes detection and deactivation of a failed port is successful 98% percent of the time, and that the time required for this is exponentially distributed with a mean of 0.1 sec. Again, the remaining 2% of the time a port failure is not successfully detected, leading to node failure. No transient restoration is attempted, i.e., all failures are considered to be permanent.

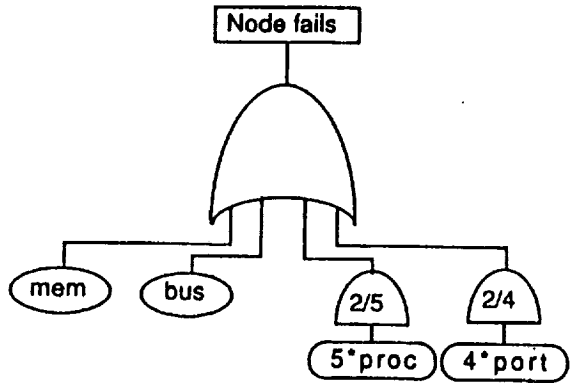

Figure 34: Fault tree model of architecture 1 processing node with hot spares

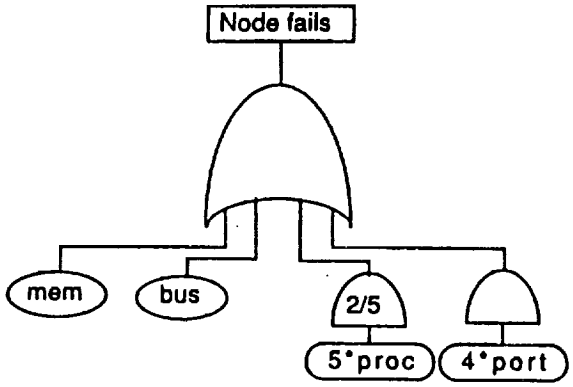

Figure 35: Fault tree model of architecture 2 processing node with hot spares

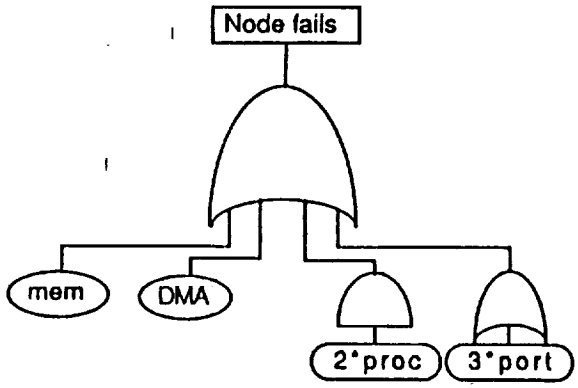

Figure 36: Fault tree model of architecture 3 processing node with hot spares

8.4 Fault tree models

8.4.1 Hot spares

Figures 34 and 35 model the processing nodes in architectures 1 and 2 when the spare processor in the node is a hot spare (the spare is powered on and operating all the time) and hence fails at the same rate as the active processors. The fault trees differ only in the modeling of port failures, as architecture 1 fails when 2 of the four ports fail (hence the 2/4 gate), while architecture 2 doesn't fail until all four ports have failed (hence the AND gate). Figure 36 depicts a fault tree model for the processing nodes of architecture 3.

8.4.2 Cold spares

Power consumption by a multiprocessor with spare nodes can be reduced by having the spares be *cold spares*, unpowered until they are needed to replace a failed active processor. A cold spare processor cannot fail until it is activated and brought online. In HARP this type of configuration is modeled using the Cold Spare gate, as depicted in figure 37 by a fault tree for architecture 2. The cold spare gate ensures that the spare processor does not fail until one of the 4 active processors fail. The 2/5 gate in parallel with the cold spare gate maintains the requirement that 2 processor failures cause the node to fail. Such a configuration not only reduces power consumption, but also enhances the reliability of the processing node.

8.4.3 Warm spares

Instead of being unpowered, the spare may be partially powered up. It may then fail before being activated but at a lesser rate than the active processors. Such a processor is called a *warm spare* and can be modeled in HARP using the *Sequence Enforcing gate* as shown in figure 38 for architecture 2. In this example two pseudo-components (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 failure of a processor (either active or spare), these two pseudo-components are "turned off" as far as the fault tree is concerned by the Functional Dependency 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 (i.e. the processors of this basic event are now permitted to fail). Note that because this basic event is also an input to the top OR gate of the fault

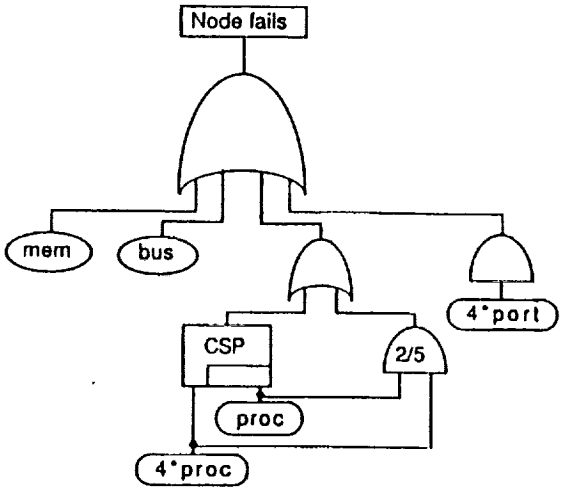

Figure 37: Fault tree model of architecture 2 processing node with cold spares

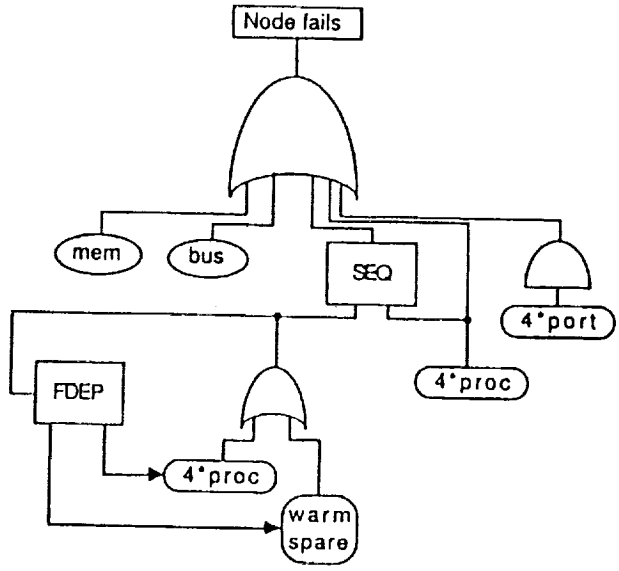

Figure 38: Fault tree model of architecture 2 processing node with warm spares

tree, a subsequent failure of any of the 4 processors will cause the node to fail, again maintaining the requirement that failure of 2 of the 5 processors cause node failure. Although a spare does not fail while unpowered, upon power up and activation there can be some probability that the spare does not operate properly. Such a situation can be modeled as a warm spare.

8.5 Results

Figure 39 compares the 10 year unreliabilities of the processing nodes of each of the three architectures assuming all of them use hot spares. The unreliability of the architecture 3 processing nodes is much lower that those for architectures 1 and 2, reflecting that the reliability of individual processors for architecture 3 is much greater than that of the others and 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.

Figure 40 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 expected, since the failure rate of the spare during its inactive period decreases in that order.

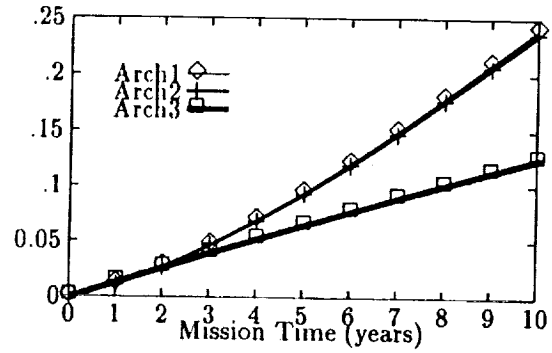

Figure 39: Comparison of node unreliabilities of all three architectures using hot spares

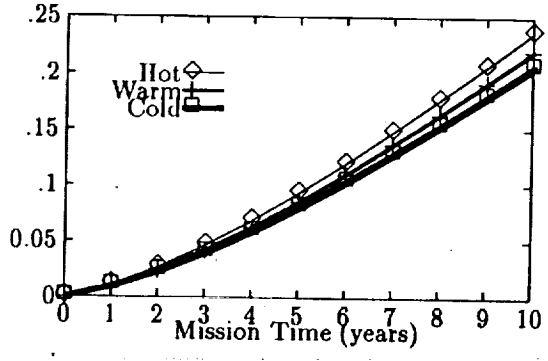

Figure 40: Unreliability of architecture 2 processing nodes with various types of spares

9 References

[1] S. J. Bavuso, Joanne Bechta Dugan, K. S. Trivedi, E. M. Rothmann, and W. E. Smith. Analysis of typ-

ical fault-tolerant architectures using HARP. *IEEE Transactions on Reliability*, R-36(2):176–185, June 1987.

[2] Salvatore Bavuso and Sandra Howell. A graphical language for reliability model generation. In *Proceedings 36th Annual Reliability and Maintainability Symposium*, 1990.

[3] Salvatore J. Bavuso and Joanne Bechta Dugan. Hirel: Reliability/availability integrated workstation tool. In *Proceedings of the Reliability and Maintainability Symposium*, pages 491–500, January, 1992.

[4] Salvatore J. Bavuso and Anna Martensen. A fourth-generation reliability predictor. In *Proceedings of the Reliability and Maintainability Symposium*, pages 11–16, January, 1988.

[5] Stephen W. Behnen, William A. Whitehouse, Richard J. Farrell, F. Mark Leahy, and LeRoy E. Moen. Advanced system integration demonstrations (ASID) system definition. Technical report, AF Wright Aeronautical Laboratories, 1984.

[6] M. A. Boyd, M. Veeraraghavan, Joanne Bechta Dugan, and K. S. Trivedi. An approach to solving large reliability models. In *AIAA/IEEE Digital Avionics Systems Conference, San Jose, CA*, October 1988.

[7] Mark A. Boyd and Jesus Tuazon. Fault tree models for fault tolerant hypercube multiprocessors. In *Proceedings of the Reliability and Maintainability Symposium*, January 1991.

[8] S. C. Chau and A. Liestman. Proposal for a fault-tolerant binary hypercube architecture. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-19*, pages 323–330, June 1989.

[9] E. Chow, J. Peterson, and H. Madan. Hyperswitch network for the hypercube computer. In *Digest of the 13th Symposium on Computer Architecture*, pages 90–99, May 1988.

[10] Joanne Bechta Dugan, Salvatore Bavuso, and Mark Boyd. Dynamic fault tree models for fault tolerant computer systems. *IEEE Transactions on Reliability*, September 1992.

[11] Joanne Bechta Dugan, Salvatore Bavuso, and Mark Boyd. Fault trees and sequence dependencies. In *Proceedings of the Reliability and Maintainability Symposium*, pages 286–293, January, 1990.

[12] Joanne Bechta Dugan, Salvatore J. Bavuso, and Mark A. Boyd. Fault trees and markov models for reliability analysis of fault tolerant systems. *Reliability Engineering and System Safety*, 39:291–307, 1993.

[13] Joanne Bechta Dugan and K. S. Trivedi. Coverage modeling for dependability analysis of fault-tolerant systems. *IEEE Transactions on Computers*, 38(6):775–787, 1989.

[14] Joanne Bechta Dugan, K. S. Trivedi, Mark K. Smotherman, and Robert M. Geist. The hybrid automated reliability predictor. *AIAA Journal of Guidance, Control and Dynamics*, 9(3):319–331, May-June 1986.

[15] Joanne Bechta Dugan, Malathi Veeraraghaven, Mark Boyd, and Nitin Mittal. Bounded approximate reliability models for fault tolerant distributed systems. In *Proceedings 8th Symposium on Reliable Distributed Systems*, pages 137–147, 1989.

[16] E. Feldman and P. S. Babcock. Reliability evaluation of AIPS I/O networks. Technical Report AIPS-87-15, C. S. Draper Laboratory, Inc., Cambridge, MA, June 1987.

[17] Richard E. Harper. Reliability analysis of parallel processing systems. In *Proceedings of the 8th Digital Avionics Systems Conference*, pages 213–219, 1988.

[18] Richard E. Harper, Jaynarayan H. Lala, and John J. Deyst. Fault tolerant parallel processor architecture overview. In *Proceedings of the 18th Symposium on Fault Tolerant Computing*, pages 252–257, 1988.

[19] E. J. Henley and H. Kumamoto. *Probabilistic Risk Assessment*. IEEE Press, 1982.

[20] Ying-Wah Ng and Algirdas Avizienis. A model for transient and permanent fault recovery in closed fault-tolerant systems. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-6*, pages 182–187, June 1976.

[21] D. A. Rennels. On implementing fault-tolerance in binary hypercubes. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-16*, pages 344–349, July 1986.

[22] D. P. Siewiorek and R. S. Swarz. *The Theory and Practice of Reliable System Design*. Digital Press, Bedford, MA, 1982.

[23] K. S. Trivedi, Robert Geist, Mark Smotherman, and Joanne Bechta Dugan. Hybrid modeling of fault-tolerant systems. *Computers and Electrical Engineering, An International Journal*, 11(2 & 3):87–108, 1985.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1993	Technical Memorandum

4. TITLE AND SUBTITLE
Tutorial: Advanced Fault Tree Applications Using HARP

5. FUNDING NUMBERS
WU 505-66-21

6. AUTHOR(S)
Joanne Bechta Dugan
Salvatore J. Bavuso
Mark A. Boyd

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA TM-102747

11. SUPPLEMENTARY NOTES
Dugan: Duke University, Durham, NC (presently at University of Virginia, Charlottesville, VA); Bavuso: NASA Langley Research Center, Hampton, VA; Boyd: Duke University, Durham, NC (presently at NASA Ames Research Center, Moffett Field, CA)

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

Reliability analysis of fault tolerant computer systems for critical applications is complicated by several factors. In this tutorial, we discuss these modeling difficulties and describe and demonstrate dynamic fault tree modeling techniques for handling them. Several advanced fault tolerant computer systems are described, and fault tree models for their analysis are presented. HARP (the Hybrid Automated Reliability Predictor) is a software package developed at Duke University and NASA Langley Research Center that is capable of solving the fault tree models presented into this tutorial.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Fault Tree, Reliability, Tutorial, Fault-Tolerance, Markov chains	29
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited