NASA CR-132552

(NASA-CR-132552) DEFINITION AND TRADE-OFF STUDY OF RECCNFIGURABLE AIRBORNE DIGITAL COMPUTER SYSTEM ORGANIZATIONS Final Report, 9 Oct. 1973 - 8 Nov. 1974 (Ultrasystems, Inc., Newport Beach, Calif.) 301 p HC $9.25 G3/60

N75-15325

Unclas
06803

DEFINITION AND TRADE-OFF STUDY OF RECONFIGURABLE AIRBORNE DIGITAL COMPUTER SYSTEM ORGANIZATIONS

FINAL REPORT

NOVEMBER 1974

PREPARED UNDER

NASA CONTRACT NAS1-12793

FOR

NASA LANGLEY RESEARCH CENTER
HAMPTON, VIRGINIA

BY

ultrasystems

NEWPORT BEACH, CALIFORNIA

RECONFIGURABLE COMPUTER
SYSTEMS STUDY

FINAL REPORT

CONTRACT SCHEDULE ITEM III-E

prepared for

Langley Research Center
National Aeronautics and Space Administration
Hampton, Virginia 23665

Contract NAS1-12793

by

Ultrasystems, Inc.
500 Newport Center Drive
Newport Beach, California 92660

November 1974

~~73/6.27-19~~

FOREWORD

This Final Report was prepared by Ultrasystems, Incorporated, Newport Beach, California, under National Aeronautics and Space Administration contract NAS1-12793. The work was performed between 9 October 1973 and 8 November 1974. Sponsorship for this work was provided by the Flight Instrumentation Division, Electronics Directorate, Langley Research Center. The Project Monitor was Mr. J. Larry Spencer with technical assistance provided by other members of the Aircraft Electronics Research Section.

The Ultrasystems' Study Leader was Ralph B. Conn. Other participants in the study were Dr. A. A. Avižienis, Mr. H. O. Lévy, Dr. P. M. Merryman, Mr. S. R. Pond, Dr. D. A. Rennels, Dr. J. A. Rohr, and Mr. K. L. Whitelaw.

Publication of this report does not constitute NASA approval of the findings or conclusions indicated in the report. It is published for the exchange of information and stimulation of ideas.

RECONFIGURABLE COMPUTER SYSTEMS STUDY

FINAL REPORT

TABLE OF CONTENTS

PAGE

1.0 SUMMARY AND INTRODUCTION 1-1

1.1 OBJECTIVE, ACCOMPLISHMENTS, AND CONCLUSIONS 1-1

1.1.1 Objective 1-1

1.1.2 Accomplishments 1-1

1.1.3 Conclusions 1-2

1.2 INTRODUCTION 1-4

1.3 SYSTEM ORGANIZATION CONCEPTS 1-5

1.3.1 "Mostly" - Software Configurations 1-6

1.3.2 Hardware - Aided Software Configurations (HASW) . 1-8

1.3.3 Mostly-Hardware Configurations 1-9

1.4 EXECUTIVE STRUCTURE 1-10

1.5 MEASURES OF FAULT-TOLERANCE 1-13

1.6 ANALYTIC MODELING 1-15

1.7 SIMULATION 1-16

1.8 COMBINED ANALYTIC-SIMULATIVE TECHNIQUE 1-20

1.9 RECOMMENDATIONS 1-21

2.0 SYSTEM ORGANIZATION CONCEPTS 2-1

2.1 "MOSTLY"-SOFTWARE REDUNDANT CONFIGURATION (MSW) 2-2

2.1.1 Internal Communications for Fault Detection and Transient Recovery 2-3

2.1.1.1 Voting and Synchronization 2-5

2.1.1.2 Transient Recovery Techniques 2-7

2.1.1.3 Transient Recovery - Implementation 2-9

2.1.1.4 Utilization of Transient Recovery Techniques . . 2-12

TABLE OF CONTENTS (Cont'd)

PAGE

2.1.2 Redundant I/O Structures for Communications with Fault Masking 2-12

2.1.2.1 Dedicated Busses 2-13

2.1.2.2 Non-Dedicated Busses (Non-Dedicated Sensors . . . 2-16

2.1.2.3 Non-Dedicated Switched Busses 2-17

2.1.3 Executive Program Considerations 2-20

2.1.3.1 Voter Module 2-21

2.1.3.2 Input/Output Module 2-25

2.1.3.3 Error-Handler Module 2-26

2.1.4 Applications Programs Considerations 2-26

2.1.5 Machine Features and RETs 2-28

2.1.5.1 Machine Features 2-28

2.1.5.2 Reliability Enhancement Techniques 2-29

2.1.5.3 Machine Options 2-29

2.1.5.4 Fault Detection 2-30

2.1.5.5 Transient Recovery 2-31

2.1.5.6 Internal Cross Connections 2-31

2.1.5.7 I/O Structures 2-31

2.1.5.8 Bus Fault-Detection 2-32

2.2 HARDWARE-AIDED SOFTWARE CONFIGURATIONS (HASW) 2-33

2.2.1 The External Electronics Module (EEM) 2-35

2.2.1.1 Case 1 - No Faults 2-36

2.2.1.2 Case 2 - One Computer Disagrees 2-36

2.2.1.3 The Transient Recovery Mechanism 2-38

TABLE OF CONTENTS (Cont'd)

PAGE

2.2.1.4 Case 3 - Multiple Faults 2-39

2.2.2 Residual Duplex and Augmented Voting Redundancy . 2-40

2.2.3 A Recovery Algorithm: Hardware-Aided Software Configuration 2-40

2.2.3.1 Hardware EEM Functions 2-41

2.2.3.2 The Software Function 2-44

2.2.4 Utilization of Redundant EEM Units 2-45

2.2.4.1 Non-Dedicated Redundant EEMs 2-45

2.2.4.2 Dedicated EEMs 2-47

2.3 MOSTLY-HARDWARE CONFIGURATIONS 2-48

2.3.1 Augmented EEM Units 2-49

2.3.1.1 The State Control Function 2-49

2.3.2 Recovery Control 2-49

2.3.2.1 Memory Copy Implementation 2-52

2.3.2.2 Rollahead - Rollback - Copy Implementation . . . 2-54

2.3.3 A Comparison of MHW and HASW Implementations . . 2-54

3.0 EXECUTIVE STRUCTURE 3-1

3.1 DESIGN GOALS 3-1

3.2 SKELETON MODULES 3-2

3.2.1 Scheduler 3-4

3.2.2 Input-Output Driver 3-4

3.2.3 Interrupt Processor 3-5

3.2.4 Machine Error Handler 3-6

3.2.5 Interaction 3-6

TABLE OF CONTENTS (Cont'd)

PAGE

3.3 SCHEDULING MECHANISMS 3-7

3.3.1 Synchronous 3-8

3.3.2 Synchronous with Asynchronous Overlay 3-10

3.3.3 Hybrid 3-10

3.3.4 Hybrid with External Interrupts 3-10

3.3.5 Constrained Asynchronous 3-12

3.3.6 Comparison and Contrast 3-12

3.4 ADAPTABILITY 3-13

3.5 SOFTWARE STRUCTURE AND FAULT-TOLERANCE IMPLEMENTATION . . 3-14

3.5.1 Software Structure Considerations for a Duplex System 3-14

3.5.1.1 Executive Scheduling Mechanisms 3-14

3.5.1.2 Typical Computational Cycle 3-16

3.5.1.3 Redundancy Requirements 3-18

3.5.1.4 Tradeoffs 3-19

3.5.2 Software Structure Considerations for a TMR System 3-21

3.5.2.1 Executive Scheduling Mechanisms 3-21

3.5.2.2 Typical Computational Cycle 3-22

3.5.2.3 Redundancy Requirements 3-23

3.5.2.4 Tradeoffs 3-24

4.0 MEASURES OF FAULT-TOLERANCE 4-1

4.1 THE CONCEPT OF FAULT-TOLERANCE 4-1

4.1.1 The Reliability Problem for Computers 4-1

4.1.2 "Fault-Intolerant" Design for Reliable Operation . 4-2

TABLE OF CONTENTS (Cont'd)

PAGE

4.1.3 Design of "Fault-Tolerant" Systems 4-3

4.2 QUANTITATIVE SPECIFICATION OF FAULT-TOLERANCE 4-5

4.2.1 Classification of Measures 4-5

4.2.2 Discrete Fault Tolerance (DFT) 4-6

4.2.3 Reliability 4-8

4.2.4 Survivability 4-9

4.2.5 Quantitative Measures of Survivability 4-11

5.0 ANALYTIC MODELING 5-1

5.1 MODELING APPROACH 5-1

5.1.1 General 5-1

5.1.2 Solution Approach 5-1

5.2 TRANSIENT FAULTS 5-2

5.2.1 Transient Arrival 5-2

5.2.2 Transient Duration 5-4

5.3 TRANSIENT RECOVERY MODEL 5-4

5.3.1 Components of Transient Recovery 5-4

5.3.2 Fault Detection 5-5

5.4 ANALYSIS OF AN ENHANCED TMR CONFIGURATION 5-6

5.4.1 Definitions and Assumptions 5-6

5.4.2 Failure Probability 5-8

5.4.3 Transient Leakage 5-13

5.4.4 Transient Coverage 5-16

5.4.5 Simplifying Assumptions for Shorter Mission Times 5-17

5.4.6 Extension of TMR Modeling to N Computers 5-18

TABLE OF CONTENTS (Cont'd)

PAGE

5.4.6.1 Fault/Recovery State Diagram 5-18

5.4.6.2 Definitions and Review 5-20

5.4.6.3 Finding Failure Probability for Four Computers . . 5-22

5.4.6.4 Finding the Failure Probability for Five Computers 5-23

5.4.6.5 Generalization to N Computers 5-25

5.4.7 Recovery Start Delay 5-25

5.5 MODELING OF GENERAL CONFIGURATIONS 5-28

5.5.1 The Recovery Process 5-28

5.5.1.1 Coverage 5-28

5.5.1.2 Transient Leakage 5-29

5.5.1.3 Permanent Recovery 5-29

5.5.1.4 Notation System 5-30

5.5.2 Analysis of a Duplex Configuration 5-30

5.5.2.1 Definitions and Assumptions 5-30

5.5.2.2 Fault/Recovery State Diagram 5-31

5.5.2.3 Failure Probability 5-32

5.5.3 Extension to N Computers 5-34

5.5.3.1 State Diagram 5-34

5.5.3.2 Failure Probability Determination 5-36

5.5.3.3 General Solution 5-37

5.5.4 Simplifying Assumptions 5-40

5.5.4.1 Simplex 5-40

5.5.4.2 Duplex 5-41

5.5.4.3 Enhanced TMR 5-41

TABLE OF CONTENTS (Cont'd)

PAGE

5.5.4.4 Adaptive TMR 5-42

5.6 MARKOV CHAIN ANALYSIS METHOD 5-42

5.6.1 Mathematical Model 5-42

5.6.1.1 Development of the Differential Equation . . . 5-44

5.6.1.2 Solution Procedure 5-47

5.6.1.3 Closed Form Solution 5-48

5.6.1.4 Power Series Evaluation of P(t) 5-49

5.6.2 Application to the Duplex Configuration 5-50

5.6.2.1 Determination of the Transition Matrix 5-50

5.6.2.2 Closed Form Solution for Duplex Configuration . . 5-53

5.6.2.3 Approximation for Small Mission Times 5-46

5.6.3 Application to Adaptive TMR Configuration . . . 5-58

5.6.3.1 Determination of the Transition Matrix 5-58

5.6.3.2 Approximations for Small Mission Times 5-59

5.6.4 Programs 5-60

5.6.4.1 Projector Method 5-60

5.6.4.2 Power Series Method 5-61

5.6.5 Conclusions 5-62

6.0 SIMULATION 6-1

6.1 OBJECTIVES OF SIMULATION 6-1

6.1.1 Configuration Fault-Tolerance 6-1

6.1.2 Determination of Global Parameters Used in Analytical Modeling 6-1

6.1.3 Fault Environment 6-2

TABLE OF CONTENTS (Cont'd)

PAGE

6.2 GENERAL ORGANIZATION OF THE SIMULATION 6-2

6.2.1 General Approach 6-2

6.2.2 Organization of the Simulator 6-3

6.3 INPUTS/OUTPUTS 6-8

6.3.1 Inputs 6-8

6.3.1.1 Detection Probabilities 6-10

6.3.1.2 Self-Test Program Efficiency 6-12

6.3.1.3 Dedicated/Non-Dedicated EEMs 6-12

6.3.1.4 Existing Recovery Algorithms 6-12

6.3.1.5 Unacceptable Recurrence Intervals 6-12

6.3.1.6 Program Integrity 6-12

6.3.1.7 Memory Copy Efficacy 6-13

6.3.2 Output 6-13

6.4 STATE DIAGRAM 6-14

6.4.1 Normal Operation (3 or more Units) 6-14

6.4.2 Rollahead (or State Vector Transfer 6-17

6.4.3 Memory Copy 6-17

6.4.4 System Restart 6-17

6.4.5 Introduction of a Spare 6-18

6.4.6 Normal Operation (2 Units) 6-18

6.4.7 Rollback 6-18

6.4.8 Diagnosis 6-19

6.4.9 Normal Operation (Simplex) 6-19

6.4.10 Rollback in Simplex 6-19

TABLE OF CONTENTS (Cont'd)

PAGE

6.4.11 System Failure 6-20

6.5 SIMULATOR IMPLEMENTATION 6-21

6.5.1 Fault Generation 6-21

6.5.1.1 Introduction 6-21

6.5.1.2 Parameters 6-21

6.5.1.3 Description of the Fault Table 6-23

6.5.1.4 General Organization of the Fault Generator . . 6-23

6.5.1.5 Determination of the Occurrence Time of the Faults According to a Poisson Distribution Function . . 6-23

6.5.1.6 Determination of the Duration 6-27

6.5.1.7 Determination of the Occurrence Time of the Faults According to a Burst Distribution Function 6-27

6.5.2 Normal Operation (3 or More Units) 6-27

6.5.3 Rollahead 6-27

6.5.4 Other States 6-31

6.5.5 Introduction of the Scheduling Mechanisms . . . 6-31

6.5.5.1 Synchronous Scheduling 6-33

6.5.5.2 Detection of Faults 6-33

6.5.5.3 Iteration Losses 6-33

6.5.5.4 Asynchronous Scheduling 6-34

6.5.6 EEM Faults 6-34

6.5.6.1 Dedicated EEMs 6-34

6.5.6.2 Non-Dedicated EEMs 6-34

6.5.7 Input-Output Faults 6-35

6.5.7.1 Dedicated Buses 6-35

TABLE OF CONTENTS (Cont'd)

PAGE

6.5.7.2 Non-Dedicated Buses 6-38

6.6.1 Fault Generator 6-38

6.6.2 Simulator 6-38

6.7 SAMPLE RUN 6-39

7.0 PARAMETERS 7-1

7.1 SIMULATOR 7-1

7.1.1 STP Efficiency 7-1

7.1.1.1 STP Requirements 7-1

7.1.1.2 Efficiency Estimation 7-2

7.1.1.3 Typical Computers 7-3

7.1.2 Program Integrity 7-3

7.1.2.1 CPU Faults 7-3

7.1.2.2 Memory 7-3

7.1.2.3 PI Estimation 7-4

7.1.3 BITE Efficiency 7-6

7.1.3.1 CPU BITE Efficiency 7-6

7.1.3.2 Memory BITE Efficiency 7-6

7.2 ANALYTIC MODEL 7-7

7.2.1 Computer Effective Failure Rate 7-7

7.2.2 Recoverability 7-8

7.2.3 Transient Leakage 7-8

8.0 COMPLEMENTARY ANALYTIC-SIMULATIVE TECHNIQUE 8-1

8.1 OVERALL STRUCTURE 8-1

8.2 RCS ENGINEERING ANALYSIS 8-1

TABLE OF CONTENTS (Cont'd)

PAGE

8.3 SIMULATION 8-3

8.4 ANALYTIC MODELING 8-3

9.0 CONFIGURATION ANALYSES AND TRADE-OFF STUDIES 9-1

9.1 GENERAL 9-1

9.2 PARAMETERS USED FOR EVALUATION 9-2

9.2.1 Mostly-Software Configurations 9-2

9.2.1.1 Physical Parameters 9-2

9.2.1.2 Software Characteristics 9-4

9.2.1.3 Parameters Affecting Fault Tolerance in the Computers 9-4

9.2.1.4 Parameters Affecting Fault Tolerance in the External Hardware 9-4

9.2.1.5 Transient Fault Recovery Parameters 9-4

9.2.1.6 Permanent Fault Recovery Parameters 9-5

9.2.1.7 Modeling Parameters 9-5

9.2.2 Hardware-Aided-Software Configurations 9-5

9.2.2.1 Physical Parameters 9-5

9.2.2.2 Software Characteristics 9-6

9.2.2.3 Parameters Affecting Fault Tolerance in the Computers 9-6

9.2.2.4 Parameters Affecting Fault Tolerance in the External Hardware 9-6

9.2.2.5 Fault Recovery Parameters 9-6

9.2.2.6 Modeling Parameters 9-6

9.2.3 Mostly-Hardware Configurations 9-8

TABLE OF CONTENTS (Cont'd)

PAGE

9.3 GENERATION OF RESULTS 9-9

9.3.1 Assessed Configurations 9-9

9.3.1.1 Quintuplex Configurations 9-9

9.3.1.2 Quadruplex Configurations 9-13

9.3.1.3 Triplex Configurations 9-13

9.3.2 Effect of Redundancy 9-22

9.3.3 Effect of Non-Unity Recoverability 9-24

9.3.4 Effects of Adaptivity 9-26

9.3.5 Effects of RETs 9-26

9.3.5.1 DRO Versus NDRO 9-26

9.3.5.2 Effects of BITE 9-32

9.3.5.3 Effects of Diagnostics 9-34

9.3.5.4 Codes and I/O Wraparound 9-34

9.3.5.5 Reasonableness Tests and Sensor Redundancy Management 9-35

9.3.5.6 Voters, Adaptive Voters, and Comparators . . . 9-38

9.3.5.7 Dedicated/Non-Dedicated I/O Units 9-39

9.3.5.8 Independent Hardware Monitor 9-39

9.3.6 Effects of Transients 9-40

9.3.6.1 Introduction of Transient Recovery 9-40

9.3.6.2 Transient Recovery Algorithms 9-40

9.3.6.3 Influence of Transient Duration 9-41

9.3.6.4 Influence of Bursts of Transients 9-41

9.3.7 Scheduling Effects 9-42

RECONFIGURABLE COMPUTER SYSTEMS STUDY
FINAL REPORT

TABLE OF CONTENTS (Cont'd)

PAGE

9.4 CONCLUSIONS 9-45

REFERENCES

THIS PAGE INTENTIONALLY LEFT BLANK

RECONFIGURABLE COMPUTER SYSTEMS STUDY

FINAL REPORT

LIST OF FIGURES

PAGE

1.1-1 EFFECTS OF RCS REDUNDANCY AND ADAPTABILITY ON FAILURE PROBABILITY 1-3

1.7-1 SIMULATOR STATE DIAGRAM 1-17

2.1-1 MSW INTERNAL CROSS-CONNECTIONS 2-4

2.1-2 SYNCHRONIZATION AND VOTING SCENARIO 2-6

2.1-3 DEDICATED BUSSES 2-14

2.1-4 NON-DEDICATED, SWITCHED BUS CONFIGURATION 2-18

2.1-5 VOTER MODULE LOGIC 2-22

2.1-6 INPUT - OUTPUT LOGIC - ADDITIONS 2-23

2.1-7 ERROR MODULE LOGIC - ADDITIONS 2-27

2.2-1 EXTERNAL HARDWARE INTERFACE: HARDWARE-AIDED SOFTWARE CONFIGURATION 2-34

2.2-2 THE EXTERNAL ELECTRONICS MODULE (EEM) 2-37

2.2-3a RECOVERY SOFTWARE - (ROLLAHEAD) HARDWARE - AIDED CONFIGURATION 2-42

2.2-3b RECOVERY SOFTWARE - (MEMORY COPY) HARDWARE-AIDED CONFIGURATION 2-43

2.2-4 REDUNDANT EEM IMPLEMENTATIONS 2-46

2.3-1 AUGMENTED EEM FOR MOSTLY-HARDWARE CONFIGURATIONS 2-51

2.3-3 AUGMENTED EEM RECOVERY ALGORITHMS 2-53

3.2-1 THE EXECUTIVE MODULES USE INTERMODULE CALLS FOR SERVICES WHICH ARE REQUIRED BY ONE MODULE BUT PROVIDED BY ANOTHER . . 3-3

3.3-1 EXECUTIVE SCHEDULING MECHANISM TYPICAL TIME LINES 3-9

3.3-2 COMPARISON OF EXECUTIVE SCHEDULING MECHANISMS 3-11

4.2-1 QUADDED DIODES, d=1 4-7

LIST OF FIGURES (Cont'd)

PAGE

4.2-2 COMPUTERS IN TMR CONFIGURATION 4-7

5.1-1 FAULT RECOVERY STATE DIAGRAM OF A TMR CONFIGURATION 5-3

5.3-1 PROBABILITY DENSITY FUNCTION OF THE TIME TO FAULT DETECTION . 5-7

5.3-2 UNIFORM-EXPONENTIAL APPROXIMATION TO THE FAULT DETECTION TIME DENSITY FUNCTION 5-9

5.4-1 FAULT RECOVERY MODEL OF A TMR CONFIGURATION 5-11

5.4-2 EXTENSION OF ENHANCED TMR MODEL TO N COMPUTERS 5-21

5.4-3 FAILURE PROBABILITY FOR FIVE COMPUTERS 5-27

5.5-1 FAULT OCCURRENCE/RECOVERY STATUS STATE DIAGRAM FOR A DUPLEX CONFIGURATION 5-33

5.5-2 FAULT OCCURRENCE/RECOVERY STATUS STATE DIAGRAM FOR 1-5 COMPUTER CONFIGURATIONS 5-35

5.6-1 MARKOV CHAIN EXAMPLE 5-43

5.6-2 STATE DIAGRAM FOR THE DUPLEX CONFIGURATION 5-51

6.2-1 SIMULATOR STATE DIAGRAM 6-5

6.2-2 GROSS ORGANIZATION OF THE SIMULATION 6-6

6.2-3 PRINCIPLES OF A FAULT DRIVEN SIMULATION (BOX 3 OF FIGURE 6.2-2) 6-7

6.2-4 RCS HANDLING OF FAULTS (BOXES 3, 4, 5 OF FIGURE 6.2-3) . . . 6-9

6.3-1 LIST OF INPUT PARAMETERS 6-11

6.4-1 SIMULATOR DETAILED STATE DIAGRAM 6-16

6.5-1 GENERAL ORGANIZATION OF THE FAULT GENERATOR 6-24

6.5-2 GENERATION OF THE OCCURRENCE OF THE FAULTS IN ONE MODULE (POISSON DISTRIBUTION) 6-28

6.5-3 NORMAL OPERATION STATE I 6-29

6.5-4 ROLLAHEAD STATE II FLOWCHART 6-30

6.5-5 EXAMPLE OF DEDICATED BUS CONFIGURATION 6-37

LIST OF FIGURES (Cont'd)

PAGE

6.7-1 SOFTWARE TMR WITHOUT MEMORY COPY 6-41

8.1-1 FAULT-TOLERANCE MEASURES CAN BE PRODUCED THROUGH A COMBINATION OF ENGINEERING ANALYSIS, SIMULATION, AND ANALYTIC MODELING . 8-2

8.4-1 CAST SUMMARY DIAGRAM 8-5

9.3-1 QUINTUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES 9-10

9.3-2 QUINTUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES 9-12

9.3-3 QUINTUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR ADAPTIVE BUSSES 9-14

9.3-4 QUADRUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES 9-16

9.3-5 QUADRUPLEX FAILURE PROBABILITY VERSUS EXTENDED MISSION TIME FOR NON-ADAPTIVE BUSSES 9-17

9.3-6 QUADRUPLEX FAILURE PROBABIITY VERSUS MISSION TIME FOR ADAPTIVE BUSSES 9-18

9.3-7 TRIPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES 9-20

9.3-8 TRIPLEX FAILURE PROBABILITY VERSUS EXTENDED MISSION TIME FOR NON-ADAPTIVE BUSSES 9-21

9.3-9 PROBABILITY VERSUS EXTENDED MISSION TIME FOR 5, 4, 3 AND 2 COMPUTERS 9-23

9.3-10 EFFECTS OF NON-UNITY RECOVERABILITY ON FAILURE PROBABILITY FOR EXTENDED MISSION TIMES 9-25

9.3-11 EFFECT OF ADAPTABILITY ON FAILURE PROBABILITY FOR EXTENDED MISSION TIMES 9-28

9.3-12 10-HOUR FAILURE PROBABILITY VERSUS TRANSIENT FAULT RATE . . 9-30

9.3-13 10-HOUR FAILURE PROBABILITY VERSUS TRANSIENT FAULT RATE . . 9-31

9.3-14 10-HOUR FAILURE PROBABILITY VERSUS TRANSIENT FAULT RATE . . 9-33

9.3-15 COMPARISON OF CODED SIMPLEX, DUPLEX AND TMR BUSSES 9-37

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

PAGE

1.3-I SOFTWARE OVERHEAD OF FAULT-TOLERANT CONFIGURATIONS . . . 1-9

1.7-I LIST OF RCS SIMULATOR INPUTS 1-18

1.7-II RCS SIMULATOR OUTPUTS 1-19

2.3-I SOFTWARE OVERHEAD OF FAULT-TOLERANT CONFIGURATIONS . . . 2-48

3.5-I SOFTWARE EFFECTS ON v_2w_2 3-20

5.4-I SUMMARY OF EQUATIONS FOR THE TMR CONFIGURATION 5-19

9.2-I LIST OF INPUT PARAMETERS FOR MOSTLY SOFTWARE CONFIGURATIONS 9-3

9.2-II LIST OF INPUT PARAMETERS FOR HARDWARE CONFIGURATIONS . . 9-7

9.3-I SUMMARY OF QUINTUPLEX CONFIGURATION ASSESSMENTS 9-11

9.3-II SUMMARY OF QUADRUPLEX CONFIGURATION ASSESSMENTS 9-15

9.3-III SUMMARY OF TRIPLEX CONFIGURATION ASSESSMENTS 9-19

9.3-IV SUMMARY OF THE EFFECTS OF ADAPTABILITY 9-27

9.3-V LEAKAGE COEFFICIENTS 9-29

9.3-VI EFFECTS OF BITE 9-32

9.3-VII FAILURE PROBABILITIES AFTER 10 AND 100 HOURS FOR 4-MR TMR WITH AND WITHOUT DIAGNOSTICS 9-34

9.3-VIII EFFECTS OF REASONABLENESS TESTS 9-36

9.3-IX EFFECTS OF NON-DEDICATED SENSORS 9-38

9.3-X EFFECTS OF SCHEDULING 9-43

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS AND ABBREVIATIONS

A	Height of uniform portion of uniform-exponential density $f_{T_d}(t)$
AEEM	Augmented External Electronic Module
AGE	Aerospace Ground Equipment
BITE	Built-In Test-Equipment
CAST	Complementary Analtyic-Simulative Technique
c_T	Transient coverage in TMR
DFBW	Digital Fly-By-Wire
DFT	Discrete Fault-Tolerance
DMA	Direct-Memory Access
D_T	Transient duration
DUP	Duplex - A two computer configuration
EEM	External Electronics Module
EH	External Hardware
F	Set of failed machines
F	System failure probability (1-S or 1-R)
FCS	Flight Control System
FD	Fault Detected
F_p	Failure probability due to a permanent or transient fault when in the one computer faulty state (TMR model)
F_T	Failure probability due to a fault during a recovery process (TMR model)
$f_{T_d}(t)$	Probability density function of the fault detection time
HASW	Hardware-Aided Software
I/O	Input-Output
MFP	Mission Failure Probability (F)
MHW	Mostly Hardware

LIST OF SYMBOLS AND ABBREVIATIONS (Cont'd)

MSP	Mission Success Probability (1-F)
MSW	Mostly-Software
N	Set of working and failed machines
NDRO	Non-Destructive Read-Out
NMR	N-Tuple Modular Redundancy
OR	Output Ready
PI	Program Integrity
PS	Program Survivability -- synonymous with program integrity
RCS	Reconfigurable Computer System
RET	Reliability Enhancement Technique
ROM	Read-Only Memory
RTI	Real-Time Interrupt
S	Survivability (1-F when transient faults are included)
SIM	Simplex - A single computer configuration
STP	Self-Test Program
T	Mission time
T_c	Time between state vector comparisons
T_d	Time between fault arrival and fault detection
T_D	Delay between fault detection and beginning of recovery procedure
TMR	Triple Modular Redundancy
TMR-A	Triple Modular Redundancy - Adaptive
TMR-E	Triple Modular Redundancy - Enhanced
TMR-H	Triple Modular Redundancy - All Hardware
TMR-HA	Triple Modular Redundancy - Hardware Aided
TMR-S	Triple Modular Redundancy - All Software

LIST OF SYMBOLS AND ABBREVIATIONS (Cont'd)

T_r	Recovery time $T_v + T_w$
T_R	Time to accomplish recovery procedure
T_Δ	Total recovery time $T_u + T_v + T_w$
T_τ	Interarrival time between faults
T_u	Time from fault occurrence to detection
T_v	Time from fault detection to diagnosis
T_w	Time from fault diagnosis to recovery
U	Uniform random deviate between 0 and 1
u_i	Detectability with i operating computers
v_i	Diagnostibility with i operating computers
W	Set of working machines
w_i	Recoverability with i operating computers
α	Fault detection rate for exponential portion of the uniform-exponential approximation to the fault detection time density
γ	Rate parameter for the duration of a transient fault
δ	Fault detection rate for the exponential approximation to the fault detection time density
κ	Ratio of transient to permanent fault rates τ/λ
λ	Permanent fault occurrence rate
σ_ℓ	Leaky transient plus permanent fault rate $\lambda + \ell_T\tau$ in TMR
σ_i	Permanent plus leaky transient rate with i operating computers $\lambda + \ell_i\tau$
σ_t	$\lambda + \tau$
σ_u	Uncovered transient plus permanent fault rate $\lambda + (1-c_T)\tau$ in TMR
τ	Transient fault occurrence rate
τ_ℓ	Non-leaky transient occurrence rate $(1-\ell_T)\tau$

LIST OF SYMBOLS AND ABBREVIATIONS (Cont'd)

ℓ_i	Transient leakage with i operating computers
ℓ_t	Transient leakage in TMR

1.0 SUMMARY AND INTRODUCTION

1.1 OBJECTIVE, ACCOMPLISHMENTS, AND CONCLUSIONS

1.1.1 Objective

The objective of this study was to provide concepts and engineering data from which a highly-reliable, fault-tolerant, reconfigurable computer system (RCS) for aircraft applications could be designed. For the purposes of this study, an RCS is defined to be a redundant configuration of off-the-shelf avionics computers which achieves fault-tolerance through use of a variety of recovery techniques. A principal study goal was the development and application of reliability and fault-tolerance assessment techniques. Particular emphasis was placed on the needs of an all-digital, fly-by-wire control system appropriate for a passenger-carrying airplane.

1.1.2 Accomplishments

The accomplishments of Contract NAS1-12793 are summarized in the following five-item list.

1. A complementary analytic-simulative technique (CAST) for calculation of predicted failure probabilities of multicomputer systems was evolved.
2. Measures of fault-tolerance applicable to general fault-tolerant computer systems were defined.
3. CAST was applied to 39 example computer system configurations to provide insight into the important aspects of these configurations, as well as demonstrate the efficacy of the approach.
4. A set of customer-provided reliability-enhancement techniques (RETs) was expanded and their individual effectiveness was evaluated.
5. A set of control laws for a digital fly-by-wire flight control system was translated into flow charts and computer sizing and timing for these were estimated (see Appendix A).

1.1.3 Conclusions

The conclusions reported below were obtained by use of CAST. They are based on a ten-hour flight and failure rates thought to be applicable to the off-the-shelf avionics computers studied. The reconfigurable computer systems were assumed to be composed of as many as five machines.

As shown in Figure 1.1-1, the greatest improvement in system survivability is obtained by increased redundancy. Each increment of redundancy decreases the 10-hour failure probability by approximately two orders of magnitude. The greatest failure probability decrease occurs when changing from triplex to quadruplex, e.g., a 200-fold improvement. Increasing redundancy also increases cost in terms of power, weight, and volume not only due to the added units but due also to the increased complexity of intercommunications modules, external electronics modules, and bus switches.

Increasing redundancy has diminishing returns if there are errors in permanent-recovery algorithm design. This error penalty becomes more severe with added redundancy. Using simpler recovery algorithms, i.e., those involving less RCS adaptivity, is a possible way of ensuring error-free recovery. However, the increase in failure probability for air-transport-type missions due to decreased adaptivity (e.g., not adapting the system down to one computer) is less than that caused by decreased redundancy or recoverability.

Since redundancy has such a large effect on failure probability, external hardware should have an equivalent redundancy to prevent external failures from depressing the overall survivability.

The techniques reported here devote much attention to the modeling of transient faults. The results show that a knowledge of the transient environment results in effective transient recovery features. Underestimating transient duration results in many transients being recorded as permanent, while overestimating transient duration leaves the system unduly vulnerable to further faults.

Finally, subject to the qualifications and assumptions described in the first paragraph of this subsection, configuration assessment has shown that hardware-aided software configurations provide a lower probability of failure than mostly-hardware or mostly-software configurations.

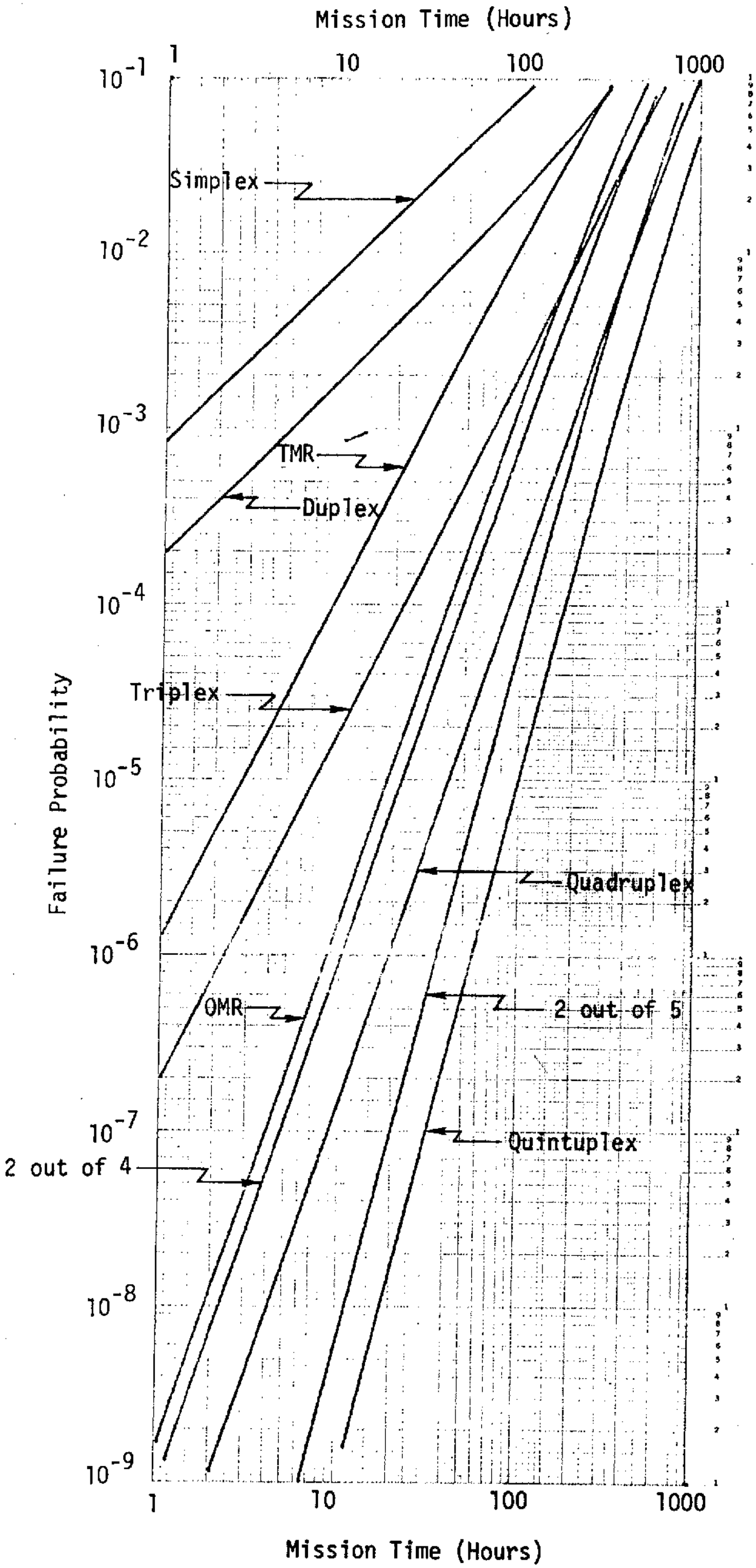


FIGURE 1.1-1

EFFECTS OF RCS REDUNDANCY AND ADAPTABILITY ON FAILURE PROBABILITY

1.2 INTRODUCTION

The configuration types to which CAST is applicable are symmetrical configurations of five or fewer, synchronized computers. The term symmetrical is used here to indicate that no one of the computers is used in a supervisory or executive mode. Each of the computers executes the same program in synchronism with the other machines. The synchronism may be "loose," or tight," depending on the mechanization of the configuration, i.e., the configuration may be one in which the fault-tolerance functions are implemented largely in software, they may be implemented in a software-hardware combination; or they may be implemented mostly in hardware. For all of these, the mechanisms for: the obtaining of input data from the sensors; the error-detection process; and the supplying of outputs to the actuators are considered to be part of the configuration. Consideration of software reliability was not considered to be within the purview of this study.

The architecture of fault-tolerant computing systems is heavily influenced by the key requirements of reliability, maintenance intervals, time for fault recovery, structure of the computations to be performed, and cost or maximum allocation of power, weight, and volume. The avionics application of this study is characterized by the following salient attributes:

1. Extreme Reliability Requirements, including "fail-safe" capability - lives are endangered upon failure.
2. Short Inter-Maintenance Interval - flights seldom exceed 10 hours.
3. Short Fault-Recovery Time - on the order of milliseconds to prevent degradation of control functions.
4. Moderate Computational Requirements - real-time control, well within the capacity of candidate machines.
5. Ample Power, Weight and Volume Allocations - a number of redundant computers may be employed.

To meet the reliability requirements of the avionics application it is necessary to attain a very high value of "coverage" in the computer design. It has been shown, that coverage, defined as the conditional probability, given that a fault occurs, that the fault is properly detected and the subsequent "recovery" is successful, is the most sensitive parameter affecting the reliability of a fault-tolerant digital system. With imperfect coverage, addition of redundant units gives little increase in reliability. For the aircraft application, coverage must closely approach unity in order to meet the stringent reliability requirements.

As a consequence of the high coverage requirement, a preferred approach to fault-tolerant computer configurations is massive redundancy. That is, performing the same computations with several computers and comparing their outputs in order to provide nearly perfect fault detection and isolation. When this approach is coupled with a sound recovery algorithm, high coverage is assured. This approach has obvious cost advantages if off-the-shelf computers, with minimal internal modifications and external supporting hardware, can be utilized. Not only can development costs be saved, but also support software and test procedures can be procured with the computers.

1.3 SYSTEM ORGANIZATION CONCEPTS

During the RCS study general models and specific examples of fault-tolerant computer configurations which are appropriate for implementation using "whole" computer massive redundance were formulated. These models are sufficiently general to serve as the basis for discussion of various redundancy options and reliability enhancement techniques. The more promising options for each configuration were modeled analytically and simulated to determine their effectiveness.

Three general categories of configurations utilizing "whole" computer redundancy were formulated. The first, termed the mostly-software approach, utilizes software for fault detection, voting, recovery and synchronization. External hardware is held to a minimum. The second, the hardware-aided software approach, shares fault detection and recovery between software and external hardware which supplies fault detection and voting. The third, the mostly-hardware approach, utilizes hardware to perform fault detection and recovery with the goal of minimizing the amount of special software required for fault-tolerance. These categories of configurations were examined in detail during the study.

1.3.1 "Mostly" - Software Configurations

The salient feature of "mostly" software (MSW) configurations is that external hardware is held to a minimum. Comparison of outputs for fault-detection and isolation is carried out by software. The interconnecting and synchronization techniques, as well as techniques for fault detection and recovery are quite similar for configurations of three or more computers. And thus a general model is presented which is inclusive of systems with three, four and five computers.

The simplest fault response is to ignore transients and for the agreeing machines to ignore the subsequent outputs of the machines which disagree. However, since transient correction is essential in meeting the stringent reliability requirements of the avionics application, it was necessary to examine more sophisticated recovery algorithms.

The recovery algorithms must respond to the following three fault conditions.

1. Permanent Fault - In the case of a permanent fault a transient recovery attempt will be unsuccessful and it is necessary to recognize this condition, typically by repeated disagreements, and terminate attempts at transient recovery. The subsequent outputs of the machine are ignored.

2. Transient Fault -- Program Undamaged - There exist a set of transient faults which can be corrected by one of two simple procedures. One of these recovery techniques, using segmented programs, is designated "rollahead". The second of these recovery techniques is called "rollback."

3. Transient Faults -- Program Damaged - Transient faults which result in damage to instructions or constants stored in memory, cannot be corrected by rollback, restart, or rollahead techniques. Correction of these fault-effects requires reloading memory, a process which results in a much longer recovery time.

To effect transient recovery* a mechanism must exist for transferring correct information to the memory of the damaged computer. A salient feature of the mostly-software configuration is that there is no hardware mechanism by which agreeing computers can take control of the disagreeing machine to force rollahead, update memories, etc. Thus even faulty computers must have a limited degree of autonomy and, in order to provide transient recovery, it was assumed that transient-damaged computers must maintain a small interrupt handling routine in order to correct this class of faults.

If memory protection (addressing interlocks) and NDRO technology is employed, a majority of transient faults will be rollahead or rollback correctable. However, unless ROM is employed for instructions and constants, there remains a probability of transients which require memory copy techniques for correction. Thus rollahead techniques should be backed up with the capability of copying memory contents to provide adequate coverage. A typical hybrid transient correction approach would be (1) attempt rollahead, then if a disagreement recurs, (2) attempt memory copy, and if fault still persists, (3) consider the computer to contain a permanent fault.

The I/O structure of the mostly-software configurations must provide:

1. Communication with sensors and activators;
2. Masking of faulty computers;
3. Precisely timed events from commands from computers which may be unsynchronized by a number of microseconds;
4. Redundancy and single-point-failure protection within the I/O structure.

Avionics systems typically contain sensors and actuators at widely separated locations. The recent trend has been to employ highly multiplexed I/O on order to reduce weight and complexity associated with cabling. Thus bus models were assumed in the analysis of I/O structures. It was assumed that two or more redundant busses are employed with a number of redundant interfaces. Two types of bus structure were considered.

*Transient recovery is defined as a recovery effected by the subsystem such that the number of properly operating computers and their identities before the fault occurrence and following the recovery are the same.

The first utilizes a bus dedicated to each of the computers with synchronization and voting performed in the peripheral units. The second treats the redundant bus structure as an integral unit in which voting, synchronization, and bus redundancy management is carried out by a special bus controller. In this case, individual busses and peripheral devices are not dedicated to any specific computer.

1.3.2 Hardware - Aided Software Configurations (HASW)

Hardware-aided configurations are characterized by the use of external hardware for fault-detection and synchronization. The goals of this approach are to (1) increase speed of computation and simplify software by performing the task of comparing state vectors and outputs in hardware, and (2) to allow the use of off-the-shelf computers with minimal I/O facilities.

The set of N computers is connected to a set of I/O busses through a special External Hardware Interface. This interface may be a single, massively-redundant structure or a set of identical modules dedicated to either individual computers or busses. The non-redundant building-block element of the external hardware interface is designated the External Electronics Module (EEM). The EEM accepts and buffers outputs from the computers, synchronizes the machines, provides voting for outputs, provides for inter-computer communications, and buffers returning inputs.

In order to effect transient recovery, a communication path must be established such that the agreeing computers can enter data into the memory of the faulty machine and command its restart. An adaptive voting capability is utilized within the EEM to allow this intercommunication. Transient recovery algorithms are similar to those used in the MSW configurations.

System failure occurs in the HASW configurations when all but two computers have failed and one of the remaining computers suffers either an uncorrectable transient or a permanent failure. When two computers remain functional, this condition is designated the Residual Duplex Configuration. Techniques for recovery from failures in the residual duplex configuration and continuing computation with a single simplex computer were developed during the study.

1.3.3 Mostly-Hardware Configurations

Mostly-hardware configurations are structured in such a way as to minimize the amount of software required to support fault detection and recovery. Table 1.3-I indicates the additional supporting software functions employed in software, hardware-aided and mostly-hardware configurations.

	Mostly-Software	Hardware-Aided Software	Mostly-Hardware
Fault Detection by Comparison	X		
Synchronization	X		
Transient Recovery	X	X	
Recording and Masking Permanently Faulty Modules	X	X	

TABLE 1.3-I SOFTWARE OVERHEAD OF FAULT-TOLERANT CONFIGURATIONS

The special software features associated with hardware-aided configurations are:

1. Rollback/Rollahead structured programs.
2. Identifying recurring faults and the decision to employ rollahead, memory copy, or classify a computer as permanently faulty.
3. Control of data transfers for rollahead and memory copy.
4. Disabling (fault response) faulty machines.
5. Diagnostic programs for recovery when only two computers remain functional.
6. A "warm" restart capability. A restart point at which computation can be resumed with a minimum of variables required for initialization. (Employed to minimize downtime for transfer of variables at the end of a memory copy.)

Mostly-hardware configurations perform the functions associated with the previously discussed hardware-aided software configurations along with implementing one or more of the above functions in hardware.

The principal difference between the mostly hardware and hardware-aided software configurations is that in the former the system state information and recovery decision mechanism resides in a central "hard core".

1.4 EXECUTIVE STRUCTURE

Four design goals were established for the executive. These goals specify general guidelines for the executive design as well as indicating a particular application in which the executive could be used. These goals are to design an executive which:

1. Can be readily adapted as an executive model for all RCS configurations under consideration.
2. Is general enough to support any reasonably foreseeable avionics application.
3. Makes clearly visible all the features required to support a digital flight control application.
4. Makes available the necessary parameters for configuration evaluations.

Since this study is directed toward multicomputer systems rather than encompassing multiprocessors, a single executive can be designed for use in each of the computers of the configuration. Thus, the first design goal ensures that the executive which is designed can be used in all computers of all configurations being considered, adapted as required by the configuration.

The computational environment imposed by air transport applications is such that the majority of computations must be performed periodically, although the computations performed will vary with the phase of the flight and the mode(s) used. Thus the computational requirement imposed by the avionics environments in which the computer systems being considered will operate involves primarily periodic, cyclical tasks of varying complexity, rate, and duration. The processing of occasional aperiodic tasks is also required. The executive has been designed to meet both of these requirements and thus be generally applicable to all avionics applications.

The executive skeleton consists of four distinct modules, each providing one of the four basic facilities required in an executive for an avionics computer. The four modules are the scheduler, the input-output driver, the interrupt processor, and the machine error handler.

The choice of a scheduling mechanism for an executive is the single most important decision in the design. The selection of the scheduling mechanism affects other modules in various degrees. For the avionics application, there exists a complete spectrum of executive scheduling mechanisms ranging from totally synchronous to constrained asynchronous.

The scheduling mechanisms considered can be differentiated by the following three characteristics:

1. Fixed vs variable processing time intervals;
2. Fixed vs variable task execution order;
3. Polled vs interrupt-driven aperiodic event registration.

The synchronous executive, while limited in terms of flexibility and growth, is conceptually very simple and its behavior is completely predictable. The synchronous mechanism utilizes fixed time intervals, fixed execution order, and polled aperiodic event registration. The constrained-asynchronous mechanism is the most flexible scheduling mechanism usable for an avionics application. The constrained-asynchronous executive schedules tasks on a demand basis. It thus provides a more flexible structure which permits growth to be achieved more easily. This mechanism utilizes variable processing intervals, variable task execution order, and interrupt registration of aperiodic events (even during periodic processing). There are a number of intermediate designs which utilize various combinations of the above approaches.

Thus it can be seen that the scheduling mechanisms considered range from the synchronous in which everything is fixed to the constrained asynchronous where everything is variable. The synchronous mechanism is the easiest to verify because everything is fixed. As more asynchronism is introduced, verification becomes more and more difficult because of more and more variability.

The asynchronous mechanism, in which almost everything is variable, can never be totally verified because the number of combinations of events is very large. All that can be done is to test all branches in a reasonable number of ways.

The choice of an executive scheduling mechanism is made on the basis of the environment, the machine capabilities, and the applications programs requirements. Once the choice of an executive scheduling mechanism is made, the other portions of the executive can be considered. In the HASW configurations this information and control mechanism is distributed and replicated within the software of the individual computers. The tradeoff between the two implementation types is largely a matter of cost.

Implementation cost in the mostly hardware case includes not only augmentation of the EEM units, but also a mechanism for protecting against and correcting transient errors in the augmented EEMs. A process of voting on all internal states (NMR synchronization) is required as well as a well-defined AEEM restart in case of information loss. To protect against transients it is advisable that the AEEM control states be maintained in non-volatile storage. Thus the augmented EEM is considerably more complex than the HASW EEM without augmentation.

1.5 MEASURES OF FAULT-TOLERANCE

Reliability theory defines the reliability of a system as the probability of correct operation up to the "mission time", T, given that the system was operating correctly at the mission starting time. The work on measures of fault-tolerance applicable to an RCS is based on the fact that computer systems are a special case among all physical systems because in their case "correct operation" means the correct execution of a set of programs, rather than the continued functioning of a set of components of the system.

The following four criteria form an operational definition of "correct execution of a set of programs:"

1. The program and their data are not altered or halted by faults;
2. The results of operations do not contain fault-caused errors;
3. The execution time of each program does not exceed a specified limit;
4. The storage capacity that is available for each program remains above a specified minimum value.

There are three distinct quantitative measures that can be applied to measure the fault-tolerance of a computer system. They are:

1. The Discrete Fault Tolerance d
2. The Reliability R(t)
3. The Survivability S(t)

The discrete fault tolerance (DFT) d is a deterministic measure that specifies how many faults of a given class can be tolerated by a computer system or by a module of the system. The remaining two measures - reliability R(t) and survivability S(t) - are probabilistic measures that predict the probability of the system continuing its correct operation over a specified time interval.

DFT is defined as the ability of a Module Set M to operate correctly for at least d faults within the Module Set. It is important to note that

DFT is not a function of time, i.e., the probability of continued correct operation is stated to be unity as long as not more than d faults from the fault set occur within the module set M.

The reliability R(t) also refers to a set F of permanent faults that can occur in the hardware module set M. It is defined as the probability that the set M will not experience a disabling hardware failure during a specified "mission time" interval $0 \leq t \leq T$.

It is known from experience that computer systems are also subject to transient faults, which can terminate the correct execution of a set of programs without causing a disabling hardware failure in the module set M_i. Our goal was to incorporate the survival probability with respect to the occurrence of transient faults into one probabilistic measure of fault-tolerance that also contains the reliability R(t). This measure is called the survivability S(t) of the module set M_i.

Previous work has established that three fault-tolerance activities must be successfully executed before the system returns to its functional state after a fault event. It was found convenient to partition the probability of successful system response to a fault into three components:

1. Detectability, denoted by u and defined as the probability that fault is detected, given that it occurs;
2. Diagnostibility, denoted by v and defined as the probability that the faulty module is correctly identified, given that the fault has been detected;
3. Recoverability, denoted by w and defined as the probability that the operational state is successfully re-established, given that the fault has been located.

Thus the survivability S(t) is seen to be an overall measure of fault-tolerance of a computer system, while detectability, diagnostibility, and recoverability give detailed insight into the system behavior and can be used for more precise specification of the fault-tolerance desired in a computer system.

1.6 ANALYTIC MODELING

The analytic modeling effort was directed toward the specific inclusion of transient faults and the inclusion by use of parameters, of the software structure and the system failure criteria.

The problem was approached by preparing state diagrams representing the fault/recovery status of the system. Transient faults were assumed to arrive at an average rate τ which is constant over the life of the system. Similarly, based on physical reasoning and mathematical tractability, an exponential density function was chosen to represent transient duration.

The three-stage transient recovery sequence, consisting of detection, recovery-start delay, and recovery, was formulated and the necessary parameters defined. The concept of imperfect detection was formalized using a probability density function of the detection time. An important concept, that of transient leakage, was formulated and defined. Transient leakage, ℓ_T, is the probability that a transient fault is interpreted as a permanent.

The modeling of specific computer systems was begun by considering an enhanced TMR configuration. An enhanced TMR systems possesses the capability of recovering from a transient fault. Following the obtaining of the results for the enhanced TMR system, the work was extended to N computers, first considering the case involving a linear degradation of the system (i.e. 5 computers to 4 computers to 3, etc.), and then formulating the more general case where an N-computer system can go directly to the system failure state. A recursive expression for the survivability of an N-computer configuration was developed and then, by mathematical induction, it was shown that the survivability can be expressed as a linear combination of exponential functions. An iterative expression was then developed for determining the coefficients for the linear combination.

The final aspect of the analytic modeling was the formulation of a model using the Markov chain analysis method. By assuming that state transitions occur continuously, it was possible to develop a vector differential equation representing the system and obtain an expression for the state probabilities at time t. As was to be expected, the results agreed with those obtained earlier. However, they provide the basis for a set of simply-formulated computer programs which are useful in obtaining numerical results.

1.7 SIMULATION

The function of the simulator developed during this contract is to produce: 1) parameters for use in analytic models of RCS; 2) the fault-tolerance effectiveness of each of a wide variety of RCS configurations; and 3) the behavior of a configuration in various fault environments.

The general organization of the simulator was formulated so that the end-product would be versatile and flexible. An efficient simulation was developed by designing a "fault-driven" simulator, rather than one that simulates the continuous operation of the system. The simulator was written in FORTRAN IV and currently runs on a CDC-6600 computer.

The approach taken to the formulation of the simulator is similar to that utilized in the analytic modeling in that a state diagram is used to describe the programs requirements. A simplified state diagram is shown in Figure 1.7-1.

The simulator program is structured to simulate the detection of faults within a computer system and the computer system's successful/unsuccessful recovery actions taken in response to the detected faults. Each simulated mission is assigned a mission time. A simulation run consists of the repetitive continued simulation of a designated number of missions (each with the same mission time). As stated earlier the simulation is fault-driven. Nothing happens in the simulator until a fault occurs. This is very important in terms of simulator efficiency. The computer time spent in one run is roughly proportional to the number of faults and not to the simulated mission time.

A good measure of the detail included in a system simulation is the number of parameters that must be specified for each run. It can be seen from Table 1.7-I that the RCS simulator is very detailed. The outputs produced by the simulator are listed in Table 1.7-II. As can be seen from this table, outputs are provided to the user that give detailed insight into the system behavior.

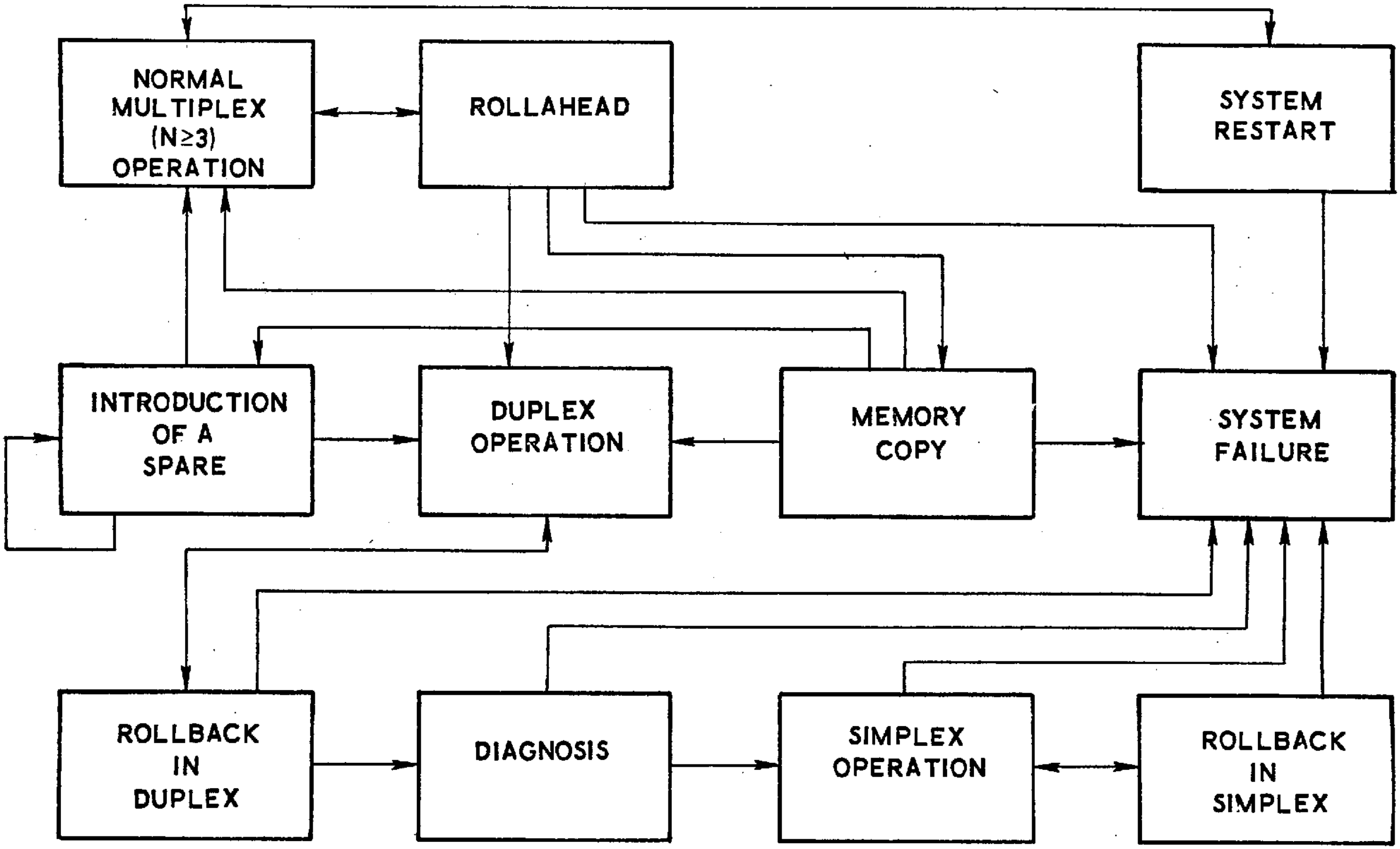


FIGURE 1.7-1 SIMULATOR STATE DIAGRAM

TABLE 1.7-I LIST OF RCS SIMULATOR INPUTS

NUMBER OF SIMULATED MISSIONS

MISSION DEPENDENT PARAMETER

- Mission Time

MACHINE DEPENDENT PARAMETERS

- Permanent Failure Rates
- BITE Detection Probability of a CPU Fault
- BITE Detection Probability of a Memory Fault
- Self-Test Program Efficiency
- Self-Test Program Duration

CONFIGURATION-DEPENDENT PARAMETERS

- Number of Computers
- Number of Spares
- Dedicated/Non-Dedicated EEMs (External Electronic Modules)
- Probability that an EEM Fault Hits the Bus
- Number of Non-Dedicated EEMs
- Dedicated/Non-Dedicated Busses
- Number of External Devices
- Coverage and Relative Failure Rate of each Device and of the Busses
- Applicable Recovery Algorithms
- Recovery Algorithm Characteristics
- Duration
- Unacceptable Recurrence Interval
- Maximum Number of Rollbacks
- Program Integrity
- Memory-Copy Efficacy

SCHEDULING PARAMETERS

- Iteration Period
- Time Between Comparisons
- Major and Minor Cycle Durations
- Asynchronous/Synchronous Mechanism

ENVIRONMENT DEPENDENT PARAMETERS

- Transient Failure Rates
- Transient Failure Duration

TABLE 1.7-II RCS SIMULATOR OUTPUTS

NUMBER OF SYSTEM FAILURES

CAUSES OF SYSTEM FAILURES

- Excessive-Length Recovery
- Non-Isolated Faults
- Simplex Mode Failures
- EEM Failures
- I/O and Bus Failures

NUMBER OF SWITCHES TO - Quadruplex
- Triplex
- Duplex
- Simplex

TRANSIENT COVERAGES IN MULTIPLEX, DUPLEX, SIMPLEX

DIAGNOSTIBILITY IN DUPLEX

PROPORTION OF CATASTROPHIC FAULTS

NUMBER OF MISSED ITERATIONS

1.8 COMBINED ANALYTIC-SIMULATIVE TECHNIQUE

The analytic modeling approach described in Section 1.6 and the simulation technique described in Section 1.7 each has its strengths and limitations. However when these two system evaluation approaches are combined, and supplemented by some engineering analysis, a very powerful technique results.

This Complementary Analytic-Simulative Technique (CAST) evolved as it became evident that neither analysis nor simulation alone could satisfy all the RCS evaluation requirements. Analytic modeling provides flexibility and rapid, economical data-generation. However the solutions for some configurations are very cumbersome and in certain cases the mathematical model formulated is intractable. Simulation permits computer system details to be included easily, but data generation is slow and expensive. CAST permits the user to obtain the best features of both analytic modeling and simulation.

The RCS engineering analysis is performed to provide six categories of information to the analytic modeling and the simulation. These information categories are:

1. Configuration Particulars
2. Fault Environment
3. System Failure Criteria
4. Software Structure
5. Recovery Features
6. Test Features

The results produced by the simulator are:

1. Permanent-fault coverage
2. Transient-fault coverage
3. Detectability
4. Diagnostibility
5. Recoverability

1.9 RECOMMENDATIONS

Based on the work summarized here and repotted in detail in Sections 2 through 9, the following two actions are recommended.

1. Apply CAST to a specific configuration of interest. The combined analytic-simulative technique should be applied to a specific aircraft or spacecraft application that requires a highly reliable computing capability. Preferably this would be an application that has progressed far enough in the preliminary design stage so that the software structure has been formulated, application program size and execution times have been estimated, subsystem failure criteria have been postulated, and specific sets of sensors and actuators have been selected. Application of CAST to a specific system will illustrate its utility.

2. Introduce additional complexity into the analytic model in order to reduce the cost of the necessary simulation runs. The complexities to be considered are:

   a. Spare computers;

   b. Dedicated busses;

   c. Recovery-procedures complexity;

   d. Explicit failure criteria;

   e. Software structure; and

   f. Burst-fault environment.

THIS PAGE INTENTIONALLY LEFT BLANK

2.0 SYSTEM ORGANIZATION CONCEPTS

The architecture of fault-tolerant computing systems is heavily influenced by several key requirements of their application. Among these are: 1) reliability, 2) maintenance intervals, 3) time for fault recovery, 4) structure of the computations to be performed, and 5) cost or maximum allocation of power, weight, and volume. The avionics application of this study is characterized by the following salient attributes:

1. Extreme Reliability Requirements, including "fail-safe" capability - lives are endangered upon failure.
2. Short Inter-Maintenance Interval - flights seldom exceed 10 hours.
3. Short Fault Recovery Time - on the order of milliseconds to prevent degradation of control functions.
4. Moderate Computational Requirements - real-time control, well within the capacity of candidate machines.
5. Ample Power, Weight and Volume Allocations - a number of redundant computers may be employed.

In order to meet the reliability requirements of the avionics application it is necessary to attain a very high value of "coverage" in the computer design. It has been shown, both by analytic means and by simulation, that the most sensitive parameter affecting the reliability of a fault-tolerant digital system is "coverage", defined as the conditional probability, given that a fault occurs, that the fault is properly detected and the subsequent "recovery" is successful. [BOUR 69]. In many cases it can be shown that increasing coverage by 1% can improve the reliability of a fault-tolerant computer to a greater extent than using an additional spare computer. Conversely, with imperfect coverage, addition of redundant (spare units) gives little increase in reliability For the aircraft application, coverage must closely approach unity in order to meet the stringent reliability requirements.

As a consequence of the high coverage requirement, a preferred approach to fault-tolerant computer configurations is massive redundancy. That is, operating several computers to perform the same computations, and comparing their outputs in order to provide nearly perfect fault detection and isolation. When this approach is coupled with a sound recovery algorithm, high coverage is assured. This approach has obvious cost advantages if off-the-shelf computers, with minimal internal modifications and external supporting hardware, can be utilized. Not only can development costs be saved, but also support software and test procedures can be procured with the computers.

The purpose of this section is to provide general models and specific examples of fault-tolerant computer configurations which are appropriate for implementation using "whole" computer massive redundancy. The models are intended to be sufficiently general to serve as a basis for discussion of redundancy options and reliability enhancement techniques. The more promising options for each configuration will be modeled analytically and through simulation to determine their effectiveness.

Under the constraints of 1) little or no modifications to the off-the-shelf computer elements, and 2) application of redundancy at the "whole" computer level, there are three key interfaces to the digital computer for its implementation into a redundant configuration. These are: 1) I/O interfaces including interrupts and AGE, 2) software, and 3) synchronization (which, though not an explicit physical interface, bears heavily upon the design of the configuration).

There are three general categories of configurations utilizing "whole" computer redundancy. The mostly-software approach utilizes software for fault detection, voting, recovery and synchronization. External hardware is held to a minimum. The hardware/software approach shares fault detection and recovery between software and external hardware which supplies fault detection and voting. The mostly-hardware approach utilizes hardware to perform fault detection and recovery with the goal of minimizing the amount of special software required for fault-tolerance. The following sub-sections are directed toward an examination of these categories of configurations.

2.1 "MOSTLY"-SOFTWARE REDUNDANT CONFIGURATION (MSW)

The salient feature of mostly-software configurations is that external hardware is held to a minimum. Comparison of outputs for fault-detection and isolation is carried out by software. The interconnecting and synchronization techniques, as well as techniques for fault detection and recovery are quite similar for configurations of three or more computers. And thus a general model is presented which is inclusive of systems with three, four and five computers.

Mostly software configurations are characterized by 1) inter-computer communications utilized for fault-detection, transient recovery, and synchronization, and 2) a redundant I/O structure which can convey "correct" information to and from peripheral devices in the presence of computer or I/O faults. These two characteristics tend to define the MSW redundant configurations which are described below:

2.1.1 Internal Communications for Fault Detection and Transient Recovery

The internal cross-connections associated with software redundant configurations are shown in Figure 2.1-1. Each computer generates data and control outputs which are made available to the other N-1 machines. It is assumed that synchronization is carried out by software and that exactly-synchronized clocks for data transfers cannot be guaranteed. (The candidate computers employ asynchronous memory cycles, an attribute which does not allow synchronizing clocks without internal hardware modifications). The following is a description of the two sets of internal cross-connections.

1. Data Transfer Paths - It is necessary to transfer outputs, and state variables generated within program segments, between computers for checking and voting by software. The data paths employed can be implemented in one of several fashions, each of which can tolerate the lack of clock synchronization:

   a. Parallel (Held) - Each computer's output remains until sampled by other computers.

   b. Serial or Parallel (Transmitted) - Output is sent to latching registers within the receiving computers for synchronization.

   c. DMA transfer facilities between computers under control of the outputting modules. Software synchronization is only necessary for initiation (or completion) of a block transfer. Individual

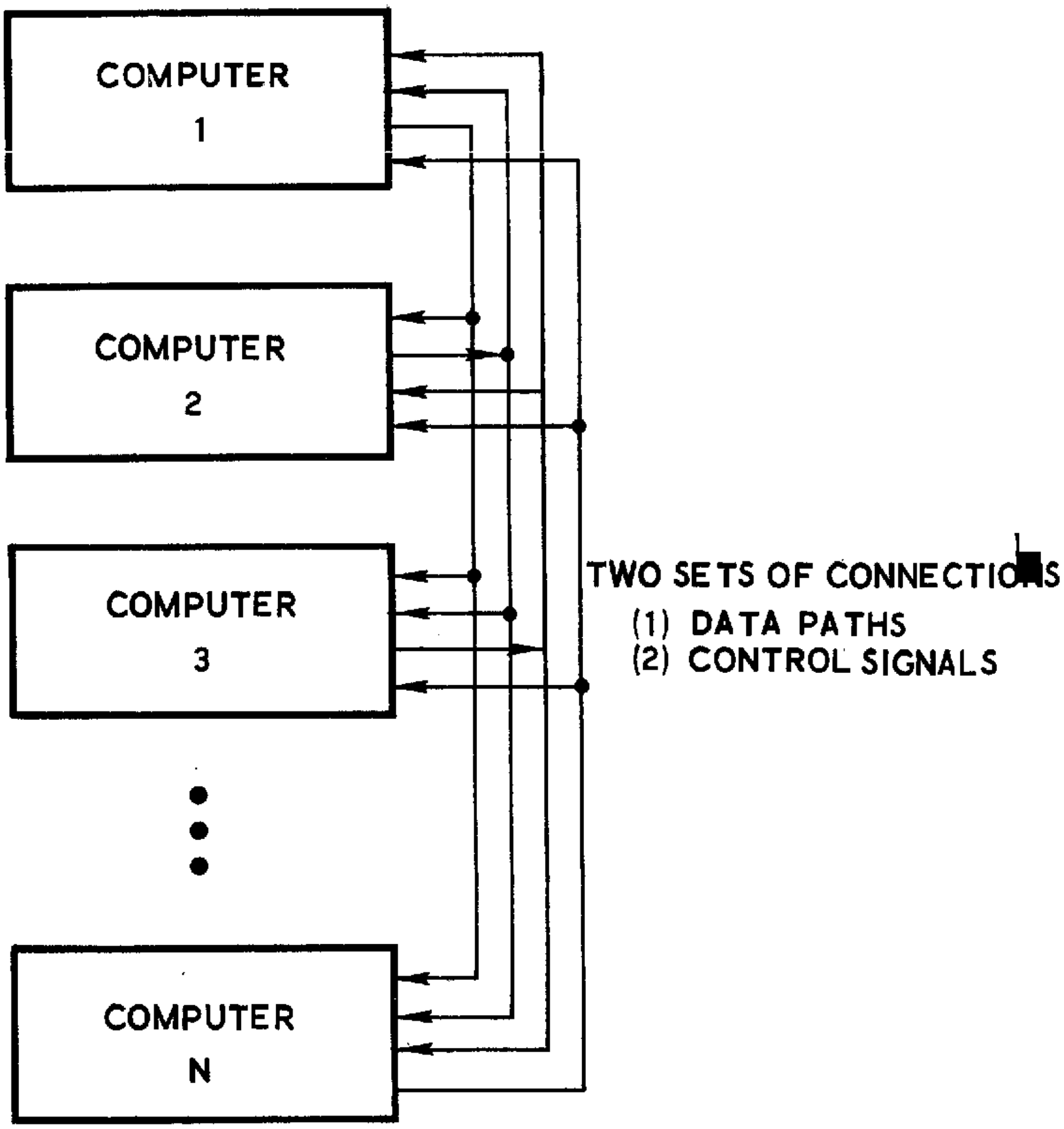


FIGURE 2.1-1 MSW INTERNAL CROSS-CONNECTIONS

word transfers are transparent to the software. (This approach is expensive in hardware but offers the more rapid comparison/voting process).

2. Control Signals - It is necessary for each computer to supply the other machines with control signals for synchronization and fault recovery. Examples of control signals are 1) output ready, 2) fault detected, etc.

An example of the comparison and synchronization process is given in the following paragraphs.

2.1.1.1 Voting and Synchronization

The following description of voting and synchronization is centered around the activities of properly functioning processors. It is assumed that there is a set of N processors, each programmed to perform an identical computation, loosely synchronized within a few instructions of each other. The properly functioning computers comprise the subset W and have stored internally the identity of the working machines W and that of the machines assumed to have failed F. ($N=W\cup F$). A vote $V_i(W,F)$ is performed in each of the properly functioning computers which provides: 1) a voted result V_r or an indication of indeterminable output (such as when all inputs disagree), and 2) an indication of one or more disagreeing units d_i. A number of voting algorithms are possible, e.g. NMR, hybrid, adaptive, etc..

A typical scenario of the voting and synchronization process is indicated in Figure 2.1-2. When any properly functioning computer reaches a point in the program where comparison of results is required, it transfers its values to the other computers and waits for one of two events:

1. The other working computers complete the transfer indicated by their Output Ready (OR) levels or,

2. A time-out overflow occurs.

In either case a vote is taken on the transferred information in all the properly functioning computers in the set W. If all the computers agree with the voted result, computations continue else a corrective action is taken. This approach corresponds to the fault exit in Figure 2.1-2.

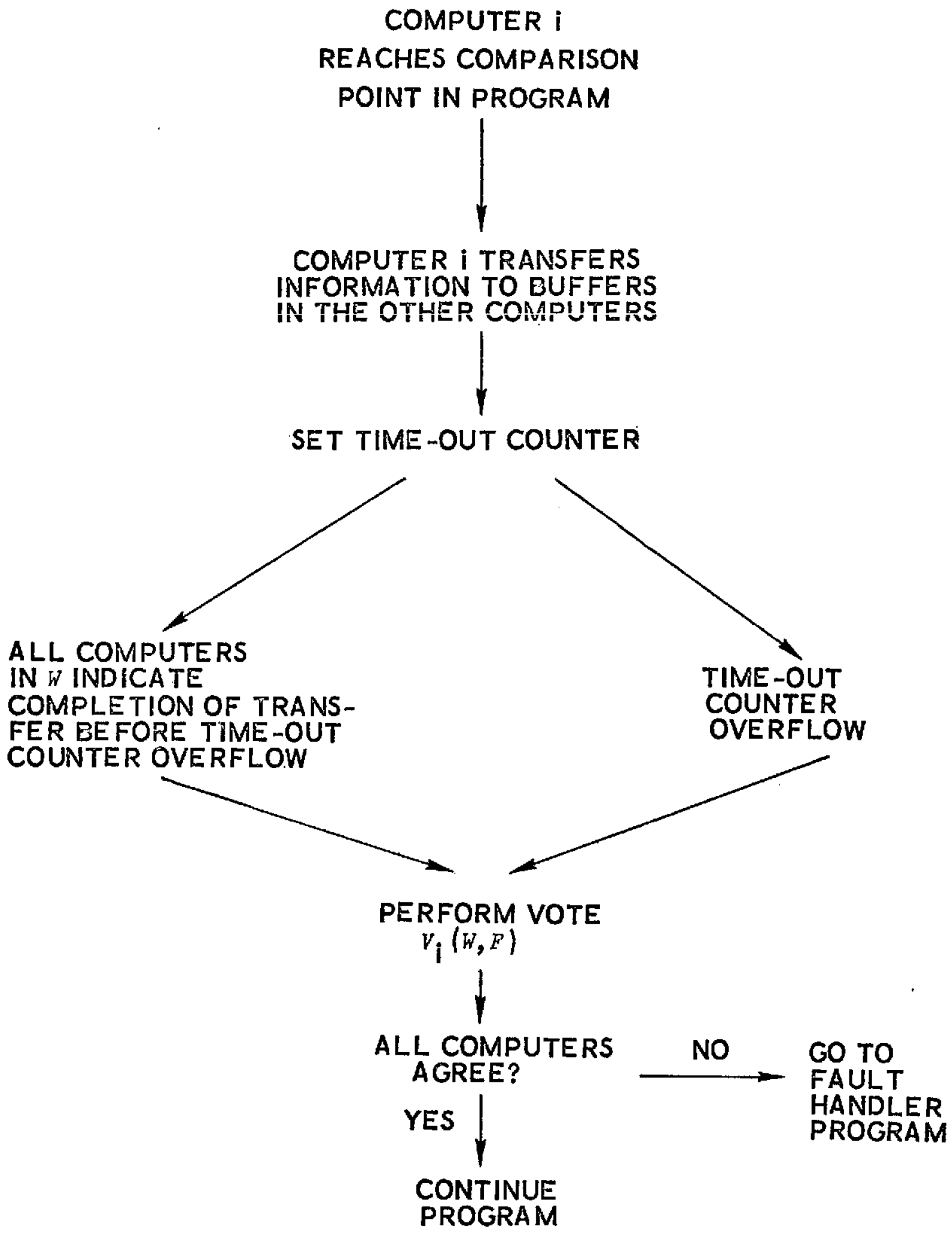

COMPUTER i REACHES COMPARISON POINT IN PROGRAM
COMPUTER i TRANSFERS INFORMATION TO BUFFERS IN THE OTHER COMPUTERS
SET TIME-OUT COUNTER
ALL COMPUTERS IN W INDICATE COMPLETION OF TRANSFER BEFORE TIME-OUT COUNTER OVERFLOW
TIME-OUT COUNTER OVERFLOW
PERFORM VOTE $V_i(W,F)$
ALL COMPUTERS AGREE?
NO
GO TO FAULT HANDLER PROGRAM
YES
CONTINUE PROGRAM

FIGURE 2.1-2 SYNCHRONIZATION AND VOTING SCENARIO

The simplest fault response is to ignore transients and for the agreeing machines to delete the machines which disagree from the set W and to ignore their subsequent outputs. However, since transient correction is essential in meeting the stringent reliability requirements of the avionics application, it is necessary to examine more sophisticated recovery algorithms.

2.1.1.2 Transient Recovery Techniques

The condition which causes a computer in the set W to be in disagreement with the other machines is either 1) an erroneous computation, or 2) the disagreeing machine has gotten out of step with the others. It is useful to classify the causes of this condition into the following three categories and list the implications of each on the transient recovery process:

1. Permanent Fault - In the case of a permanent fault a transient recovery attempt will be unsuccessful and it is necessary to recognize this condition, typically by repeated disagreements, and terminate attempts at transient recovery. The machine is reclassified from the set W to the set F and its subsequent outputs are ignored.

2. Transient Fault -- Program Undamaged - There exist a set of transient faults which can be corrected by either one of two simple procedures, program rollahead and program rollback. For both of these procedures, the program is segmented and associated with each segment is a set of global variables designated the State Vector. The state vector contains necessary and sufficient input data to properly execute the associated program segment. Furthermore the state vector is not modified by its associated program segment (e.g. call by value) such that the program segment can be re-started. During correct computation of the N^{th} program segment, the state variables for the next $(N+1^{th})$ segment are generated (ROHR 73).

   The first technique, designated "rollahead", takes advantage of one or more "correct" machines. When a fault occurs in a computer, the next state vector (including the location counter) of the "correct" machines is loaded into its memory at the end of the program segment in which the fault occurred. Since the corrected state vector corresponds to the variables which are necessary to start the next program segment, the faulty machine can be corrected without repeating the segment in which the fault occurred. The desired outputs are available from the correct machines.

The second of these recovery techniques is called "rollback". In this case, if the computers disagree upon output or comparison of information the current program segment is re-started. If the state vector and program have not been damaged, the program segment will be executed correctly after the restart.

While program rollahead takes place nearly instantaneously, program rollback results in a delay required to re-compute a transient-damaged program segment. However, since rollback does not require a transfer of information from "correct" computers, it can be utilized in duplex configurations where the "correct" computer cannot be immediately identified. Thus rollahead is the preferred approach to correction of this class of transient faults when three or more computers are functional. Rollback is required in the residual duplex condition where only two machines remain functional or when only one computer is working.

3. Transient Faults -- Program Damaged - Transient faults which result in damage to instructions or constants stored in memory, cannot be corrected by rollback, restart, or rollahead techniques. Correction of these faults requires reloading memory, a process which results in a much longer recovery time.

Memory address protection and NDRO memory are two RETs employed to prevent transients of type (3) above, as well as to reduce the total number of memory transients. The next section explores the implementation of transient recovery techniques in software redundant configurations.

2.1.1.3 Transient Recovery - Implementation

In order to effect transient recovery a mechanism must exist for transferring correct information to the memory of the damaged computer. A salient feature of the software configuration is that there is no hardware mechanism by which agreeing computers can take control of the disagreeing machine to force rollahead, update memories, etc. Thus even faulty computers must have a limited degree of autonomy and, in order to provide transient recovery, it is assumed that transient-damaged computers must maintain a small interrupt handling routine in order to correct this class of faults.

Two examples are given below of transient correction mechanisms employed in the software-redundant configuration. Each requires three principal actions:

1. The disagreeing computer must be notified that it is out of step. It can ascertain this locally by performing a test at the occurrence of each real-time interrupt (RTI), or can be notified via interrupts from the other computers in *W*.
2. The "good" computers must effect transfer of correction information.
3. The faulty computer must load this information and re-synchronize with the other computers.

It is assumed that each computer has capability of loading comparison values into dedicated buffers in the other computers independent of the program in those machines (e.g. DMA).

Program Rollahead

"Instantaneous" transient recovery can be achieved if segmented programs are employed and rollahead is implemented. At the end of each segment of program, the state variables (those global variables utilized by subsequent program segments) are exchanged and compared in the various computers. If a computer in *W* suffers a transient fault during the program segment, which does not damage instructions or constants, it can utilize the state vector from the other machines and continue with the next program segment. The address of the next segment must be included within the state vector to indicate the point at which the faulty machine should commence execution.

The principal problem of the rollahead implementation is notification of the faulty machine so that it can utilize the corrected state vector and start the next program segment in step with the other machines. Three cases exist:

1. If the machine has only data damage but is still in step with the other machines, the software vote $N_i(W,F)$ will indicate its disagreement and the program can automatically choose the state vector sent from another machine in W and continue.

2. If the machine is out of step and attempts a comparison before the other machines, this condition can be recognized and a wait initiated for access to the state vector from the other computers and a subsequent rollahead.

3. If the machine is out of step and does not perform a comparison with the others of state vectors, then it must be alerted to this fact in order to execute the rollahead. This can be done utilizing "fault detected" signals as interrupts in the following fashion.

   Each computer generates a "fault detected" (FD) signal which is received as an interrupt by the other computers. This interrupt is permanently masked (by all computers in W) from computers designated as failed (F).

   Prior to performing a comparison, each computer masks out the FD interrupts. If after completion of the transfer, one computer fails to respond with an output ready (OR) signal, the other computers send the FD signal, thus notifying the computer which is out of step. The validity of these interrupts can be verified by the interrupted computer by checking for several OR signals.

Thus the idea of this approach is:

a. A computer not in or near the process of comparison enables FD interrupts from the other machines.

b. If the other machines perform a comparison without activity from one or more machines their FD interrupts are raised.

c. Erroneous FD interrupts are identified and masked by verification of output-ready signals.

A machine which is alerted as to being out of step can utilize the state vector from the other computers to perform a rollahead.

Memory Copy

At the occurrence of the RTI, the computers in set W, check the results sent for comparison during the last minor cycle. If one of the computers has gotten out of step due to a transient, it will then recognize that its comparison data differs from that of the other good machines. Under this condition the faulty computer enters the UPDATE mode. (It is important that ROM or memory-protect hardware be employed to preserve the integrity of this RTI-driven program).

Upon recognizing a computer from the set W which is in disagreement, the remaining (agreeing) computers transfer programs, constants, and that variable data necessary to restart computations to the disagreeing computer. Two characteristics of this transfer are listed below:

1. Since it is necessary to maintain normal computations, the transfer of programs and fixed constants takes place at a low duty cycle and thus recovery takes on the order of seconds.

2. Computations must be stopped during the transfer of that variable information required to restart the disagreeing computer. The UPDATE program is flagged by receipt of this variable data and it resumes normal computation at the next RTI if the transient was corrected.

If the disagreeing computer continues to produce erroneous results, after the transient recovery attempt the remaining computers in set W, reclassify the machine as permanently faulty and ignore its outputs.

2.1.1.4 Utilization of Transient Recovery Techniques

If memory protection (addressing interlocks) and NDRO technology is employed, a majority of transient faults will be rollahead or rollback correctable. However, unless ROM is employed for instructions and constants, transients may cause memory-contents damage which require memory copy techniques for correction. Thus rollahead techniques should be backed up with the capability of copying memory contents to provide adequate coverage. A typical hybrid transient correction approach would be 1) attempt rollahead, then if a disagreement recurs, 2) attempt memory copy, and if fault still persists, 3) consider the computer to contain a permanent fault, reclassify from W to F and ignore further outputs from that machine in comparisons.

The previous discussion was related to internal cross-connections associated with the software redundant configurations. It is these connections which are utilized for comparison of variables for fault detection, voting to establish the correct value of information in the various machines, and data transfer for transient correction. The next sub-section deals with various redundant structures for I/O.

2.1.2 Redundant I/O Structures for Communications with Fault Masking

Beside the primary task of providing communication with peripheral units, the I/O structure of the software redundant configurations must:

1. Provide masking of faulty computers.
2. Provide precisely timed events from commands from computers which may be unsynchronized by a number of microseconds.
3. Provide redundancy and single point failure protection within the I/O structure.

Avionics systems typically contain sensors and actuators at widely separated locations. The recent trend has been to employ highly multiplexed I/O in order to reduce weight and complexity associated with cabling. Thus bus models will be employed in the following discussion of I/O structures. It is assumed that two or more redundant busses are employed with a number of redundant interfaces. Two types of bus structure will be discussed. The first utilizes a bus dedicated to each of the computers with synchronization and voting performed in the peripheral units. The second will treat the redundant bus structure as an integral unit in which voting, synchronization, and bus redundancy management is carried out by a special bus controller. In this case, individual busses and peripheral devices are not dedicated to any specific computer.

2.1.2.1 Dedicated Busses

Figure 2.1-3 shows the connections employed in an I/O configuration with dedicated busses. Each bus may be bidirectional or employ a separate set of return lines. Each bus is controlled by its associated computer and is only synchronized within a few instruction times of the other busses. Two types of I/O modules may be attached to the bus 1) dedicated and 2) non-dedicated.

2.1.2.1.1 Dedicated Sensors

A set of identical sensors may be dedicated one per each bus line and operate independently. This results in differing values being returned to the various computers. And thus it is necessary to exchange input values from a set of redundant sensors (using the interal cross-connections described in the previous sections) between the computers in W and to compute a common value for use in subsequent computations so that the machines will generate identical outputs and not appear faulty. This process of computing a common value for the various sensors is a critical RET in the utilization of sensor redundancy which must (1) exclude values from faulty machines (ϵF), (2) exclude inputs from sensors previously determined as faulty, and (3) use reasonableness checks, averaging, etc. to establish a "best" sensor value for common utilization.

The use of dedicated sensors has the advantage of simplicity, but also has several disadvantages:

1. The number of redundant sensors is constrained to the number of computers, which prevents optimum balance of redundancy for modules of differing reliability. This approach is not applicable for configurations of more than three computers due to the requirement of excess redundancy in sensors.

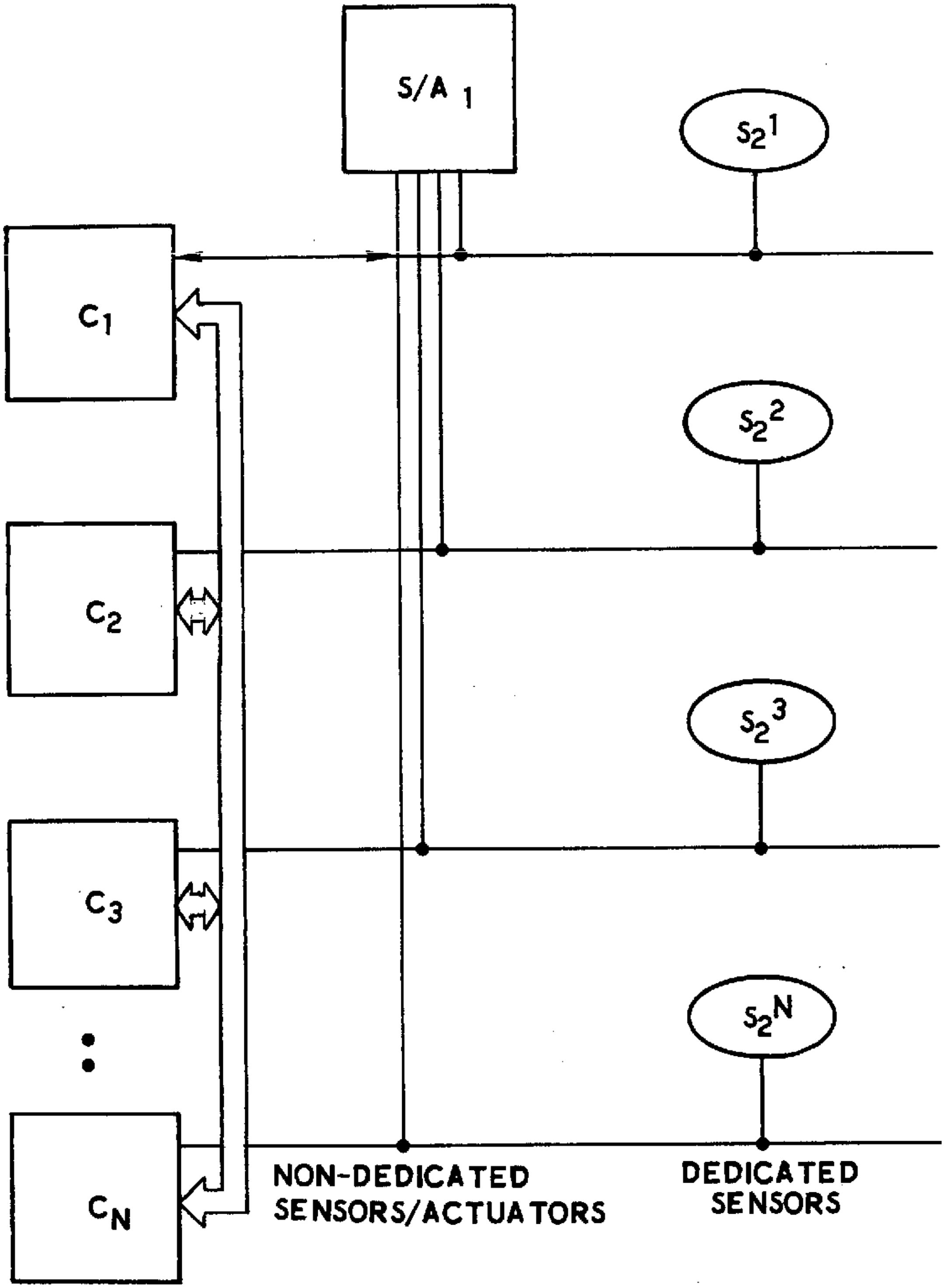


FIGURE 2.1-3 DEDICATED BUSSES

2. If one computer fails, its associated sensors are effectively disabled.

2.1.2.1.2 Non-Dedicated Sensors and Actuators

A non-dedicated sensor/actuator interface, provides communication between all the computers and the redundant sensors and actuators. The bus interface allows each of the computers to address any specific member of a redundant set of sensors or actuators and the returning information is coherent, i.e. the same number is returned to all computers. This differs from the case of dedicated I/O where each computer addresses a different peripheral device within a redundant set and receives slightly different information. Redundant sensor data is obtained by addressing several redundant devices in sequence.

Bus interfaces to non-dedicated sensors or actuators are moderately complex since they have the following requirements:

1. The interface receives identical commands and data from busses associated with properly functioning computers, and, most likely, incorrect outputs from failed computers.

2. The agreeing outputs are not precisely synchronized but will occur within a worst-case time interval Δt.

3. For commands which change the state of peripheral subsystems, a vote must be provided in the interface to mask out faulty commands. (A typical implementation is to respond only to two or more identical commands, occurring on different busses within the acceptance interval Δt).

4. Independent data streams must be supplied to each computer upon receipt of a set of identical input-request commands. In order to meet the timing requirements of a number of candidate machines, information must be returned immediately upon receipt of its request (i.e. each input transfer is synchronized to the associated computer). Thus voting may not be employed for information requests if the first computer to make an I/O request cannot wait for subsequent requests and voting.

5. The I/O interface should supply identical data to all machines making an information request within the acceptance interval Δt. This can be accomplished by synchronization with the RTI. (All status vectors and sensor measurements are latched and only allowed to change upon occurrence of the RTI; all programmed I/O is constrained to occur between these changes).

Non-dedicated modules have several advantages indicated below:

1. Individual actuators within redundant sets require a guarantee of correctness from the computer system before executing a command. Thus the I/O module voting capability is necessary.
2. A computer failure does not disable non-dedicated sensors and actuators.

Several disadvantages exist and are discussed in the following paragraphs.

1. Latching of information to guarantee identical sensor values to the various machines is not inconsistent with the real-time control application, but does represent a degree of added complexity.
2. Cross checking of input information must still be performed, or a faulty input from an I/O module will appear as a computer fault due to disagreeing results.
3. As indicated above, a non-dedicated I/O interface is a device of considerable complexity, but this complexity is not significantly greater than that required by replicating simplex bus interfaces as employed with dedicated devices.

2.1.2.2 Non-Dedicated Busses (Non-Dedicated Sensors)

A non-dedicated redundant bus structure can be treated as a self-contained unit for conveying information between the computers and peripheral devices. One or more busses carry identical information to and from the sensors and actuators, and individual sensors are accessed within redundant sets. To utilize redundancy, a set of identical sensors must be sampled sequentially and a selection (or voting) process performed in software. Individual bus lines are not dedicated to any specific computer but, information may be obtained from any of the computers by a process of switching or voting. A good first-order approximation is to treat the non-dedicated bus structure as a series term in the system reliability expressions (see Section 9).

Two options are possible for implementation of non-dedicated busses:

1. Switched Busses - Each of several busses can be connected to any of the computers, in standby or massive redundant configurations (see Figure 2.1-4). With standby redundancy, only one bus is utilized, and the remaining busses serve as spares. In the massive redundant case, the various busses are connected to different computers, allowing voting and error correction in the peripheral sensor and actuator interfaces.

2. Voted Outputs - Each bus output is derived as a vote of the various computer outputs. As with switched outputs above, the redundant busses can be employed in standby or massive redundant configurations.

Switched busses require limited amounts of hardware and can employ software synchronization. Thus switched-bus structures are discussed in this section which treats mostly-software computer configurations, being consistent with the requirement of minimum external hardware. Busses driven by voted outputs are more complex, requiring a degree of hardware synchronization. Redundant busses which include output voting are described in Section 2.2 which considers hardware-aided computer configurations.

2.1.2.3 Non-Dedicated-Switched Busses

A non-dedicated switched bus configuration is shown in Figure 2.1-4.

Case 1 - Standby Redundancy

One computer is designated MASTER and the others are designated auxiliary machines. All I/O is initiated and carried out by the master machine. Synchronization of the computers and transfer of input information, as well as comparison of information prior to output is performed using the internal cross connections as described in 2.1.1.

The switching function, i.e. assignment of a bus to one of the computers, is carried out in the following fashion. Each computer generates

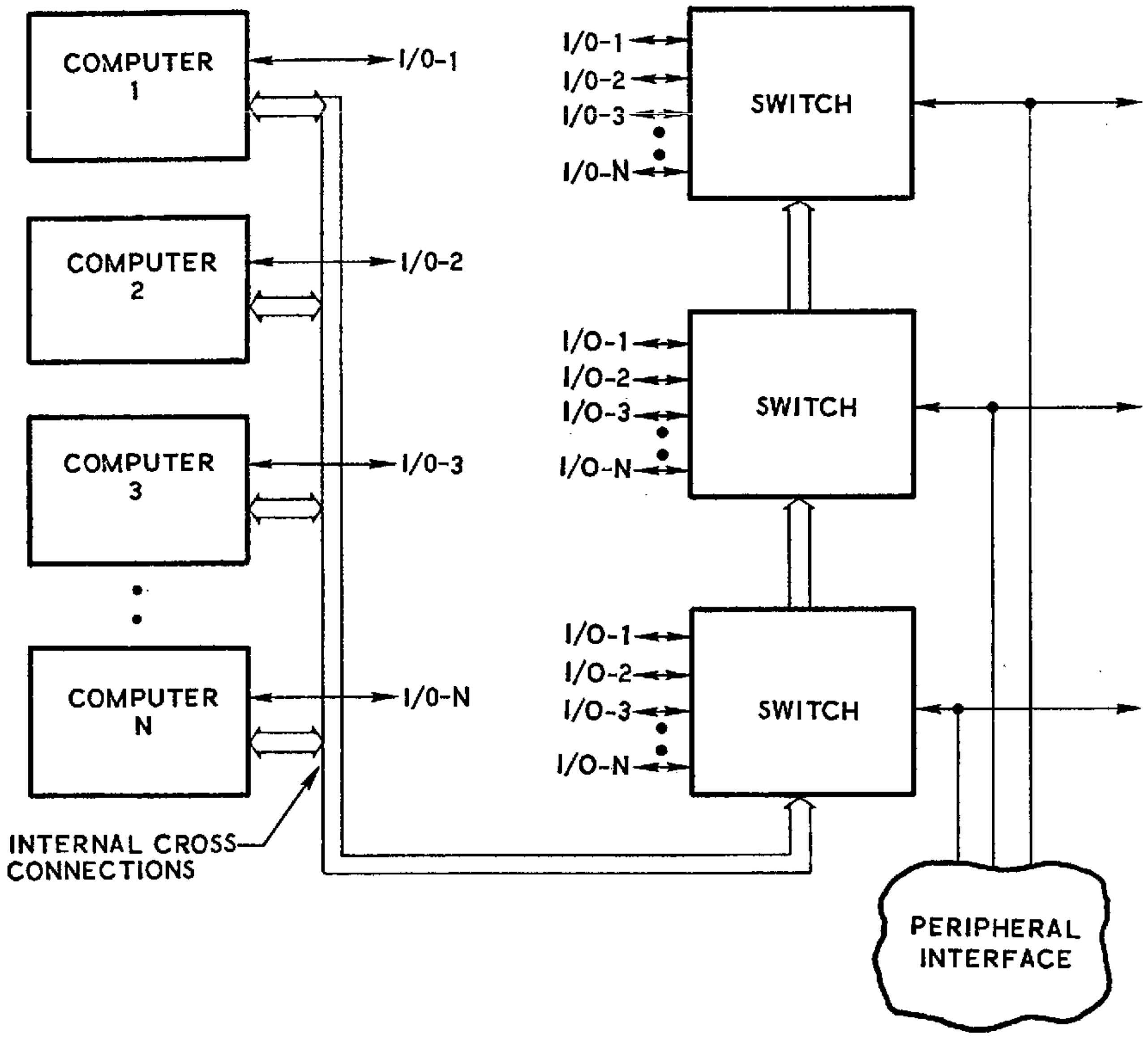


FIGURE 2.1-4 NON-DEDICATED, SWITCHED BUS CONFIGURATION

several signals to specify which switch (bus) is to be active, and which computer (master) is to be connected. Voting is provided in the switch to select the proper command when several computers disagree. (The vote performed in the switches should be adaptive and can be implemented in such a way that the switch only responds to computers in *W*, as defined by software voting algorithms. This is a problem associated with specific implementations).

1. Fault Detection Detection of faults in the computers is accomplished by comparison of variables using the internal cross-connections as described in Section 2.1. Each computer generates the information to be output and a cross-comparison is made to verify proper computation. If all machines in *W* agree, the master machine proceeds with the I/O operation. Faults in the bus may be detected using a) a wrap around check at each RTI, or b) by utilizing error detecting codes appended to words before transmission.

2. Fault Correction In the case of a disagreement of the computer designated master, when state vectors are compared before output, the remaining computers in *W* designate a new master machine by commanding the active bus to switch to a different computer. Transient correction techniques are applied to the disagreeing machine (rollback, rollahead, memory-copy as described in Section 2.1) and if the disagreement is not corrected the machine is deleted from the working set *W*.

   If bus failure is detected by a wrap-around or parity check, the computers in *W* activate a spare bus and continue computation.

Case 2 - Multiple Identical Outputs

A minor variation on Case 1 is to utilize all redundant busses to convey I/O information between the master machine and the peripheral sensors and actuator interfaces. A vote can then be performed at the peripheral interfaces and at the computer input to correct input and output information in the presence of bus failures.

In both configurations (1 and 2) all I/O is controlled by the master machine offering the distinct advantage that the peripheral interfaces are not required to synchronize multiple data streams. The standby redundant bus structure further simplifies peripheral interfaces in that voting is not required.

Case 3 - Multiple Computer Outputs

This configuration corresponds to the case of a set of active busses, connected to at least three different computers. This approach allows correction of computer and bus fault if voting is employed in the bus interface units. Several computers must be designated master, and fault detection and recovery algorithms become more complex.

2.1.3 Executive Program Considerations

The unique characteristics of the executive and applications programs running on MSW computer configurations will be considered in this and the following section. Previous sections have considered the hardware required for a "mostly" software redundant configuration as well as some of the recovery strategies which are applicable. Here the executive augmentation which is required for the MSW computer configurations will be described in enough detail to illustrate the feasibility of the approach.*

The executive augmentation which will be described is applicable to any NMR or NMR-adaptive MSW computer configuration. With some further additional modifications, operation on a duplex configuration would be possible.

The three major augmentations to a standard or skeleton executive are a voter module, intercomputer communication routines in the input/output module, and memory reload on the error handler module. The voter is a completely new module which does not appear in a skeleton executive. The intercomputer communication routines and memory reload capability are also new, but they are additions to existing modules, namely, input/output and error handler, rather than new modules themselves. Each module will now be discussed individually.

*The general requirements and applicable executive structures are described in Section 3.

2.1.3.1 Voter Module (Figure 2.1-5)

The voter module is the heart of the MSW computer configurations. All comparisons of data and all decisions as to which computers are fault-free are made in the voter module. When the voter module encounters a suspected error in a computer which was previously considered to be fault-free, the error handler is utilized to attempt to bring the computer back into the working state.

The voter module can be used in two different ways, depending on the hardware configuration in which it is used. If the bus structure is such that bus voting is required, then the voter module does this. If the hardware does the bus voting, then the voter is used only to compare program state vectors at predetermined program segment points. For either type of voting, the use of voter module is the same.

The voter assumes that the computers in the configuration are numbered. An example of the numbering scheme for three computers is shown in Figure 2.1-6. Each computer checks on the computer ahead of it to ensure that it has received data from that computer. If a computer determines that it needs to reload itself, it reloads from the computer behind it. The first computer is assumed to follow the last computer to make the connection a closed ring of computers.

The voter module is first activated when all data should have been received. The voter first checks to see if all computers have sent data. If not, a check is made to determine if data have been obtained from the next computer in the ring. If not, a fault is tallied against the next computer. If the fault count for the next computer has reached or exceeded a specified limit, n, then that computer is forcefully reloaded on the assumption that it has failed and cannot recover. If data have been received from the next computer, its fault tally is cleared to zero.

The justification for including the reload capability is the recovery of a machine whose program or constants have been damaged by a transient error. If a working computer has detected repeated errors or missing responses in another computer, the memory of the faulty machine can

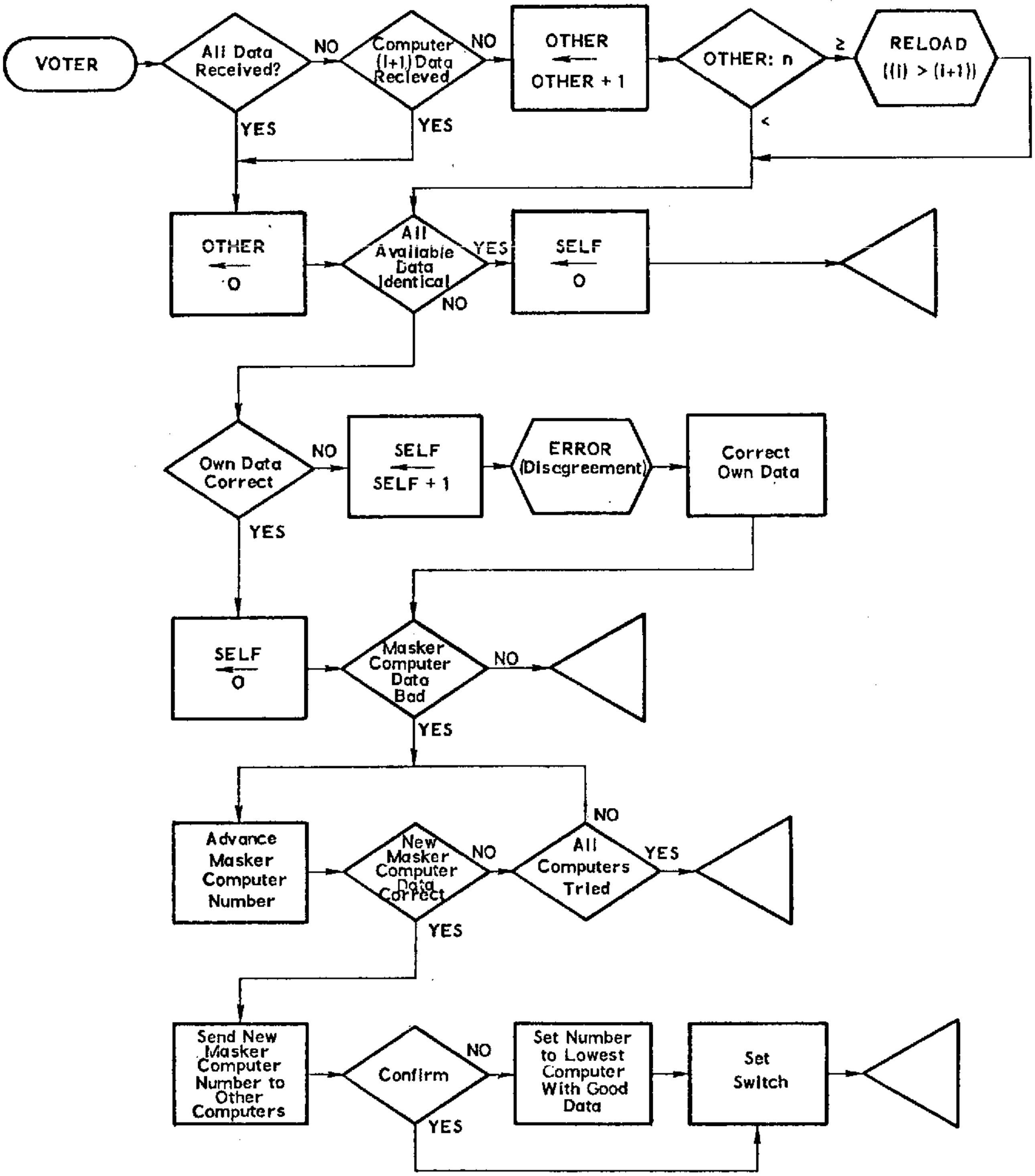


FIGURE 2.1-5 VOTER MODULE LOGIC

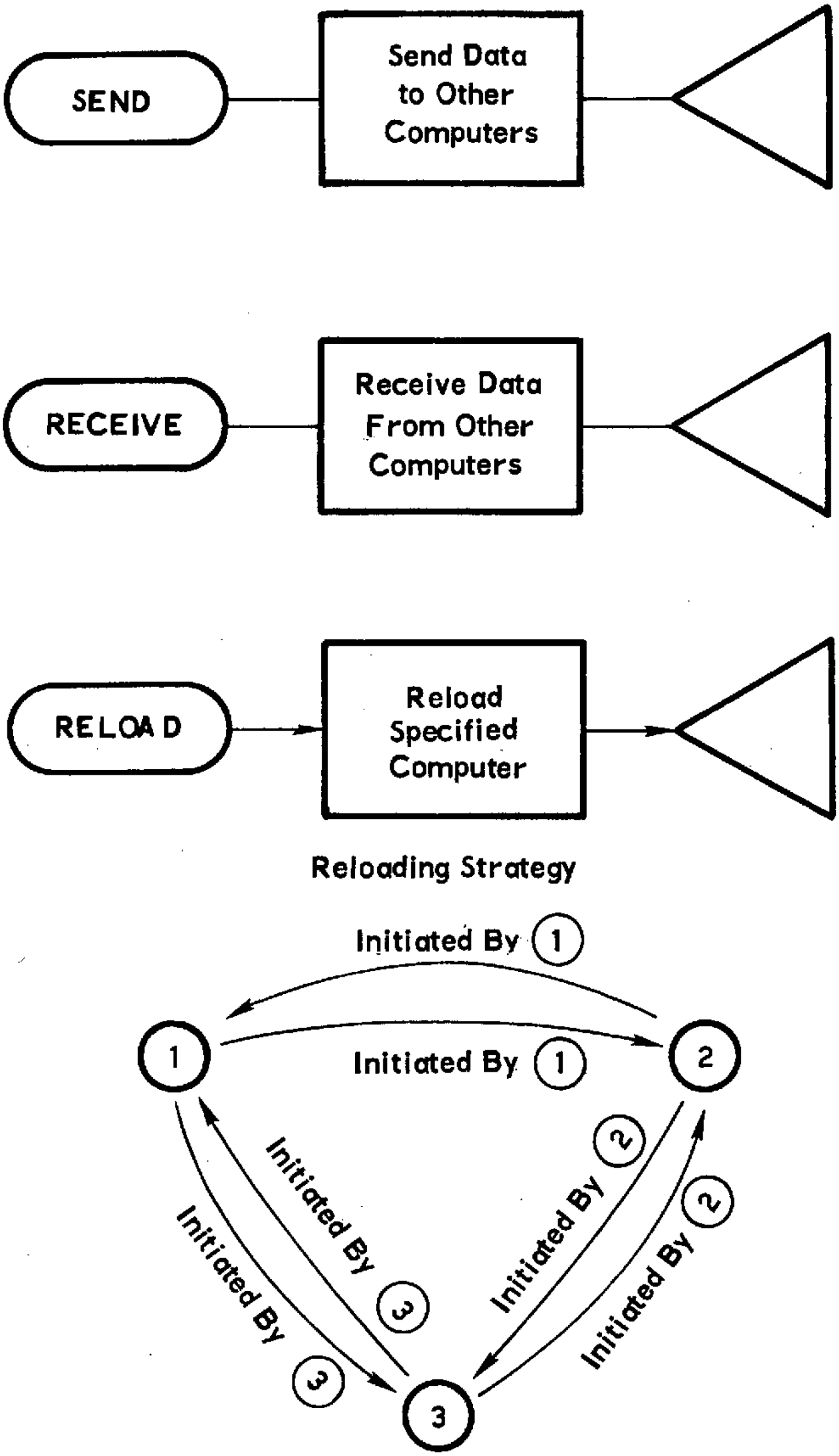


FIGURE 2.1-6 INPUT - OUTPUT LOGIC - ADDITIONS

be reloaded by the working machine. If errors persist, the faulty machine has a permanent, hardware fault. If memory reloading restores the faulty machine to working status, then memory damage had occurred and was repaired by the reload.

The next step in the voting procedure is to compare the data which are available. If all data agree, the self-error tally is cleared to zero and the voting procedure is terminated. If all available data do not agree, then further action is required. If the computer determines that its own data are in disagreement with the data from other computers, it tallies an error against itself and calls its error handler, indicating that its own data do not agree with that of the other computers in the configuration. It then corrects its own data and continues. If the computer's own data are not faulty, then it clears its self-error tally and continues.

For some MSW computer configurations, the voter module will be required to perform additional tasks. If one of the computers in the configuration is designated as a master computer, then the voter module in each of the computers must check on the behavior of the master computer. If the master computer's data are good, then no further action is required. If, however, the master computer's data are bad, then the next working computer in the ring is designated as a new master computer. If the new master computer's data are bad, the process repeats until all computers have been tried as master computers.

If and when a new master computer is obtained which has good data, then the number of the new master computer is sent to all other computers for voting and confirmation. If confirmation is received, that computer which was selected as the new master is used. If no confirmation is received, that computer which was selected as the new master is used. If no consensus can be obtained, the current master computer is used. In any case, the new master computer is set by whatever means is required by the configuration. This may be only setting tables in all the computers of the configuration or it may be actually setting hardware to designate the new master computer.

2.1.3.2 Input/Output Module

The augmentation required for the input/output module of the executive in a MSW computer configuration consists of data interchange routines and reloading routines. The data interchange routines consist of transmitting routines which transmit data to another computer and receiving routines which receive data from another computer. Similarly, reloading routines are required which transmit a memory load from one computer to another and which receive the memory load. It is desirable to have the receiving routine be implemented as much by hardware as is possible.

The data interchange routines are used to exchange input and output data for bus voting in configurations where bus voting is done by software and for program state vector voting in all MSW computer configurations. The data interchange routines in the sending computer initiate the handshaking activity required to establish a connection between computers. When the transmission has been completed, the sending computer proceeds to send the data to the next computer in the ring. It may be possible on some hardware to broadcast the data to all computers simultaneously so then only one transmission will be required regardless of the number of machines in the configuration.

The reload procedure is used to reload memory as the last resort when all other recovery mechanisms fail. If a machine is well enough to know it needs a reload, it will request it. Else, another computer will forcefully do the reload. In either case the computer being reloaded should do the minimum possible role in the reload. This may require software, but read-only memory or handwired control is preferable. The same hardware which is used for initial loading by AGE could very likely be used for the function.

The reload strategy which is used always loads the next higher numbered computer in the ring from the next lower number. The routine which transmits a reload is a relatively slow, low-priority routine. It transmits words as the facilities are available until the computer being reloaded has received all the program and inactive data words. When this has been accomplished, computation is halted, the state vector is transmitted at the highest possible speed and all machines begin computations at the same point.

2.1.3.3 Error-Handler Module (Figure 2.1-7)

The augmentation required for the error handler module of the executive in an MSW computer configuration consists of the processing which handles voter disagreement errors when the subject computer disagrees with the other computers. The first step which is taken is to record the error for later analysis if required. Then a comprehensive self-test is attempted. If the test fails, the computer returns to attempting normal operations, though it will probably be considered to be a faulty machine by the other computers in the configuration. If the self-test is successful and does not indicate any hardware malfunction, program memory damage is indicated. Thus a check is made of the number of self-errors. If a specified number, m, of self-errors has occurred, a self-reload is initiated to return the computer to normal operation.

2.1.4 Applications Programs Considerations

The major effect on applications programs of using a MSW computer configuration is the rollahead/rollback requirement which implies a need for program segmentation. It is assumed here that, as discussed in the preceding section, any voting which is required for sensor data or actuator control is done by the executive routines.

The segmentation requirement for rollahead/rollback will have varying impacts on the applications programs. Many programs will require only minimum modification to run in a rollahead/rollback environment. These programs are ones which run for only a short time when activated and require no internal segmentation. The programs may be run often, but they complete each time and can establish a new rollahead/rollback point when they complete. Another type of program which will require minimum modifications is one which requires no state vector data between activations. This type of program always uses the most current input data for its computations. Even if it is a relatively long program, it must be restarted with fresh input data if an error occurs.

The type of program which is most effected by recovery in a rollahead/rollback environment is the type which requires a relatively long time to run and has many variables in its state vector. For this type of program, careful segmentation is required to establish rollahead/rollback

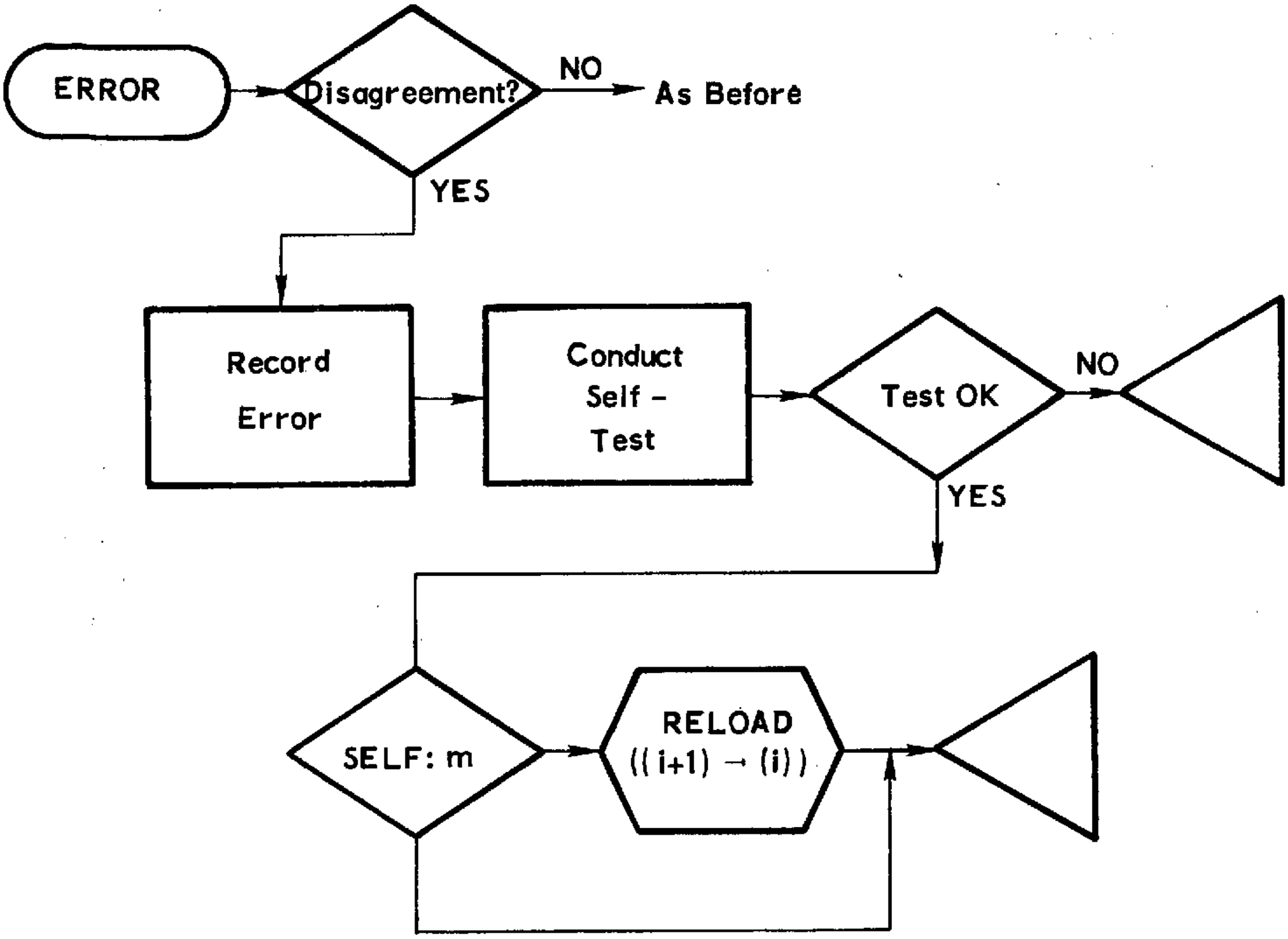


FIGURE 2.1-7 ERROR MODULE LOGIC - ADDITIONS

points with minimum size state vectors. Also, the amount of data in the state vector at the end of each activation of the program should be minimized.

One other consideration affects applications programs. This is the executive scheduling mechanism. If a synchronous-type executive is used, the rollahead/rollback structure of the applications programs can correspond temporally to their successive activations. If an asynchronous-type executive is used, an order of magnitude of complexity is introduced due to the number of programs which may be active. When an asynchronous-type executive is used in a rollahead/rollback environment, both the executive and the applications programs must be structured to minimize the size and frequency of state vector updates.

2.1.5 Machine Features and RETs

The previous discussion of software redundant configurations dealt with the general interconnections, synchronization, and recovery techniques required when external hardware is held to a minimum. Several features are characteristic of these configurations. Specifically, 1) an internal cross connection network for exchange of information between computers, 2) software techniques for synchronization, comparison checking and transient recovery in the redundant computers, and 3) dedicated and non-dedicated I/O structures. The principal objective of this architectural description is to clarify the features required for implementation of the software redundant configurations and to identify applicable RETs and their influence on reliability modeling. The next section is a qualitative discussion of those machine features necessary to implement MSW configurations with off-the-shelf hardware requiring minimal additional circuitry.

2.1.5.1 Machine Features

The salient requirement for an off-the-shelf machine for MSW implementations is the need for sufficient I/O capabilities to support the internal cross connections and I/O busses. This implies:

1. Signal/Interrupt Facilities for synchronization, and failure notification and bus redundancy control where applicable.

2. Digital (word) I/O for support of I/O busses. Serial or byte-serial transfer of information is consistent with the relatively low data rates associated with the avionics application and the requirement of high bus reliability. A DMA capability (under control of the processor for reliability) though not essential will result in simplication of software and an easing of processor timing (speed) requirements.

3. Internal Cross-Connections The machine should support the capability to send information to the other computers and a multiple-port facility to receive information from the other machines. If the computers are not precisely (hardware) synchronized, it is necessary for incoming information from other computers to be buffered. The most effective way to provide this buffering and to expedite the cross-transfer of incoming I/O information is to utilize DMA structures for the transfer of information between machines.

4. Time-Out Counting In order to proceed when one machine fails to generate data at a comparison (or rollback) point it is necessary to utilize a time-out count. This counter may be implemented either by special hardware or in software.

2.1.5.2 Reliability Enhancement Techniques

There are a considerable number of optional features and techniques which may be included in the machines for reliability enhancement. Examples are NDRO memory, wrap around bus checks, coding techniques, program rollahead, etc. The following is a discussion of the implication of these features on modeling parameters. The following is a listing of a number of these optional RETs and their effects from the standpoint of reliability modeling.

2.1.5.3 Machine Options

1. NDRO Memory - Significantly reduces the probability of transients which damage the contents of memory. Greatly reduces the probability of transients which damage programs and data in memory thus making most memory transients correctable by program rollahead.

2. Memory Address Protection - Limits the extent of damage due to a processor transient. Prevents processor transients from damaging protected memory areas, typically programs and fixed data, thus making most processor transients rollahead correctable.

2.1.5.4 Fault Detection

1. Comparison - Information is compared between the computers at rollback points and before outputs. Thus faults effecting the currently active area of program and memory are detected with probability approaching unity. This fault detection only occurs at discrete intervals of time (typically once every minor cycle) allowing considerable memory damage to occur before detection of a transient.

2. Built-In-Test Equipment - RETs associated with built-in tests such as memory parity checking, address protection, illegal operation traps, etc. have a probability of detecting faults which is considerably less than unity, however, those covered by the BITE are detected sooner thus reducing possible memory damage. When comparison is employed, BITE does not improve the probability of fault detection but does in some percentage of faults allow for more rapid detection.

3. Duplex Fault Detection - When only two computers remain functional, the probability of recovery from failure depends upon the effectiveness of diagnostics utilized to isolate the faulty machine and the effectiveness of BITE. Typically the coverage of diagnostics indicated by manufacturers includes the use of BITE, without which software diagnostics would be larger, slower, and less effective.

4. Memory Comparison There can exist faults in memory areas seldom utilized. The maximum duration of these lurking faults can be bounded if memory is compared at a low rate between machines on a periodic basis. Lurking faults tend to increase the chance of double failures and thus should not be allowed to remain within the system.

2.1.5.5 Transient Recovery

1. Program Rollback - Given that a transient is detected there exists a probability of successful rollback corresponding to the probability that only local variables are damaged in memory. This probability is significantly increased by the use of NDRO memory and address protection. Associated with program rollback is a recovery time during which the program segment is repeated during which computation is halted and the system remains vulnerable to additional faults.

2. Program Rollahead - Has the same probability of transient recovery as program rollback, but has a much smaller recovery time associated with transfer of state vector information.

3. Memory Copy - Should correct all transients if information from other computers is correct and no additional faults occur during the long copy duration (in the order of seconds).

2.1.5.6 Internal Cross Connections

1. Failure Treatment Failures in the internal cross connections have the identical effect of failures in computers for nearly all failure modes. Failure of an output drivers and input receivers are associated with their related computers as is the corresponding failure rates.

2.1.5.7 I/O Structures

Dedicated and non-dedicated I/O busses and peripheral interfaces are architectural features requiring comparative simulations for evaluation. Dedicated and non-dedicated I/O are included in the simulator and treated in the following fashion:

1. Non-dedicated I/O allows separation of the computers, bus structures, and sets of redundant peripherals such that the system reliability is reasonably expressed as the product of their individual reliabilities.

2. Dedicated I/O allows interactive failures such that the computers, bus, and I/O must be simulated together to account for dependent failures. (For example failure of a computer may disable a large set of sensors).

2.1.5.8 Bus Fault-Detection

Two techniques are commonly utilized for bus fault detection: wraparound checks and error detecting codes. The use of bus fault detection techniques must be examined in conjunction with the manner in which redundancy is applied to the I/O buses. If multiple buses are provided with voting at the receiving modules, fault detection and correction is readily implemented in the associated voters as long as three buses are functioning properly. However, when only two buses remain functional, the failure cannot be resolved by a voting process and additional fault detection techniques must be employed to identify the one which is functioning properly. If a standby redundant bus configuration (one active with spares) is utilized fault detection is essential to identify the fault and activate a spare bus.

1. Wraparound checks are characterized by periodically generated test outputs which are returned from one or more peripheral devices to verify proper functioning of the bus. While these checks have very high coverage with respect to permanent bus faults, they have two disadvantages: (a) faults are not detected instantaneously, but only at the occurrence of the next periodic check; (b) transient faults are not detected.

2. Coding techniques provide concurrent fault detection with a coverage determined by the failure modes of the bus and the amount of redundancy employed in each message. The simplest fault detecting code is multiple parity bits which can be implemented to provide high coverage at minimal cost [ULTRA 74].

In order to prevent erroneous commands from being accepted by actuators (in the case of standby or residual duplex buses), coding techniques should be employed. The use of error detecting codes allows TMR buses to function after the failure of two component buses. The effect of this RET is modeled in Section 9.

2.2 HARDWARE - AIDED SOFTWARE CONFIGURATIONS (HASW)

Hardware-aided configurations are characterized by the use of external hardware for fault-detection and synchronization. The goals of this approach are to (1) increase speed of computation and simplify software by performing the task of comparing state vectors and outputs in hardware, and (2) to allow the use of off-the-shelf computers with minimal I/O facilities. A block diagram representing the general class of hardware-aided configurations is shown in Figure 2.2-1.

A set of N computers is connected to a set of I/O busses through a special External Hardware Interface (EH). This interface may be a single, massively-redundant structure or a set of identical modules dedicated to either individual computers or busses but in either case the following functions are required:

1. Detection of faults by comparison of outputs and notification of the computers of the identity of the disagreeing unit by generating levels or interrupts.
2. Synchronization of computers (when applicable) and buffering bus information.
3. Exchanging of information between computers in order to effect transient correction.
4. Supplying the bus(ses) with correct information in the presence of faulty computers. (This is done using either switching or voting as in the MSW case).

The computer I/O interface to the EH interface is assumed to have the following properties (which are simple but consistent with a large number of computers.)

1. Programmed output consists of a single word including device address, data, and an associated strobe signal. The output may consist of parallel, serial or byte-serial data.
2. Programmed input accepts a single word which, upon issuance of a command output, is sampled after a short fixed time (in the order of a few microseconds).

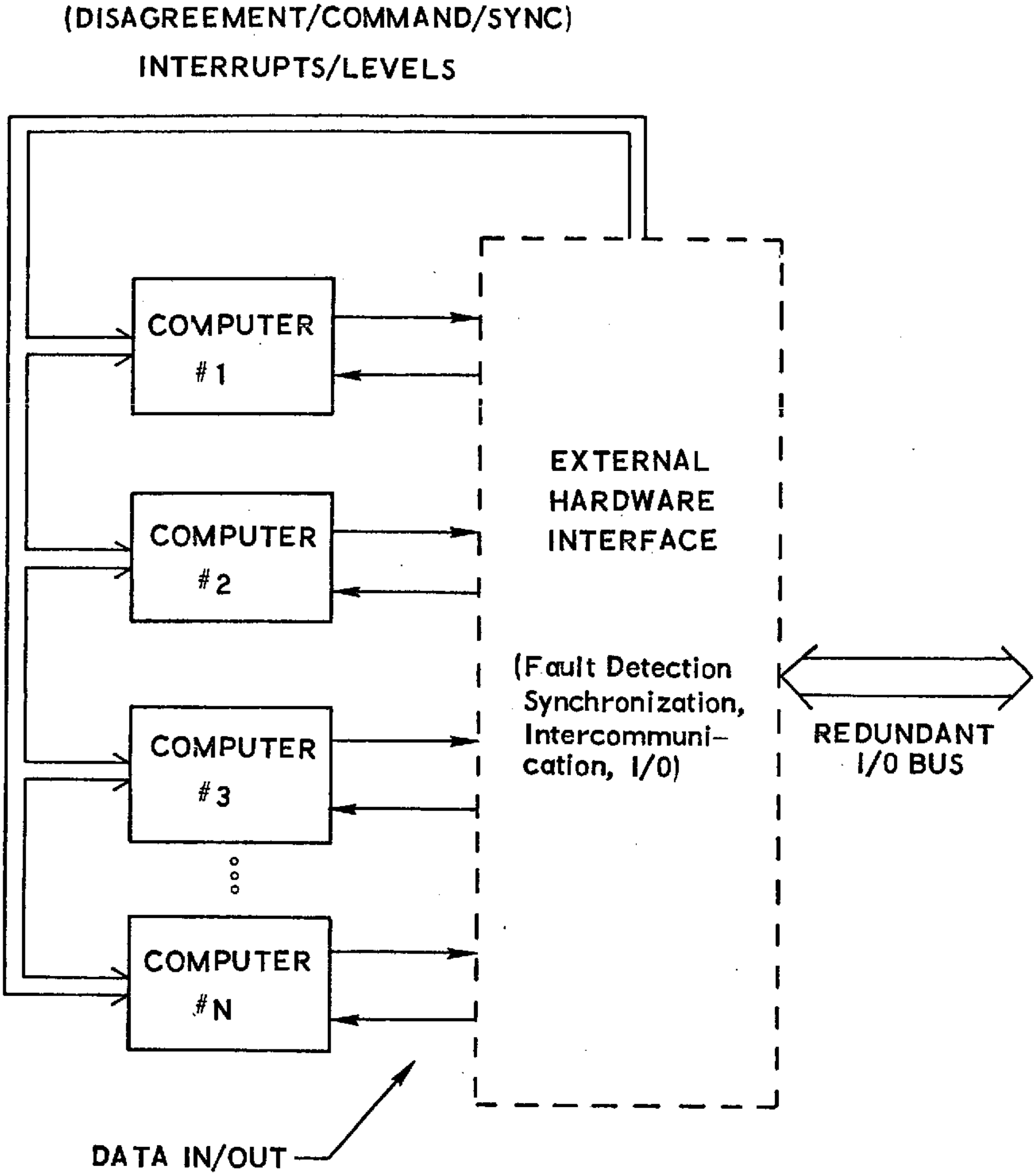

(DISAGREEMENT/COMMAND/SYNC)
INTERRUPTS/LEVELS
COMPUTER #1
COMPUTER #2
COMPUTER #3
COMPUTER #N
EXTERNAL HARDWARE INTERFACE
(Fault Detection Synchronization, Intercommunication, I/0)
REDUNDANT I/0 BUS
DATA IN/OUT

FIGURE 2.2-1 EXTERNAL HARDWARE INTERFACE: HARDWARE-AIDED SOFTWARE CONFIGURATION

3. A DMA input accepts data, as delivered from external devices and stores this information, within a worst-case time of a few memory cycles, in pre-defined sequential locations.

A byte-serial or serial computer interface is preferred if the I/O data rate is acceptably low in order to reduce connections between the computers and EH interface.

The EH interface must provide buffering for incoming bus information for the following reasons. First a typical processor expects "immediate" availability of requested data within a few microseconds of issuance of an input command. This is often not consistent with delays imposed by a bus structure. Secondly, a typical avionics bus is serial, self-clocking (Manchester) and synchronized by the sending unit, while the typical processor accepts parallel or at best byte-serial information. Thus the approach is to command the sensors to return data to one or more buffer registers in the EH interface and all computer input commands access registers within the interface.

Similarly output information is buffered for transmission on the bus for three reasons: (1) to allow parallel or byte-serial to serial conversion (where applicable), (2) to hold information from loosely synchronized computers until all outputs are available for comparison, and (3) to provide a mechanism for holding information for transfer between computers.

The simplest means for describing the functioning of the EH interface is to consider the non-redundant case. The following section will present a simplex model of the external hardware associated with a hardware aided redundant configuration. Subsequent sections will deal with options for its redundant implementation and example configurations.

2.2.1 The External Electronics Module (EEM)

The non-redundant building-block element of the external hardware interface is designated the External Electronics Module (EEM), and is shown in Figure 2.2-2. The EEM accepts and buffers outputs from the computers, synchronizes the machines, provides voting for outputs, provides for inter-computer communications, and buffers returning inputs. Examples are given for the case of a non-dedicated EEM with multiple buffers for inputs, and the case where an EEM is dedicated to each computer and has only one input buffer. The following paragraphs describe the functioning of the EEM and its interaction with the redundant computers to effect fault detection and recovery.

2.2.1.1 Case 1 - No Faults

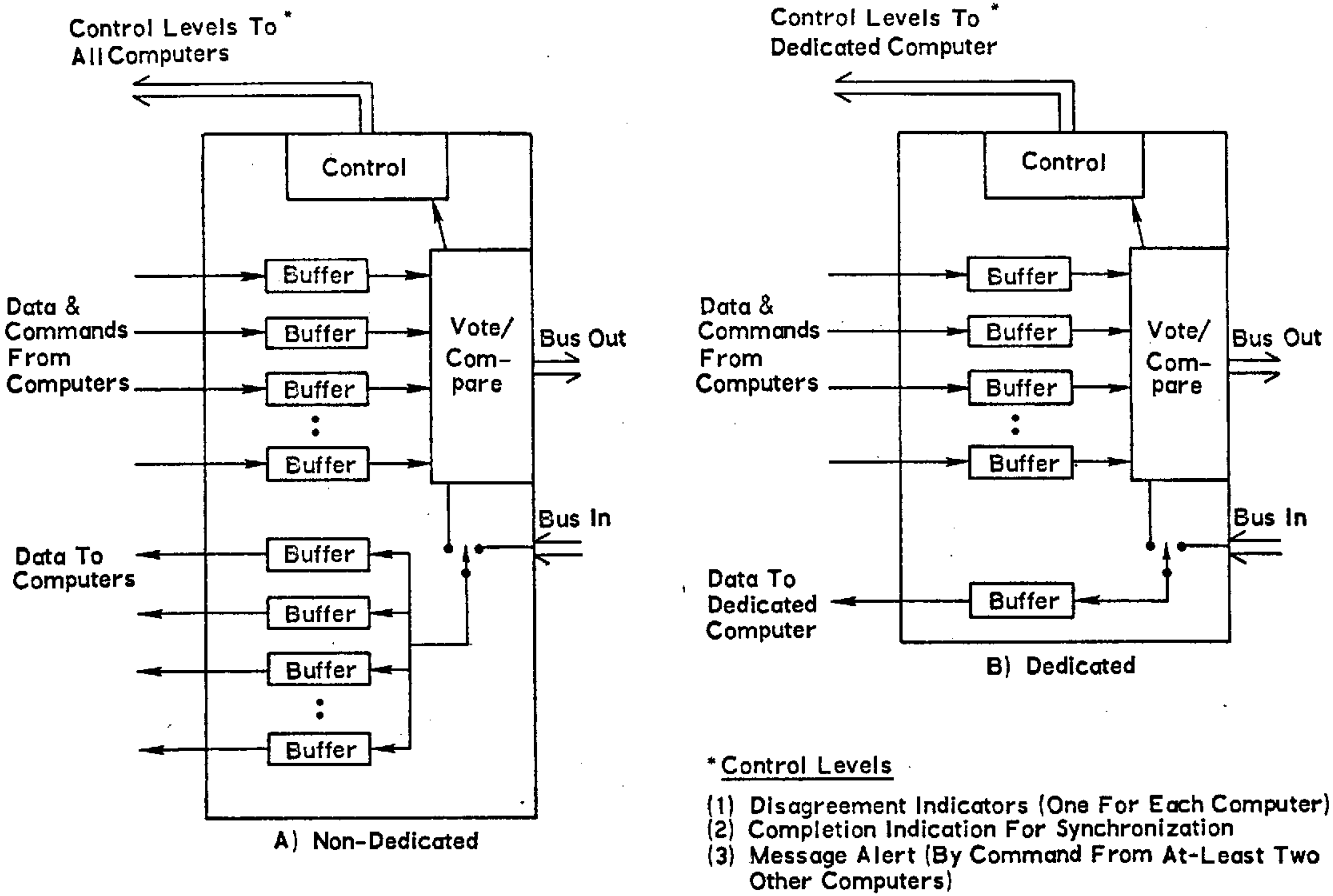
At the point of outputting or comparison of state vector information (by outputting to a non-existent peripheral) the computers load the EEM output buffers and halt. Upon receiving two agreeing outputs, the EEM waits for a "worst-case" time interval (e.g., a few instruction times) during which the other computers should have completed the output. At this point all outputs are compared, the result of the comparisons are conveyed to the computers via the disagreement indicators and a completion interrupt is sent to the computers to re-establish computation. If the output was addressed to a sensor or actuator, the output buffers are voted and conveyed to the bus and, if a data return is commanded, the returning information is loaded into the input buffer. Each computer has control of its associated input buffer for subsequent access to its returned information.

(The information on the bus is derived as an adaptive 2-out-of-n vote. Two agreeing output commands must occur within a very short time interval before a command is allowed to each the bus. Two nearly simultaneous identical outputs from faulty computers are considered to be a sufficiently remote possibility that it is not protected against in the EEM). For the case of a dedicated EEM, information is only returned to its associated computer and thus only one input buffer is provided.

2.2.1.2 Case 2 - One Computer Disagrees

In this case N-1 computers complete an identical output and a faulty computer either fails to output or outputs incorrect information. The machines are restarted by the EEM by raising the completion interrupt, and simultaneously the disagreement level associated with the faulty computer is sent to all computers. The disagreement indication causes two actions:

1. If capable of doing so, the disagreeing computer is halted by its disagreement interrupt.
2. The agreeing machines have two options; (1) classifying the disagreeing machine as faulty, masking out its disagreement



A) Non-Dedicated

B) Dedicated

*Control Levels

(1) Disagreement Indicators (One For Each Computer)
(2) Completion Indication For Synchronization
(3) Message Alert (By Command From At-Least Two Other Computers)

FIGURE 2.2-2 THE EXTERNAL ELECTRONICS MODULE (EEM)

interrupt and ignoring its further outputs, or (2) attempting a transient recovery.

Typically the first disagreement is considered to be caused by a transient and a transient recovery is attempted under control of the agreeing computers using rollahead, memory copy techniques, rollback, or restart. Recurrence of the disagreement after one or more transient recovery attempts is considered evidence of a permanent fault, the disagreement indication is masked in the working computers, and the faulty computer is subsequently ignored.

2.2.1.3 The Transient Recovery Mechanism

In order to effect transient recovery, a communication path must be established such that the agreeing computers can enter data into the memory of the faulty machine and command its restart. The adaptive voting capability is utilized within the EEM to allow this intercommunication as described below:

1. The EEM will accept commands addressed to individual computer units from the agreeing machines. If two or more agreeing commands are received within a designated time window, the information received will be transferred directly to the input buffer of the addressed machine and the corresponding message-alert signal(s) is transmitted. (Note that in the case of dedicated EEMs, each EEM only accepts commands for its associated computer).
2. The message-alert signal may be a single interrupt to a subroutine which interprets the incoming information as memory addresses, corresponding data, and transfer addresses. This approach relies on the existence of a small interrupt routine in the memory of the faulty machine which interprets messages from the other computers and effects transient recovery operations. A second approach is to utilize microprogramming through AGE interface commands so that the response to commands from the other machines resides in protected memory.

Transient recovery algorithms are similar to those used in the previous software redundant configurations (see 2.1.3 and 2.1.4). When a computer disagrees upon output of a state vector, the agreeing computers transfer the correct value of the state vector and the computation starting address to the disagreeing machine. If the rollahead is unsuccessful, as indicated by recurrence of a disagreement from the same computer within a pre-defined time period, an (optional) memory copy may be attempted. The agreeing computers transfer the contents of memory to the disagreeing machine and attempt a restart. If the previous transient recovery attempts are unsuccessful, the disagreeing machine is adjudged to contain a permanent fault, its disagreement interrupt is disabled in the working machines, it is commanded into a halt state, and its subsequent disagreement indications (if further outputs should occur) are ignored.

2.2.1.4 Case 3 - Multiple Faults

There exist two cases of multiple fault conditions; (1) the expected case where one or more computers have previously failed and the "next" single failure occurs, and (2) the case where more than one failure occurs simultaneously.

1. If one or more machines have failed previously and a single computer fails, a single disagreement indication from that machine is acknowledged by the remaining good computers since the disagreement indications of previously faulty machines have been masked out. Transient correction is carried out identically to Case 2 above.

2. Recovery from multiple, simultaneous disagreements can be effected if at least two computers remain in agreement, as the two remaining computers can initiate transient recovery in the other machines. If all computers disagree, no output is acknowledged by the EEM which requires two agreeing outputs to respond. A period without I/O activity corresponds to this condition and may be detected by logic within the EEM modules which would cause a system restart in all computers.

System failure occurs in the previous configurations when all but two computers have failed and one of the remaining computers suffers either an uncorrectable transient or a permanent failure. When two computers remain functional, this condition is designated the Residual Duplex Configuration. The following section is a discussion of techniques for recovery from failures in the residual duplex configuration and continuing computation with a single simplex computer.

2.2.2 Residual Duplex and Augmented Voting Redundancy

In order to effect recovery when only two computers remain functional it is necessary to employ program rollback for transient correction, diagnostics for permanent fault isolation, and modifications to the EEM to override the output vote and allow the remaining single computer access to the I/O bus. The following is a description of the response to faults in the residual duplex configuration.

1. The EEM units are notified by command from the two functioning processors (before occurrence of a fault) that only two computers remain and the identity of those two processors is stored.
2. The two remaining computers modify the interrupt handling routines associated with their disagreement interrupts such that a program rollback is performed upon disagreement. (In the residual duplex mode the EEM accepts outputs from either machine and, if a disagreement occurs, it inhibits the output and indicates disagreement to both computers).

   Program rollahead is no longer applicable since it is not known which machine contains correct information and thus the rollback segment is re-executed using the previously stored (N-1th) state vector.
3. If the disagreements continue, the EEM commands a diagnostic in both machines which includes hardware tests and checksum verification of programs. The machine which delivers a properly computed (pre-determined) output will be allowed to continue the computations. If both computers pass the diagnostic or if both fail one computer is chosen and a system restart is attempted. (The latter cases have a significant chance of failure).

2.2.3 A Recovery Algorithm: Hardware-Aided Software Configuration

The following section describes an example recovery mechanization for a Hardware-Aided Software RCS configuration. It is intended to indicate the interaction between hardware and software in a typical recovery mechanization.

2.2.3.1 Hardware EEM Functions

The EEM provides the following signals to (1) its associated computer if it is dedicated or (2) all computers if it is non-dedicated.

1. I/O-Complete Interrupt Indicates that an output has been completed by at least two agreeing computers and an elapsed time Δt has occurred to allow all other machines to output. As all computers halt after output the I/O complete interrupt serves as a restart/synchronization signal.

2. Disagreement Indicator Signals are sent simultaneiously with the I/O complete interrupt. There is one Disagreement Indicator for each computer which indicates that computer's disagreement with the threshold-voted result.

3. Rollback Interrupt When the EEM is notified that only two computers remain functional, subsequent disagreements of the two machines result in a Rollback Interrupt. This interrupt, sent to the two residual duplex computers, causes a program rollback.

4. Diagnose Interrupt If rollbacks prove unsuccessful in the residual duplex configuration, the EEM commands a diagnosis of the two machines. A machine which successfully completes the diagnostic is connected to all buses and continues in simplex.

5. Message Alert Interrupt Two agreeing computers may transmit a word to any other computer by specifying a unique device ID upon output. The data word is transferred to the input buffer associated with the addressed computer by the EEM and a Message Alert. The Message Alert interrupt is sent to that machine (see Figure 2.2-3). In the case of dedicated units, the EEM associated with the addressed machine recognizes the device I/D and loads its associated input buffer.

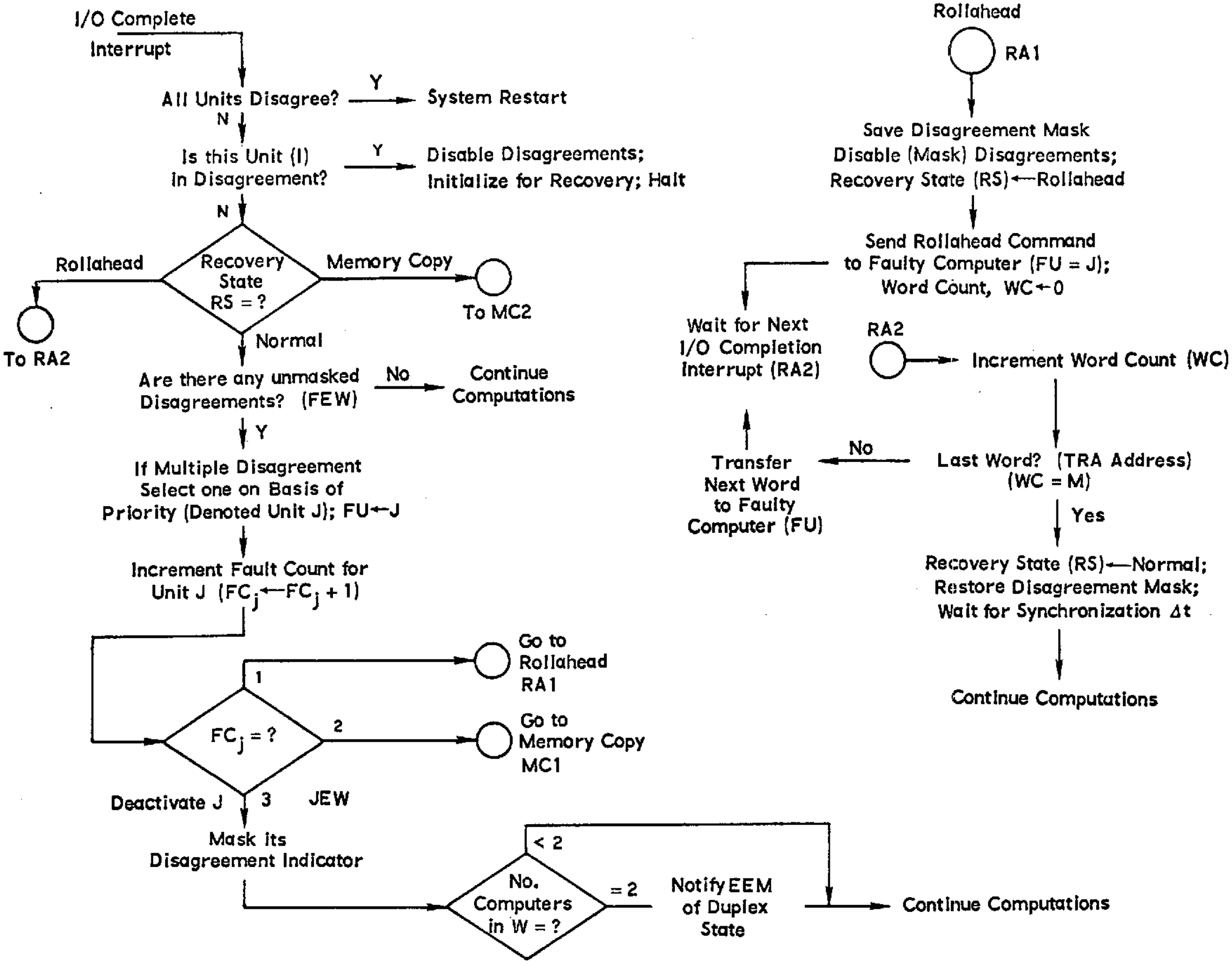


FIGURE 2.2-3a RECOVERY SOFTWARE - (ROLLAHEAD) HARDWARE-AIDED CONFIGURATION

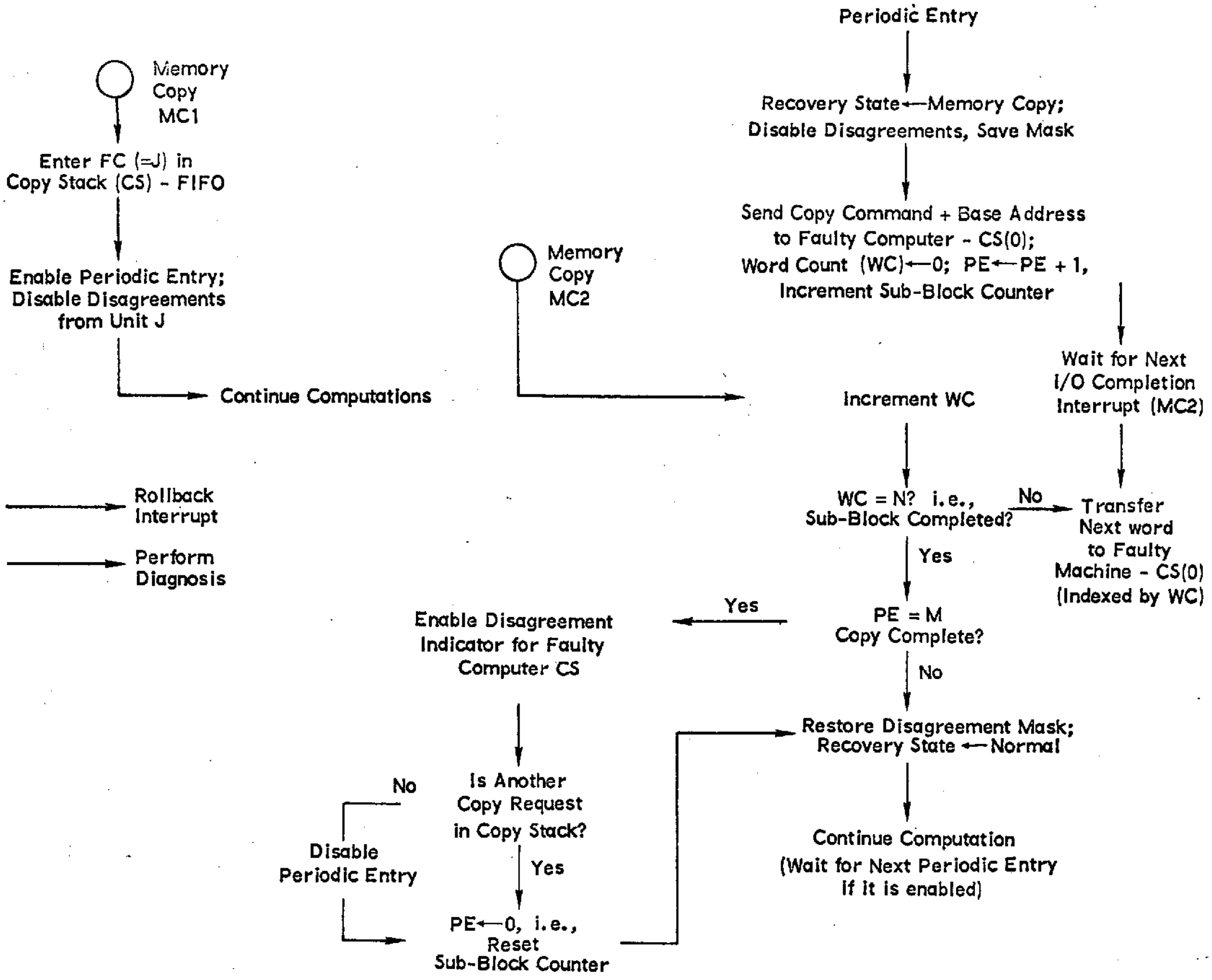

Memory Copy MC1
Enter FC (=J) in Copy Stack (CS) - FIFO
Enable Periodic Entry; Disable Disagreements from Unit J
Continue Computations
Rollback Interrupt
Perform Diagnosis
Memory Copy MC2
Periodic Entry
Recovery State ←— Memory Copy; Disable Disagreements, Save Mask
Send Copy Command + Base Address to Faulty Computer - CS(0); Word Count (WC) ←— 0; PE ←— PE + 1, Increment Sub-Block Counter
Wait for Next I/O Completion Interrupt (MC2)
Increment WC
WC = N? i.e., Sub-Block Completed?
No
Transfer Next word to Faulty Machine - CS(0) (Indexed by WC)
Yes
PE = M Copy Complete?
Yes
Enable Disagreement Indicator for Faulty Computer CS
No
Restore Disagreement Mask; Recovery State ←— Normal
Continue Computation (Wait for Next Periodic Entry if it is enabled)
Is Another Copy Request in Copy Stack?
No
Disable Periodic Entry
Yes
PE ←— 0, i.e., Reset Sub-Block Counter

FIGURE 2.2-3b RECOVERY SOFTWARE - (MEMORY COPY) HARDWARE-AIDED CONFIGURATION

2.2.3.2 The Software Function

Figure 2.2-3 represents a block diagram of the software algorithms utilized to carry out fault recovery. The software in each functioning machine maintains a record of those computers which are properly functioning and those considered to have failed. The disagreement indicators are permanently masked (ignored) for previously discarded machines. The recovery software has four states: normal, rollahead, memory copy, and wait, which are discussed below:

1. Normal During normal operation at the completion of each output, the computer checks the unmasked disagreement indicators and if there are no disagreements it resumes normal computations. In the case of disagreements, the following actions are taken:

   a. If the computer is itself in disagreement, it halts (if possible) and waits for rollahead or memory copy information from the other computer.

   b. If a different machine is in disagreement, it examines an error tally associated with the faulty machine (FC_j). If that machine has not previously disagreed, the rollahead state is entered and the current state vector and restart address is transferred to the disagreeing machine. If a previous rollahead has been unsuccessful within some time interval Δt, a memory copy is attempted.

2. Rollahead The current state vector and restart address are transmitted from the agreeing machines to the disagreeing machine. Fault indications are masked during the rollahead with two exceptions. If one of the working machines disagree during the transfer, it enters the wait state.

3. Memory Copy The programs, constants, and a minimal set of variables required to permit a system restart are transferred to the disagreeing machine at a low duty cycle. Since the memory copy cannot be allowed to consume more than a few

percent of the available processing time, the copy information is transmitted periodically in short blocks. Thus a memory copy is initiated by activating a periodic entry to the transfer program. The Memory Copy State is only entered during the transfer of each short block. The final block contains variables necessary to perform a "warm" system restart.

4. Wait State A machine which recognizes that it is in disagreement upon receiving an I/O complete interrupt or BITE error indication halts until loaded by the other working machines with a program rollahead or memory copy. If a memory copy is unsuccessful the disagreeing machine is ignored. The fault tallies associated with each machine not designated as permanently faulty ($FC_j \leq 2$) are cleared if sufficient time passes without recurrence of a fault.

Two additional interrupts are shown in Figure 2.2-3 as single arrows which are utilized by the EEM to command program rollback and diagnosis when only two computers remain functional.

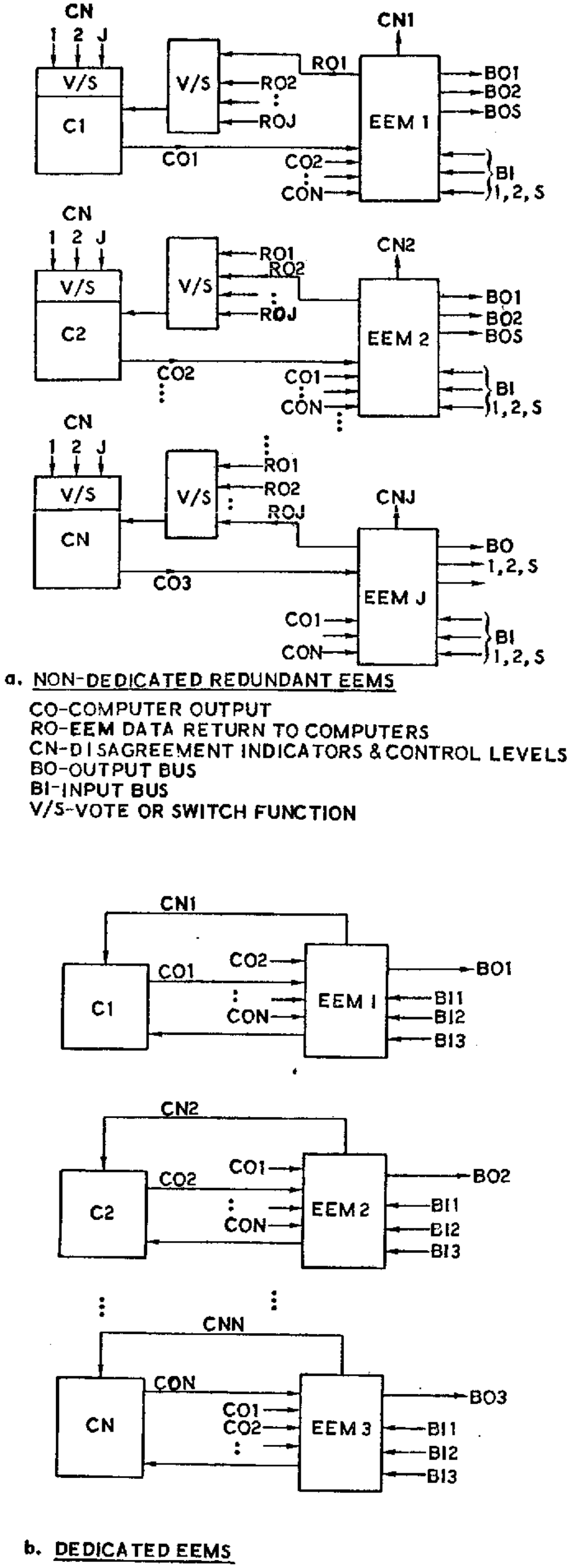
In the case of non-dedicated EEM's an additional level of complexity is added to the software. A set of outputs is received from each EEM and an adaptive vote must be taken on these inputs in order to eliminate the effects of defective EEM units on the recovery process. This is straightforward and not included in the figure.

2.2.4 Utilization of Redundant EEM Units

Two general approaches to implementing redundant EEM units are shown in Figure 2.2-4. The first approach employs non-dedicated redundancy such that any of several EEMs can be utilized by each computer. The second approach utilizes a separate EEM dedicated to each computer.

2.2.4.1 Non-Dedicated Redundant EEMs

As shown in Figure 2.2-4a, there are N computers and J EEM units. Each EEM accepts outputs and commands from all the computers and inputs from all buses. Therefore, when functioning properly, each EEM delivers identical data and control information to the computers. By a process of voting and/or



a. NON-DEDICATED REDUNDANT EEMS

CO-COMPUTER OUTPUT
RO-EEM DATA RETURN TO COMPUTERS
CN-DISAGREEMENT INDICATORS & CONTROL LEVELS
BO-OUTPUT BUS
BI-INPUT BUS
V/S-VOTE OR SWITCH FUNCTION

b. DEDICATED EEMS

FIGURE 2.2-4 REDUNDANT EEM IMPLEMENTATIONS

switching, the computers can ignore the outputs of one or more faulty (disagreeing) EEM units. (Typically the input to the computers from EEM units would be voted using an adaptive software vote requiring one more than half of the inputs to agree from EEM units which have not previously been adjudged faulty).

If each EEM can be commanded to output on one of several (s) redundant bus lines failures in the EEM can be decoupled from bus failure, resulting in a totally non-dedicated system.

2.2.4.2 Dedicated EEMs

In this case an EEM unit is dedicated to each computer resulting in considerable simplification of the EEM (see Figure 2.2-2) and the associated interconnections. This approach provides adequate reliability if the failure rate of the EEM and associated bus is small with respect to the failure rate of the computer. In this case failure of an EEM may result in disabling the associated computer and bus. Failures which cause improper control signals result in failure of the computations on the associated computer, while failures in voting for output disable the associated bus. Non-dedicated sensors and actuators are recommended using this configuration.

2.3 MOSTLY-HARDWARE CONFIGURATIONS

Mostly-hardware configurations are structured in such a way as to minimize the amount of software required to support fault detection and recovery. Table 2.3-I indicates the additional supporting software functions employed in software, hardware-aided and mostly hardware configurations.

	Mostly- Software	Hardware-Aided Software	Mostly- Hardware
Fault Detection by Comparison	X		
Synchronization	X		
Transient Recovery	X	X	.
Recording and Masking Permanently Faulty Modules	X	X	.

TABLE 2.3-I SOFTWARE OVERHEAD OF FAULT-TOLERANT CONFIGURATIONS

The special software features associated with hardware-aided configurations are:

1. Rollback/Rollahead structured programs.
2. Identifying recurring faults and the decision to employ rollahead, memory copy, or classify a computer as permanently faulty.
3. Control of data transfers for rollahead and memory copy.
4. Disabling (fault response) faulty machines.
5. Diagnostic programs for recovery when only two computers remain functional.
6. A "warm" restart capability. A restart point at which computation can be resumed with a minimum of variables required for initialization. (Employed to minimize downtime for transfer of variables at the end of a memory copy.)

Mostly-hardware configurations perform the functions associated with the previously discussed hardware-aided software configurations along with implementing one or more of the above functions in hardware.

The associated hardware recovery elements will be considered as extensions to the EEM previously discussed. And the augmented EEM's will be protected utilizing non-dedicated redundancy as shown in Figure 2.2-4.

2.3.1 Augmented EEM Units

The primary augmentation of the EEM units to effect mostly-hardware control over the recovery process is the addition of a "system state" store consisting of a working/failed flip flop and a tally count for each computer. Using this information the augmented EEM acts as a finite-state machine. The current system state and disagreement indications are combinatorially mapped into a specific recovery action. Similarly the fault indications are utilized to update the system state. This mechanization is shown in Figure 2.3-1. The state control is similar for a large number of implementations while the recovery net implementation is custom to the particular recovery algorithms employed. Thus an informal description of the state control function will be given below, followed by descriptions of the custom recovery control for the two example configurations in the following sections.

2.3.1.1 The State Control Function

The state control function is to update the system state store to indicate the current status of the redundant computers within the system. Associated with each computer is a tally count which is advanced upon its disagreement with the other working machines. The tally counts are periodically reset if a sufficient period of time has elapsed to indicate that a transient has been corrected. Typically, after the first disgareement, (T_i=1) a transient recovery is attempted. If subsequent disagreements recur, the count is advanced ($T_i \leftarrow T_i+1$), additional recovery techniques are attempted and, after a prescribed number of unsuccessful recovery attempts (T_i=N), the computer is adjudged permanently faulty ($W \rightarrow F$) and further recovery attempts are discontinued.

2.3.2 Recovery Control

A logical starting point in the description of the recovery functions carried out by the augmented EEM is the control levels between the AEEM and computer units. The following control levels are a "typical" set generated by the augment EEM.

1. I/O Complete Interrupt (ICI) - Same as HASW configuration provides synchronizing restart after computers output.

2. Send Rollahead (SR) - Causes computer to output a state vector and rollahead address. This information is received from all computers thus commanded, voted in the AEEM and loaded into the input register of the disagreeing machine.

3. Receive Rollahead (RR) - Is sent to disagreeing computer to accept rollahead information from its input buffer. (Details of control can be implemented several ways. A typical case is to treat SR, RR, and ICI with the following interpretation: $SR \cdot \overline{ICI}$ - send first word, $SR \cdot ICI$ - send intermediate word. The last word is a transfer address for rollahead restart. The AEEM is notified by a special device code. Similar interpretations are utilized by the receiving computer ($RR \cdot \overline{ICI}$ - first, $RR \cdot ICI$ - intermediate, and $\overline{RR} \cdot ICI$ - last word and start.))

4. Send Copy Block (SCB) - Causes computer to output a block of data for memory copy.

5. Receive Copy Block (RCB) - Is sent to disagreeing computer to accept memory copy information from its input buffer. (Several detailed implementations are possible similar to the example in 3 above).

6. Halt - Halts the commanded machine.

7. Cold Start - If all computers disagree the AEEM commands a cold start of all machines currently classified as working.

8. Diagnose - Causes computer to execute a self-diagnostic including program check-sums.

Two recovery strategies will be discussed in the following sections. The first employs only memory copy techniques for transient recovery and, at the cost of slower recovery of the disagreeing machine, eliminates the requirement for rollahead/rollback structured programs. Unfortunately, this approach

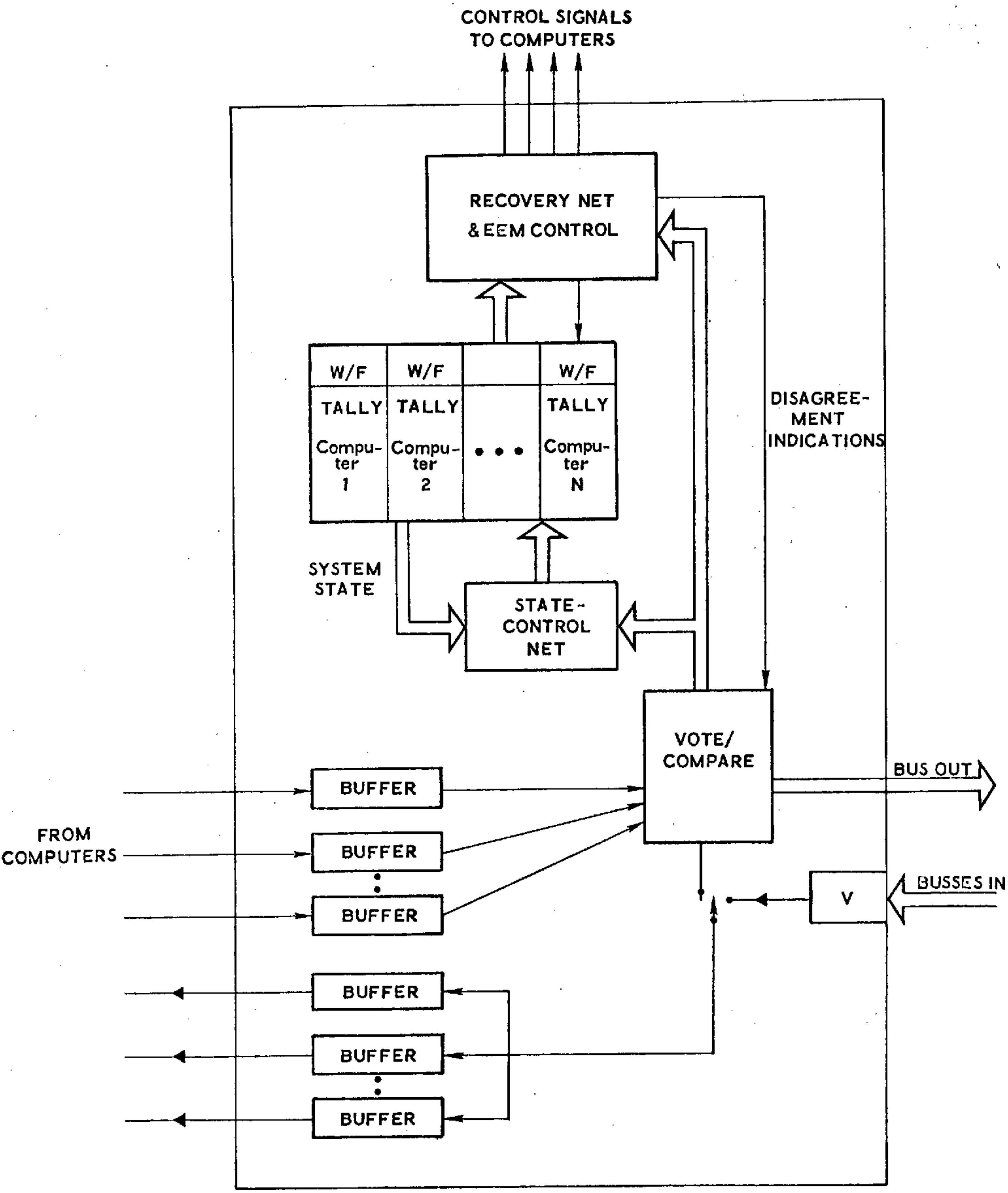


FIGURE 2.3-1 AUGMENTED EEM FOR MOSTLY-HARDWARE CONFIGURATIONS

requires system restart to recover from transients in the residual duplex configuration. The second approach utilizes program rollahead, rollback, and memory copy techniques.

2.3.2.1 Memory Copy Implementation

The memory copy recovery algorithm is indicated in Figure 2.3-3a. If a computer disagrees upon output (D_i) and it has no previous disagreements within a fixed time interval Δt ($TC_i=0$), then a memory copy is attempted. Periodically on the order of every millisecond a small block of program data is transferred to the disagreeing machine. This transfer is maintained at a low duty cycle so not as to degrade the computations of the remaining computers. After transfer of the final block, which consists of variables necessary to restart all computers in an initialized state, the memory copy state is terminated and the disagreeing computer is brought back into operation. If the disagreement recurs within a short time ($TC_i=1$ and D_i) the computer is assumed to contain a permanent fault, is classified faulty, and the voter is commanded to ignore any further outputs from the machine.

Two approaches are suggested for multiple faults:

1. If the state control is disabled during a memory copy additional faults in other computers are ignored during the memory copy. As long as sufficiently many additional computers do not fail such that the voted output remains valid, correction of additional faults can be deferred until the current copy is completed.

2. A second approach is to re-initialize the memory copy when an additional fault occurs, and perform the memory copy into both (or all) the defective machines.

Residual Duplex

Items 3, 4, and 5 in the memory copy algorithm (Figure 2.3-3a) correspond to actions taken in the residual duplex and simplex configurations, i.e. when only two or one computers remain functional. The only mechanism available for transient correction when one or two computers remain functional is a system restart (3,5). If in the residual duplex case a

1) If $TC_i=0$ and D_i and $\rho w>2$ then Memory Copy*

2) If $TC_{i,j}=0$ and $D_{i,j}$ and $\rho w>2$ then $W_i \rightarrow F_i$

3) If $TC_{i,j}=0$ and $D_{i,j}$ and $\rho w=2$ then Restart

4) If $T_{i,j}=1$ and $D_{i,j}$ and $\rho w=2$ then Diagnose**

5) If $T_i=d$ and D_i and $\rho w=1$ then Restart

a) MEMORY COPY ALGORITHM

1) If $TC_i=0$ and D_i and $\rho w>2$ then Rollahead

2) If $TC_i=1$ and D_i and $\rho w>2$ then Memory Copy

3) If $TC_i=2$ and D_i and $\rho w>2$ then $W_i \rightarrow F_i$

4) If $TC_{i,j}=0$ and $D_{i,j}$ and $\rho w=2$ then Rollback

5) If $TC_{i,j}=1$ and $D_{i,j}$ and $\rho w=2$ then Diagnose**

6) If $TC_i=d$ and D_i*** and $\rho w=1$ then Rollback

b) ROLLAHEAD-ROLLBACK-COPY ALGORITHM

---

*Memory Copy

1) If P and Blockcount $< k$ then Transfer Data Block
2) If P and Blockcount $=k$ then Exit Memory Copy

**Diagnose Select machine which delivers correct result

1) If i=ok, j=Fail then $W_j \rightarrow F_j$
2) If i=ok, j=ok then $W_j \rightarrow F_j$
3) If i=Fail, j=Fail then $W_j \rightarrow F_j$

} Select One Arbitrarily

***BITE-Indicated Fault

DEFINITIONS

p=periodic interval

k=maximum number of blocks

D_i=disagreement in i^{th} computer

TC_i=tally count for i^{th} computer

ρw=number of working computers

$W_i \rightarrow F_i$=classify i^{th} computer as failed

FIGURE 2.3-3 AUGMENTED EEM RECOVERY ALGORITHMS

system restart is unable to correct the disagreement between the two machines, a diagnosis is commanded (4) and, if it is successful, the system degrades to simplex operation.

2.3.2.2 Rollahead - Rollback - Copy Implementation

A recovery algorithm which includes rollahead, rollback, and memory copy techniques is indicated in Figure 2.3-3b. When more than two computers are functioning (ρw72) the transient recovery sequence consists of program rollahead followed (if unsuccessful) by memory copy. If the fault persists after a memory copy it is assumed permanent, the computer is classified as faulty, its outputs are excluded from the vote, and further recovery attempts are abandoned.

When two computers remain functional (ρw=2) a rollback is attempted for transient correction. If unsuccessful, both computers are commanded to run a diagnostic. One is selected and, if successful, the computation continues in simplex with program rollback upon BITE-detected faults for partial transient protection.

2.3.3 A Comparison of MHW and HASW Implementations

The principal difference between the mostly hardware and hardware-aided software configurations is that in the former the system state information and recovery decision mechanism resides in a central "hard core" In the HASW configurations this information and control mechanism is distributed and replicated within the software of the individual computers. The tradeoff between the two implementation types is largely a matter of cost.

Implementation costs in the mostly hardware case includes not only augmentation of the EEM units, but also a mechanism for protecting against and correcting transient errors in the augmented EEMs. A process of voting on all internal states (NMR synchronization) is required as well as a well defined AEEM restart in case of information loss. To protect against transients it is advisable that the AEEM control states be maintained in non-volatile storage. Thus the augmented EEM is considerably more complex than the HASW EEM without augmentation.

The additional software necessary for HASW implementations with respect to the mostly hardware case is estimated to require on the order of 1000-2000 words of storage. (Note that program diagnostics and in many cases rollback-structured programs are required in both implementations).

Memory is allocated for most of the candidate computers in 8k modules, thus the machine is procured with 16k or 24k or 32k words. Often under these circumstances, additional memory exists beyond that required for the flight programs and thus the implementation of HASW routines can be placed in this extra "core space" at nominal extra cost. If this is the case (as expected) for the RCS application, the HASW-type implementation is the most cost-effective.

THIS PAGE INTENTIONALLY LEFT BLANK

3.0 EXECUTIVE STRUCTURE

The structure of the executive designed for the RCS study is described in this section. The design goals for the executive are discussed first. Then each module of the executive is described. Several scheduling mechanisms are then presented and compared. Finally, the adaptability of the design is considered.

3.1 DESIGN GOALS

Four design goals have been established for the executive. These goals specify general guidelines for the executive design as well as indicating a particular application in which the executive could be used. These goals are to design an executive which:

1. Can be readily adapted as an executive model for all RCS configurations under consideration.
2. Is general enough to support any reasonably foreseeable avionics application.
3. Makes clearly visible all the features required to support a digital flight control application.
4. Makes available the necessary parameters for configuration evaluations.

The RCS configurations under consideration in this study all use hardware, software, and program-execution redundancy in varying degrees to obtain fault tolerance. The modeling undertaken as part of the RCS work is of a much broader scope than that attempted in the past. This modeling not only gives explicit consideration to both permanent and transient faults, but also treats the actions of the relevant software. In considering the relevance of the various portions of the software, it was recognized early that the details of the actual applications programs have little to do with the fault-tolerance of an RCS. However the executive structure is important and thus must be considered.

Since this study is directed toward multicomputer systems rather than encompassing multiprocessors, a single executive may be designed for use in each of the computers of the configuration. Thus, the first design goal ensures that the executive which is designed can be used in all computers of all configurations being considered, adapted as required by the configuration.

An earlier study [RATN 73] has shown that the computational environment imposed by avionics applications is such that the majority of computations must be performed periodically, although the computations performed will vary with the phase of the flight and the mode(s) used. Thus the computational requirement imposed by the avionics environments in which the computer systems being considered will operate involves primarily periodic, cyclical tasks of varying complexity, rate, and duration. The processing of occasional aperiodic tasks is also required. The executive has been designed to meet both of these requirements and thus be generally applicable to all avionics applications. The structure of the executive which has been designed is applicable to all applications, but details are application-dependent to varying degrees. The digital flight control application is being used as an example of a specific application to ensure that the general design can be specified in some degree of detail for one specific application. Thus, the second and third design goals, are complementary.

3.2 SKELETON MODULES

The design of an executive for a fault-tolerant computer system includes all the features required for an executive designed for a conventional (i.e., non-fault-tolerant) computer plus those unique features required in the software to implement the fault-tolerant capability designed into the fault-tolerant system. The Reconfigurable Computer Systems (RCS's) under consideration by this study are, by definition, fault-tolerant computer systems. Although each different RCS organization will impose different requirements on the executive, a common skeleton can provide the basis for the entire series of RCS executives since the same applications programs will be performed on all RCS's.

The executive skeleton consists of four distinct modules, each providing one of the four basic facilities required in an executive for an avionics computer. The four modules are the scheduler, the input-output driver, the interrupt processor, and the machine error handler. Each of the modules is described in the following subsections, followed by a discussion of the interaction between pairs of the modules. A block diagram of the executive skeleton is shown in Figure 3.2-1.

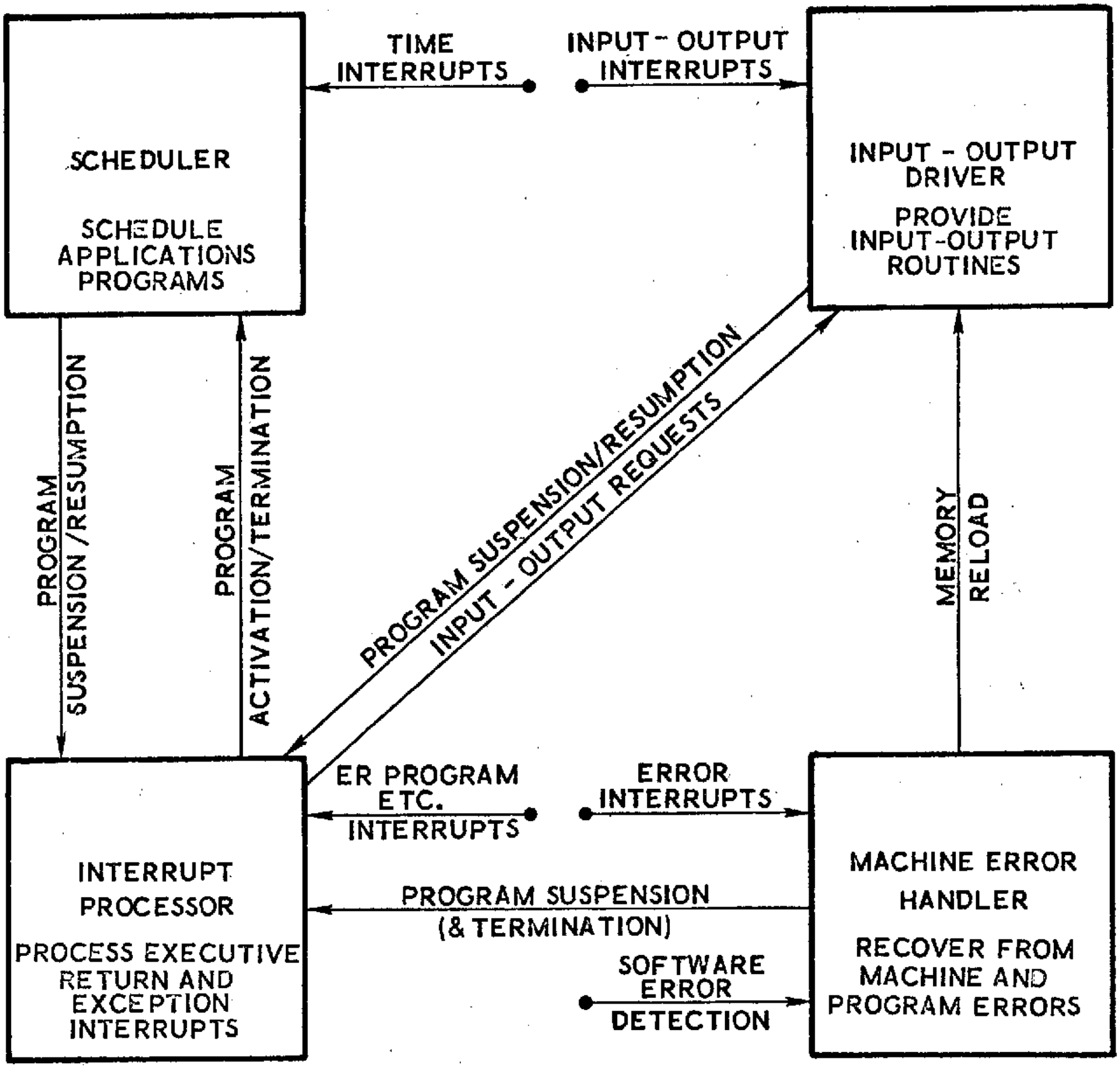

TIME INTERRUPTS
INPUT - OUTPUT INTERRUPTS
SCHEDULER
SCHEDULE APPLICATIONS PROGRAMS
INPUT - OUTPUT DRIVER
PROVIDE INPUT-OUTPUT ROUTINES
PROGRAM SUSPENSION /RESUMPTION
PROGRAM ACTIVATION/TERMINATION
PROGRAM SUSPENSION/RESUMPTION
INPUT - OUTPUT REQUESTS
MEMORY RELOAD
ER PROGRAM ETC. INTERRUPTS
ERROR INTERRUPTS
INTERRUPT PROCESSOR
PROCESS EXECUTIVE RETURN AND EXCEPTION INTERRUPTS
PROGRAM SUSPENSION (& TERMINATION)
SOFTWARE ERROR DETECTION
MACHINE ERROR HANDLER
RECOVER FROM MACHINE AND PROGRAM ERRORS

FIGURE 3.2-1 THE EXECUTIVE MODULES USE INTERMODULE CALLS FOR SERVICES WHICH ARE REQUIRED BY ONE MODULE BUT PROVIDED BY ANOTHER

3.2.1 Scheduler

The scheduler provides for periodic execution of programs in the computer. Most applications programs run periodically, although the frequency and duration of each program may be different. Collectively, the applications programs must deliver correct periodic outputs at the time required. The scheduler insures that each program will be run when required.

The flight of an aircraft can be partitioned into phases and modes. Phases of flight include takeoff, ascent, cruise, descent, landing, etc. The modes of flight include manual and autopilot. The subset of active programs (out of the set of all loaded programs) is determined by the phase and mode of the flight. Whenever a phase or mode change is made, the subset of active programs may change.

Once the subset of active programs is determined, the programs are scheduled for periodic actuation. Tables and queues are maintained to accomplish the scheduling function. A priority is also maintained for each active program to determine whether an asynchronous program activation request will be honored when it occurs.

3.2.2 Input-Output Driver

The input-output driver provides the external communication capability for the other executive modules and the applications programs. Although input-output could be performed directly by the requesting program in many cases, the use of the executive skeleton as the basis for additional fault-tolerant features requires that all input and output be provided by this module of the executive. Both aircraft-computer communication and intercomputer communication are provided. The former is characterized by primarily one-word (single) data transfers while the latter is characterized by primarily multi-word (block) data transfer.

The aircraft-computer communication consists of sensor and control panel inputs and actuator and display console outputs. These communications generally require only one computer word (or a part of a word) to hold all the data. Also, these communications usually require no waiting for the device. Thus, single word input/output is usually accomplished immediately and no further action is required after returning to the calling program.

The intercomputer communication consists of data to be verified or memory words to be loaded. When verification data are exchanged for checking or voting, all pertinent state vector information must be communicated. When a recovery is being attempted through use of a memory reload, a significant portion of the memory may be affected, and thus, the volume of data may be rather large. Furthermore, the communication channel (e.g., bus) may not be immediately available. Thus, block data transfers are usually queued when the routine is called but completed as the channel becomes available. The completion of the data transfer may set a status variable or actuate a dormant program.

3.2.3 Interrupt Processor

The interrupt processor services all interrupts which are not handled directly by other executive modules. Interrupts processed by this module include executive requests and program exception interrupts such as fixed-point overflow, floating-point exponent overflow, and division by zero. This module also provides program suspension and resumption routines.

The executive request interrupt is processed by this module because several different requests are multiplexed into the one interrupt. After decoding the parameter of the request, the interrupt processor invokes the program which will process the specific request which was received. The program may be in another module or in the same module. After the request has been processed, control returns to the program which made the request unless the request resulted in a scheduling alteration.

The program exception interrupts are often maskable under program control. When one of them occurs, the interrupt processor first determines whether or not the interrupt will have any effect. If the interrupt is to be ignored, the program is resumed immediately. If an algorithm is specified (e.g., setting the result to zero for floating-point exponent underflow), it is applied immediately and the program is resumed. If the program is to be terminated because of the condition, control is transfered to the scheduler.

The program suspension and resumption routines are required for saving and restoring the machine state when an interrupt occurs. Some machines provide alternate register sets for the executive, thus minimizing the functions of this routine. However, executive pointers, tables, and queues must always be updated when switching programs.

3.2.4 Machine Error Handler

The machine error handler processes all program and machine errors which occur. Even computers which are not designed with fault tolerance as a design goal usually include address checks, memory parity checks, and other self-checks. Some computers include self-test hardware which can be used when an error is indicated. This module utilizes whichever hardware features are available on the particular machine, combined with software, to act upon any errors which occur.

Software errors (e.g., address out-of-range) which occur indicate that either the program contains a design error or the program or its data has been affected by a fault. Since programs used in aircraft applications must be thoroughly verified before being put into use, the assumption must be made that a hardware fault has occurred. (This decision could be modified for program checkout on the hardware.) Thus, whenever a software error occurs, the same procedure is followed as when a hardware error occurs.

Hardware errors (e.g., memory parity check) indicate that a failure has occurred in the hardware. The resolution of the failure depends on the fault-tolerance capability provided in the computer. If self-test is available, the module may be able to determine the error and reconfigure to eliminate the error or set an indicator to cause erroneous outputs to be ignored. If only minimal fault tolerance is provided, a degraded mode of operation may still be possible.

3.2.5 Interaction

Each of the modules of the executive skeleton implements a logically complete and distinct set of functions. Every one of the modules, however, requires functions in at least one other module. Thus, executive modules communicate with each other just as applications programs communicate with the executive modules. A brief discussion of the interaction follows. The interaction is indicated on the block diagram of the executive skeleton shown in Figure 3.2-1.

The scheduler is activated primarily by the time interrupts. The scheduler is also used by the interrupt processor to activate and terminate programs as phase and mode changes occur. It uses the program suspension and resumption routines in the interrupt processor.

The input-output driver is activated primarily by input-output interrupts. The interrupt processor passes applications programs input-output requests to this module. The machine error handler can also request this module to reload the memory. This module uses the program suspension and resumption routines in the interrupt processor.

The interrupt processor is activated primarily by the executive request and program exception interrupts. It is also used by all other executive modules for program suspension and resumption. It uses the scheduler to activate and terminate programs and passes input-output executive requests to the input-output driver.

The machine error handler is activated by hardware and software error detection mechanisms. This module uses the interrupt processor for program suspension and the input-output driver for memory reloading.

3.3 SCHEDULING MECHANISMS

The choice of a scheduling mechanism for an executive is the single most important decision in the design. The selection of the scheduling mechanism affects other modules in various degrees. For the avionics application, there exists a complete spectrum of executive scheduling mechanisms ranging from totally synchronous to constrained asynchronous. A study has been made [TSOU 73] which investigated five distinct scheduling mechanisms and their applicability to the space shuttle and space tug, applications with computational requirements not too different from those assumed for this study. The five mechanisms are investigated in detail and compared to each other as candidates for the executives in various onboard computers. The mechanisms are described here in sufficient detail to permit investigation of the effects on simulation of selecting one scheduling mechanism over another for a general avionics application.

The scheduling mechanisms considered can be differentiated by the following three characteristics:

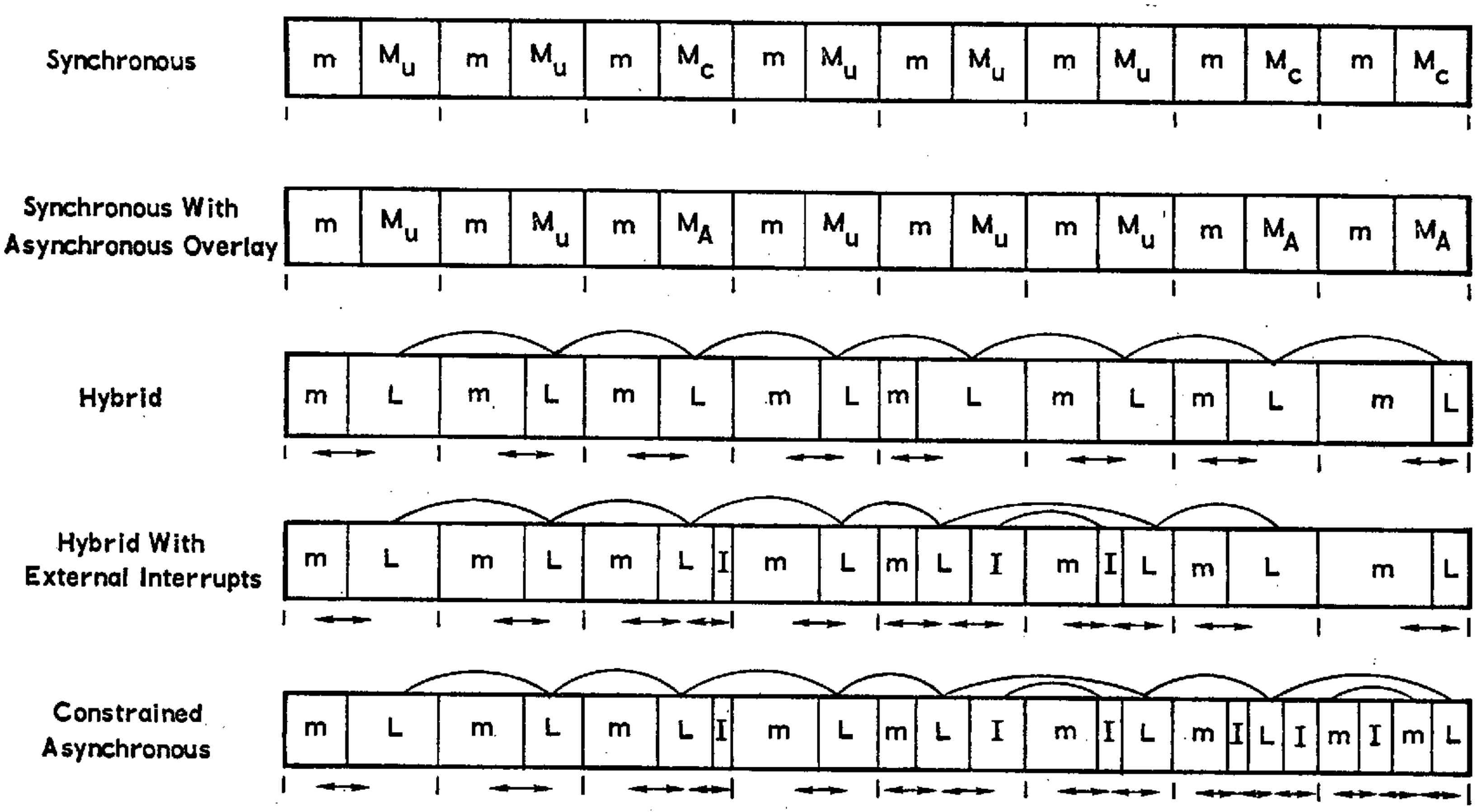
1. Fixed vs variable processing time intervals;
2. Fixed vs variable task execution order;
3. Polled vs interrupt-driven aperiodic event registration.

The synchronous executive, while limited in terms of flexibility and growth, is conceptually very simple and its behavior is completely predictable. The synchronous mechanism utilizes fixed time intervals, fixed execution order, and polled aperiodic event registration. The constrained-asynchronous mechanism is the most flexible scheduling mechanism usable for an avionics application. The constrained-asynchronous executive schedules tasks on a demand basis. It thus provides a more flexible structure which permits growth to be achieved more easily. This mechanism utilizes variable processing intervals, variable task execution order, and interrupt registration of aperiodic events (even during periodic processing). There are a number of intermediate designs which utilize various combinations of the above approaches. These are the synchronous-with-asynchronous-overlay, hybrid, and hybrid-with-external-interrupt mechanisms provide more and more flexibility. It is not sufficient to differentiate between these scheduling mechanisms by any one characteristic alone; all characteristics must be considered when comparing the mechanisms. A description of each of the five scheduling mechanisms being considered follows. Typical time lines for the various mechanisms are illustrated in Figure 3.3-1. A comparison of characteristics of the mechanisms is given in Figure 3.3-2.

3.3.1 Synchronous

The synchronous scheduling mechanism divides processing time into fixed-length slots. Two types of computations are distinguished: minor-cycle and major-cycle. Minor cycle computations are high-frequency tasks while major cycle computations are low-frequency tasks, often of longer duration than minor cycle tasks. The minor and major cycle computations are allocated to fixed time slots, arranged to form one complete cycle for all major cycle tasks. Each time slot is activated by an external interrupt. Some idle time usually exists at the end of each time slot.

The minor cycle tasks may be scheduled in one of two ways. Either the *beginning* of *every* time slot is used for minor cycle tasks or *alternate* time slots are devoted *entirely* to minor cycle tasks. The major cycle tasks are performed, one per time slot, either in the time remaining after minor cycle computations in the first case or in the slots between minor cycle computations in the second case. Two types of major cycle tasks exist:


Synchronous
m M_u m M_u m M_c m M_u m M_u m M_u m M_c m M_c
Synchronous With Asynchronous Overlay
m M_u m M_u m M_A m M_u m M_u m M_u m M_A m M_A
Hybrid
m L m L m L m L m L m L m L m L
Hybrid With External Interrupts
m L m L m L I m L m L I m I L m L m L
Constrained Asynchronous
m L m L m L I m L m L I m I L m I L I m I m L

LEGEND

m: Minor Cycle Computations

M_u: Unconditional Major Cycle

M_c: Conditional Major Cycle Computations

M_A: Aperiodic Major Cycle Computations

L: List-Processed Computations

I: Interrupts (Except Real-Time)

I: Minor Cycle or Periodic Computation Initiation

←→: Variable Boundary

FIGURE 3.3-1 EXECUTIVE SCHEDULING MECHANISM TYPICAL TIME LINES

unconditional and conditional. The unconditional major cycle tasks are executed every time their time slot occurs. The conditional major cycle tasks are performed only if specified conditions are met.

3.3.2 Synchronous with Asynchronous Overlay

The synchronous-with-asynchronous-overlay scheduling mechanism eliminates the idle time associated with conditional major cycle tasks which are not executed in the synchronous scheduling mechanism. With the overlay, unconditional major cycle tasks are allocated fixed slots as before. Conditional tasks, however, are scheduled by a master controller during time slots which are allocated to it. In this way, conditional major cycle tasks are executed on a priority basis rather than having to wait until the fixed time slot arrives for processing. The synchronous-with-asynchronous-overlay scheduling mechanism is the first to schedule any tasks other than by a fixed time slot.

3.3.3 Hybrid

The hybrid scheduling mechanism combines the minor-cycle scheduling of the synchronous scheduling mechanism with the list processing of the asynchronous mechanism. An external interrupt is used to initiate the minor cycle computations. The minor cycle tasks include a subsystem scanner which registers events required to service subsystems requesting attention. As soon as the minor cycle tasks are completed, the list processing (asynchronous) mechanism regains control. Aperiodic tasks may be registered by the subsystem scanner or by other aperiodic tasks. The hybrid scheduling mechanism is the first to use the list processing mechanism which schedules aperiodic tasks and the first to have variable minor cycle processing times.

3.3.4 Hybrid with External Interrupts

The hybrid-with-external-interrupts scheduling mechanism replaces the subsystem scanner, in the minor cycle tasks of the hybrid mechanism, with interrupts which are enabled during the asynchronous processing. The hybrid-with-external-interrupts scheduling mechanism is the first to use any interrupts other than a real-time interrupt which initiate minor cycle computations.

Scheduling Mechanism Characteristic	Synchronous	Synchronous with Asynchronous Overlay	Hybrid	Hybrid with External Interrupts	Constrained Asynchronous
Processing Time Intervals	Fixed	Fixed	Fixed/ Variable	Fixed/ Variable	Variable
Task Execution Order	Fixed	Fixed/ Variable	Fixed/ Variable	Fixed/ Variable	Variable
Cycle Position for Periodic Computations	Fixed	Fixed	Fixed	Fixed	Variable
Cycle Position for Aperiodic Computations	Fixed	Fixed/ Variable	Variable	Variable	Variable
Aperiodic Event Registration	Polled	Polled	Polled	Interrupt	Interrupt
Idle Time Distribution	Dispersed	Dispersed	Concentrated	Concentrated	Concentrated

FIGURE 3.3-2 COMPARISON OF EXECUTIVE SCHEDULING MECHANISMS

3.3.5 Constrained Asychronous

The constrained-asynchronous scheduling mechanism is the most flexible which could be used in an avionics application. The constraints applied include required precise, periodic computations; debugged, cooperating programs; and a minimum number of interrupts. When this scheduling mechanism is used, interrupts are enabled most of the time. The tasks are scheduled using a cyclic list which contains entries such as interrupt-scheduled tasks, studies, queues, and ordered lists. A real-time interrupt initiates a special task which processes the high-frequency tasks which are on a special ordered list. The constrained asynchronous scheduling mechanism is the only one to allow interrupts at almost all times, even during the periodic high-frequency computations.

3.3.6 Comparison and Contrast

The five executive scheduling mechanisms which are being considered here range from the totally synchronous mechanism to the constrained asynchronous mechanism with three other designs between. The synchronous mechanism is the least flexible but easiest to verify. All computations are scheduled at a fixed time and are limited to a specified amount of processing time. No asynchronous processing is allowed and conditional tasks are allocated time even if they are not run. The synchronous-with-asynchronous-overlay mechanism improves on this by allocating some of the time slots to aperiodic event scheduling. Thus tasks which are conditional can be scheduled only when the conditions exist which as required for processing. The hybrid mechanism goes one step further by allocating all non-minor-cycle time to asynchronous scheduling. This is the first use of the list-processing algorithms employed in the asynchronous mechanism. The hybrid-with-external-interrupts mechanism uses interrupts to register aperiodic events rather than the polling mechanism used in the other three mechanisms. The asynchronous mechanism allows interrupts even during periodic events. A cyclic list is used to schedule tasks. Periodic tasks are activated by a high-priority interrupt.

Thus it can be seen that the scheduling mechanisms considered range from the synchronous in which everything is fixed to the constrained asynchronous where everything is variable. The synchronous mechanism is the easiest to verify because everything is fixed. As more asynchronism is introduced, verification becomes more and more difficult because more and more variability. The asynchronous mechanism, in which almost everything is variable, can never be totally verified because the number of combinations of events is very large. All that can be done is to test all branches in a reasonable number of ways.

The choice of an executive scheduling mechanism is made on the basis of the environment, the machine capabilities, and the applications programs requirements. Once the choice of an executive scheduling mechanism is made, the other portions of the executive can be considered.

3.4 ADAPTABILITY

Adaptability is an important design goal for the executive. This goal can be interpreted in two different contexts: configuration adaptability and scheduling adaptability. Configuration adaptability is the ability to adapt to any configuration considered by the RCS study. Each different configuration imposes a slightly different set of requirements upon the executive although each configuration accomplishes the same tasks for the avionics environment in which it resides. Thus adaptations of the same executive skeleton should provide executives for all configurations. Scheduling adaptability is the ability to use any scheduling mechanism from totally synchronous to constrained asynchronous. The synchronous scheduling mechanism implements a strictly fixed time and order for task processing while the constrained asynchronous scheduling mechanism permits demand scheduling except for strictly-periodic, high-frequency events.

The executive skeleton described in this section provides both configuration adaptability and scheduling adaptability. The same basic set of four modules (scheduler, input-output driver, interrupt processor, and machine error handler) is used for all executives. Additional modules and/or enhancement of the basic four modules are provided to satisfy the requirements imposed by the various configurations. The various scheduling mechanisms can be implemented internally within the scheduler module with minor changes in the interrupt handler module. It should be noted that even for the totally synchronous executive, machine error conditions must be processed immediately when they occur.

The RCS configurations being considered by this study are all to be used in an avionics environment. The results of the study, however, should be usable for other environments such as spacecraft, process control, etc. Thus, although the executive is designed to support an avionics environment, it is not constrained to that environment. Even in the avionics environment, a wide range of applications exists. Furthermore, the purpose of undertaking the design of an executive is to provide a model for the simulator which is being written. Thus the level of design of the executive is that which is necessary to support a model for the simulator. The level which has been chosen, the module level, provides complete generality while still specifying the structure and functions of the executive.

3.5 SOFTWARE STRUCTURE AND FAULT-TOLERANCE IMPLEMENTATION

3.5.1 Software Structure Considerations for a Duplex System

It is important to study the impact of the executive scheduling on a duplex system since adaptive configurations are able to degrade to duplex.

The structure of the software used in a duplex computer configuration has an effect on the probability of recovery from a fault given detection of an error. The software features which must be investigated to determine the effects include the executive scheduling mechanism, a typical cycle of minor-and major-cycle computations, redundancy requirements, and tradeoffs which must be made between conflicting requirements.

3.5.1.1 Executive Scheduling Mechanisms

Evaluation of scheduling mechanisms which could be used in a duplex computer configuration leads to a dichotomy of the mechanisms presented in Section 3.3. The breakpoint is the inclusion of an asynchronous mechanism which does not require segmenting of major cycle computations into pieces which can be completed between Real Time Interrupts (RTI's). The synchronous and synchronous-with-asynchronous-overlay mechanisms do require such segmentation. These mechanisms will be called synchronous-type mechanisms in this discussion. The hybrid, hybrid-with-external-interrupts, and constrained-asynchronous mechanisms do not require segmentation of major-cycle computations and will be called asynchronous-type mechanisms.

The requirement that all programs running on a duplex system must incorporate rollback has a significant effect on the choice of a scheduling mechanism. For minor-cycle computations, a rollback point can be established at the end of each iteration. If new data are input immediately prior to the next computation cycle, the input mechanism must be protected separately. In any case, it is assummed that the previous state vector and the new input data are correct and available at the beginning of a minor-cycle computation. If the minor cycle completes successfully, a new state vector is stored and the major-cycle computations are performed. If an error occurs, the minor-cycle computations can be repeated, possibly at a cost of delaying the major-cycle computations. Thus the rollback structure of minor-cycle computations corresponds with the scheduling of the computations for any of the scheduling mechanisms (except for the constrained-asynchronous mechanism where minor-cycle computations can be interrupted briefly).

For major-cycle computations, the choice of a synchronous-type or asynchronous-type scheduling mechanism has significant consequences. If a synchronous-type scheduling mechanism is chosen, then the major-cycle computations can have a rollback point at the end of each segment. At the end of a segment the state vector is known and is usually reasonably small. Each segment of a major-cycle computation is similar to the minor-cycle computations in that the rollback structure and the scheduling of the segment correspond. If an asynchronous-type scheduling mechanism is chosen, the situation is entirely different. Rollback points will not correspond temporally to processing segments. Zero, one, or several rollback points may be established during one computation period. Furthermore, there may not be time to complete a rollback before another minor-cycle is initiated. A rollback point could be forced by storing every variable the program uses when an RTI occurs at a cost of time and memory.

The interaction between the minor-cycle computations and the major-cycle computations depends on the choice of a scheduling mechanism and comparison period for major-cycle computations. If a synchronous-type scheduling mechanism is chosen, the rollback structure corresponds to the scheduling so each execution segment is independently protected by its own rollback mechanism. If an asynchronous-type scheduling mechanism is chosen, a fault can cause a rollback in more than one program segment. If a fault occurs during a major-cycle computation which is interrupted by a minor-cycle computation, the minor-cycle computation can get an error, causing rollback and then the major-cycle computation can get an error when it reaches a rollback point, resulting in another rollback. The comparison period for major-cycle computation determines the length of rollbacks. If a comparison is made only when an output occurs, the entire computation must be repeated if an error occurs. If rollback points with comparisons are inserted at a reasonable number of places in each major-cycle computation, errors will be caught sooner and rollbacks will be shorter. Also, the chance of an error affecting both a minor and major cycle is reduced if more rollback points are used. However, the time required for the complete major-cycle computation is increased.

3.5.1.2 Typical Computational Cycle

A typical computational cycle consists of all processing between two consecutive Real Time Interrupts (RTI's). Immediately following an RTI, minor-cycle processing is performed. After the minor-cycle processing is completed, the remaining time before the next RTI is used for major-cycle processing.

The minor-cycle processing consists of high-frequency activities which must be performed regularly. These activities include input, calculation, checking, and output of high-frequency control information and memory copy when it is active. The input, calculation, checking, and output must be protected by rollbacks. Furthermore, the original input should by used again if a rollback occurs. A way of accomplishing this is to use three minor-cycle periods for a complete iteration. A minor cycle would begin by issuing the output data generated by the calculations during the last minor cycle. Calculations would then be performed on input data obtained during the last minor cycle with checking of the results. Then input would be obtained for the next minor cycle. Finally, a rollback point would be established for all minor-cycle processing up to that point.

The last portion of a minor cycle is devoted to memory copy when it is active. If the other computer has suffered a suspected memory failure, and if memory copy is considered a viable option for the application, the memory of one computer can be copied to the other. When the memory copy is active, however, the system is running in the simplex mode, since one computer contains a fault. Thus no rollback is included in the programs which implement memory copy. A fault in the sending computer during memory copy would constitute a second fault, while another fault in the receiving computer would not change its faulty status. During each minor cycle, as many words as possible would be transmitted, subject to the limitation that the multiplexed bus must be available for input and output during the next minor cycle.

When the minor-cycle processing is successfully completed, major-cycle processing is initiated or resumed, depending on the scheduling mechanism. The major cycle continues until it completed or is terminated by another RTI, again depending on the scheduling mechanism. Major-cycle processing consists of low-frequency activities which occur regularly and aperiodic activities which respond to external or internal stimuli. If no other major-cycle processing is required, self-test is run. The self-test may also be scheduled as one of the regular major-cycle activities to insure a minimum amount of self-test. Major-cycle computations tend to be longer than minor-cycle computations. Major-cycle computations would be likely to have several rollback points in a single task. As an absolute minimum, rollback points would be required after input, calculation and output. The rollback point after input would protect the input. More than one rollback point during calculation, with checking, would prevent completely restarting the task in case an error occurred. Finally, a rollback point after output would prevent repeated outputs.

The rollback structure of a period between two RTI's has three distinct sections. During the minor-cycle processing preceding the memory copy, all computations can be protected by a single rollback point at the end. Once this rollback point is established, the second section, the memory copy, is initiated if it is active. As stated above, the memory copy is not protected by rollback. Finally, the major-cycle computations each have one or more rollback points which may or may not correspond to RTI intervals, depending on the scheduling mechanism.

3.5.1.3 Redundancy Requirements

The use of duplex redundancy imposes three requirements on the hardware and software which are not required in a simplex system. First, increased memory is needed for state vector storage and comparison programs. Second, increased execution speed is required for time to do the same function. Third, rollback must be included in the software. In addition to these three requirements, reconfiguration mechanisms must be provided if the duplex operation is part of an adaptive system which degrades from TMR to duplex to simplex.

The increased memory required for duplex redundancy is used to store the state vectors at program rollback points and for programs which compare state vectors control reconfiguration. The state vectors must be saved by double buffering or an equivalent technique which preserves the information from a previous rollback point until a new one is established. Programs are also required to store the state vector and to reload it from the same area when a rollback is required. These programs may be executive routines provided for use by application programs or code in each of the application programs themselves. It is preferable for the code to exist in a set of executive routines to minimize duplication and to provide for adaptive redundancy by reconfiguration.

The increased speed is required to provide time for storing and comparing the state vectors. The time which is required to store the state vectors depends on the length of each state vector which is stored and the frequency at which they are stored. Increased time is also needed to compare the state vectors when they are stored and/or compare outputs before they are issued to aircraft actuators. When a failure occurs, time is needed to reload the state vector, repeat the computation which was in progress when the error occurred.

Rollback capability must be provided for programs running on a duplex computer configuration in order to determine the faulty computer when an error occurs. When the results of the two computers differ, each must rerun the program first to check for a transient error. If the faulty computer cannot be determined, a self-test must be used. When the faulty computer is determined, it must be removed from actively controlling the aircraft. Then the remaining good computer must continue in the simplex mode.

An extra redundancy requirement is imposed if adaptive redundancy is being used. Some additional speed and memory will be required for reconfiguration. The system life will be maximized by progressively degrading from TMR to duplex to simplex as failures occur. If a failure is due to memory damage, memory copy may be used to correct the fault and upgrade to a mode using one more computer. Changing modes, however, requires special considerations. Adaptive TMR requires rollahead while duplex requires rollback. The same code can be used to save and compare the state vectors. The primary difference is whether the segment in which the error occurred is repeated or the state vector is corrected and the program continues.

3.5.1.4 Tradeoffs

The three major tradeoffs which must be considered when determining the value of ρ for a duplex configuration are the scheduling mechanism, the frequency of rollback points, and the lengths of the state vectors. In all cases, the software influences the v_2w_2 factor (see Section 5.5.1), the probability of correct recovery given detection and correct diagnosis of the faulty unit. A summary of the tradeoffs and the effects on v_2w_2 is presented in Table 3.5-I.

The choice of a scheduling mechanism has a significant effect on v_2w_2. If a synchronous scheduling mechanism is used, the rollback segments can be established at the end of each time slot. Furthermore, a synchronous scheduling mechanism has idle time distributed to a part of each time slot. Thus the synchronous scheduling will yield a higher value of v_2w_2 since recovery will be faster (at most one rollback series is needed per error) and the time is available for immediate rollback without delaying other computations (due to the distributed idle time).

ITEM	LARGER v_2w_2	SMALLER v_2w_2
SCHEDULING MECHANISM	SYNCHRONOUS	ASYNCHRONOUS
	Faster Recovery	Slower Recovery
	Distributed Idle Time	Concatenated Idle Time
Number of Rollback Points	More	Less
Length of Rollback Segments	Shorter	Longer
	Faster Recovery	Slower Recovery
State Vector Length	Shorter	Longer
	Faster	Slower

TABLE 3.5-I SOFTWARE EFFECTS ON v_2w_2

The frequency of rollback points also affects v_2w_2. The greater the number of rollback points which are inserted, the faster the recovery will be when a rollback is required. At the extreme, the length of the state vector and the time required to store it can become significant, but in most cases this is small relative to the processing time of the rollback segment. A greater number of rollback points also increases v_2w_2 when transient duration is considered. If a transient is of relatively-long duration, more than one rollback may be required before the transient fault disappears. The shorter the length of the rollback segment (greater the number of rollbacks), the faster the successful one will complete.

Finally, the lengths of the state vectors affects v_2w_2 in two ways. Shorter state vectors require less memory which results in a larger v_2w_2 value. Also, the shorter the state vector, the shorter the time needed to save and restore it, resulting in greater probability of recovery in time.

3.5.2 Software Structure Considerations for a TMR System

Although this section considers the TMR case, its conclusions are also valid for 4-MR and 5-MR systems.

This section considers for TMR the same topics which were considered for the duplex configuration in Section 3.5.1. The probability of recovery given detection, v_2w_2, is used only in adaptive TMR when running in the duplex mode. The topics discussed in this section are executive scheduling mechanisms, minor-and major-cycle computational requirements, redundancy requirements, and tradeoffs between conflicting requirements.

3.5.2.1 Executive Scheduling Mechanisms

The executive scheduling mechanisms which could be used in a TMR configuration again are separated into the synchronous-type which require segmentation of major-cycle computations into pieces which can be completed between Real Time Interrupts (RTI's) and the asynchronous-type which do not require segmentation. Programs running on an adaptive TMR configuration, which can degrade to duplex, require rollback segmentation. When the system is used in the TMR mode, rollahead is used when an error occurs, since a majority vote is possible. When the system degrades to the duplex mode, rollback is used when an error occurs.

For minor-cycle processing, a rollahead/rollback point can be established at the end of the minor-cycle computation period for any of the scheduling mechanisms. For major-cycle processing, however, rollahead/rollback points will correspond temporally to processing segments only if a synchronous-type scheduling mechanism is used. If an asynchronous-type scheduling mechanism is used, zero, one, or several rollahead/rollback points may be established during one computation period.

If a synchronous scheduling mechanism is used, an error which occurs will only affect the active processing, since rollahead/rollback points will be established at the end of each processing segment. If an asynchronous-type scheduling mechanism is used, however, an error may affect an interrupted major-cycle computation as well as a minor-cycle computation. In the TMR configuration no major problem exists when this occurs, since rollahead requirements are not critical in comparison to rollback requirements. The time required to update the state vector is small relative to the time needed to repeat a program segment.

3.5.2.2 Typical Computational Cycle

The structure of a typical computational cycle, which consists of all the processing between two Real Time Interrupts (RTI's), does not vary with configuration changes. Minor-cycle processing always immediately follows the RTI, after which the major-cycle processing uses the remaining time before the next RTI.

Although the structure of a typical computational cycle is more dependent on the scheduling mechanism used than on the amount of redundancy the system is using (TMR, duplex, etc.), the details of the rollahead/rollback and reconfiguration code are different. In the TMR configuration, the faulty computer can be identified by a majority vote. A rollahead is used first to attempt to correct the faulty computer. If this fails, a memory copy can be tried. In any case, no significant amount of time is lost while applying these procedures. The two computers which do not have any errors continue to perform the required computations while also attempting to correct the faulty computer.

3.5.2.3 Redundancy Requirements

Redundancy requirements for TMR include memory, speed and rollahead. Increased memory is required for programs which compare the state vectors at rollahead points. Increased execution speed is required for state vector correction. Rollahead is used to correct a faulty state vector when it is damaged by an error. The use of adaptive TMR also requires reconfiguration mechanisms which can change the system to a duplex mode of operation and back to TMR.

The only requirement for additional memory in a TMR configuration over simplex is for the checking and reconfiguration programs. One important function is the comparison of state vectors at the end of a rollahead segment. If an error is detected, the erroneous state vectors corrected by a majority vote and computation continues (rollahead). If adaptive TMR is used, provisions must also be included for reconfiguring to duplex mode when an error is detected. The system will run in the duplex mode while attempting to restore the faulty computer to operational status. If the faulty computer recovers, the TMR mode of operation can be resumed, otherwise duplex operation continues.

A moderate speed increase is needed for state vector comparison, rollahead and reconfiguration if used. The state vector comparison is a short task. If an error is detected, the state vector can be corrected by using a majority vote among the three computers. Although this task is also short, it is essential that it be performed rapidly, since all system computation is suspended during rollahead. The reconfiguration process also must be rapid when it is required, since no other activity may be in progress during reconfiguration.

The incorporation of rollahead in programs running on a TMR configuration provides the capability of restoring a faulty computer to service when a transient error occurs. When an error is detected in a state vector, the state vectors in the two good computers can be used to correct the state vector in the faulty computer. If an error remains after rollahead, the system can be reconfigured to duplex while the memory of the faulty computer is reloaded. If the error is removed, the system can again be reconfigured to TMR.

3.5.2.4 Tradeoffs

The two major tradeoffs which must be considered for an adaptive TMR configuration are rollahead/rollback structure and the memory copy option.

There are essentially four different TMR configurations which can be considered, each increasing the recovery mechanisms provided. The basic configuration is classical TMR with no recovery for a faulty machine. Some improvement can be expected if rollback is included in each of the computers with comparison with the state vector of the other machines. A better configuration includes rollahead rather than rollback so that an erroneous state vector is corrected by a majority vote and no time is lost for recovery. Finally, the inclusion of memory copy would permit correcting damaged memory words in a computer. All configurations except the classical TMR can be adaptive so that they degrade to duplex and then simplex as permanent failures occur.

The use of adaptive TMR requires that programs include rollback capability for use when running in the duplex mode. If rollback rather than rollahead is used for TMR, the same mechanisms can be used for duplex and TMR operation. The availability of the third computer, however, provides a voting capability which can be used to eliminate the time penalty imposed by rollback. The cost is in extra memory for the rollahead mechanism and extra complexity which is introduced into programs which must include provisions for using either rollahead or rollback.

The inclusion of a memory copy capability requires time, memory, and complexity. Time must be provided during each minor cycle when memory copy is active to transmit words between computers. Memory is required for buffer space and for programs which accomplish the data transfer. Also, additional reconfiguration complexity is introduced. The advantage of using memory copy, however, is that a computer in which a transient fault damages memory may be corrected and the configuration returned to TMR, resulting in correction of errors by rollahead rather than rollback. Thus, there will be three good computers operating in case a fault occurs in one of the computers which did not have the initial memory fault.

The same considerations which were discussed in Section 3.5.1 pertaining to effects of software on ρ are also applicable to adaptive TMR degraded to the duplex mode. As can be seen from Figure 3.5-1, all software considerations ultimately lead to a consideration of the recovery time. With three computers running, at least two of which are good, recovery is immediate by voting, thus software considerations are not involved until the system degrades to duplex operation.

THIS PAGE INTENTIONALLY LEFT BLANK

4.0 MEASURES OF FAULT-TOLERANCE

4.1 THE CONCEPT OF FAULT-TOLERANCE

4.1.1 The Reliability Problem for Computers

The reliability problem in computer operation is caused by imperfections in the physical implementation of the logic structure. Reliability theory defines the reliability of a system as the probability of correct operation up to the "mission time" t=T, given that the system was operating correctly at the "starting time" t=0. Computer systems are a special case among all physical systems because in their case "correct operation" means the correct execution of a set of programs, rather than the continued functioning of a set of components of the system. It is the purpose of this section to present a unified view of those aspects of computer system design that are specifically directed toward the assurance of correct program execution in the presence of physical imperfections (called *faults*) in the components of the system. (AVIZ 72)

The following four criteria form an operational definition of "correct execution of a set of programs:"

1. The programs and their data are not altered or halted by faults;
2. The results of operations do not contain fault-caused errors;
3. The execution time of each program does not exceed a specified limit;
4. The storage capacity that is available for each program remains above a specified minimum value.

It is to be noted that the definition excludes the question of correctness of the programs and of accuracy of the algorithms, both of which are separate fields of study.

The set of programs and data, the definitions of required operations, the time limits for program execution, and the storage requirements are specified by the users of the system. The goal of the designer is to raise the system reliability (i.e., the probability of correct execution of these programs) to an acceptably high value, given that faults may occur during execution. Such faults are caused by three classes of physical events that affect the hardware:

1. Permanent failures of computer components;
2. Intermittent malfunctions of components;
3. External interference with computer operation.

When these events occur, they cause logic faults, defined as the deviations of one or more logic variables within the computer system from their design-specified values.

4.1.2 "Fault-Intolerant" Design for Reliable Operation

There exist two complementary approaches that can be employed to attain satisfactory reliability of computer systems. They are designated as: 1) fault-intolerance, and 2) fault-tolerance, respectively.

Fault intolerance is an approach that aims to reduce the probability of occurrence of the first fault during a specified time interval $0 \leq t \leq T$ to an acceptably low value. In this approach the system is designed without redundancy, and every component of the system must function correctly in order to assure correct program execution. The procedures which lead to the attainment of "fault-intolerant" reliable systems are:

1. The most reliable components are selected for the system within the existing cost and performance constraints;
2. Proven techniques are employed for the interconnection of components and assembly of subsystems;
3. The system is packaged to screen out the expected forms of external interference;
4. Quantitative prediction of the system reliability is made on the basis of known or predicted failure distributions and rates for the components and interconnections.

In the "purely" fault-intolerant (non-redundant) design, the probability of fault-free hardware operation is equated to the probability of correct program execution. This design is characterized by the decision to invest all the reliability resources into procurement of high-reliability components and refinement of assembly and packaging techniques.

4.1.3 Design of "Fault-Tolerant" Systems

An alternative to the "purely" fault-intolerant approach is offered by the use of various forms of redundancy. Known as fault-tolerance, this is an approach that increases computer system reliability by the use of design techniques that allow faults to occur without disrupting the continued correct execution of its programs. Fault-tolerance does not entirely eliminate the need for reliable components; instead, it offers the option of allocate part of the reliability resources to the use of redundancy. The end goal of a fault-tolerant design is either: 1) a system reliability prediction that cannot be attained by the purely fault-intolerant design; or 2) a system reliability prediction that matches the purely fault-intolerant design at a lower overall cost of implementation.

A fault-tolerant computer system is defined as possessing the following three attributes:

1. It consists of a set of components (hardware) and programs (software);
2. It is initially free of design errors;
3. It executes its set of programs correctly in the presence of faults.

The first attribute stresses the fact that the ability of a computer system to continue operating correctly in the presence of faults depends not only on the properties of the hardware, but also on the nature of the software--both the system programs and the user (application) programs. For example, the ability to recover from the errors caused by transient faults frequently depends on special restart features incorporated in the system software as well as on proper partitioning and state vector storage of user programs.

The second attribute requires that design errors be eliminated from both hardware and software prior to the initiation of fault-tolerant computing. Design errors are caused by errors in the translation of the original system specifications into the operational forms. They are eliminated by validation of the hardware and software designs prior to their operational use. Since a complete a priori verification cannot yet be assured, computer systems also need provisions to detect and trap various abnormal conditions during operation which may be symptoms of remaining design errors. A completely fault-tolerant operation is attained only when all design errors are eliminated from the system.

The third condition for fault-tolerant computing postulates correct execution of the entire set of programs in the presence of faults. Program errors that are caused by faults in the hardware can be avoided or corrected by means of protective redundancy. Protective redundancy is introduced into the computer system in three forms:

1. Additional hardware (hardware redundancy);
2. Additional programs (software redundancy);
3. Repetition of machine operations (time redundancy).

These redundant features would not be needed in a fault-free computer. Given that faults will occur in the hardware, the redundant features provide a fault-tolerant computing system, which carries out its programs correctly in the presence of faults. Partial fault-tolerance ("graceful degradation") results when one or more programs fail to satisfy criteria (1) or (2), and also when some (or all) programs fail to satisfy criteria (3) or (4) for correct execution (Paragraph 4.1.1).

Research results and design experience that have been accumulated during the past decade show that the systematic introduction of protective redundancy to provide fault-tolerance in a computer system can be accomplished by the following design procedure.

1. The computational requirements are established and the system architecture is specified with the initial assumption that faults will not occur (the "fault-intolerant" design).
2. The classes of faults that are to be tolerated in the design of (1) are identified, and the extent of tolerance is specified for each class of faults.

3. The most cost-effective methods of protective redundancy (time, hardware, software) are chosen to cover every class of faults identified in (2), and the system architecture is modified to incorporate the redundancy.

4. Analytic or heuristic techniques are applied to estimate the extent of fault-tolerance that is provided by the methods of protective redundancy selected in (3).

5. Checkout methods are devised to test all redundancy features specified in (3). Where applicable, fault-tolerance features are extended to effect automatic maintenance of peripheral systems that are connected to or controlled by the computer.

Design experience has shown that the initial results of task (4) often lead back to (3), and that several iterations of (3) and (4) may be necessary to arrive at a satisfactory fault-tolerant system architecture. The measures applicable to task (4) are discussed in the next section.

4.2 QUANTITATIVE SPECIFICATION OF FAULT-TOLERANCE

4.2.1 Classification of Measures

There are three distinct quantitative measures that can be applied to measure the fault-tolerance of a computer system. They are:

1. The Discrete Fault Tolerance $\underline{d}$

2. The Reliability R(t)

3. The Survivability S(t)

The discrete fault tolerance (DFT) $\underline{d}$ is a deterministic measure that specifies how many faults of a given class can be tolerated by a computer system or by a module of the system. The remaining two measures - reliability R(t) and survivability S(t) - are probabilistic measures that predict the probability of the system continuing its correct operation over a specified time interval. The three measures are discussed in the following parts of this section.

4.2.2 Discrete Fault Tolerance (DFT)

DFT is defined as the ability of a Module Set M to operate correctly for at least d faults within the Module Set. The value of d is an integer:

$$d(M)\geq 1$$

in a fault-tolerant module set M.

The DFT measure applies to permanent faults that are taken from a specified Fault Set F. The fault set F must also be explicitly stated for a complete DFT specification, i.e., we have:

$$d(M,F)\geq 1$$

It is important to note that DFT is not a function of time, i.e., the probability of continued correct operation is stated to be unity as long as not more than d faults from the fault set F occur within the module set M. In practical DFT implementations for which $d(M,F)\geq 2$ is specified it may be necessary to specify a minimum time interval T between successive faults. The interval is needed in the case of dynamically redundant systems that require a recovery procedure to be completed before the next fault occurs. This gives the specification

$$d(M,F,\Delta T)\geq 2$$

In the case of $d = 1$ the value of ΔT is not important, since the second fault is assumed to lead to system failure.

As defined above, the DFT is a "worst case" specification of fault-tolerance, since d refers to the most critical set of faults. Given that faults other than the "critical faults" occur, the module set M is usually able to survive more than d faults.

The Module Set M itself refers to a redundant set of modules, in which a "module" may be an entire computer, a functional subsystem, a logic package, or even a single physical component of the system. For example, $d = 1$ applies to both a "quadded" set (M_1) of diodes (Figure 2.2-1) and to a "TMR" configuration (M_2) of complete computers (Figure 2.2-2). In both cases the fault set F includes independent failures of single units. The "unit" is one diode in M_1, and one computer or one voter in M_2.

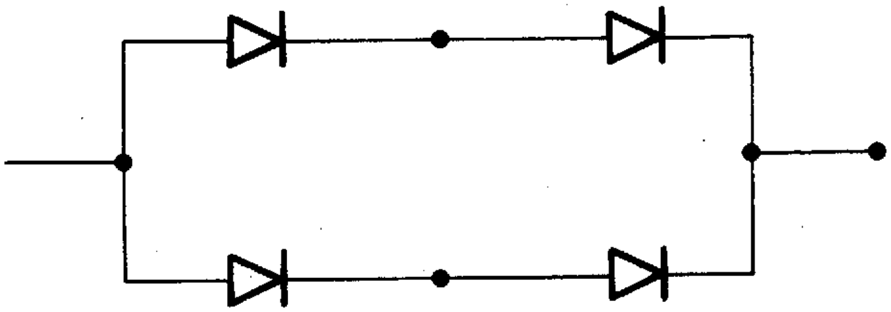


FIGURE 4.2-1 QUADDED DIODES, d = 1

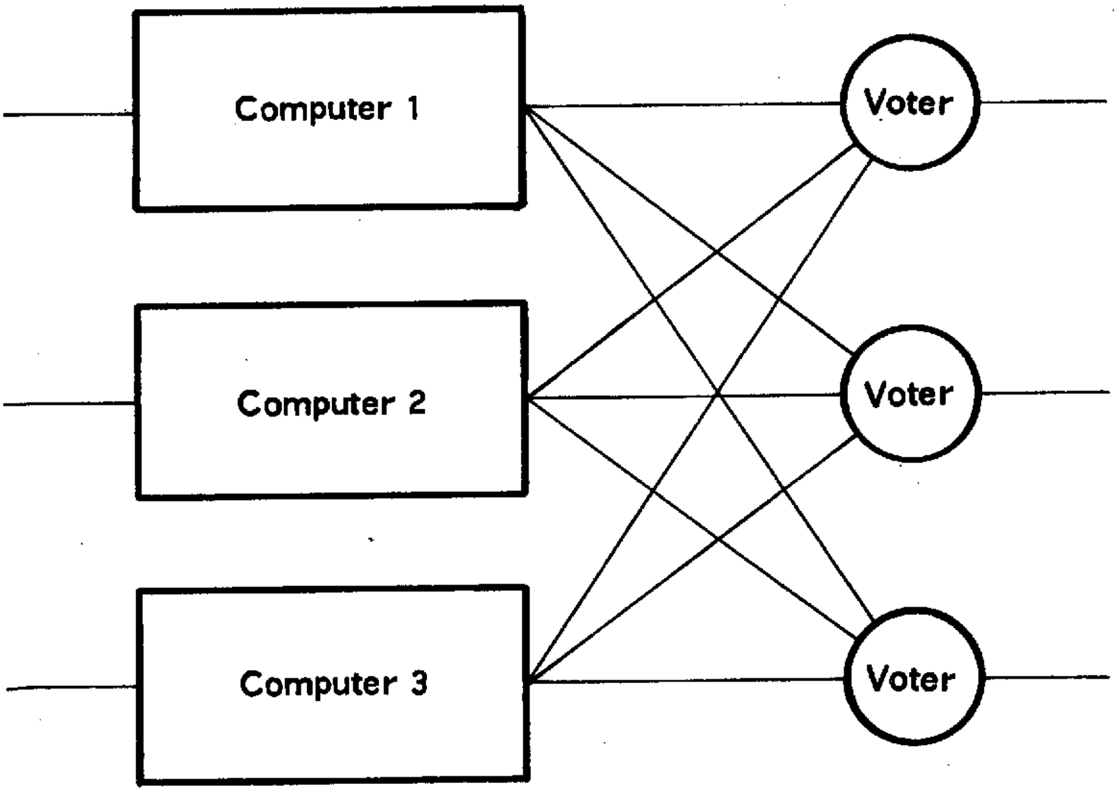


FIGURE 4.2-2 COMPUTERS IN TMR CONFIGURATION

A common extension of the DFT specification is a "fail-safe" condition for the (d+1)$^{\underline{st}}$ fault. This means that after d faults have occurred, the next "worst case" fault will lead to a systematic shutdown of the function performed by the module set M. An example is the "FO-FO-FS" specification, which translates to d=2 and a fail-safe condition for the third fault in the worst possible location within the module set M. The set M is usually composed of system "modules" in this specification.

4.2.3 Reliability

The reliability R(t) also refers to a set F of permanent faults that can occur in the hardware module set M. It is defined as the probability that the set M will not experience a disabling hardware failure during a specified "mission time" interval $0 \leq t \leq T$. When a system is composed of several module sets M_i with reliability $R_i(t)$, it is usually assumed that all module sets must survive in order for the system to survive, i.e.,

$$R(t)_{sys} = \prod_{i=1}^{n} R_i(t)$$

for a system composed of n module sets M_i $(1 \leq i \leq n)$.

The current state of the art in reliability modeling (BOUR 71) specifies the reliability of a module set M_i with respect to a fault set F in terms of seven parameters as

$$R(M_i, F) \triangleq {}^{f}_{c}R^{q}_{s}\ (T, \lambda, \mu)\ ^{*}$$

where the parameters are defined as follows:

$q \triangleq$ quota $\triangleq$ number of modules within the set M_i required to survive to time T

$s \triangleq$ sparing $\triangleq$ number of modules provided as spares within M_i

$c \triangleq$ coverage $\triangleq$ conditional probability [system recovers| module fails]

$f \triangleq$ discrete fault tolerance within one module of M_i

$\lambda \triangleq$ power-on failure rate for one module of M_i

* The symbol $\triangleq$ means "is defined as."

$\mu \triangleq$ power-off failure rate for one module of M_i

$T \triangleq$ mission time

The analytic expressions for R(t) as a function of these parameters are found in BOUR 71.

4.2.4 Survivability

It is known from experience that computer systems are also subject to transient faults, which can terminate the correct execution of a set of programs without causing a disabling hardware failure in the module set M_i. Our goal is to incorporate the survival probability with respect to the occurrence of transient faults into one probabilistic measure of fault-tolerance that also contains the reliability R(t) as defined in the preceding section. This measure will be called the survivability S(t) of the module set M_i.

In order to include transient faults, it is necessary to define a transient fault set F' which is described by two properties: their arrival characterization and their duration (AVIZ 72). Both properties are statistical in their nature. The arrival time of a transient fault is a discrete random variable, while its duration is described by a probability density function. These concepts are illustrated by specific examples in Section 5.2.

In a dynamically redundant system both transient and permanent faults require a two step fault-tolerance procedure:

1. The existence of the fault is detected.
2. A recovery action takes place.

The detection of permanent faults may be by means of periodic diagnosis or by concurrent error-detection procedures; only the latter are suitable to detect transient faults. In the present study we assume concurrent error detection by a comparison of the outputs of two or more copies of identical modules. The same comparison procedure will detect errors caused by both transient and permanent faults; however, at the time of detection it is not known which type of fault has been detected.

The recovery procedure first must distinguish whether a permanent or a transient fault has occurred, next the faulty module must be located (identified). After fault location, an appropriate corrective action is implemented. In the case of a transient fault, it consists of bringing the affected module back into correct operation ("rollback" or "rollahead"); in the case of a permanent fault, it requires the continuing of correct operation without assistance from the failed module. The failed module may be removed (e.g., in replacement, hybrid-redundant, adaptive systems), or it may remain in the working module set (e.g., in TMR systems).

In the case of dynamically redundant systems, the time requirement becomes an important parameter. We distinguish the following time intervals that affect the success of recovery of a dynamically redundant system:

1. Time interval from the occurrence of the fault to its detection. During this time the fault continues to affect the computation, and errors may proliferate in the program being executed.
2. A specified time delay before the recovery action is initiated. This delay is part of the recovery function; computing is suspended during this time. The function of the delay is to allow transient faults to end before recovery is initiated. A fault that lasts longer will be treated as a permanent fault.
3. Time interval needed to execute the recovery action. At the end of this interval the system is again in an operational state. This state is identical to the pre-fault state after a successful recovery from a transient. After a successful recovery from a permanent fault, the system enters the specified "next state", which depends on the redundancy technique employed.

The recovery action itself consists of two components:

1. The fault must be <u>located</u> by identifying the module that has been affected by the fault.
2. The operational state must be re-established by an appropriate technique.

Each component requires a time interval and has a certain probability of unsuccessful execution. Furthermore, we note that certain complications may occur after a transient fault has been detected:

1. Its duration may be long enough to overlap into the recovery period and thus create the false indication of a permanent fault.

2. A second transient fault may occur before recovery is complete. If it affects the same module, the effect is the same as that of a "long" transient discussed above; if it affects a different module, recovery may become impossible. Both of these possibilities must be incorporated in a realistic model of a fault-tolerant system.

4.2.5 Quantitative Measures of Survivability

The preceding discussion has identified several functions that must be successfully executed before the system returns to its operational state after a fault event. It is convenient to partition the probability of successful system response to a fault into three components:

1. Detectability, denoted by u and defined as the probability that fault is detected, given that it occurs;

2. Diagnostibility, denoted by v and defined as the probability that the faulty module is correctly identified, given that the fault has been detected;

3. Recoverability, denoted by w and defined as the probability that the operational state is successfully re-established, given that the fault has been located.

A time interval is associated with each one of the three measures. It may be given either as a fixed value for a "worst case" upper bound, or as a random variable with a specified density function.

In order to generate a reasonable estimate of the probabilities and their associated times, we need some fundamental information about the hardware organization and about the software of the fault-tolerant system.

The hardware information includes:

1. The description of the fault-detection mechanisms;

2. The description of the sequence of operational states after permanent faults have been identified and recovery has succeeded;

3. The description of hardware aids for fault-location and execution of recovery;
4. The description of inter-module communication paths that may serve the purposes of fault-tolerance.
5. The identification of possible related failure modes (affecting more than one module simultaneously) and their probabilities.

The software information includes:

1. Fault-tolerance features of the executive;
2. Nature of available test and diagnosis programs;
3. Scheduling mechanism for application programs;
4. Time available for recovery purposes;
5. Constraints on restarts (singular events, real-time interrupt scheme, etc.)

The probabilities and times for the survivability measures are derived from the above information. Survivability then is determined either by analytic modeling or by simulation, as described in the following Sections 5 and 6.

5.0 ANALYTIC MODELING

5.1 MODELING APPROACH

5.1.1 General

Available reliability models consider all faults to be of a permanent nature, but some faults are known to be transient. We direct our analytic modeling effort toward the inclusion of transient faults.

A transient fault is a fault that disappears some time after its arrival. During its stay it alters the contents of registers and/or memory and/or disrupts the normal sequence of program execution. We recover from a transient that has passed by restoring altered data and/or program and by bringing the recovering computer into synchronization with the fault-free computers.

5.1.2 Solution Approach

We approach the problem by drawing a state diagram representing the fault/recovery status of the system. Each state represents the number of fault-free units in the system and the level of fault recovery being undertaken. The transitions between states represent the occurrence of status changing events. The events are random in general so that the state diagram is probabilistic in nature.

Such a state diagram is illustrated in Figure 5.1-1 for an enhanced TMR configuration. Enhanced TMR differs from classical TMR in that transient fault recovery is provided. The system begins a mission time t=0 in the no-fault state, and we wish to find the probability of arriving in the system failure state before t=T, the mission length. We study this simple model to provide insight into the inclusion of transient faults into more complex models.

From the no-faults state a transient fault moves us into the transient recovery state. From the transient recovery state one of three things may occur:

1. Successful recovery -- go to No-Faults.
2. Transient mistaken for a permanent -- go to One Computer Faulty.
3. Fault in a previously fault-free computer during recovery -- go to System Failure.

Note that a fault in a fault-free computer during recovery leaves a TMR system with two faulty computers. We make the assumption that a TMR system fails if two computers are faulty.

A permanent fault will certainly be interpreted by the recovery procedure to be permanent. Therefore, for analysis purposes, a permanent fault in the no faults state moves us to the one computer faulty state without passing through the transient recovery state.

The steps to be taken in analyzing our model are:

1. Characterize transient faults,
2. Model transient recovery, and
3. Find the failure probability.

The results obtained will be estimates of system reliability and the effectiveness of recovery procedures.

5.2 TRANSIENT FAULTS

5.2.1 Transient Arrival

In this analysis we make the assumption that transient faults arrive with an average rate τ that is constant over the life of the system. With the constant rate over time, the probability of the arrival of a transient fault in a small interval of time dt is τdt. It is well-known that under these conditions (see PARZ 60, Ch. 6, Section 3, or DAVE-58 Section 7-2) the probability of exactly k transient fault arrivals between 0 and t obeys a Poisson probability law. That is

$$\Pr\{\text{ k arrivals in } (0,t)\} = e^{-\tau t}\,\frac{(\tau t)^k}{k!}$$

If we let k = 0, we have

$$\Pr\{\text{ No transients in } (0,t)\} = e^{-\tau t}$$

which is analogous to the simplex reliability equation for permanent faults.

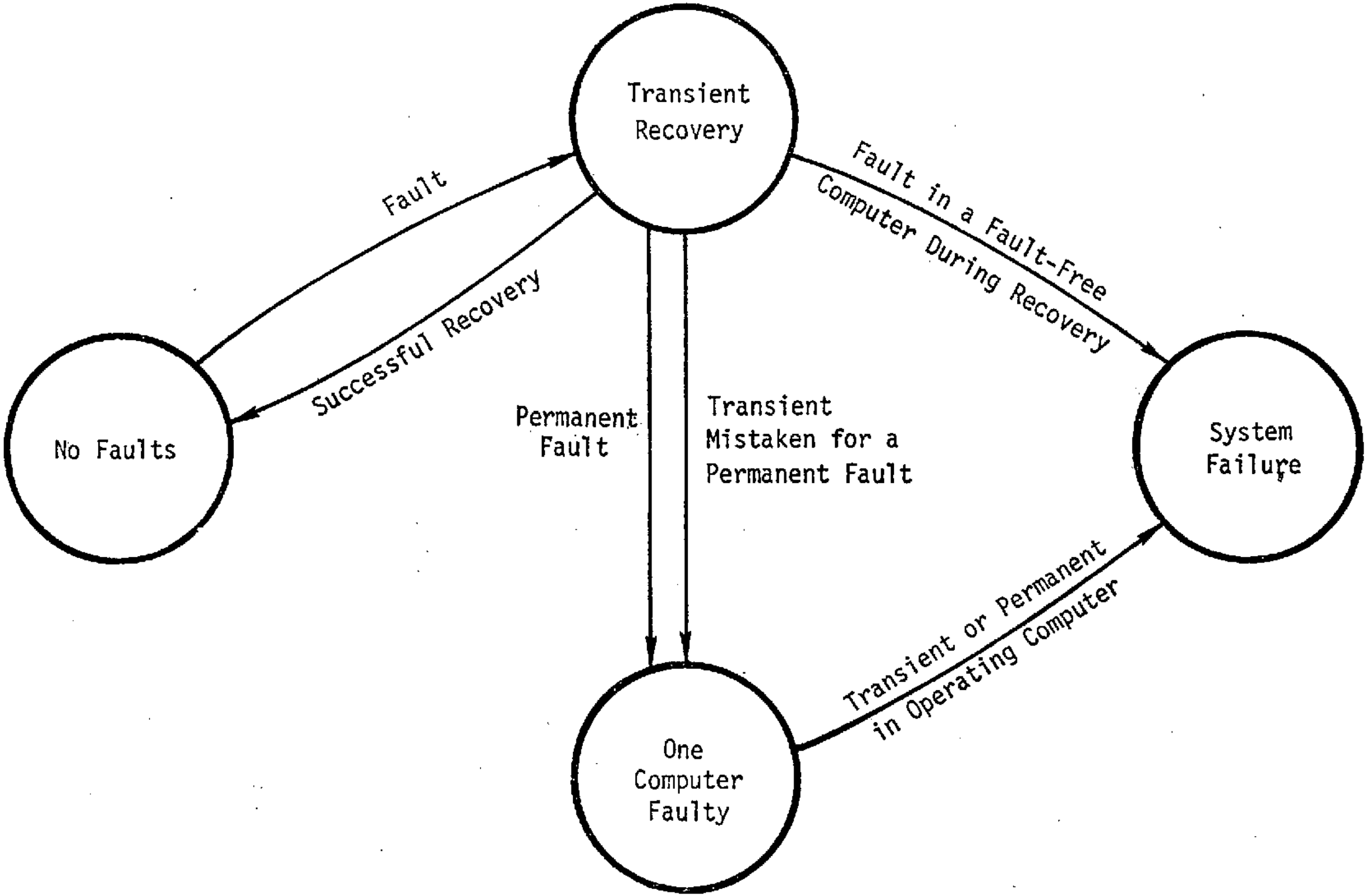


FIGURE 5.1-1 FAULT RECOVERY STATE DIAGRAM OF A TMR CONFIGURATION

5.2.2 Transient Duration

There is little known about the nature of transient faults. A reasonable assumption is that they have a definite duration. There is a dilemma concerning the probability density function of the duration: We could be of the opinion that short transients are much more likely than long transients which would lead us to an exponential density as a mathematically tractable approximation. We could also be of the opinion that there is definite mean duration with an associated spread which would lead us to the gamma, normal or Weibull densitities as an approximation. We could also say that transient faults are caused by several sources, each source with a different average duration. But there are more sources with a small duration than with a large duration. In this case, the composite density function of all the durations could be a "lumpy" exponential.

The above, along with its mathematical tractability lead us to choose the exponential density function to represent transient duration. Hereafter

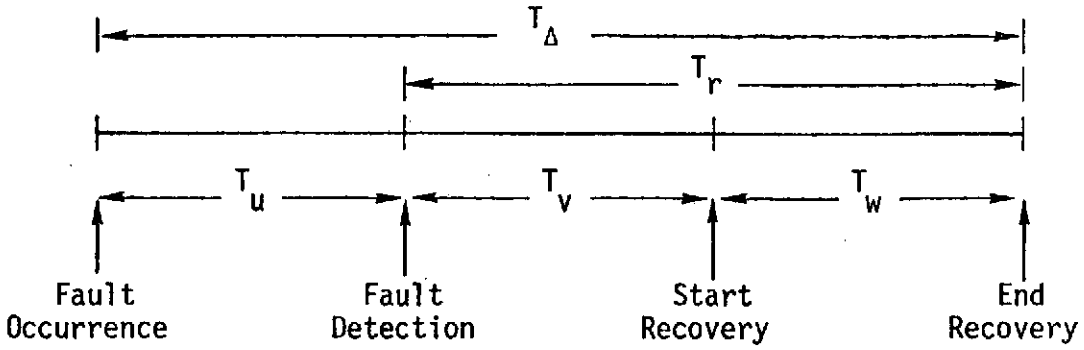
$$f_{D_T}(t) = \gamma e^{-\gamma t}$$

where f_{D_T} is the probability density function of transient fault duration.

5.3 TRANSIENT RECOVERY MODEL

5.3.1 Components of Transient Recovery

A recovery procedure is composed of three stages as shown below. First there is the detection time (T_u) between fault occurrence and detection which is random (It is random because we assume milliseconds between successive comparisons). Then there is a delay T_v before starting the recovery procedure. The delay is the diagnostic time which is a design constant for transient recovery. Then there is the recovery time T_w to accomplish the recovery procedure. The quantity T_r is convenient variable which is defined as $T_r = T_V + T_W$, the time between detection and recovery completion. The quantity T_Δ is the total time consumed between fault occurrence and recovery completion. We assume the distribution of T_u does not vary during the mission. This three stage recovery sequence is also applicable to permanent fault recovery, but is presented here in the transient fault context.



For the time being we assume that T_r is a constant. We hope to relax this restriction in the future to include more sophisticated recovery procedures.

5.3.2 Fault Detection

We now focus our attention on T_u. Let us make the following assumptions:

1. Faults are detected by comparison only.
2. The time between comparisons T_c is a constant.
3. The probability of detecting a fault at the first comparison time is independent of the point in time it arrives between comparisons.

Under these assumptions, the probability density function of T_u is a descending staircase function as shown in Figure 5.3-1. The width of each step is T_c, the time between comparisons. The origin is the time of fault occurrence. The quantity AT_c is the probability of detecting the fault on the first comparison. The area under the entire staircase is one.

There are now three approximations to the staircase that come to mind:

1. Exponential: $f_{T_u}(t) = \delta e^{-\delta t}$

2. Uniform: $f_{T_u}(t) = \frac{1}{T_c} \qquad t \varepsilon [o, T_c]$

3. Uniform-Exponential: $$f_{T_u}(t) = \begin{cases} A & t \varepsilon [o, T_c] \\ \alpha e^{-\alpha t} & t > T_c \end{cases}$$

where $f_{T_u}(t)$ is the probability density function of the detection time, T_u The exponential is a gross approximation. The uniform assumes perfect fault detection. The uniform-exponential assumes an exponential density for the detection of lurking faults and is illustrated in Figure 5.3-2. A lurking fault is an undetected error in program and constants caused by a transient in the memory.

Since f_{T_u} is a density function, we have

$$\int_0^\infty f_{T_u}(t)dt = 1$$

so for the uniform-exponential approximation we have

$$\int_0^{T_c} Adt + \int_{T_c}^\infty \alpha e^{-\alpha t}dt = 1$$

which implies

$$AT_c = 1-e^{-\alpha T_c},$$

$$\alpha T_c = -\log(1-AT_c),$$

relationships that will yield one parameter knowing the other.

5.4 ANALYSIS OF AN ENHANCED TMR CONFIGURATION

5.4.1 Definitions and Assumptions

We assume the system fails if a computer suffers either a transient or permanent fault whenever another computer either (a) has suffered a previous permanent fault or (b) is recovering from a previous transient fault. We define the following configuration parameters:

1. λ $\triangleq$* Permanent Fault Rate.
2. τ $\triangleq$ Transient Fault Rate.
3. c_T $\triangleq$ Pr {Recovery from Transient|Transient Occurs}.
4. ℓ_T $\triangleq$ Pr {Transient Fault is interpreted as a Permanent}.

* $\triangleq$ means equal by definition.

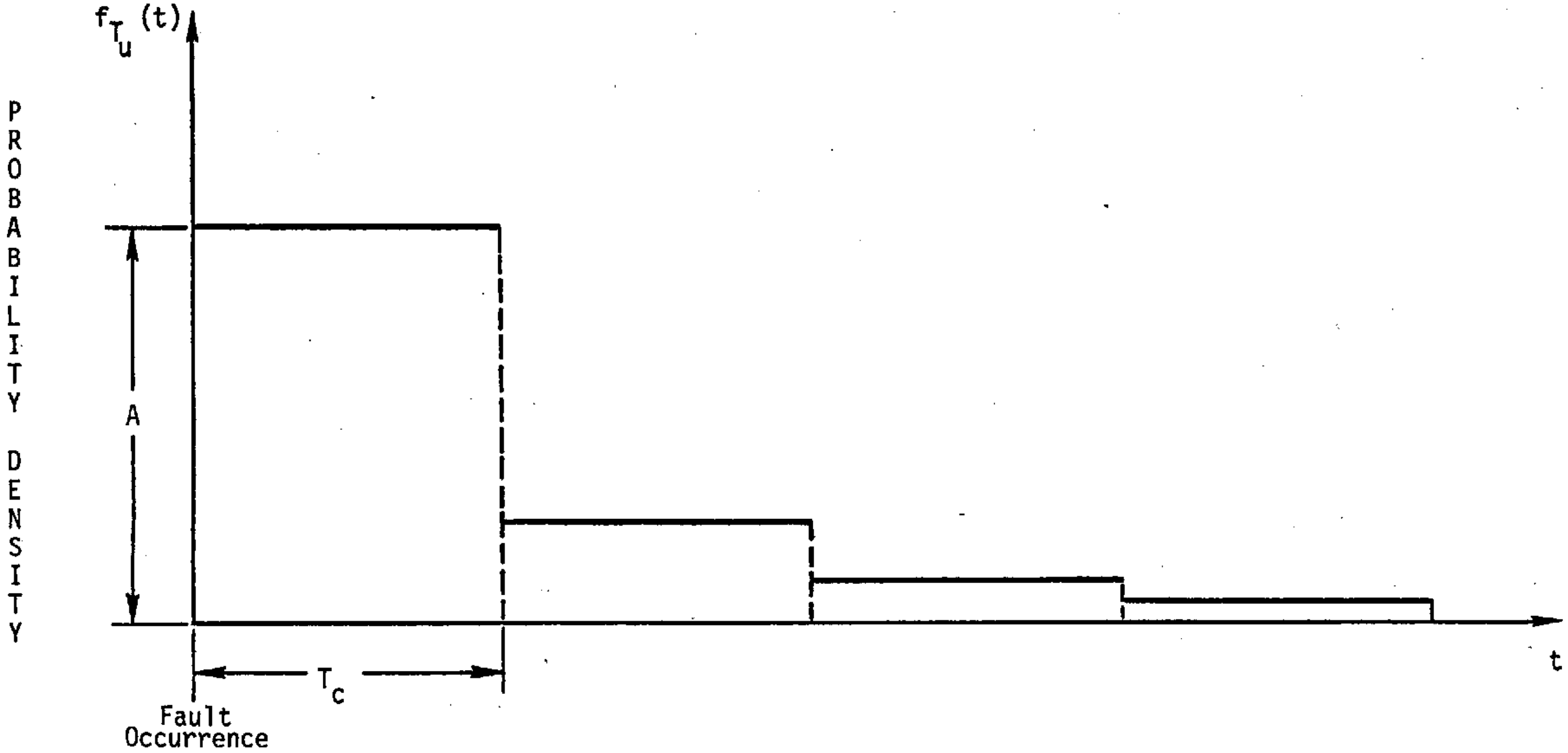


FIGURE 5.3-1 PROBABILITY DENSITY FUNCTION OF THE TIME TO FAULT DETECTION

The quantity c_T is the transient coverage in triplex, and is the probability of returning to the no faults state from the transient recovery state.

We can see that ℓ_T is given by

$$\ell_T = 1 - c_T - \text{Pr \{System Fails During Recovery from a Transient\}}.$$

The quantity ℓ_T decreases the transient coverage. Therefore, we call ℓ_T the transient leakage. We will find leakage directly, so we need to identify some of the mechanisms that contribute to it:

1. Transient duration lasting into the recovery procedure,
2. A second transient occurring in the recovery computer during the last recovery attempt before being declared a permanent, and
3. An imperfect recovery process.

This leads us to our model of the TMR configuration as shown in the fault/recovery state diagram of Figure 5.4-1.

It is important for us to distinguish between an uncovered transient and a leaked transient. Uncovered transients include both leaked transients and transients that end in system failure.

5.4.2 Failure Probability

We now find the probability of reaching the system failure state within a mission. Let F denote this probability. Then

$$F = F_P + F_T$$

where $F_P \triangleq$ Pr {System failure through the one computer faulty state}

and $F_T \triangleq$ Pr {System failure through the transient recovery state}

The two system failure events are made mutually exclusive in the following way: Let

$$\sigma_u \triangleq \lambda + (1-c_T)\tau$$

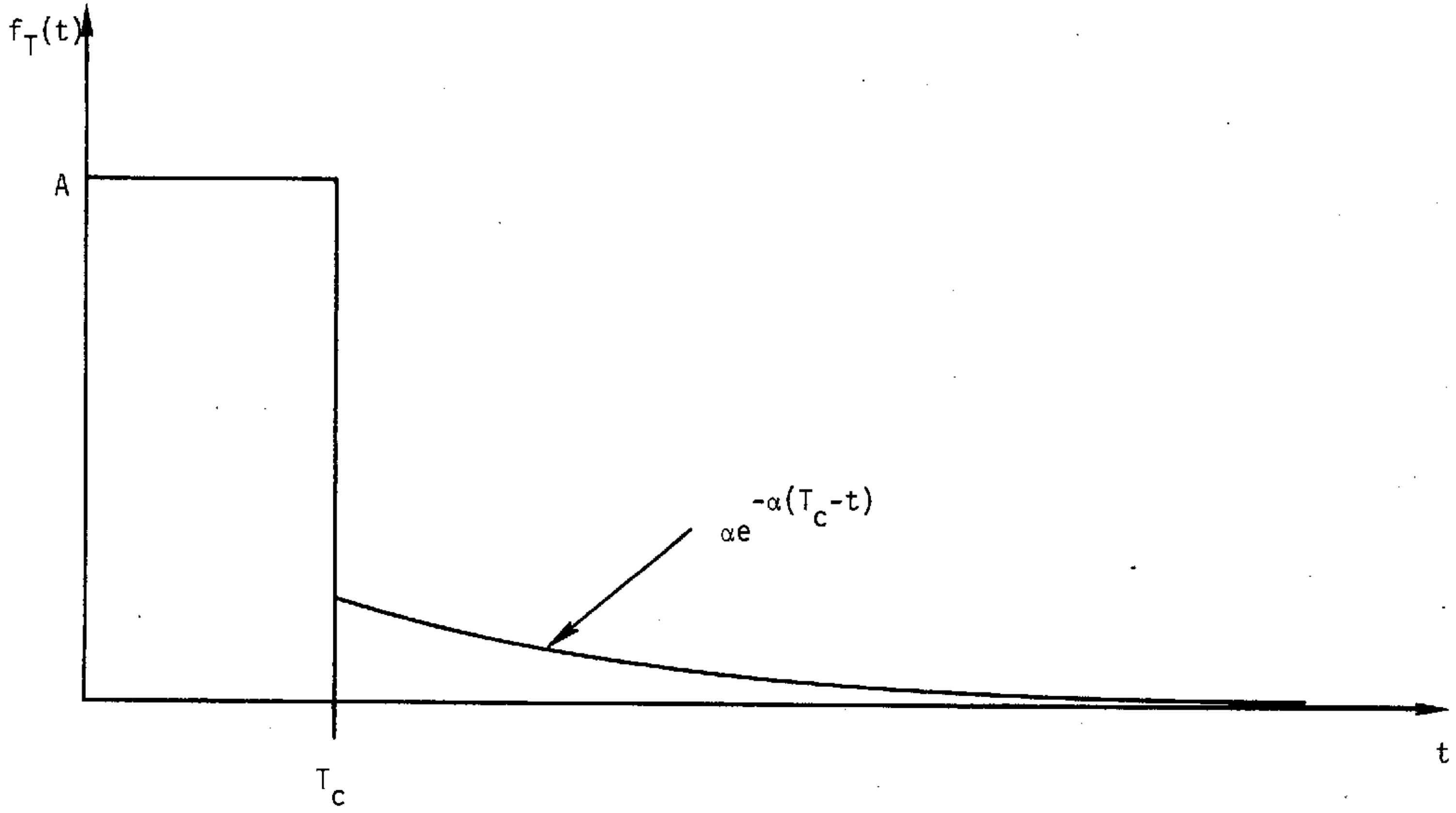


FIGURE 5.3-2 UNIFORM-EXPONENTIAL APPROXIMATION TO THE FAULT DETECTION TIME DENSITY FUNCTION

Defined in this way, σ_u is the rate parameter for the occurrence of permanents and uncovered transients. We use $e^{-\sigma_u t}$ as the probability that no permanents or uncovered transients have occurred up to time t. We then express F_P as:

$$F_P = \int_0^T \text{Pr}\ \{\text{No permanent failures } \varepsilon(0,t),\text{ no uncovered transients } \varepsilon(0,t),\text{ a permanent at } t,\text{ and a transient or permanent in another computer } \varepsilon(t,T)\}$$

$$+ \int_0^T \text{Pr}\ \{\text{No permanent failures } \varepsilon(0,t),\text{ no uncovered transients } \varepsilon(0,t),\text{ a leaked transient at } t,\text{ and a transient or permanent in another computer } \varepsilon(t,T)\}$$

$$= 3\int_0^T e^{-3\sigma_u t} \lambda \ dt\ [1-e^{-2\lambda\,(T-t)} e^{-2\tau(T-t)}]$$

$$+ 3\int_0^T e^{-3\sigma_u t} \ell_T \tau \ dt\ [1-e^{-2\lambda\,(T-t)} e^{-2\tau(T-t)}]$$

$$= \sigma_\ell \left\{ \frac{1-e^{-3\sigma_u T}}{\sigma_u} - \frac{3e^{-2\sigma_t T}}{3\sigma_u - 2\sigma_t} [1-e^{-(3\sigma_u - 2\sigma_t)T}]\right\}$$

where $\sigma_t = \lambda + \tau$

and $\sigma_\ell = \lambda + \ell_T\ \tau$

Let $\hat{F}_T(T_\Delta) \triangleq \text{Pr}\ \{\text{System fails through the transient recovery state } | T_\Delta\}$. The quantity T_Δ is a random variable since the detection time T_u is random ($T_\Delta = T_u + T_r$). Then since T_Δ is random, $\hat{F}(T_\Delta)$ is also a random variable. And $F_T = E(\hat{F}_T)$, where E denotes the expectation, so that

$$F_T = \int_0^\infty \hat{F}_T(T_r + t)\ f_{T_u}(t)\ dt$$

since the T_r portion of T_Δ is assumed to be a constant at this time. We then express $\hat{F}_T$ as

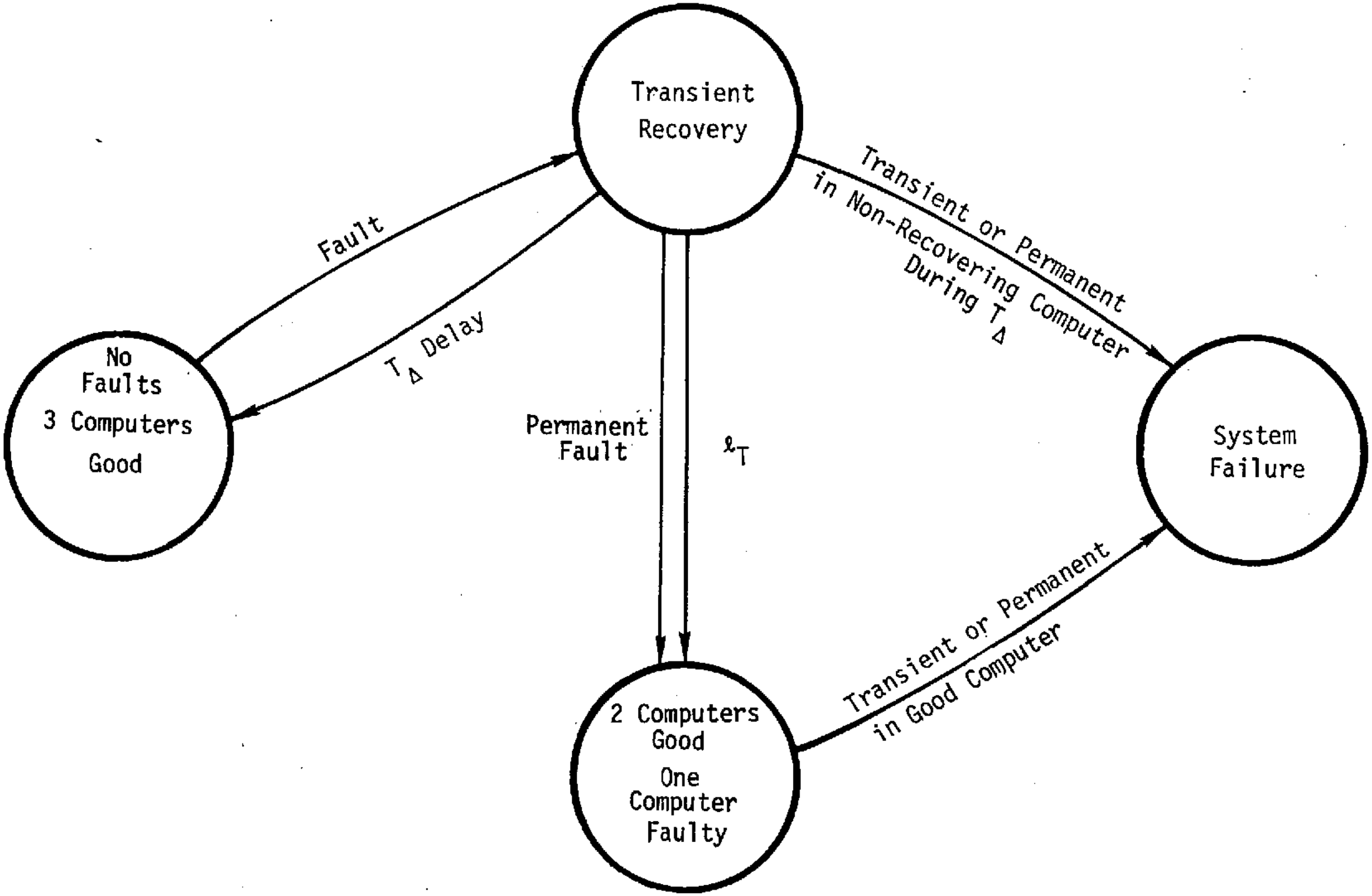


FIGURE 5.4-1 FAULT RECOVERY MODEL OF A TMR CONFIGURATION

$$\hat{F}_T = \int_0^T \text{Pr\{No permanent failures } \varepsilon(0,t)\text{, no uncovered transients } \varepsilon(0,t)\text{, a non-leaky transient at } t\text{, transient or permanent in another computer } \varepsilon(t,t+T_\Delta)\}$$

$$= 3\int_0^T e^{-3\sigma_u t}\,\tau_\ell dt\ [1-e^{-2\sigma_t T_\Delta}]$$

$$= [1-e^{-2\sigma_t T_\Delta}]\,\frac{\tau_\ell}{\sigma_u}\,[1-e^{-3\sigma_u T}]$$

$$= a[1-e^{-2\sigma_t T_\Delta}]$$

where $a \triangleq \frac{\tau_\ell}{\sigma_u}[1-e^{-3\sigma_u T}]$ and $\tau_\ell = (1-\ell_T)\tau$

The quantity τ_ℓ is the rate of non-leaky transients. We now find F_T using the above notational simplification as

$$F_T = a\int_0^\infty (1-e^{-2\sigma_t(t+T_r)})\ f_{T_u}(t)\ dt$$

$$= a - a\int_0^\infty e^{-2\sigma_t(t+T_r)}\ f_{T_u}(t)\ dt$$

Now we apply the three detection density function approximations to F_T. Using the exponential approximation, F_T becomes

$$F_T = a - a\int_0^\infty \delta e^{-2\sigma_t T_r}\ e^{-(2\sigma_t+\delta)t}dt$$

$$= a(1-\frac{\delta e^{-2\sigma_t T_r}}{\delta+2\sigma_t})$$

$$= a(\frac{2\sigma_t+\delta(1-e^{-2\sigma_t T_r})}{\delta+2\sigma_t}$$

With the uniform approximation, F_T becomes

$$F_T = a[1-\int_0^{T_c} \frac{e^{-2\sigma_t T_r}}{T_c} e^{-2\sigma_t t} dt]$$

$$= a[1- \frac{e^{-2\sigma_t T_r}}{2\sigma_t T_c} (1-e^{-2\sigma_t T_c})]$$

And using the uniform-exponential approximation, F_T becomes

$$F_T = a - a\int_0^{T_c} Ae^{-2\sigma_t T_r} e^{-2\sigma_t t} dt$$

$$-\int_{T_c}^{\infty} a\alpha e^{-2\sigma_t T_r} e^{-(2\sigma_t+\alpha)t} dt$$

$$= a[1- \frac{Ae^{-2\sigma_t T_r}}{2\sigma_t}(1-e^{-2\alpha_t T_c}) - \frac{\alpha e^{-2\sigma_t T_r} e^{-(2\sigma_t+\alpha)T_c}}{2\sigma_t+\alpha}$$

Note that if we set $A = 1/T_c$ and $\alpha = 0$, F_T becomes the same as for the case of the uniform approximation. And if we set $A = 0$, $T_c = 0$, and $\alpha = \delta$; F_T becomes the same as for the case of the exponential approximation.

5.4.3 Transient Leakage

We will model transient leakage due to two causes:

1. A second transient occurring in the recovering computer during recovery.
2. Transient duration continuing into the recovery process.

Here we consider ideal recovery procedures. A complete loading of state vector and program/constants (if DRO memory) would approach such an ideal for this modeling.

Let L_1 be the event we receive no transient in the recovery process and L_2 be the event that the transient is still active during the recovery process. Then the transient leakage becomes:

$$\ell_T = \Pr\{\bar{L}_1 \vee L_2\}$$

$$= 1 - \Pr\{L_1\} + P\{L_2\} - \Pr\{\bar{L}_1 \wedge L_2\}$$

We compute the leakage in this manner so that we can examine the two causes of leakage separately.

Let

$$\ell_1 = 1 - \Pr\{L_1\} \qquad \text{(Transient upon a transient)}$$

$$\ell_2 = \Pr\{L_2\} \qquad \text{(Excessive transient duration)}$$

then

$$\ell_1 = 1 - \int_0^\infty \Pr\{L_1 | T_\Delta = T_r + t\} \Pr\{T_\Delta = T_r + t\}\, dt$$

$$= 1 - \int_0^\infty e^{-\sigma_t(t+T_r)} f_{T_u}(t)\, dt$$

For the uniform-exponential approximation to T_u

$$\ell_1 = 1 - e^{-\sigma_t T_r}\{\int_0^{T_c} Ae^{-\sigma_t(T_r+t)} dt + \int_{T_c}^\infty \alpha e^{-(\sigma_t+\alpha)t}\, dt$$

$$= 1 - \frac{Ae^{-\sigma_t T_r}}{\sigma_t}(1-e^{-\sigma_t T_c}) - \frac{e^{-\sigma_t T_r} e^{-(\sigma_t+\alpha)T_c}}{\alpha+\sigma_t}$$

Setting $A = \frac{1}{T_c}$ and $\alpha = 0$, we have ℓ_1 for the uniform approximation

$$\ell_1 = 1 - \frac{e^{-\sigma_t T_r}}{\sigma_t T_c}(1-e^{-\sigma_t T_c})$$

And setting $A = 0$, $T_c = 0$, and $\alpha = \delta$; we have the exponential approximation case

$$\ell_1 = 1 - \frac{\delta e^{-\sigma_t T_r}}{\delta+\sigma_t}$$

$$= \frac{\sigma_t+\delta(1-e^{-\sigma_t T_r})}{\sigma_t+\delta}$$

Turning to ℓ_2,

$$\ell_2 = \int_0^\infty \Pr \{L_2 | T_\Delta = T_r+t\} \Pr \{T_\Delta = T_r+t\} \, dt$$

To find $\Pr \{L_2 | T_\Delta\}$

$$\Pr \{L_2 | T_\Delta\} = \Pr \{D_T > T_u+T_v\}$$

$$= \int_{T_u+T_v}^\infty \gamma e^{-\gamma t} \, dt = e^{-\gamma(T_u+T_v)}$$

where γ is the duration parameter as defined in Section 3.2.2.

So ℓ_2 becomes

$$\ell_2 = e^{-\gamma T_v} \int_0^\infty e^{-\gamma t} f_{T_u}(t) \, dt$$

$$= e^{-\gamma T_v} \{ \int_0^{T_c} A e^{-\gamma t} \, dt + \int_0^\infty \alpha e^{-(\gamma+\alpha)t} \, dt$$

$$= e^{-\gamma T_v} \{ \frac{A}{\gamma} (1-e^{-\gamma T_c}) + \frac{\alpha}{\gamma+\alpha} e^{-(\gamma+\alpha)T_c} \}$$

for the uniform-exponential approximation. For the uniform approximation, ℓ_2 becomes

$$\ell_2 = \frac{e^{-\gamma T_v}}{\gamma T_c} (1-e^{-\gamma T_c})$$

And for the exponential approximation, ℓ_2 becomes

$$\ell_2 = \frac{\delta e^{-\gamma T_V}}{\delta+\gamma}$$

Then we move to P $\{\overline{L}_1 \wedge L_2\}$

$$\Pr\{\overline{L}_1 \wedge L_2\} = \int_0^\infty \Pr\{\overline{L}_1 \wedge L_2 | T_\Delta = t+T_V+T_W\} \Pr\{T_{\underline{\Delta}} = t+T_V+T_W\}\, dt$$

$$= e^{-\gamma T_D} \int_0^\infty e^{-\gamma t} (1-e^{-\lambda T_r} e^{-\lambda t})\, f_{T_u}(t)\, dt$$

Computing the above for the uniform-exponential approximation and combining with ℓ_1 and ℓ_2, we have for ℓ_T

$$\ell_T = 1-e^{-\sigma_t T_r} \{\frac{A}{\sigma_t} (1-e^{-\sigma_t T_c}) + \frac{\alpha e^{-(\sigma_t+\alpha)T_c}}{\sigma_t+\alpha} - e^{-\gamma T_W} [\frac{A}{\sigma_t+\gamma} (1-e^{-(\sigma_t+\gamma)T_c}) + \frac{\alpha e^{-(\sigma_t+\gamma+\alpha)T_c}}{\sigma_t+\gamma+\alpha}]$$

Setting $A = \frac{1}{T_c}$ and $\alpha = 0$ gives the uniform case

$$\ell_T = 1 - e^{-\sigma_t T_r} [\frac{1-e^{-\sigma_t T_c}}{\sigma_t T_c} - \frac{e^{-\gamma T_V}}{(\sigma_t+\gamma)T_c} (1-e^{-(\sigma_t+\gamma)T_c})]$$

And for the exponential case

$$\ell_T = 1 - \delta e^{-\sigma_t T_r} [\frac{1}{\delta+\sigma_t} - \frac{e^{-\gamma T_V}}{\sigma_t+\gamma+\delta}]$$

5.4.4 Transient Coverage

Turning to transient coverage, we can see that it is the probability of the joint occurrence of two events. Let C_1 be the event we receive no transient in any computer during recovery and C_2 be the event that the transient is not active during the recovery process. Then the transient coverage becomes

$$c_T = P_r\{C_1 \wedge C_2\}$$

$$= \int_0^\infty P_r\{C_1 \wedge C_2 | T_\Delta = t+T_r\}\ P_r\{T_\Delta = t+T_r\}\ dt$$

$$= \int_0^\infty (1-e^{-\gamma(t+T_v)})\ e^{-3\sigma_t(t+T_r)} f_{T_u}(t)\ dt$$

which for the uniform-exponential case becomes

$$c_T = d^{-3\sigma_t T_r}\{\frac{A}{3\sigma_t}(1-e^{-3\sigma_t T_c} + \frac{\alpha e^{-(3\sigma_t+\alpha)T_c}}{3\sigma_t+\alpha} - e^{-\gamma T_v}[\frac{A}{3\sigma_t+\gamma}(1-e^{-(3\sigma_t+\gamma)T_c})$$

$$+ \frac{\alpha e^{-(3\sigma_t+\gamma+\alpha)T_c}}{3\sigma_t+\gamma+\alpha}$$

Setting $\alpha = 0$ and $A = 1/T_c$ gives us the uniform case

$$c_T = e^{-3\sigma_t T_r}[\frac{1-e^{-3\sigma_t T_c}}{3\sigma_t T_c} - \frac{e^{-\gamma T_v}}{(3\sigma_t+\gamma)T_c}(1-e^{-(3\sigma_t+\gamma)T_c})]$$

And the exponential case becomes

$$c_T = \delta e^{-3\sigma_t T_r}[\frac{1}{\delta+3\sigma_t} - \frac{e^{-\gamma T_v}}{3\sigma_t+\gamma+\delta}$$

The equations obtained so far are summarized in Table 5.4-I.

5.4.5 Simplifying Assumptions for Shorter Mission Times

The basic assumption used here is that $\lambda T<<1$. This could apply to either shorter mission times or certain transient burst environments. If $\lambda T<<1$, then $\lambda_u T<<1$ and $\lambda T_\Delta<<1$. Another reasonable assumption to make is that $\gamma,\delta>>\lambda$. Using the exponential approximation to $f_{T_d}(t)$ our expression for F is

$$F = \sigma_\ell\{\frac{1-e^{-3\sigma_u T}}{\sigma_u} - \frac{3e^{-2\sigma_t T}}{3\sigma_u-2\sigma_t}[1-e^{-(3\sigma_u-2\sigma_t)T}]\} + \frac{\tau_\ell}{\sigma_u}(1-e^{-3\sigma_u T})(\frac{2\sigma_t+\delta(1-e^{-2\sigma_t T_r})}{2\sigma_t+})$$

By using the series expansion

$$e^{-x} = 1 - x + \frac{1}{2}x^2 + \ldots$$

we have

$$F \approx 3\sigma_{\ell}\sigma_{t}T^2 + 6\tau_{\ell}\sigma_{t}T(\frac{1 + \delta T_r}{\delta})$$

Similarly,

$$\ell_1 \approx \frac{\sigma_t}{\delta}(1+\delta T_r)$$

$$\ell_2 \approx \frac{\delta e^{-\gamma T_v}}{\gamma+\delta}$$

$$\ell_T \approx \frac{\delta e^{-\gamma T_v}}{\gamma+\delta} + \frac{\sigma_t}{\delta}(1+\delta T_r)$$

and

$$1-c_T = \frac{\delta e^{-\gamma T_v}}{\gamma+\delta} + \frac{3\sigma_t}{\delta}(1+\delta T_r)$$

Note that ℓ_T and $1-c_T$ differ by $\frac{2\sigma_t}{\delta}(1+\delta T_r)$. This means that

$$\Pr\{\text{System Failure}|\text{Transient Occurs}\} \approx \frac{2\sigma_t}{\delta}(1+\delta T_r)$$

5.4.6 Extension of TMR Modeling to N Computers

5.4.6.1 Fault/Recovery State Diagram

Here we will extend the basic enhanced TMR model to 4, 5, and finally N computers. A natural extension to the fault occurrence/recovery state diagram is shown in Figure 5.4-2.

We begin at T=0 with N working computers with the computers undergoing permanent and transient faults. On receiving a fault a transient recovery procedure is initiated. In transient recovery, four conditions can result in three outcomes:

TABLE 5.4-I SUMMARY OF EQUATIONS FOR THE TMR CONFIGURATION

	UNIFORM-EXPONENTIAL*	EXPONENTIAL*	UNIFORM*
F_T	$\frac{\tau_\ell}{\sigma_u}(1-e^{-3\sigma_u T})[1-\frac{Ae^{-2\sigma_t T_r}}{2\sigma_t}(1-e^{-2\sigma_t T_c})-\frac{\alpha e^{-2\sigma_t T_r}e^{-(2\sigma_t+\alpha)T_c}}{2\sigma_t+\alpha}]$	$\frac{\tau_\ell}{\sigma_u}(1-e^{-3\sigma_u T})(\frac{2\sigma_t+\delta(1-e^{-2\sigma_t T_r}}{2\sigma_t+\delta})$	$\frac{\tau_\ell}{\sigma_u}(1-e^{-3\sigma_u T})[1-\frac{e^{-2\sigma_t T_r}}{2\sigma_t T_c}(1-e^{-2\sigma_t T_c})]$
ℓ_T	$1-e^{-\sigma_t T_r}(\frac{A}{\sigma_t}(1-e^{-\sigma_t T_c})+\frac{\alpha e^{-(\sigma_t+\alpha)T_c}}{\sigma_t+\alpha}-e^{-\gamma T_v}[\frac{A}{\sigma_t+\gamma}(1-e^{-(\sigma_t+\gamma)T_c})+\frac{\alpha e^{-(\sigma_t+\gamma+\alpha)T_c}}{\sigma_t+\gamma+\alpha}])$	$1-\delta e^{-\sigma_t T_r}[\frac{1}{\delta+\sigma_t}-\frac{e^{-\gamma T_v}}{\sigma_t+\gamma+\delta}]$	$1-e^{-\sigma_t T_r}[\frac{1-e^{-\sigma_t T_c}}{\sigma_t T_c}-\frac{e^{-\gamma T_v}}{(\sigma_t+\gamma)T_c}(1-e^{-(\sigma_t+\gamma)T_c})]$
ℓ_1	$1-\frac{Ae^{-\sigma_t T_r}}{\sigma_t}(1-e^{-\sigma_t T_c})-\frac{\alpha e^{-\sigma_t T_r}e^{-(\sigma_t+\alpha)T_c}}{\alpha+\sigma_t}$	$\frac{\sigma_t+(1-e^{-\lambda T_r})}{\sigma_t+\delta}$	$1-\frac{e^{-\sigma_t T_r}}{\sigma_t T_c}(1-e^{-\sigma_t T_c})$
ℓ_2	$e^{-\gamma T_v}[\frac{A}{\gamma}(1-e^{-\gamma T_c})+\frac{\alpha}{\gamma+\alpha}e^{-(\gamma+\alpha)T_c}]$	$\frac{\delta e^{-\gamma T_v}}{\delta+\gamma}$	$\frac{e^{-\gamma T_v}}{\gamma T_c}(1-e^{-\gamma T_c})$
c_T	$e^{-3\sigma_t T_r}\{\frac{A}{3\sigma_t}(1-e^{-3\sigma_t T_c})+\frac{\alpha e^{-(3\sigma_t+\alpha)T_c}}{3\sigma_t+\alpha}-e^{-\gamma T_v}[\frac{A}{3\sigma_t+\gamma}(1-e^{-(3\sigma_t+\gamma)T_c})+\frac{\alpha e^{-(3\sigma_t+\gamma+\alpha)T_c}}{3\sigma_t+\gamma+\alpha}]\}$	$\delta e^{-3\sigma_t T_r}[\frac{1}{\delta+3\sigma_t}-\frac{e^{-\gamma T_v}}{3\sigma_t+\gamma+\delta}]$	$e^{-3\sigma_t T_r}[\frac{1-e^{-3\sigma_t T_c}}{3\sigma_t T_c}-\frac{e^{-\gamma T_v}}{(3\sigma_t+\gamma)T_c}(1-e^{-(3\sigma_t+\gamma)T_c})]$
F_P	$\sigma_\ell\{\frac{1-e^{-3\sigma_u T}}{\sigma_u}-\frac{3e^{-2\sigma_t T}}{3\sigma_u-2\sigma_t}[1-e^{-(3\sigma_u-2\sigma_t)T}]\}$		

*APPROXIMATIONS TO FAULT DETECTION

1. Permanent fault - Go to N-1 working computers.
2. A fault occurs in a computer assisting in transient recovery - Go to N-2 working computers.
3. A transient fault is leaked - Go to N-1 working computers.
4. The time T_Δ passes (successful recovery) - Return to N working computers.

The system continues to undergo faults and with the passing of time degrades to three working computers. In three working computers, we are in the familiar enhanced TMR.

5.4.6.2 Definitions and Review

We define the following failure probability as a function of the number of computers working.

$$F_N(T) = \Pr \{ \text{System failure before time } T | N \text{ computers are working at time } 0 \}$$

And the probability of one or more of n computers suffering a fault in time T_Δ is

$$f_n(T_\Delta) \triangleq 1 - e^{-n\sigma_t T_\Delta}$$

We can express F_2 as

$$F_2(T) = 1 - e^{-2\sigma_t T}$$

where

$$\sigma_t = \lambda + \tau$$

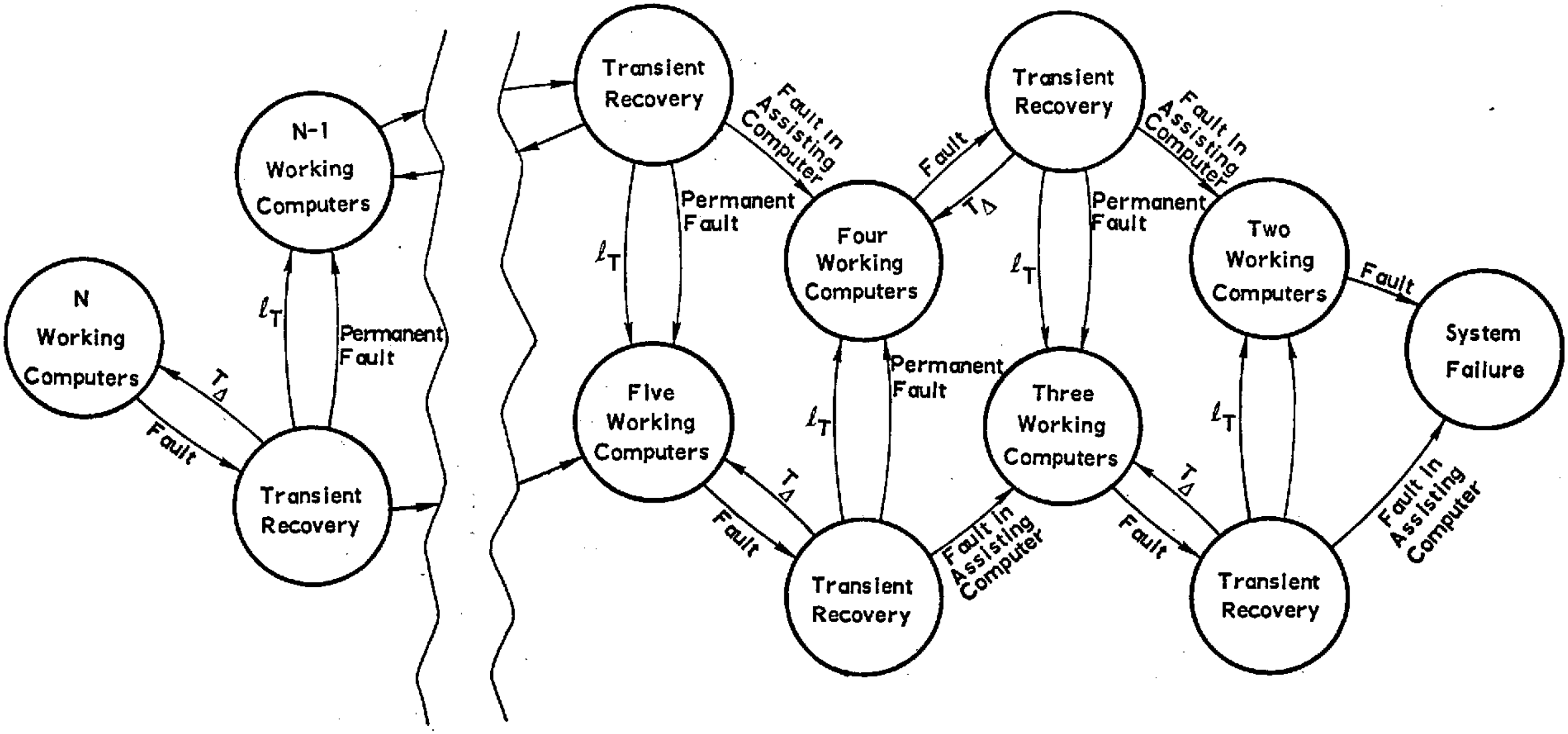


FIGURE 5.4-2 EXTENSION OF ENHANCED TMR MODEL TO N COMPUTERS

And recall that

$$F_3(T) = \int_0^T 3\sigma_\ell \, e^{-3\sigma_u t} \, (1 - e^{-2\sigma_t (T-t)})dt$$

$$+ \int_0^T 3\tau_\ell e^{-3\sigma_u t} [1 - e^{-2\sigma_t T_\Delta}] \, dt$$

$$= \int_0^T 3\sigma_\ell e^{-3\sigma_u t} \, F_2(T-t)dt$$

$$+ \int_0^T 3\tau_\ell e^{-3\sigma_u t} \, f_2(T_\Delta)dt$$

where $\sigma_\ell \triangleq \lambda + \ell_T \tau,$

$\sigma_u \triangleq \lambda + (1 - c_T)\tau,$

$f_n(T_\Delta) \triangleq [1 - e^{-n\sigma_t T_\Delta}]$

$\tau_\ell \triangleq (1-\ell_T)\tau$

The quantity f_n is the probability of receiving any fault during the time T_Δ.

5.4.6.3 Finding Failure Probability for Four Computers

From the state diagram of Figure 5.4-2, we can find $F_4(T)$ as a function of F_2 and F_3 by the sum of the probability of going to system failure through the two working computers state and the probability of going to system failure through the three working computers state. We formulate F_4 as follows:

$$F_4(T) = \int_0^T \mathrm{Pr}\ \{\text{No permanent or uncovered transients from 0 to t, a permanent or leaked transient at t, system failure from three working computers between t and T}\}$$

$$+ \int_0^T \mathrm{Pr}\ \{\text{No permanents or uncovered transients from 0 to t, a non-leaky transient at t, a fault in a computer assisting in recovery between t and } t+T_\Delta\text{, and system failure from two working computers between t and T}\}$$

$$= \int_0^T e^{-4\sigma_u t}\ 4\sigma_\ell dt\ F_3(T-t)$$

$$+ \int_0^T e^{-4\sigma_u t}\ 4\tau_\ell dt\, f_3(T_\Delta)\ F_2(T-t)$$

$$= \sigma_\ell^{\ 2} \left\{ \frac{1+3e^{-4\sigma_u T} -4e^{-3\sigma_u T}}{\sigma_u^{\ 2}} + \frac{12}{3\sigma_u - 2\sigma_t} \left[\frac{e^{-3\sigma_u T}}{\sigma_u} - \frac{e^{-2\sigma_t T}}{4\sigma_u - 2\sigma_t} \right] - \frac{12e^{-4\sigma_u T}}{\sigma_u(4\sigma_u - 2\sigma_t)} \right\} + \frac{\sigma_\ell \tau_\ell}{\sigma_u^{\ 2}}\ f_2(T_\Delta) \left[1+3e^{-4\sigma_u T} -4e^{-3\sigma_u T} \right]$$

$$+ \tau_\ell f_3(T_\Delta) \left\{ \frac{1-e^{-4\sigma_u T}}{\sigma_u} - \frac{4}{4\sigma_u - 2\sigma_t} [e^{-2\sigma_t T} - e^{-4\sigma_u T}] \right\}$$

For smaller $\sigma_t T$ this simplifies to

$$F_4(T) \approx 4\sigma_\ell^{\ 2}\ \sigma_t\ T^3$$

5.4.6.4 Finding the Failure Probability for Five Computers

In the same manner as in finding $F_4(T)$ as a function of F_3 and F_2, we can find $F_5(T)$ as a function of F_4 and F_3. So $F_5(T)$ becomes

$$F_5(T) = \int_0^T 5\sigma_\ell \, e^{-5\sigma_u t} \, F_4(T-t)dt$$

$$+ f_4(T_\Delta) \int_0^T 5\tau \, e^{-5\sigma_u t} \, F_3(T-t)dt$$

$$= \frac{\sigma_\ell^{\,2}}{3} [\sigma_\ell + \tau_\ell f_2(T_\Delta)] \, [1-6\, e^{-5\sigma_u T} + 15\, e^{-4\sigma_u T} - 10\, e^{-3\sigma_u T}]$$

$$- \frac{60\sigma_\ell^{\,3}}{\sigma_u^{\,2}(4\sigma_u - 2\sigma_t)} [e^{-4\sigma_u T} \, (1 - e^{-\sigma_u T})]$$

$$+ \frac{60\sigma_\ell^{\,3}}{3\sigma_u - 2\sigma_t} \left[\frac{e^{-3\sigma_u T}(1-e^{-2\sigma_u T})}{2\sigma_u^{\,2}} - \frac{e^{-2\sigma_t T} \, (1-e^{-(5\sigma_u - 2\sigma_T)T}}{(5\sigma_u - 2\sigma_t)(4\sigma_u - 2\sigma_t)} \right]$$

$$+ \frac{\sigma_\ell \tau_\ell f_3(T_\Delta)}{\sigma_u^{\,2}} [1+4e^{-5\sigma_u T} - 5e^{-4\sigma_u T}]$$

$$- \frac{20\sigma_\ell \tau_\ell f_3(T_\Delta)}{4\sigma_u - 2\sigma_t} \left\{ \frac{e^{-2\sigma_t T}(1-e^{-(5\sigma_u - 2\sigma_t)T})}{5\sigma_u - 2\sigma_t} - \frac{e^{-4\sigma_u T}(1-e^{-\sigma_u T})}{\sigma_u} \right\}$$

$$+ \frac{\tau_\ell f_4(T_\Delta)}{\sigma_u^{\,2}} [\sigma_\ell + \tau_\ell f_2(T_\Delta)] \, [1 + \frac{3}{2}\, e^{-5\sigma_u T} - \frac{5}{2}\, e^{-3\sigma_u T}]$$

$$- \frac{15\sigma_\ell \tau_\ell f_4(T_\Delta)}{3\sigma_u - 2\sigma_t} \left[\frac{e^{-2\sigma_t T}(1-e^{-(5\sigma_u - 2\sigma_t)T})}{5\sigma_u - 2\sigma_t} - \frac{e^{-3\sigma_u T}(1-e^{-2\sigma_u T})}{2\sigma_u} \right]$$

which for small $\sigma_t T$ simplifies to

$$F_5(T) \approx 5\sigma_\ell^{\,3} \sigma_t T^4$$

5.4.6.5 Generalization to N Computers

An examination of the integral expressions for F_4 and F_5 shows a recursive relationship between $F_N(T)$ and F_{N-1} and F_{N-2} which may be given as follows:

$$F_N(T) = \int_0^T N\sigma_\ell \, e^{-N\sigma_u t} F_{N-1}(T-t)dt$$

$$+ f_{n-1}(T) \int_0^T N\tau_\ell e^{-N\sigma_u t} F_{N-2}(T-t)dt$$

This relationship may be expanded into an (N-1)-fold convolution integral. An example of this is shown in Figure 5.4-3 for $F_5(T)$.

5.4.7 Recovery Start Delay

When formulating a transient recovery procedure we are faced with a dilemma. If we begin the recovery procedure too soon, a long transient duration could hinder recovery; and if we delay the start of the recovery procedure too long, we leave the system unnecessarily vulnerable to other faults.

We seek an optimum delay associated with this tradeoff by maximizing transient coverage. Using the exponential approximation to transient detection we have for c_T

$$c_T = \delta e^{-3\sigma_t T_r}\left[\frac{1}{\delta+3\sigma_t} - \frac{e^{-\gamma T_v}}{3\sigma_t+\gamma+\delta}\right]$$

$$= \delta e^{-3\sigma_t T_w}\left[\frac{e^{-3\sigma_t T_v}}{\delta+3\sigma} - \frac{e^{-(3\sigma_t+\gamma)T_v}}{3\sigma_t+\gamma+\delta}\right]$$

by breaking T_r into T_v+T_w

Differentiating and setting $\frac{dc_T}{dT_v} = 0$

$$\frac{dc_T}{dT_v} = -\delta e^{-3\sigma_t T_w}\left[\frac{3\sigma_t e^{-3\sigma_t T_v}}{\delta+3\sigma_t} - \frac{(3\sigma_t+\gamma)e^{-(3\sigma_t+\gamma)T_v}}{3\sigma_t+\gamma+\delta}\right] = 0$$

This yields

$$T_D = \frac{1}{\gamma}\log\left[\frac{(3\sigma_t+\gamma)(3\sigma_t+\delta)}{3\sigma_t(3\sigma_t+\gamma+\delta)}\right]$$

which will be a maximum if the second derivative is negative at that point. The second derivative is

$$\frac{d^2c_T}{dT_v^2} = \delta e^{-3\sigma_t T_r}\left[\frac{9\lambda^2}{\delta+3\sigma_t} - \frac{(3\sigma_t+\gamma)^2 e^{-\gamma T_v}}{3\sigma_t+\gamma+\delta}\right],$$

and it becomes

$$\frac{d^2c_T}{dT_v^2} = -\delta e^{-3\sigma_t T_r}\left(\frac{\gamma}{3\sigma_t+\delta}\right) < 0$$

when T_v is as given above.

Therefore, we have found a maximum. Similarly, the uniform approximation yields the following optimum delay

$$T_v = \frac{1}{\gamma}\log\left[\frac{1-e^{-(3\sigma_t+\gamma)T_c}}{1-e^{-3\sigma_t T_c}}\right]$$

$$
\begin{aligned}
F_5(T) = & \int_0^T dt 5\sigma_\ell e^{-5\sigma_u t} \int_0^{T-t} d\xi 4\sigma_\ell e^{-4\sigma_u \xi} \int_0^{T-\xi} d\zeta 3\sigma_\ell e^{-3\sigma_u \zeta} \int_0^{T-\zeta} d\eta 2\sigma_t e^{-2\sigma_t \eta} \\
& + \int_0^T dt 5\sigma_\ell e^{-5\sigma_u t} \int_0^{T-t} d\xi 4\sigma_\ell e^{-4\sigma_u \xi} \int_0^{T-\xi} d\zeta 3 \tau_\ell \; e^{-3\sigma_u \zeta} \left[1 - e^{-2\sigma_t T_\Delta}\right] \\
& + \int_0^T dt 5\sigma_\ell e^{-5\sigma_u t} \int_0^{T-t} d\xi 4 \tau_\ell \; e^{-4\sigma_u \xi} \left[1 - e^{-3\sigma_t T_\Delta}\right] \int_0^{T-\xi} d\zeta 2\sigma_t e^{-2\sigma_t \zeta} \\
& + \int_0^T dt 5 \tau_\ell \; e^{-5\sigma_u t} \left[1 - e^{-4\sigma_t T_\Delta}\right] \int_0^{T-t} d\xi 3\sigma_\ell e^{-3\sigma_u \xi} \int_0^{T-\xi} d\zeta 2\sigma_t e^{-2\sigma_t \zeta} \\
& + \int_0^T dt 5 \tau_\ell \; e^{-5\sigma_u t} \left[1 - e^{-4\sigma_t T_\Delta}\right] \int_0^{T-t} d\xi 3 \tau_\ell \; e^{-3\sigma_u \xi} \left[1 - e^{-2\sigma_t T_\Delta}\right]
\end{aligned}
$$

FIGURE 5.4-3 FAILURE PROBABILITY FOR FIVE COMPUTERS

5.5 MODELING OF GENERAL CONFIGURATIONS

Here we present a general model for the analysis of adaptive and non-adaptive configurations consisting of 1 to 5 whole computers. This model includes transient leakage as well as the components of coverage discussed in Section 4.

5.5.1 The Recovery Process

5.5.1.1 Coverage

Coverage is defined as (BOUR 71):

$$c \triangleq^{*} \Pr\{\text{System recovers} \mid \text{fault occurs}\}$$

Here we break coverage into a triplet as follows: Define

$$u \triangleq \Pr\{\text{Fault is detected} \mid \text{fault occurs}\}$$

$$v \triangleq \Pr\{\text{Fault is located} \mid \text{fault detected}\}$$

$$w \triangleq \Pr\{\text{Recovery} \mid \text{fault is located}\}$$

as in Section 4.2.5. Here u, v, and w are the detectability, diagnostability, and recoverability, respectively and

$$c = uvw$$

The quantities u, v, and w are probabilities that also have times associated with them. We define

$T_u \triangleq$ Detection time

$T_v \triangleq$ Diagnosis time

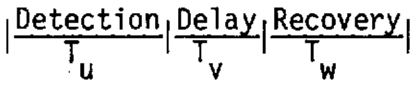
$T_w \triangleq$ Recovery time

The times may be modeled as random variables or as "worst case" values.

*$\triangleq$ means "equal by definition"

5.5.1.2 Transient Leakage

Transient fault recovery is divided into three states as shown below:

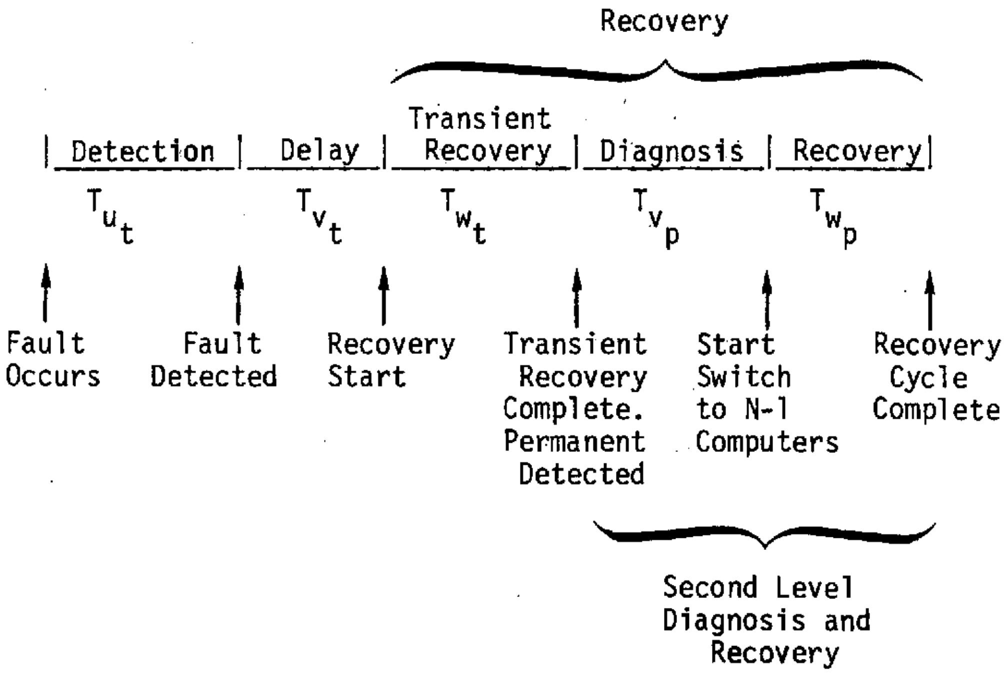


In this case detection is the fault indication generated by output comparators or by error detection RET's. Diagnostic time is a design delay to allow the transient to disappear. Recovery consists of one or more of rollback, rollahead, and memory copy depending on the number of operating computers and recovery design.

5.5.1.3 Permanent Recovery

Permanent fault recovery begins after an unsuccessful transient recovery. An uncovered transient, as well as a true permanent, may be declared as a permanent fault. After a fault is declared (or detected) as a permanent, diagnosis and recovery may proceed.

The overall picture of fault detection, diagnosis, and recovery in the presence of transients and permanents is shown below:



The subscripts t and p represent transient and permanent, respectively.

The times and probabilities are dependent on the particular configuration, the recovery procedure, and the number of computers that are working.

5.5.1.4 Notation System

Coverage and its component parts are in general different for the number of working computers in the configuration. We need to identify the number of computers and whether we are talking about transients or permanents. The notation is defined by the following table:

Type of Recovery No. of Computers	Permanent	Transient
1	$c_1, u_1, v_1, w_1, T_{u_1}, T_{v_1}, T_{w_1}$	ℓ_1 $u_s, v_s, w_s, T_{u_s}, T_{v_s}, T_{w_s}$
2	c_2, etc.	ℓ_2 u_d,
3	c_3, etc.	ℓ_3 u_t,
4	c_4, etc.	ℓ_4 u_f,
5	c_5, etc.	ℓ_5 u_q,

5.5.2 Analysis of a Duplex Configuration

5.5.2.1 Definitions and Assumptions

We define the following parameters for a duplex configuration:

1. $\ell_2 \triangleq$ Pr{Transient mistaken to be permanent while in Duplex | Transient occurs}

2. $v_2 w_2 \triangleq$ Pr{Successful adaptation to Simplex | Permanent or Leaky Transient occurs}

3. $\ell_1 =$ Pr{Transient mistaken to be permanent while in simplex | Transient occurs}

4. $\sigma_2 \triangleq \lambda + \ell_2 \tau$

The quantity ℓ_2 is the transient leakage in duplex. Transient recovery while in duplex is achieved by a rollback. Duplex transient leakage is composed then if:

1. Pr{Failure of rollback}

2. Pr{Transient duration lasts into rollback}

3. Pr{Other fault occurs during rollback}

The quantity $v_2 w_2$ is the product of the diagnostability v_2 and the recoverability w_2. Diagnostability is the probability of correctly locating a fault given the fault is detected as a permanent or uncovered transient. Recoverability is the probability of a successful adaptation to a simplex configuration given a correct location of the fault. The quantity σ_2 is the average rate of occurrence of permanent and leaky transients. Diagnosis is achieved by software self test in conjunction with BITE.

5.5.2.2 Fault/Recovery State Diagram

The fault occurrence/recovery state diagram of our duplex configuration is shown in Figure 5.5-1. From the no faults state a transient will send us the rollback state where a rollback is attempted. It is successful with probability $1-\ell_2$. If the rollback is not successful, then the fault is taken to be a permanent from where diagnosis and recovery is initiated. If the fault is permanent, then it is taken to be permanent with probability 1. In diagnosis and recovery, a recovery to simplex is achieved with probability $v_2 w_2$.

In simplex, it is possible to have some transient fault recovery. Detection is accomplished by error checking RET's (e.g., BITE). After detection, diagnosis is immediate. Recovery consists of rollback attempts.

The simplex transient leakage is then

$$\ell_1 = 1 - u_s w_s$$

where $v_s = 1$

5.5.2.3 Failure Probability

We define $F_N(T)$ as the probability of system failure before time $t=T$, given N working computers at time $t=0$. The probability of failure in simplex then becomes:

$$F_1(T) = 1 - e^{-\sigma_1 T}$$

where $\sigma_1 \triangleq \lambda + \ell_1 \tau$

If we set $\ell_1 = 1$, then $F_1(T)$ becomes the ordinary simplex failure probability.

And the probability of failure in duplex becomes:

$$\begin{aligned} F_2(T) &= \int_0^T e^{-2\sigma_2 t}\, 2\sigma_2 dt\ (1-v_2 w_2) \\ &+ \int_0^T e^{-2\sigma_2 t}\, 2\sigma_2\ v_2 w_2\ F_1(T-t)dt \\ &= (1-v_2 w_2)(1-e^{-2\sigma_2 T}) \\ &\quad + v_2 w_2 \int_0^T 2\sigma_2\ e^{-2\sigma_2 t}\ F_1(T-t)dt \end{aligned}$$

where $\sigma_2 = \lambda + \ell_2 \tau$ as before, and

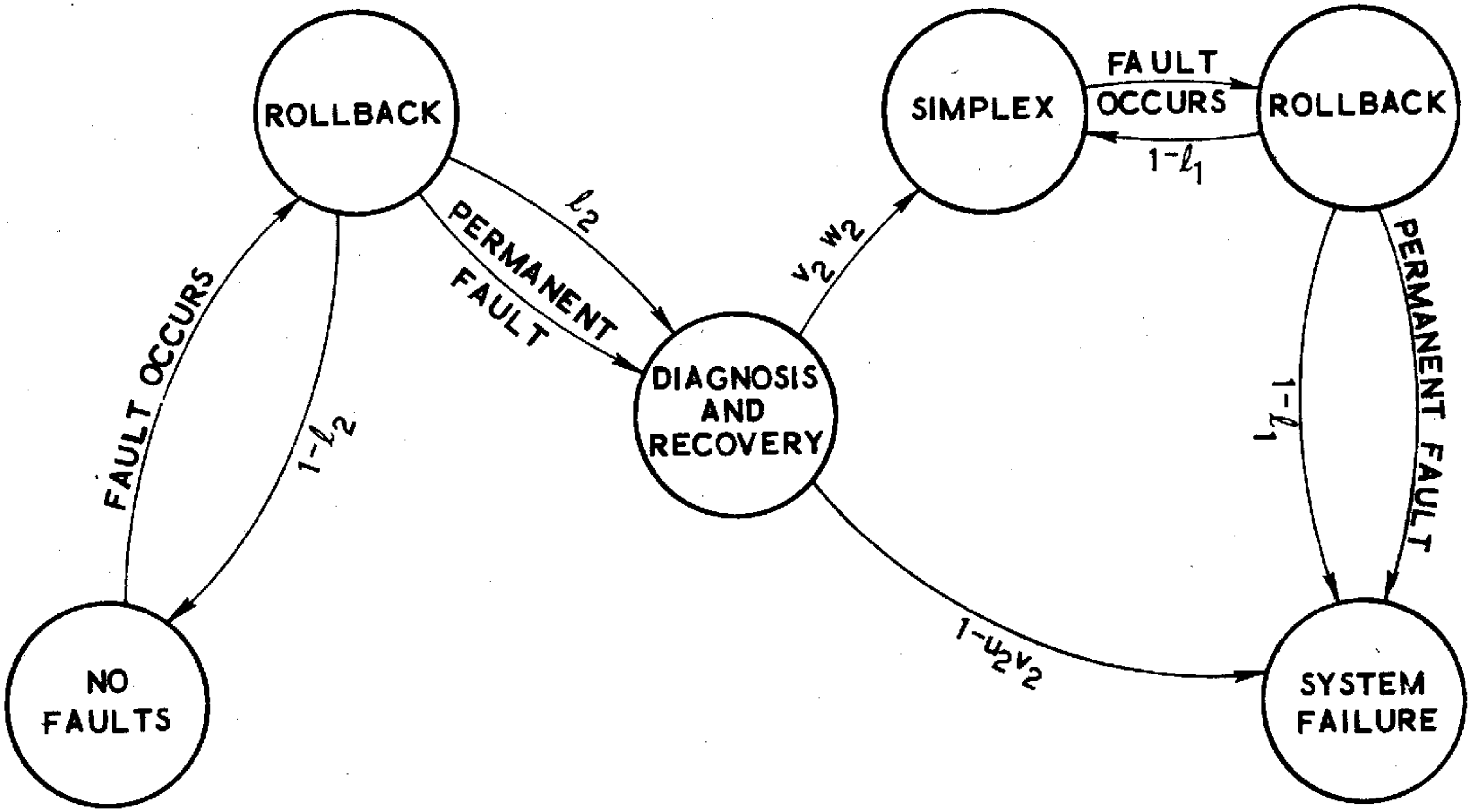


FIGURE 5.5-1 FAULT OCCURRENCE/RECOVERY STATUS STATE DIAGRAM FOR A DUPLEX CONFIGURATION

$$F_2(T) = \int_0^T e^{-2\sigma_2 T} \; 2 \; {}_2 dt \; [1 - v_2 + w_2(1 - e^{- \;_1 (T-t)})]$$

$$= 1 - e^{-2\sigma_2 T} - \frac{2\sigma_2 \; e^{-\sigma_t T}}{2\sigma_2 - \sigma_t} (1 - e^{-(2\sigma - \sigma_1)T}) \; v_2 w_2$$

If we let $\lambda = 0$ and $\ell_1 = 1$, then F(T) becomes

$$F_2(T) = 1 - e^{-2\ell_2 \tau T} + \frac{2\ell_2 e^{-\tau T}}{1-2(1-\ell_2)} (1 - e^{-(2\ell_2 - 1)\; T}) \; v_2 w_2$$

This is an approximation for the case where transient faults occur much more often than permanents. And if we let $\tau = 0$, we have for F(T)

$$F_2(T) = 1 - 2v_2 w_2 e^{-\lambda T} + (2v_2 w_2 - 1)e^{-2\lambda T}$$

which is the case when transients are not considered.

5.5.3 Extension to N Computers

5.5.3.1 State Diagram

The extension of Figure 5.5-1 is straightforward and is shown in Figure 3.5-2 for up to 5 computers. The state diagram presented in Figure 5.5-2 contains several states labeled "transient recovery." When there are three or more non-faulty computers prior to the occurrence of a fault, then the transient recovery process involves rollahead and memory copy (if utilized). The probability of success of the recovery procedure is reflected in the transient leakage parameter. The probability of a transient resulting in system failure is reflected by both the leakage (ℓ_i) and recoverability (w_i) parameters, each of which is determined through simulation runs using the simulator described in Section 6. Similar remarks apply to the duplex and simplex cases, but include the diagnostability (v_2).

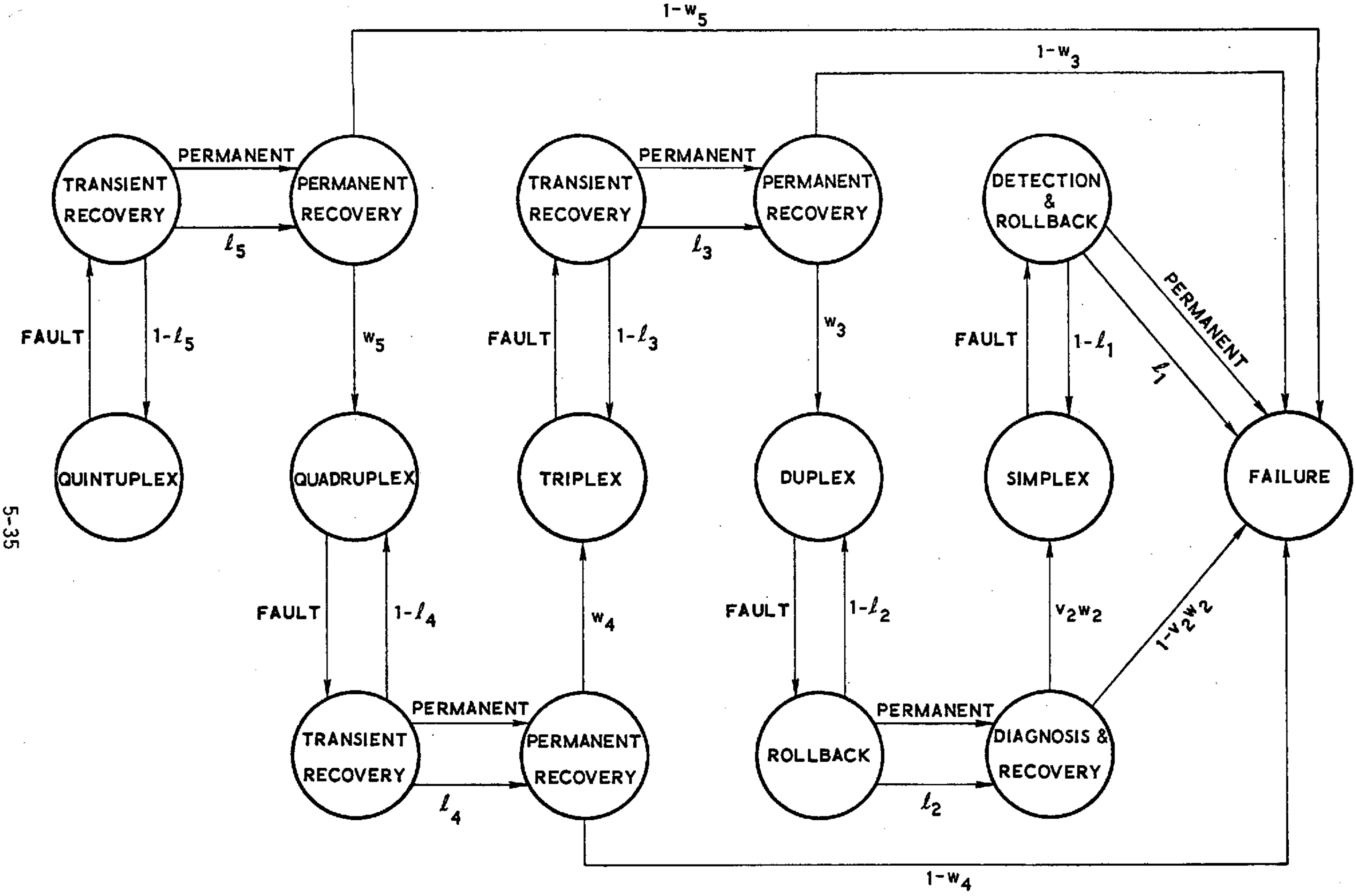


FIGURE 5.5-2 FAULT OCCURRENCE/RECOVERY STATUS STATE DIAGRAM FOR 1-5 COMPUTER CONFIGURATIONS

With 3 or more computers, we assume diagnosis is certain ($v_n=1$) since two or more faultless computers can point the finger at the faulty one; and u_n is considered to be one as in duplex since output comparison is used for fault detection. Permanent coverage then becomes

$$c_n = w_n \qquad n = 3, 4, 5$$

5.5.3.2 <u>Failure Probability Determination</u>

The failure probabilities then become

$$F_3(T) = (1-w_3)\ (1-3^{-3\sigma_3 T})$$

$$+ w_3 \int_0^T 3\sigma_3\ e^{-3\sigma_3 t}\ F_2(T-t\ dt$$

$$F_4(T) = (1-w_4)\ (1-e^{-4\sigma_4 T})$$

$$+ w_4 \int_0^T 4\sigma_4\ e^{-4\sigma_4 t}\ F_3(T-t)dt$$

$$F_5(T) = (1-w_5)\ (1-e^{-5\sigma_5 T})$$

$$+ w_5 \int_0^T 5\sigma_5\ e^{-5\sigma_5 t}\ F_4(T-t)dt$$

where $\sigma_3 = \lambda + \ell_3\tau$ as before

and $\sigma_4 \triangleq \lambda + \ell_4\tau$

$\sigma_5 \triangleq \lambda + \ell_5\tau$

If we assume $\ell_3 = \ell_4 = \ell_5$, then a general recursive expression for the failure probability can be given as

$$F_N(T) = (1-w_N)\ (1-e^{-\sigma_N T})$$

$$+ w_N \int_0^T N\sigma_N\ e^{-\sigma_N t}\ F_{N-1}(T-t)dt$$

where we replace $N\sigma_N$ by σ_N for rotational simplification.

5.5.3.3 General Solution

A general solution to $F_N(T)$ may be found by finding $S_N(T) = 1-F_N(T)$. Here we use C_N for w_n, $N \geq 3$; C_2 for $v_2 w_2$, and $C_1=0$ as well as using σ_N for $N\sigma_N$. If we substitute $1-S_N(T)$ in for $F_N(T)$ and simplify we get:

$$1 - S_N(T) = (1 - c_N)(1 - e^{-\sigma_N T}) + c_N \int_0^T \sigma_N e^{-\sigma_N t}\ (1 - S_{N-1}(T-t))dt$$

$$= (1 - c_N)(1 - e^{-\sigma_N T}) + c_N \int_0^T \sigma_N e^{-\sigma_N T} - c_N \int_0^T \sigma_N e^{-\sigma_N T} S_{N-1}(T-t)dt$$

$$= (1 - e^{-\sigma_N T}) - c_N(1 - e^{-\sigma_N T}) + c_N(1 - e^{-\sigma_N T}) - c_N \int_0^T \sigma_N e^{-\sigma_N T} S_{N-1}(T-t)dt$$

By rearranging terms we have:

$$S_N(T) = e^{-\sigma_N T} + c_N\ \sigma_N \int_0^T e^{-\sigma_N t}\ S_{N-1}(T-t)dt$$

Thus we have a recursive expression for the survivability of our N computer configuration. Since this is a convolution integral, we can re-write this as:

$$S_N(T) = e^{-\sigma_N T} + c_N \sigma_N \int_0^T e^{-\sigma_N (T-t)} S_{N-1}(t)dt$$

$$= e^{-\sigma_N T} (1 + c_N \sigma_N \int_0^T e^{\sigma_N t} S_{N-1}(t)dt)$$

Since $S_1(T) = 1 - F_1(t) = e^{-\sigma_1 T}$

We can find $S_2(T)$ by substitution. Thus

$$S_2(T) = e^{-\sigma_2 T} (1 + c_2 \sigma_2 \int_0^T e^{\sigma_2 t} e^{-\sigma_1 t} dt)$$

$$= e^{-\sigma_2 T} (1 + c_2 \sigma_2 \int_0^T e^{(\sigma_2 - \sigma_1)t} dt)$$

$$= e^{-\sigma_2 T} (1 + \frac{c_2 \sigma_2}{\sigma_2 - \sigma_1} (e^{(\sigma_2 - \sigma_1)T} - 1))dt$$

$$= \frac{c_2 \sigma_2}{\sigma_2 - \sigma_1} e^{-\sigma_1 T} + (1 - \frac{c_2 \sigma_2}{\sigma_2 - \sigma_1}) e^{-\sigma_2 T}$$

Note that both $S_1(T)$ and $S_2(T)$ can be expressed as a linear combination of exponential functions. It seems reasonable that $S_N(T)$ could also be expressed as a linear combination of exponential functions. In particular, assume

$$S_N(T) = \alpha_{N1} e^{-\sigma_1 T} + \alpha_{N2} e^{-\sigma_2 T} + \cdot \cdot \cdot + \alpha_{NN} e^{-\sigma_N T} = \sum_{j=1}^{N} \alpha_{Nj} e^{-\sigma_j T}$$

By substituting this expression for $S_N(T)$ into the recursive equation and simplifying, we obtain

$$S_{N+1}(T) = e^{-\sigma_{N+1}T} \left(1 + c_{N+1}\, \sigma_{N+1}\int_0^T e^{\sigma_{N+1}t}\, S_N(t)dt\right)$$

$$= e^{-\sigma_{N+1}T} \left(1 + c_{N+1}\, \sigma_{N+1}\int_0^T e^{\sigma_{N+1}t}\, \left(\sum_{j=1}^{N} \alpha_{Nj}\, e^{-\sigma_j t}\right)dt\right)$$

$$= e^{-\sigma_{N+1}T} \left(1 + c_{N+1}\, \sigma_{N+1}\sum_{j=1}^{N} \alpha_{Nj}\int_0^T e^{(\sigma_{N+1}-\sigma_j)t}\, dt\right)$$

$$= e^{-\sigma_{N+1}T} \left(1 + c_{N+1}\, \sigma_{N+1}\sum_{j=1}^{N} \frac{\alpha_{Nj}}{\sigma_{N+1}-\sigma_j}\left(e^{(\sigma_{N+1}-\sigma_j)T} - 1\right)\right)$$

$$= \sum_{j=1}^{N} \frac{c_{N+1}\sigma_{N+1}\alpha_{Nj}}{\sigma_{N+1}-\sigma_j}\, e^{-\sigma_j T} + \left(1 - \sum_{j=1}^{N} \frac{c_{N+1}\sigma_{N+1}\alpha_{Nj}}{\sigma_{N+1}-\sigma_j}\right) e^{-\sigma_{N+1}T}$$

$$= \sum_{j=1}^{N+1} \alpha_{N+1,j}\, e^{-\sigma_j T}$$

where

$$\alpha_{N+1,j} = \frac{c_{N+1}\sigma_{N+1}\alpha_{Nj}}{\sigma_{N+1}-\sigma_j} \qquad j=1,\ldots,N$$

$$\alpha_{N+1,N+1} = 1 - \sum_{j=1}^{N} \alpha_{N+1,j}$$

Thus by mathematical induction we have shown that the survivability of an N computer system can be expressed as a linear combination of exponential functions, and in the process have found an iterative expression for finding these coefficients. A FORTRAN program has been written to evaluate these coefficients on a computer and to plot the results on an automatic plotter.

5.5.4 Simplifying Assumptions

The formulas obtained for the failure probabilities of the various configurations can be simplified by representing the exponentials in each of them by a power series, and discarding higher order terms (i.e., let $e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \cdot\cdot\cdot$ and assume that $\frac{x^2}{2}$, $\frac{x^3}{6}$, etc. are small in comparison with x). This is a valid procedure as long as it is assumed that $\sigma_t T << 1$. (Note: This implies that $\sigma_N T < < 1$)

In the following discussion, define

$F_{3-E}(T)$ = Pr{Enhanced TMR fails before time T}

$F_{3-A}(T)$ = Pr{Adaptive TMR fails before time T}

ε = Relative error in simplified formulas (Relative error = $\frac{\text{actual error}}{\text{correct value}}$)

5.5.4.1 Simplex

$$F_1(T) = 1 - e^{-\sigma_1 T}$$

but $$e^{-\sigma_1 T} \approx 1-\sigma_1 T + \frac{\sigma_1{}^2 T^2}{2} - \frac{\sigma_1{}^3 T^3}{6} = \ldots \approx 1-\sigma_1 T$$

hence $$F_1(T) \approx \sigma_1 T$$

The relative error is given by

$$|\varepsilon| \approx \left| \frac{-\sigma_1{}^2 T^2/2 + \sigma_1{}^3 T^3/6 - \ldots}{\sigma_1 T - \sigma_1{}^2 T^2/2 + \sigma_1{}^3 T^3/6 - \ldots} \right| < \frac{\sigma_1 T/2}{1-\sigma_1 T/2} < \frac{\sigma_1 T}{2}$$

thus $$F_1(T) = \sigma_1 T \text{ with } |\varepsilon| < \frac{\sigma_1 T}{2}$$

5.5.4.2 Duplex

$$F_2(T) = 1 - e^{-2\sigma_2 T} - \frac{2\sigma_2 v_2 w_2}{2\sigma_2 - \sigma_1} (e^{-\sigma_1 T} - e^{-2\sigma_2 T})$$

$$\approx 2\sigma_2 \; 1 - v_2 w_2 T + v_2 w_2 \sigma_2 \sigma_1 T^2 \text{ where } |\varepsilon| < \sigma_2 T$$

Note that the second term can be ignored if ρ is not close to 1, i.e.,

$$F_2(T) \approx 2\sigma_2 \; (1 - v_2 w_2) T \text{ with } |\varepsilon| < \frac{\sigma_1 v_2 w_2 T}{2(1 - v_2 w_2)}$$

5.5.4.3 Enhanced TMR

$$F_{3-E}(T) = \sigma_\ell \{ \frac{1 - e^{-3\sigma_u T}}{\sigma_u} - \frac{3e^{-2\sigma_t T}}{3\sigma_u - 2\sigma_t} [1 - e^{-(3\sigma_u - 2\sigma_t)T}]\}$$

$$+ \frac{\tau}{T} (1 - e^{-3\sigma_u T}) \left(\frac{2\sigma_t + \delta \; (1 - e^{-2\sigma_t T_R})}{2\sigma_t + \delta}\right)$$

If we assume that $\sigma_\ell \approx \sigma_u = \sigma_3$ and that failures caused by transient overlaps are negligible, then we have

$$F_{3-E}(T) \approx 3\sigma_3 \sigma_t T^2 \text{ with } |\varepsilon| < \frac{5}{3} (\sigma_t T)$$

5.5.4.4 Adaptive TMR

$$F_{3-A}(T) = 1 - e^{-3\sigma_3 T} - \frac{6\sigma_3\sigma_2(1-v_2w_2) - 3\sigma_3\sigma_1}{(2\sigma_2-\sigma_1)(3\sigma_3-2\sigma_2)} (e^{-2\sigma_2 T} - e^{-3\sigma_2 T})$$

$$- \frac{6\sigma_3\sigma_2 v_2 w_2}{(2\sigma_2-\sigma_1)(3\sigma_3-\sigma_1)} (e^{-\sigma_1 T} - e^{-3\sigma_3 T})$$

$$\approx 3\sigma_3\sigma_2 \ (1-v_2w_2)T^2 + \rho\sigma_3\sigma_2\sigma_1 \ T^3 \text{ with } |\varepsilon| < \sigma_3 T$$

Note that $F_{3-A}(T) \approx 3\sigma_3\sigma_2 \ (1-v_2w_2)T^2$ with $|\varepsilon| < \frac{v_2w_2\sigma_1 T}{3(1-v_2w_2)}$

5.6 MARKOV CHAIN ANALYSIS METHOD

The basic approach is to model the computer configuration as a continuous parameter Markov chain, by assuming state transitions occur continuously. Once this is done we can develop a vector differential equation* representing the system and then obtain an expression for the state probabilities at any time t. This technique was used with the aid of computer programs to determine the state probabilities for the duplex and adaptive TMR configurations. The results agree with those of previous models.

5.6.1 Mathematical Model

Given a fault tolerant computer configuration, we model it by drawing a state diagram representing the status of the system.

In Figure 5.6-1 the nodes (S_1, S_2, S_3, S_4) represent the states and the branches represent the state transition paths. A transition occurs at each small time increment h.

* This is a compact form for a system of differential equations.

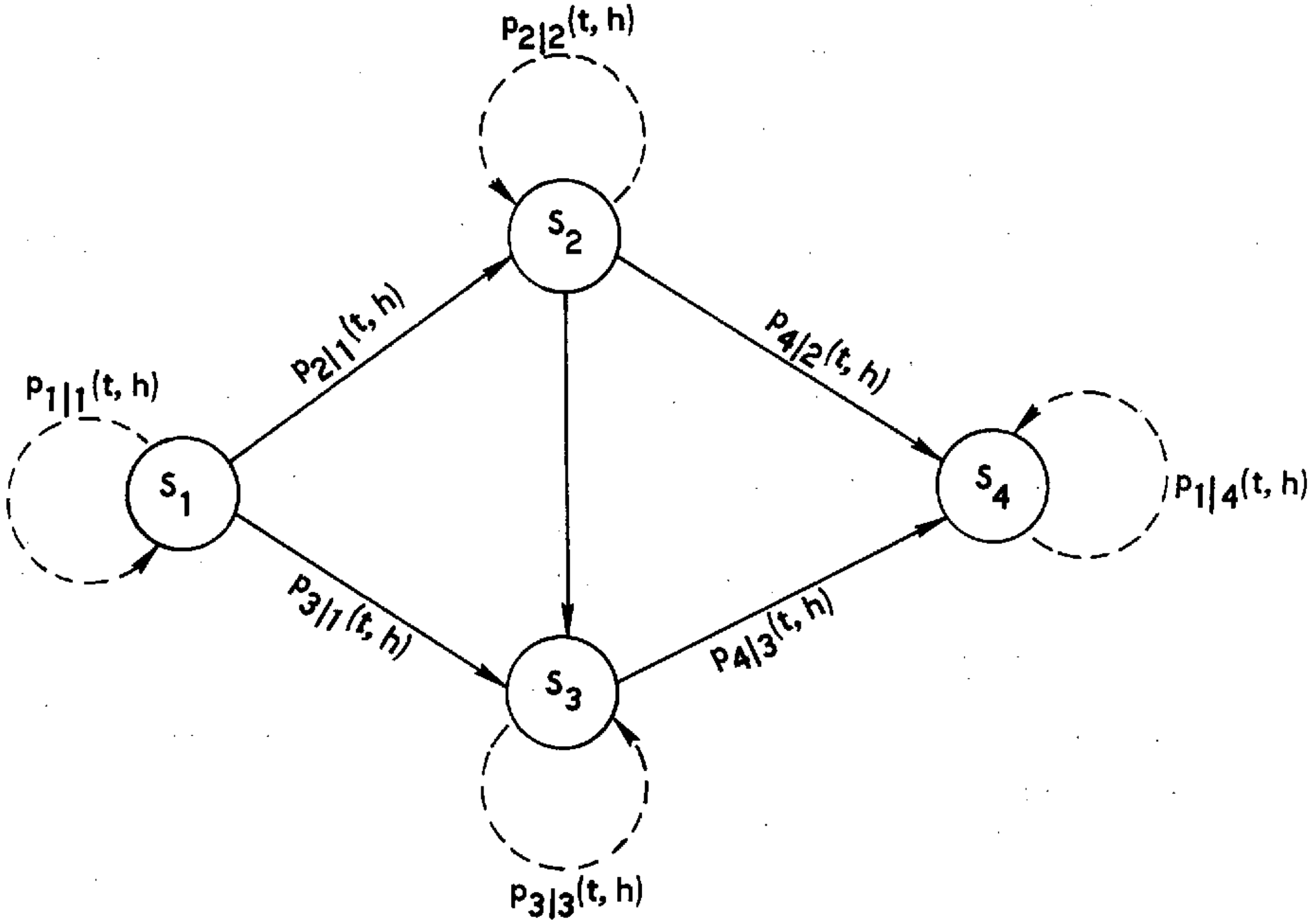


FIGURE 5.6-1 MARKOV CHAIN EXAMPLE

To each state S_i, assign a state probability function $P_i(t)$ defined as follows.

$$P_i(t) \triangleq \Pr\{\text{ The system will be in state } S_i \text{ at time } t\}$$

Also assign to each branch a conditional probability that is a function of t and h, defined by

$$p_{i|j}(t,h) = \Pr\{\text{ System in state } S_i \text{ at time } t+h | \text{it was in state } S_j \text{ at time } t\}$$

We make the assumption that $p_{i|j}(t,h)$ is independent of how we arrived in state S_j, and thus model the system as a discrete parameter Markov chain.

5.6.1.1 Development of the Differential Equation

We want to determine $P_i(t)$, the probability that the system will be in state S_i at time t. Because of the Markov assumption,

$$P_i(t) = \sum_j p_{i|j}(t,h)P_j(t) \qquad (1)$$

Given that a state S_i is occupied at time t, state S_j must be occupied at time t+h. Hence,

$$\sum_j p_{j|i}(t,h) = 1 \qquad \text{For } i = 1, 2, 3, \ldots N$$

By solving for $p_{i|i}(t,h)$, it follows that

$$p_{i|i}(t,h) = 1 - \sum_{\substack{j \\ j \neq i}} p_{j|i}(t,h)$$

By substituting this equation into Equation 1 we obtain

$$P_i(t+h) = \sum_{\substack{j \\ j \neq i}} p_{i|j}(t,h)P_j(t) + [1 - \sum_{\substack{j \\ j \neq i}} p_{j|i}(t,h)]P_i(t)$$

An equivalent expression obtained by subtracting $P_i(t)$ from both sides is

$$P_i(t+h) - P_i(t) = \sum_{\substack{j \\ j \neq i}} p_{i|j}(t,h)P_j(t) - P_i(t) \sum_{\substack{j \\ j \neq i}} p_{j|i}(t,h)$$

By defining

$$\Phi_{ij}(t,h) = \begin{cases} p_{i|j}(t,h) & \text{for } i \neq j \\ - \sum\limits_{\substack{j \\ j \neq i}} p_{j|i}(t,h) & \text{for } i=j \end{cases}$$

We can write the above equation as

$$P_i(t+h) - P_i(t) = \sum_j \phi_{ij}(t,h)P_j(t)$$

or in vector notation, we have

$$\underline{P}(t+h) - \underline{P}(t) = \underline{\Phi}\,(t,h)\underline{P}(t) \qquad (2)$$

where

$$\underline{P}(t) \triangleq \begin{bmatrix} P_1(t) \\ P_2(t) \\ \circ \\ \circ \\ \circ \\ P_N(t) \end{bmatrix}$$

and

$$\underline{\Phi}(t,h) = \begin{bmatrix} \Phi_{11}(t,h) & \Phi_{12}(t,h) & \circ\ \circ\ \circ & \Phi_{1N}(t,h) \\ \Phi_{21}(t,h) & \Phi_{22}(t,h) & \circ\ \circ\ \circ & \Phi_{2N}(t,h) \\ \circ & & & \\ \circ & & & \\ \circ & & & \\ \Phi_{N1}(t,h) & \Phi_{N2}(t,h) & \circ\ \circ\ \circ & \Phi_{NN}(t,h) \end{bmatrix}$$

Equation 2 represents an iterative relationship, that may be used to obtain the various state probabilities for any time t. However, we can obtain a result that is easier to evaluate if we make some further simplifications. Assume that state transitions occur continuously, so the system can be modeled as a continuous parameter Markov chain. The above result is then extended for a continuous model as follows.

By dividing both sides of Equation 2 by h and taking the limit as h approaches zero, we obtain

$$\lim_{h \to o} \frac{1}{h}(\underline{P}(t+h) - \underline{P}(t)) = \lim_{h \to o} \frac{1}{h}\, \underline{\Phi}\, (t,h) \qquad (3)$$

Now, let

$$\underline{\beta}(t) \triangleq \lim_{h \to o} \frac{1}{h}\, \underline{\Phi}(t,h)$$

Then since

$$\frac{d}{dt}\, \underline{P}(t) \triangleq \lim_{h \to o} \frac{1}{h}\, (\underline{P}(t+h) - P(t))$$

We can rewrite Equation 3 as

$$\frac{d}{dt}\, \underline{P}(t) = \underline{\beta}(t)\underline{P}(t)$$

If we make the further assumption that $\underline{\beta}(t)$ is independent of time, we obtain

$$\frac{d}{dt}\underline{P}(t) = \underline{\beta}\;\underline{P}(t) \qquad (4)$$

which actually represents the system of N linear homogeneous differential equations

$$\frac{dP_i(t)}{dt} = \sum_j \beta_{ij} P_j(t) \qquad \text{for } i = 1, 2, \ldots, N$$

5.6.1.2 Solution Procedure

To solve Equation 4, we define an operator $e^{\underline{\beta}t}$ as follows

$$e^{\underline{\beta}t} \triangleq \underline{I} + \underline{\beta}t + \frac{1}{2}\underline{\beta}^2t^2 + \frac{1}{6}\underline{\beta}^3t^3 + \ldots* \qquad (5)$$

Then
$$\frac{d}{dt}\, e^{\underline{\beta}t} = \frac{d}{dt}\underline{I} + \underline{\beta}\frac{d}{dt}t + \frac{1}{2}\underline{\beta}^3\frac{d}{dt}t^2 + \frac{1}{6}\underline{\beta}^3\frac{d}{dt}t^3 + \ldots$$

$$= \underline{\beta}\,(\underline{I} + \underline{\beta}\,t + \frac{1}{2}\underline{\beta}^2t^2 + \ldots)$$

$$= \underline{\beta}\; e^{\underline{\beta}t}$$

It then becomes apparent that the solution to Equation 4 is

$$\underline{P}(t) = (e^{\underline{\beta}t})\underline{P}(o) \qquad (6)$$

Since

$$\frac{d}{dt}\underline{P}(t) = (\underline{\beta} e^{\underline{\beta}t})\underline{P}(o) = \underline{\beta}\;\underline{P}(t)$$

*Here $\underline{I}$ is the identity matrix, and $\underline{\beta}^m$ is defined by the recursive relationship $\underline{\beta}^m = \underline{\beta} \cdot \underline{\beta}^{m-1}$ and $\underline{\beta}^1 = \underline{\beta}$.

and

$$\underline{P}(o) = (e^{\underline{\beta}\, o})\underline{P}(o) = \underline{I}\ \underline{P}(o) = \underline{P}(o)$$

$e^{\underline{\beta}\, t}$ can either be evaluated by means of the above series, or in closed form by a spectral expansion.

5.6.1.3 Closed Form Solution

The approach here is to expand $\underline{\beta}$ on a set of matricies $\{\underline{Q}_1, \underline{Q}_2, \ldots, \underline{Q}_N\}$ called projectors. Projectors have the following properties (see DENN 67 Ch 2).

1. $\underline{Q}_i^2 = \underline{Q}_i \cdot \underline{Q}_i = \underline{Q}_i$
2. $\underline{Q}_i \cdot \underline{Q}_j = \underline{0}$ for $i \neq j$
3. $\underline{Q}_1 + \underline{Q}_2 + \ldots + \underline{Q}_N = \underline{I}$
4. $\underline{\beta} = \alpha_1 \underline{Q}_1 + \alpha_2 \underline{Q}_2 + \ldots + \alpha_N \underline{Q}_N$ where $\{\alpha_i\}$ is the set of eigenvalues of $\underline{\beta}$.

Using these properties, it is easy to show that

$$\underline{\beta}^m = \alpha_1^m \underline{Q}_1 + \alpha_2^m \underline{Q}_2 + \ldots + \alpha_N^m \underline{Q}_N$$

and thus

$$\begin{aligned} e^{\underline{\beta} t} &= \underline{I} + \underline{\beta} t + \tfrac{1}{2}\underline{\beta}^2 t^2 + \ldots \\ &= (\underline{Q}_1 + \ldots + \underline{Q}_N) + (\alpha_1 \underline{Q}_1 + \ldots + \alpha_N \underline{Q}_N)t + \tfrac{1}{2}(\alpha_1^2 \underline{Q}_1 + \ldots + \\ &\quad \alpha_N^2 \underline{Q}_N)t^2 + \ldots \\ &= (1 + \alpha_1 t + \tfrac{1}{2}\alpha_1^2 t^2 + \ldots)\ \underline{Q}_1 + \ldots + (1 + \alpha_N t + \tfrac{1}{2}\alpha_N^2 t^2 + \ldots) \\ &= e^{\alpha_1 t}\ \underline{Q}_1 + \ldots + e^{\alpha_N t}\ \underline{Q}_N \end{aligned}$$

A matrix can be expanded on projectors having the above properties, if and only if it has a set of N linearly independent eigenvectors. This will always be the case if the matrix has N distinct eigenvalues.

If it is assumed that the configuration can only get worse (it can only go from N computers to N-1 computers, and not vice versa), then the state diagram will have no loops in it. In this case, the transition matrix $\underline{\beta}$ can always be written in lower triangular form (all elements above the diagonal are 0) for a proper ordering of states. Since the eigenvalues of a triangular matrix are just its diagonal elements, the projectors $\{\underline{Q}_1, \underline{Q}_2, \ldots, \underline{Q}_N\}$ can be easily calculated as follows:

1. $\alpha_i = \beta_{ii}$

2. $\underline{Q}_i = \dfrac{\varphi_i(\underline{\beta})}{\varphi_i(x)}$ where $\varphi_i(x) \triangleq \prod_{\substack{k=1 \\ k\neq i}}^{N} (x-\alpha_k)$

$\underline{P}(t)$ then turns out to be

$$\underline{P}(t) = \sum_{i=1}^{n} e^{\alpha_i t} \underline{Q}_i \underline{P}(o)$$

5.6.1.4 Power Series Evaluation of $\underline{P}(t)$

$\underline{P}(t)$ can also be evaluated directly using the power series expansion for $\underline{P}(t)$. This approach is useful for determining algebraic approximations for the state probabilities. It is also a much more general procedure, and can be used for a system whose transition matrix is not easily expanded as a linear combination of projector matricies.

From Equations 5 and 6 we have

$$\underline{P}(t) = (\underline{I} + \underline{\beta}t + \frac{1}{2}\underline{\beta}^2 t^2 + \frac{1}{6}\underline{\beta}^3 t^3 + \ldots)$$

$$= \underline{P}(o) + (\underline{\beta}\,\underline{P}(o))t + (\frac{1}{2}\underline{\beta}^2\underline{P}(o))t^2 + (\frac{1}{6}\underline{\beta}^3\underline{P}(o))t^3$$

$$= \underline{A}_o + \underline{A}_1 t + \underline{A}_2 t^2 + \underline{A}_3 t^3 + \ldots$$

where the $\underline{A}_i$'s are column vectors defined by the iterative relationship

$$\underline{A}_o = \underline{P}(o)$$

$$\underline{A}_i = \frac{1}{i}\,\underline{\beta}\,\underline{A}_{i-1}$$

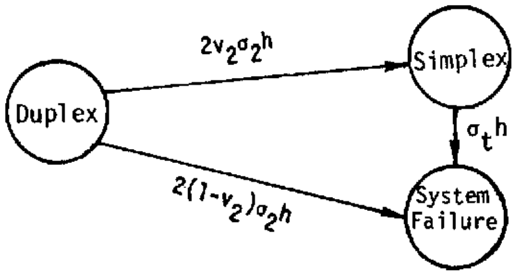
5.6.2 Application to the Duplex Configuration

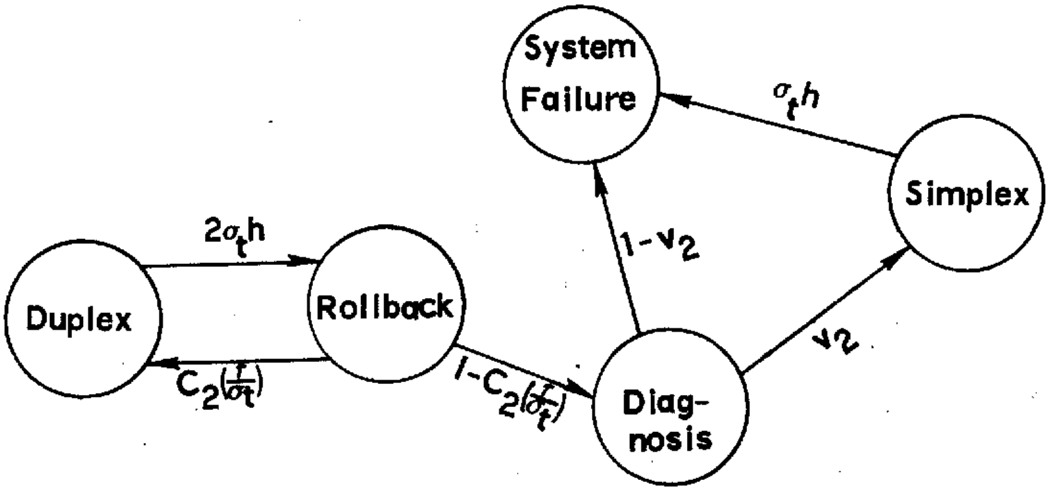
To determine the state probabilities for the duplex configuration, we make several assumptions.

1. Permanent and transient failures occur independently with mean rates λ and τ respectively.
2. The failure occurrences have an exponential density function.
3. The system is continuous - i.e., the time spent in the transient recovery states is negligible, and a multiple fault cannot occur.

5.6.2.1 Determination of the Transition Matrix

In Figure 5.6-2, the important states are duplex, simplex and system failure. The time spent in the other 2 states (rollback and diagnosis) is negligible (zero if it is assumed that the system is continuous), hence we combine them with the duplex state to obtain the following state diagram. In this example, we assume $w_2 = 1$ and $\ell_1 = 1$.





λ = Permanent fault rate

τ = Transient fault rate

σ_t = $\tau + \lambda$ = Total fault rate

$\rho = v_2 w_2$ = Pr {System switches successfully from duplex to simplex}

C_2 = Pr {Transient error corrected by rollback} (= $1-\ell_2$)

$\sigma_t h$ = Pr {Fault occurs in time increment h}

τ/σ_t = Pr {Fault is transient}

σ_2 = $\lambda + (1-C_2)\tau = \sigma_t \ (1-C_2(\frac{\tau}{\sigma_t}))$

FIGURE 5.6-2 STATE DIAGRAM FOR THE DUPLEX CONFIGURATION

Where the transition duplex → simplex actually corresponds to the sequence of transitions.

Duplex → Rollback → Diagnosis → Simplex

and the transition

Duplex → System Failure

corresponds to the sequence of transitions

Duplex → Rollback → Diagnosis → System Failure

The new conditional transition probabilities are the product of the conditional probabilities along the path, hence since

$$\Pr\{\text{Duplex} \rightarrow \text{Rollback}\} = 2\sigma_t h$$

$$\Pr\{\text{Rollback} \rightarrow \text{Diagnosis}\} = 1 - C\left(\frac{\tau}{\sigma}\right)$$

and $\Pr\{\text{Diagnosis} \rightarrow \text{System Failure}\} = 1-\rho$

We have

$$\Pr\{\text{Duplex} \rightarrow \text{System Failure}\} = (2\sigma_t h)\left(1 - C_2\left(\frac{\tau}{\sigma_2}\right)\right)(1-v_2)$$

$$= 2(1-v_2)\sigma_2 h$$

Similarly we obtain $\Pr\{\text{Duplex} \rightarrow \text{Simplex}\} = 2v_2\sigma_2 h$

Let S_1 = Duplex

S_2 = Simplex

S_3 = System Failure

Then from the above figure we obtain

$$p_{2|1}(t,h) = 2v_2\sigma_2 h \quad \Rightarrow \quad \phi_{21}(t,h) = 2v_2\sigma_2 h$$

$$p_{3|1}(t,h) = 2(1-v_2)\sigma_2 h \quad \Rightarrow \quad \phi_{31}(t,h) = 2(1-v_2)\sigma_2 h$$

$$p_{3|2}(t,h) = \sigma_t h \quad \Rightarrow \quad \phi_{32}(t,h) = \sigma_t h$$

Also $\phi_{11}(t,h) = -(p_{2|1}(t,h) + p_{3|1}(t,h)) = -2\sigma_2 h$

$\phi_{22}(t,h) = -\ \sigma_t h$

and $\phi_{33}(t,h) = 0$

Thus the transition matrix is $\underline{\Phi}(t,h) = \begin{bmatrix} -2\sigma_2 h & 0 & 0 \\ 2v_2\sigma_2 h & -\sigma_t h & 0 \\ 2(1-v_2)\sigma_2 h & \sigma_t h & 0 \end{bmatrix}$

$$\underline{\beta} = \lim_{h\to o} \frac{1}{h}\underline{\Phi}(t,h) = \begin{bmatrix} -2\sigma_2 & 0 & 0 \\ 2v_2\sigma_2 & -\sigma_t & 0 \\ 2(1-v_2)\sigma_2 & \sigma_t & 0 \end{bmatrix}$$

5.6.2.2 Closed Form Solution for Duplex Configuration

$$\underline{P}(t) = (e^{\underline{\beta}t})\underline{P}(o) \qquad \text{where } \underline{P}(t) = \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

and $P_1(t)$ = Pr { System is in duplex state at time t}

$P_2(t)$ = Pr { System is in simplex state at time t}

$P_3(t)$ = Pr { System has failed by time t}

Earlier it was stated that

$$\underline{P}(t) = \sum_{i=1}^{N} e^{\alpha_i t}\,\underline{Q}_i\underline{P}(o)$$

where $\alpha_i = \beta_{ii}$

$$\underline{Q}_i = \frac{\varphi_i(\underline{\beta})}{\varphi_i(\alpha_i)}$$

and $$\varphi_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{N} (x-\alpha_i)$$

For the special case where $\underline{\beta} = \begin{bmatrix} -2\sigma_2 & 0 & 0 \\ 2v_2\sigma_2 & -\sigma_t & 0 \\ 2(1-v_2)\sigma_2 & \sigma_t & 0 \end{bmatrix}$

The eigenvalues are

$$\alpha_1 = -2\sigma_2$$

$$\alpha_2 = -\sigma_t$$

$$\alpha_3 = 0$$

and the $\varphi_{i's}$ are

$$\varphi_1(x) = x(x+\sigma_t)$$

$$\varphi_2(x) = x(x+2\sigma_2)$$

$$\varphi_3(x) = (x+\sigma_t)(x+2\sigma_2)$$

so $\varphi_1(\alpha_1) = \varphi_1(-2\sigma_2) = 2\sigma_2(2\sigma_2-\sigma_t)$

$$\varphi_2(\alpha_2) = \varphi_2(-\sigma_t) = -\sigma_t(2\sigma_2-\sigma_t)$$

$$\varphi_3(\alpha_3) = \varphi_3(o) = 2\sigma_t\sigma_2$$

Also

$$\varphi_1(\underline{\beta}) = \begin{bmatrix} -2\sigma_2 & 0 & 0 \\ 2v_2\sigma_2 & -\sigma_t & 0 \\ 2(1-v_2)\sigma_2 & \sigma_t & 0 \end{bmatrix} \times \begin{bmatrix} \sigma_t-2\sigma & 0 & 0 \\ 2v_2\sigma_2 & 0 & 0 \\ 2(1-v_2)\sigma_2 & \sigma_t & \sigma_t \end{bmatrix} = \begin{bmatrix} -2\sigma_2(\sigma_t-2\sigma\ ) & 0 & 0 \\ -4v_2\sigma_2 & 0 & 0 \\ 2\sigma_t\sigma_2-4(1-v_2)\sigma_2^2 & 0 & 0 \end{bmatrix}$$

$$\varphi_2(\underline{\beta}) = \begin{bmatrix} 0 & 0 & 0 \\ -2v_2\sigma_2 & -\sigma_t(2\sigma_2-\sigma_t) & 0 \\ 2v_2\sigma_2\sigma_t & \sigma_t(2\sigma_2-\sigma_t) & 0 \end{bmatrix}$$

and

$$\varphi_3(\underline{\beta}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2\sigma_2\sigma_t & 2\sigma_2\sigma_t & 2\sigma_2\sigma_t \end{bmatrix}$$

Thus the projectors are

$$\underline{Q}_1 = \frac{\varphi_1(\underline{\beta})}{\varphi_1(\alpha_1)} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{-2v_2\sigma_2}{2\sigma_2-\sigma_t} & 0 & 0 \\ -(1-\frac{2v_2\sigma_2}{2\sigma_2-\sigma_t} & 0 & 0 \end{bmatrix}$$

$$\underline{Q}_2 = \begin{bmatrix} 0 & 0 & 0 \\ \frac{2v_2\sigma_2}{2\sigma_2-\sigma_t} & 1 & 0 \\ \frac{-2v_2\sigma_2}{2\sigma_2-\sigma_t} & -1 & 0 \end{bmatrix}$$

and

$$\underline{Q}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

A closed form solution is thus

$$\underline{P}(t) = e^{-2\sigma_2 t}\,\underline{Q}_1\,\underline{P}(o) + e^{-\sigma_t t}\,\underline{Q}_2\,\underline{P}(o) + \underline{Q}_3\,\underline{P}(o)$$

where $\underline{P}(o) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ since the probability of being in the duplex state at t=o is one.

Hence,

$$\underline{P}(t) = e^{-2\sigma_2 t} \begin{bmatrix} 1 \\ \frac{-2v_2\sigma_2}{2\sigma_2-\sigma_t} \\ -(1-\frac{2v_2\sigma_2}{2\sigma_2-\sigma_t}) \end{bmatrix} + e^{-\sigma_t t} \begin{bmatrix} 0 \\ \frac{2v_2\sigma_2}{2\sigma_2-\sigma_t} \\ \frac{-2v_2\sigma_2}{2\sigma_2-\sigma_t} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Thus $P_1(t)$ = Pr { System is in duplex state at time t}

$$= e^{-2\sigma_2 t}$$

$P_2(t)$ = Pr { System is in simplex state at time t}

$$= \frac{2v_2\sigma_2}{2\sigma_2-\sigma_t} [e^{-\sigma_t t} - e^{-2\sigma_2 t}]$$

$P_3(t)$ = Pr { System has failed by time t}

$$= 1 - (1- \frac{2v_2\sigma_2}{2\sigma_2-\sigma_t})\ e^{-2\sigma_2 t} - \frac{2v_2\sigma_2}{2\sigma_2-\sigma_t}\ e^{-\sigma_t t}$$

Note that $P_3(t)$ agrees with the other results.

5.6.2.3 Approximation for Small Mission Times

A quick approximation for small mission times, suitable for hand computation, can be obtained by evaluating the first few terms of the power series expansion.

The transition matrix is

$$\underline{\beta} = \begin{bmatrix} -2\sigma_2 & 0 & 0 \\ 2v_2\sigma_2 & -\sigma_t & 0 \\ 2(1-v_2)\sigma_2 & \sigma_t & 0 \end{bmatrix}$$

Earlier it was shown that $\underline{P}(o) \approx \underline{A}_o + \underline{A}_1 t + \underline{A}_2 t^2 + \ldots$

where $\underline{A}_i = \frac{1}{i}\,\underline{\beta}\,\underline{A}_{i-1}$ and $\underline{A}_o = \underline{P}(o)$

Using this for the duplex case we have

$$\underline{A}_o = \underline{P}(o) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\underline{A}_1 = \underline{\beta}\,\underline{A}_o = \begin{bmatrix} -2\sigma_2 \\ 2v_2\sigma_2 \\ 2(1-v_2)\sigma_2 \end{bmatrix}$$

$$\underline{A}_2 = \frac{1}{2}\,\underline{\beta}\,\underline{A}_1 = \begin{bmatrix} 2\sigma_2{}^2 \\ -2v_2\sigma_2(\sigma_2+\sigma_t) \\ -2(1-v_2)\sigma_2{}^2+v_2\sigma_2\sigma_t \end{bmatrix}$$

Thus $$\underline{P}(t) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + t \begin{bmatrix} -2\sigma_2 \\ 2v_2\sigma_2 \\ 2(1-v_2)\sigma_2 \end{bmatrix} + t^2 \begin{bmatrix} 2\sigma_2{}^2 \\ -2v_2\sigma_2(\sigma_2+\sigma_t) \\ -2(1-v_2)\sigma_2{}^2+v_2\sigma_2\sigma_t \end{bmatrix}$$

So we have

$$P_1(t) = \Pr\{\text{System in duplex state}\} \approx 1-2\sigma_d t + 2\sigma_d{}^2 t^2$$

$$P_2(t) = \Pr\{\text{System in simplex state}\} \approx 2v_2\sigma_2 t - 2v_2\sigma_2(\sigma_2+\sigma_t)t^2$$

$$P_3(t) = \Pr\{\text{System has failed}\} \approx 2(1-v_2)\sigma_2 t(1-\sigma_2 t)+ (v_2\sigma_2\sigma_t)t^2$$

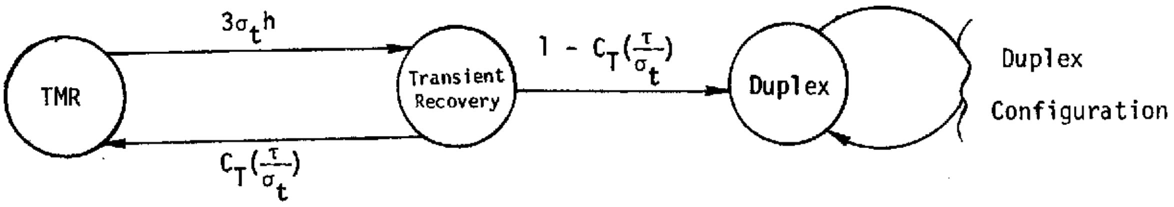
The approximation for $P_3(t)$ agrees with the result obtained in Section 5.5.4.2.

5.6.3 Application to Adaptive TMR Configuration

The same assumptions are made in this analysis as were made earlier for the analysis of the duplex configuration.

5.6.3.1 Determination of the Transition Matrix

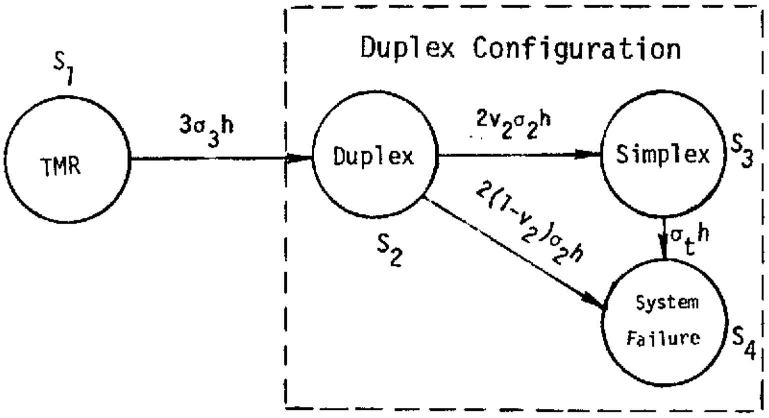
The state diagram for the adaptive TMR configuration is



$3\sigma_t h$
TMR
Transient Recovery
$1 - C_T(\frac{\tau}{\sigma_t})$
Duplex
Duplex Configuration
$C_T(\frac{\tau}{\sigma_t})$

where C_T= Pr{ Transient error corrected by rollahead} (= $1-\ell_3$)

The time spent in the transient recovery state is negligible, so we combine it with the TMR state to obtain



The transition matrix is thus

$$\underline{\beta} = \left[\begin{array}{c|ccc|} -3\sigma_3 & 0 & 0 & 0 \\ \cline{2-4} 3\sigma_3 & -2\sigma_2 & 0 & 0 \\ 0 & 2v_2\sigma_2 & -\sigma_t & 0 \\ 0 & 2(1-v_2)\sigma_2 & \sigma_t & 0 \\ \cline{2-4} \end{array}\right]$$

Note that the outlined portion surrounds the transition matrix for the duplex configuration.

5.6.3.2 Approximations for Small Mission Times

An approximation for the state probabilities can be obtained by evaluating the first 4 terms of the power series expansion. Thus

$$A_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\underline{A}_1 = \begin{bmatrix} -3\sigma_3 \\ 3\sigma_3 \\ 0 \\ 0 \end{bmatrix}$$

$$\underline{A}_2 = \frac{1}{2} \begin{bmatrix} 9\sigma_3^{\,2} \\ -(9\sigma_3^{\,2} + 6\sigma_3\sigma_2) \\ 6v_2\sigma_3\sigma_2 \\ 6(1-v_2)\sigma_3\sigma_2 \end{bmatrix}$$

$$\underline{A}_3 = \frac{1}{6} \begin{bmatrix} -27\sigma_3{}^3 \\ 27\sigma_3{}^3 + 2\sigma_2(9\sigma_3{}^2+6\sigma_3\sigma_2) \\ -2v_2\sigma_2(9\sigma_3{}^2+6\sigma_3\sigma_2) -6v_2\sigma_3\sigma_2\sigma_t \\ -2(1-v_2)\sigma_2(9\sigma_3{}^2+6\sigma_3\sigma_2)+6v_2\sigma_3\sigma_2\sigma_t \end{bmatrix}$$

Thus since $\underline{P}(t) \approx \underline{A}_o + t\underline{A}_1 + t^2\underline{A}_2 + t^3\underline{A}_3$

we have

$$\begin{aligned} P_4(t) &= \Pr\{\text{ System has failed by time t}\} \\ &\approx \tfrac{1}{2}(6(1-v_2)\sigma_3\sigma_2)t^2 + \tfrac{1}{6}\,(2-(1-v_2)\sigma_2(9\sigma_3{}^2+6\sigma_3\sigma_2) + 6v_2\sigma_3\sigma_2\sigma_t)t^3 \\ &\approx (1-v_2)\sigma_3\sigma_2t^2\ [3-(3\sigma_3+2\sigma_2)t] + v_2\sigma_3\sigma_2\sigma_tt^3 \\ &\approx 3(1-v_2)\sigma_3\sigma_2t^2 + v_2\sigma_3\sigma_2\sigma_tt^3 \text{ assuming } (3\sigma_3+2\sigma_2)t \ll 3 \end{aligned}$$

5.6.4 Programs

Several programs have been written in APL to aid in calculating state probabilities for various mission times. These programs have been tested for both the duplex and adaptive TMR, and the results agree with those obtained earlier for the Interim Report.

5.6.4.1 Projector Method

```
         ∇PROJECTOR[⎕]∇
       ∇ PROJECTOR A;I;N;P;J;K;Q
[1]      I←(ρA)ρ1,(N←1↑ρA)ρ0
[2]      EVS← 1 1 ⍉A
[3]      TRANS←(0,ρA)ρ0
[4]      P←((N,ρA)ρA)-EVS∘.×I
[5]      J←1
[6]     LP1:K←2
[7]      R←(~I[J;])/⍳N
[8]      Q←P[R[1];;]
[9]     LP2:Q←Q+.×P[R[K];;]
[10]     →(N>K←K+1)/LP2
[11]     TRANS←TRANS,[1] Q÷×/(~I[J;])/EVS[J]-EVS
[12]     →(N≥J←J+1)/LP1
       ∇
```

This program determines the eigenvalues and projectors of a triangular matrix - A. The resulting eigenvalues are returned in the vector - EVS, and the projectors are returned in a multidimensional array - TRANS.

```
         ∇EVAL1[⎕]∇
       ∇ R←IC EVAL1  T
  [1]    R←(*T∘.×EVS)+.×TRANS+.×IC
       ∇
```

This program uses the results returned by PROJECTOR to determine the state probabilities for various mission times. The initial condition - IC, and a vector of mission times - T are the arguments of EVAL1. An array of state probabilities is returned as a result.

5.6.4.2 Power Series Method

```
         ∇TRAN[⎕]∇
       ∇ R←IC TRAN  A;M
  [1]    M←A+.×IC
  [2]    R←M,[0.5] IC
  [3]    I←2
  [4]   LP:M←A+.×M÷I
  [5]    R←M,[1] R
  [6]    →(NT≥I←I+1)/LP
       ∇
```

This program returns the set of column vectors $\underline{A}_0$, $\underline{A}_1$, $\underline{A}_2$, ... (see Section 5.6.1.4) in the form of a matrix. The number of terms (column vectors) is determined by a global variable - NT. The initial conditions - IC, and the transition matrix - A are the left and right arguments of TRAN respectively.

```
         ∇EVAL[⎕]∇
       ∇ R←A EVAL T
  [1]    R←(((ρT),1)ρT)⊥A
       ∇
```

Given a matrix of column vectors [$\underline{A}_0$, $\underline{A}_1$, $\underline{A}_2$, ..., $\underline{A}_N$] and a vector of times - T, this program returns the sum $\underline{A}_0 + \underline{A}_1 t + \underline{A}_2 t^2 + \underline{A}_N t^N$ for each t. It is used in conjunction with TRAN to obtain $\underline{P}(t)$.

```
         ∇ERR[⎕]∇
       ∇ R←T ERR  A
  [1]    R←A EVAL  T
  [2]    R←(R-(((ρρA)↑1)↓A) EVAL  T)÷R
       ∇
```

This program gives an upper bound on the error of the power series expansion. The maximum time - T is the left argument, and the matrix of column vectors - A is the right argument.

5.6.5 Conclusions

The above procedure for determining the failure probabilities of a fault tolerant computer has several advantages over the earlier approach.

1. All state probabilities are obtained as a function of time. This allows a more detailed study of a given computer configuration.

2. The model is general for any number of computers only the transition matrix is needed to obtain numerical results on a computer.

3. It is much easier to obtain approximations for the state probabilities as a function of time.

6.0 SIMULATION

6.1 OBJECTIVES OF SIMULATION

The function of the simulator portion of CAST has been described briefly in the Summary and is treated in more detail in Section 8.1. Translating this function into simulation objectives yields the following three items. The simulator should produce:

1. The fault-tolerance effectiveness of each of a wide variety of reconfigurable computer system configurations;
2. Global parameters for use in analytic modeling;
3. The behavior of a configuration in various fault environments.

The requirements imposed on the simulator design by these three objectives are examined in the following paragraphs.

6.1.1 Configuration Fault-Tolerance

Measures of fault-tolerance have been defined in Section 4. The simulator should be able to produce these for a wide variety of configurations. This requirement can be satisfied in a reasonable way by structuring the simulator such that the various fault-detection and recovery algorithms are implemented as subroutines. Thus a configuration can be described by specifying the applicable set of subroutines, plus the necessary parameters. This simulator structure provides versatility and modularity, and minimizes the impact of addition of new subroutines.

6.1.2 Determination of Global Parameters Used in Analytical Modeling

Global parameters are those required when using the analytic model for analysis of a configuration. An example will help in understanding what we mean. The reader may recall that in Section 5.4.1, the transient coverage in triplex, c_T has been defined as the conditional probability that a triplex system recovers, given that a transient has occurred.

If a configuration is analyzed by mathematical modeling, c_T is one of the input parameters of the model. However, it is difficult for the designer to evaluate c_T, since it may depend on:

- the location of the transient fault
- their occurrence rate τ
- the time between occurrence and detection of a fault
- the recovery algorithm used

By introducing these factors into the simulation, and gathering statistics describing the computer system reaction to transient faults, c_T can be estimated by computing the ratio of the number of successful recoveries from transient faults to the total number of transients.

Thus, for the configurations where the mathematical modeling is applicable, one simulation run gives an estimate of these parameters of the modeling. Then using the model, the reliability, R(t), of the configuration can be easily determined for any given time t.

6.1.3 Fault Environment

The fault environment provided in the simulator should be sufficiently versatile to provide all expected possibilities to test the recovery algorithm utilized in the configuration under simulation. Thus low or high failure rates, existence and duration of transient bursts, long transients, mathematical fault-distribution functions, etc. must be provided. Implementation of this fault environment should be accomplished so as to provide maximum flexibility of environment choice by the user.

6.2 GENERAL ORGANIZATION OF THE SIMULATION

6.2.1 General Approach

Certain aspects of the general approach to the design of the simulator are implicit in objectives 1 and 3, namely the need for versatility and flexibility. There is a third, as-yet-unstated requirement, and that is for an efficient implementation that results in a reasonable computer-cost per run.

The versatility and flexibility requirements can be satisfied by designing a modular simulator that is easily modified (flexibility), and that models many configuration and fault-environment possiblities (versatility). Since we are concerned with behavior of the computer system following occurrence of a fault, we can obtain an efficient implementation by designing a "fault-driven" simulator, rather than one that simulates the continuous operation of the system.

Having chosen the general structure of the simulator, the next choice is that of implementation language. There are three contending possibilities. These are the computer system simulation languages such as ASPOL, ECSS, CSS-II, etc.; the discrete-event languages such as SIMSCRIPT, GPSS, SIMULA, etc.; and finally the general purpose languages such as FORTRAN. The computer system simulation languages offer ease of inclusion of peripheral devices such as tapes and discs, and the gathering of statistics as to their use. However this is not an issue for the RCS study. Similarly, the discrete-event simulation languages offer easy simulation of user queues and related items but these are not a factor in the type of simulation considered here. Thus we are left with the general purpose languages. Since FORTRAN is available both to Ultrasystems and NASA Langley, and provides the possibility of good program efficiency, this is the language that was chosen.

6.2.2 Organization of the Simulator

The approach taken to the formulation of the simulator is somewhat similar to that described in KRUU 63. Utilizing an extension of this approach, the computer system is seen as a finite state automaton. A state is defined by:

1. The number of good computers.
2. The action performed by the system at a given time.

A simplified state-diagram of the computer system is presented in Figure 6.2-1. This diagram shows all the states and the transitions between states, but does not show all state entry and exit conditions as does Figure 6.4-1. Basically the computer system states can be divided into five categories. These are:

NORMAL OPERATION
- Multiplex ($N \geq 3$)
- Duplex ($N=2$)
- Simplex ($N=1$)

TRANSIENT-FAULT RECOVERY
- Rollahead
- Memory Copy

Rollback
System Restart

PERMANENT-FAULT RECOVERY
Introduction of a Spare

DIAGNOSIS
Diagnosis

FAILURE
System Failure

A more detailed state diagram and the related details are provided in Section 6.4.

The simulator program consists of a collection of FORTRAN IV computer programs (to be run in a CDC 6600 CYBERNET computer environment) organized and designed to satisfy the objectives of simulation (Section 6.1). The main routine in charge of directing the processing flow of the simulation is designated the Driver. A collection of subroutines are accessible to the Driver via FORTRAN CALL statements. Each of the computer system states (Section 6.4) are represented by a subroutine. Other supportive subroutines perform statistics gathering and probability generating functions. The gross organization of the simulation is presented in Figure 6.2-2.

The simulator program is structured to simulate the detection of faults within a computer system and the computer system's successful/unsuccessful recovery actions taken in response to the detected faults. Each simulated mission is assigned a mission time. A simulation run consists of the repetitive continued simulation of a designated number of missions (each with the same mission time).

The initialization box of Figure 6.2-2 encompasses mainly of reading the run parameters and generating the fault table (see Section 6.5).

The simulation box is detailed in Figure 6.2-3. As stated earlier the simulation is fault driven. Nothing happens in the simulator until a fault occurs. This is very important in terms of simulator efficiency. The computer time spent in one run will be roughly proportional to the number of faults and not to the simulated mission time.

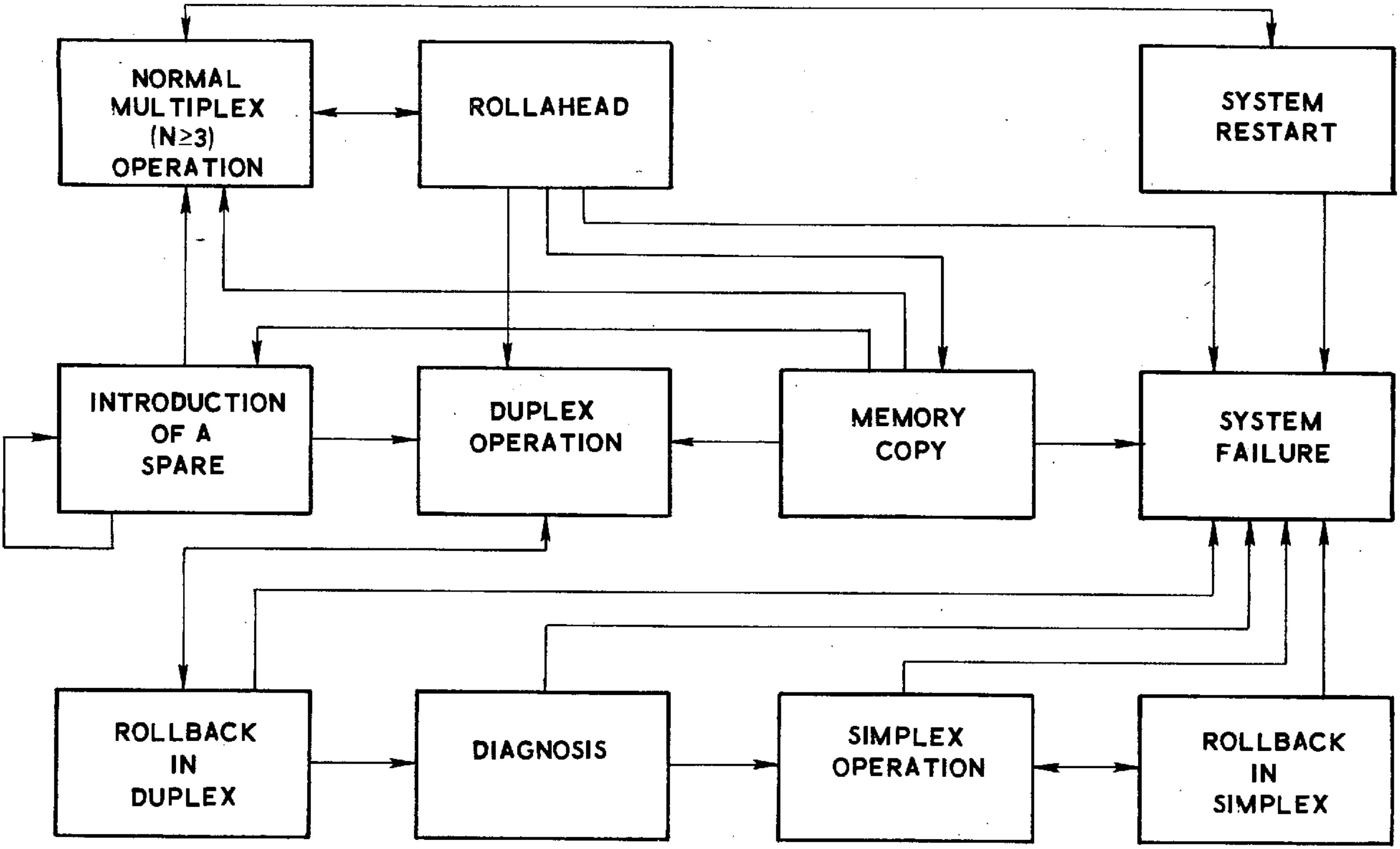


FIGURE 6.2-1 SIMULATOR STATE DIAGRAM

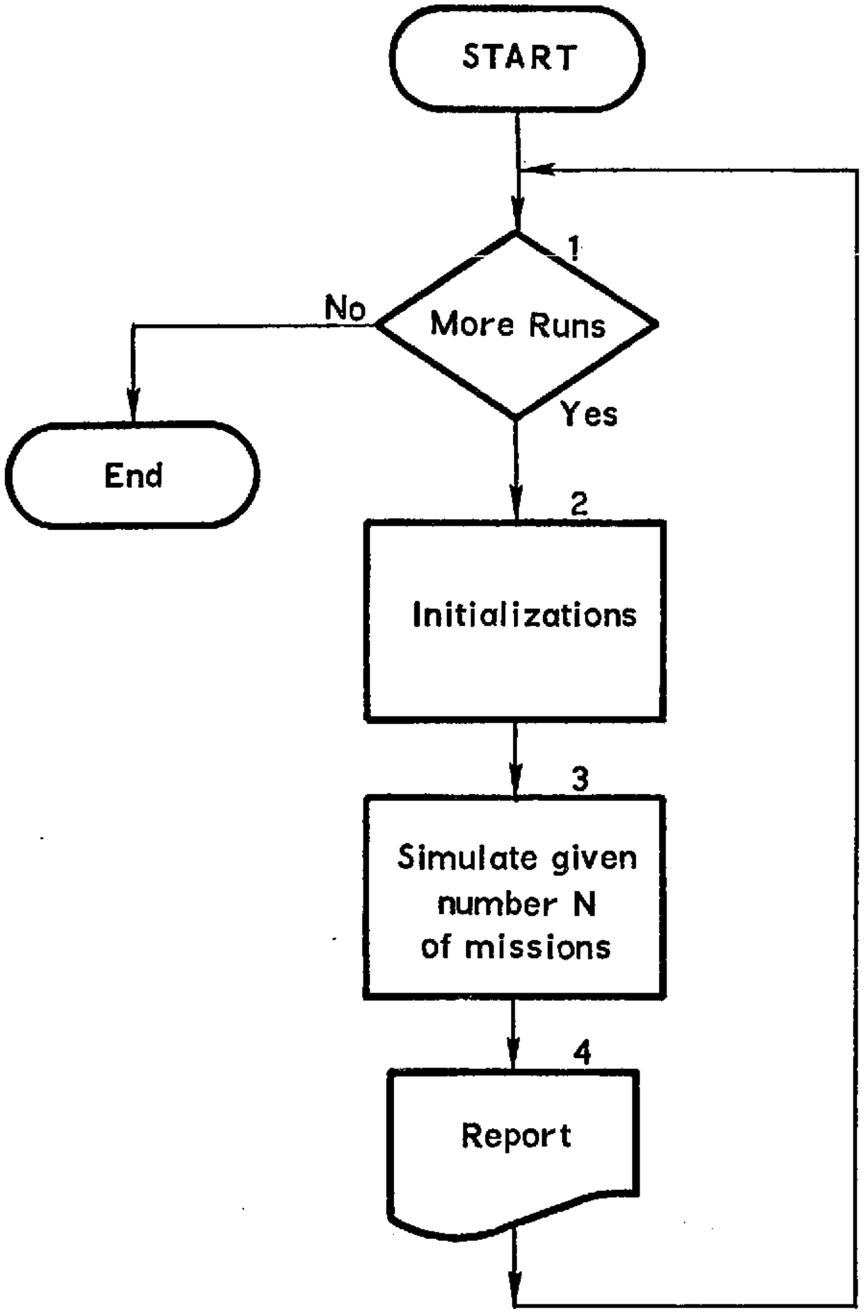


FIGURE 6.2-2 GROSS ORGANIZATION OF THE SIMULATION

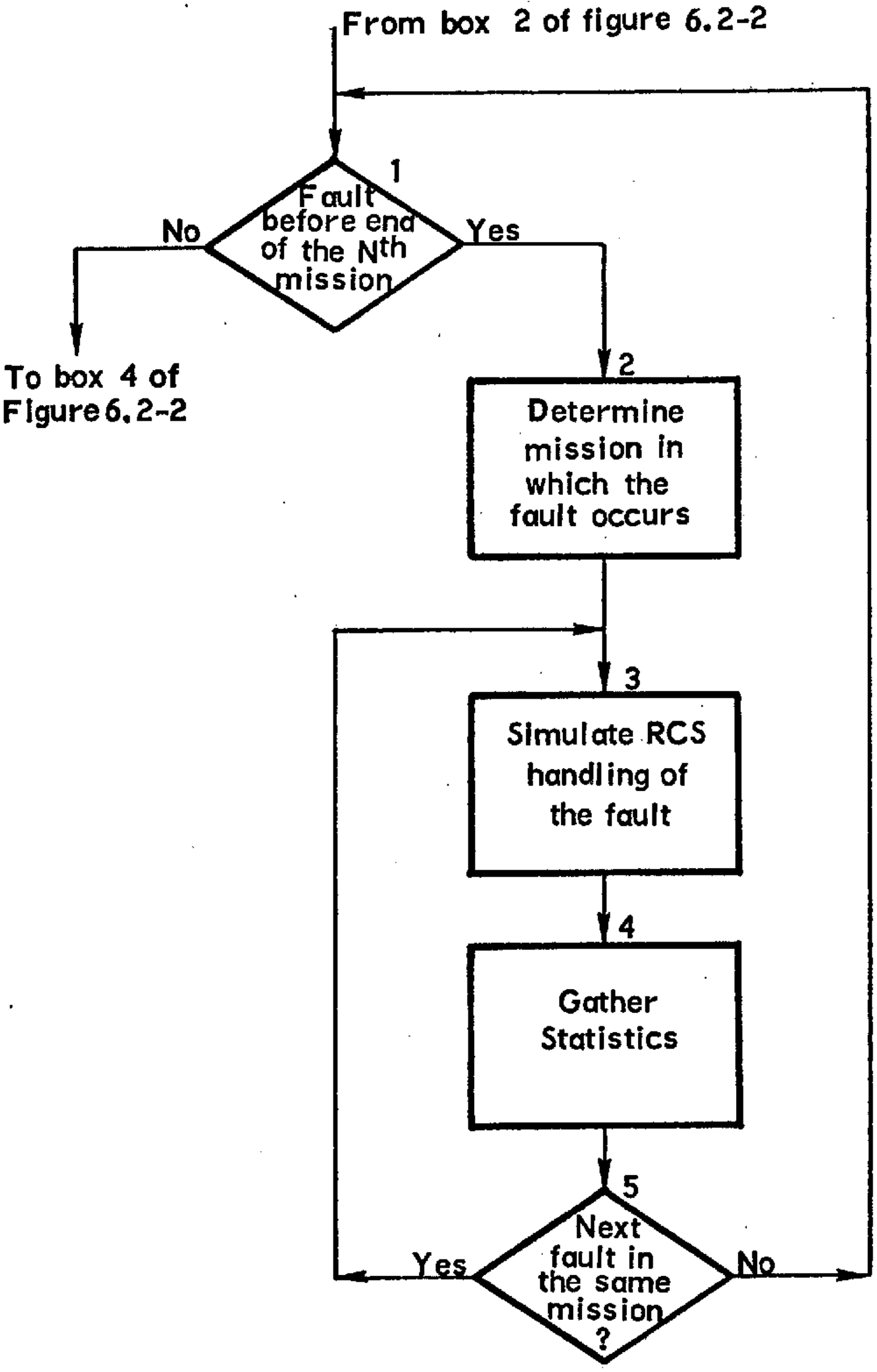

From box 2 of figure 6.2-2
1
Fault before end of the Nth mission
No
Yes
To box 4 of Figure 6.2-2
2
Determine mission in which the fault occurs
3
Simulate RCS handling of the fault
4
Gather Statistics
5
Next fault in the same mission ?
Yes
No

FIGURE 6.2-3 PRINCIPLES OF A FAULT DRIVEN SIMULATION (BOX 3 OF FIGURE 6.2-2)

Figure 6.2-4 shows how the simulator makes the transition between the various states. For example, when simulating a triplex configuration, simulation begins in State I. The fault table is scanned, a fault is found and its detection time is determined. The next state is determined (variable NEXT) and the transition occurs. The simulation continues in a similar fashion for each state until the mission ends.

6.3 INPUTS/OUTPUTS

6.3.1 Inputs

The parameters of a simulation run are listed below. An asterisk indicates that an explanation of this parameter is given in a following subsection.

Number of simulated missions

Mission dependent parameter

- Mission time

Machine-dependent parameters

- Permanent failure rates
- BITE Detection probability of a CPU fault*
- BITE Detection probability of a memory fault*
- Self-test program efficiency*
- Self-test program duration

Configuration-dependent parameters

- Number of computers
- Number of spares
- Dedicated/Non dedicated EEMs (External Electronics Modules)*
- Probability that an EEM fault hits the bus
- Number of non-dedicated EEMs
- Dedicated/Non-dedicated buses (see Section 6.5.7)
- Number of non-dedicated buses
- Number of external devices
- Coverage and relative failure rate of each device and of the buses
- Applicable recovery algorithms*
- Recovery algorithm characteristics

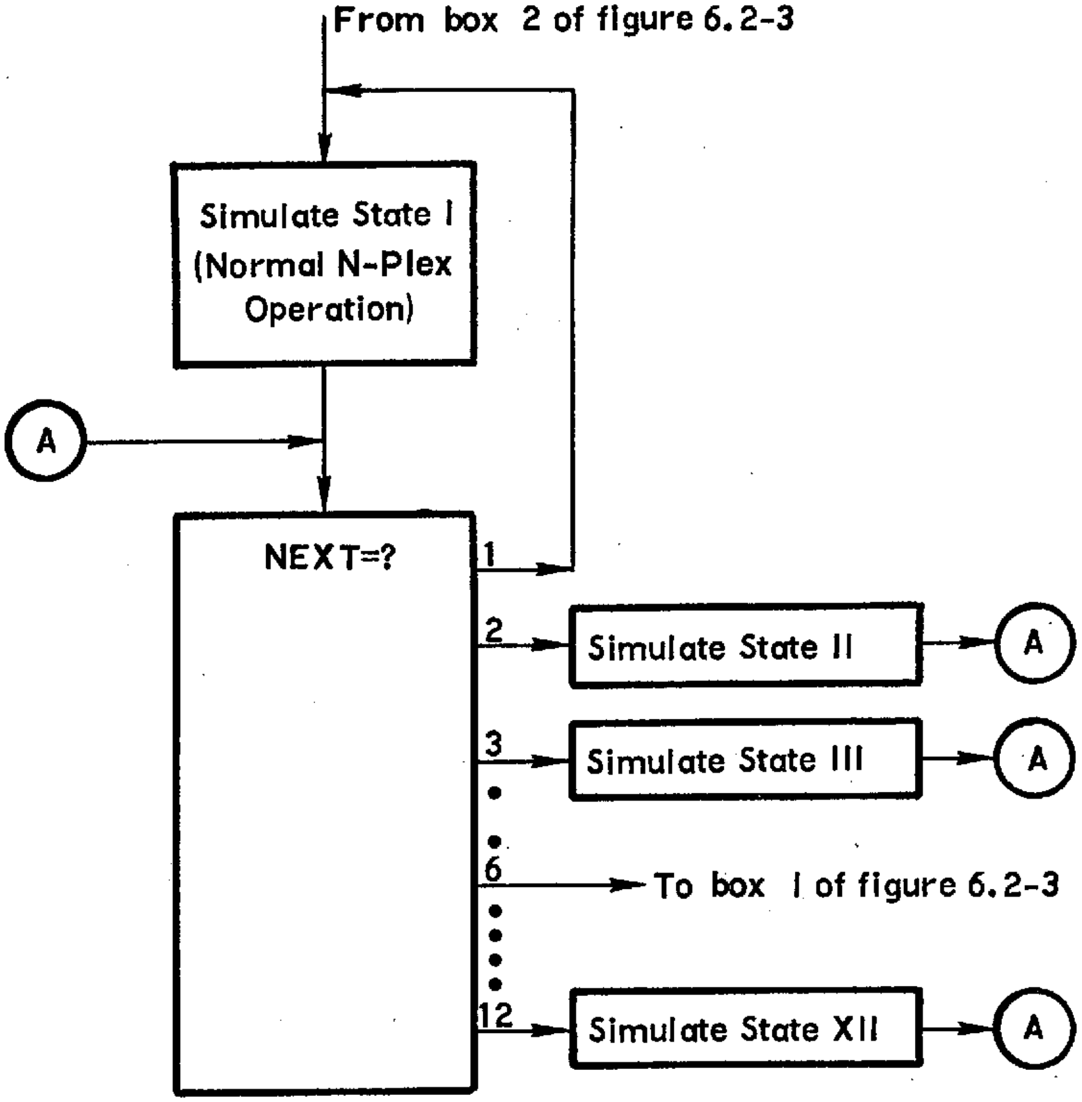


FIGURE 6.2-4 RCS HANDLING OF FAULTS (BOXES 3, 4, 5 OF FIGURE 6.2-3)

Duration
Unacceptable recurrence interval*
Maximum number of rollbacks
Program Integrity*
Memory-copy Efficacy

Scheduling parameters
Iteration period
Time between comparisons
Major and minor cycle durations
Asynchronous/synchronous mechanism

Environment dependent parameters
Transient failure rates
Transient failure duration

This list of inputs is presented in a slightly different and more detailed way in Figure 6.3-1. Filled in spaces contain either the implicit value of a parameter or its name. For example, among the transient fault recovery parameters, we see that the duration of a rollback is the time between comparisons and that its efficiency for a CPU transient fault is always 100%.

6.3.1.1 Detection Probabilities

These are the probabilities that a computer detects its own faults (except through diagnosis). This is not significant for N-M-R configurations ($N \geq 3$) since all faults are detected and located through voting or comparison. However, these probabilities become critical in duplex and simplex. In duplex, faults are detected through comparisons. However, if no other RETs are available, it is not possible to isolate the faulty computer. In simplex, these RETs are necessary, since they provide the only way to detect transient faults.

For simplex operation the detection probability of CPU faults is low. Faults in the CPU usually cause only a wrong output which will not be detected by BITE. However some will be detected. Those are the ones which cause a forbidden address to be computed or those which modify the computing sequence in such a manner that a go/no-go counter detects them. Intuitively, we can set this detection probability in the range from 5 to 20%.

1) PHYSICAL PARAMETERS

a) Design Decisions

Number of Computers	
Dedicated/Non Dedicated EEMs	
Number of EEMs	(1)
Dedicated/Non Dedicated Buses	
Number of Buses	(1)
Number of External Devices Per Bus	
Number of Spare Computers	

b) Failure Characteristics

		λ[3]	μ[3]	τ[3]	$1/\gamma$[3]
CPU	(I)		NA[4]		
Memory	(II)		NA[4]		
EEM[12]	(III)		NA[4]		
Bus and External Devices	(IV)		NA[4]		
Impact of EEM Failure on Bus					
$\frac{\text{Number of Faults in the Bus}}{\lambda_{IV}}$					
$\frac{\text{Number of Faults in Each External Device}}{\lambda_{IV}}$					

2) SOFTWARE CHARACTERISTICS

Synchronous/Asynchronous Scheduling Mechanism	
Iteration Period	
Minor Cycle Duration	
Major Cycle Duration (In terms of iterations)	
Time Between Comparisons	
Maximum Down Time	
Relative Size of Minor Cycle Program	
Interrupt Rate	

3) PARAMETERS AFFECTING FAULT DETECTION AND ISOLATION IN THE COMPUTERS

a) Detection Efficiency

Number of Computers	3 or more	2	1
Comparisons[7]	100%	100%	0%
CPU BITE			
Memory BITE			

b) Isolation Efficiency[9]

Number of Computers	3 or more	2
Comparisons[7]	100%[10]	0%[10]
CPU BITE		
Memory BITE		
Diagnosis[8] (STP)		

4) PARAMETERS AFFECTING FAULT DETECTION AND ISOLATION IN THE EXTERNAL HARDWARE .

a) EEM [11]

Number of EEMs	3 or more	2	1
Detection Efficiency	100%[10]	100%[10]	0%[10]
Isolation Efficiency	100%[10]	EEMCoverage[16]	N-A.[14]

b) Bus and External Devices

- Bus

Number of Buses	3 or more	2	1
Detection Efficiency	100%[10]	100%[10]	0%[10]
Isolation Efficiency	100%[10]	Bus Coverage	N-A.[14]

- For Each External Device

Number of Redundant Devices	3 or more	2	1
Detection Efficiency	100%[10]	100%[10]	0%[10]
Isolation Efficiency	100%[10]	Sensor Coverage	N.A.[14]

5) TRANSIENT FAULT RECOVERY PARAMETERS

	Efficiency for CPU Fault	Efficiency for Memory Fault	Duration	Maximum Number of Trials	Time Limit	In Use
Rollahead	100%[10]	Program Survivability	Delay[15] + Rollahead Duration	NA[14]	Recurrence Interval	Yes[10]
Rollback			Time Between Comparisons	Max. Number of Rollbacks	NA[14]	Yes[10]
Memory Copy	NA[14]	Memory Copy Efficacy	Memory Copy Duration	NA[14]	Recurrence Interval	
System Restart Duration	100%[10]	100%[10]	System Restart Duration	NA[14]	NA[14]	Yes[10]

6) PERMANENT FAULT RECOVERY PARAMETERS

Always done by switching off or ignoring the faulty computer, once the permanent has been recognized. If a spare is available, it is switched in.

Number of Computers	3 or more	2	1
Efficiency	100%[10]	STP Efficiency	0%[10]
Duration	0[10] or Spare Conditioning Time	Diagnosis + Switch-off Duration	NA[11]

FOOTNOTES

[1] Implicit parameter when each EEM/bus is dedicated to one computer.

[2] The bus by itself is considered as an external device.

[3] λ: permanent fault rate; μ: dormant fault rate. τ: transient fault rate; $1/\gamma$: mean transient duration.

[4] Non applicable when there is no spare.

[5] In this example, it is assumed that the four external devices have the same fault rate.

[6] Non applicable when synchronous scheduling.

[7] These parameters are implicit. However, the 100% efficiency is reached only after the whole memory has been exercised, i.e. after a full major cycle. Thus BITE feature may speed up detection.

[8] This parameter is valid only for permanent faults. Diagnosis is no help with transient faults.

[9] Isolation efficiency is 100%, once the fault has been detected in simplex.

[10] Implicit parameters.

[11] These parameters are not applicable with dedicated EEM: in this case, faults in the EEM are considered as equivalent to CPU faults.

[12] These parameters are 0 with a software TMR since there is no EEM.

[13] Rollahead is used for 3 or more computers. Rollback is used for 2 or 1 computers.

[14] Non applicable.

[15] Rollahead is preceded by an imposed delay to allow the transient to dissipate.

[16] Irrelevant when EEM are dedicated.

FIGURE 6.3-1 LIST OF INPUT PARAMETERS

The main technique to detect a memory fault is parity encoding. When it exists, the probability of detecting a memory fault is 80%. When it does not exist, this probability is quite smaller.

6.3.1.2 Self-Test Program Efficiency

Self-test programs (diagnosis) are run in a duplex system where a fault has been detected but not isolated. Note that if the fault is transient, the self-test does not diagnose it, since it likely will have dissipated when the test is run.

6.3.1.3 Dedicated/Non-Dedicated EEMs

If the configuration includes some additional hardware for the External Electronics Module, the consequence of faults in this hardware has to be assessed. We partitioned the configurations in two classes. In the first class (dedicated EEMs), we assume that a fault in the EEM is equivalent to a fault in the computer and some times on the corresponding bus. In the second one (non-dedicated EEMs), we assume that EEMs are independent from the computers. The system can work as long as one computer and one EEM are good. Note that the dedicated case includes software TMR.

6.3.1.4 Existing Recovery Algorithms

In the present simulator, the recovery procedure for a NMR system is the state vector transfer. Memory copy is optional.

6.3.1.5 Unacceptable Recurrence Intervals

Once a recovery procedure has failed for a certain fault, it is useless to attempt to recover through the same procedure. Some other one has to be chosen. If after completion of a recovery procedure, a fault recurs in the same computer after a time less than the unacceptable recurrence interval, the system decides that the recovery procedure was unsuccessful and attempts something else. Usually, the recurrence intervals will be chosen equal to the duration of one major cycle (Section 5). The rationale is that the memory is thoroughly exercised in one major cycle.

6.3.1.6 Program Integrity

This probability is listed with the other recovery algorithm characteristics because a rollback and a rollahead (state vector transfer) cannot succeed when there is a program memory damage. Program integrity is strongly linked to the type of memory: an NDRO memory is much better in this respect than a DRO memory. The fact that there is no need to restore the information makes it very unlikely that a transient fault damages instructions or constants. In addition, in most NDRO applications, the write voltage for the program memory is disabled except when altering the program under AGE control.

6.3.1.7 Memory Copy Efficacy

This is the probability that a memory copy corrects a transient fault. They only reason why it should not succeed is that the transient had hit the little (micro) program initiating the memory copy. This is very unlikely since this program should reside in a read only memory or microstore.

6.3.2 Output

Output consists of the following results.

1. Number of system failures.
2. Causes of system failures:
   - Too long unavailability. Failures caused by repeated recovery procedures lasting too long.
   - Non isolated faults. In duplex, even though a fault may be detected, it is possible that the diagnosis routines are unable to isolate the faulty computer.
   - Simplex failures in simplex mode, permanent faults, undetected or unrecovered transients cause system failure.
   - EEM failures.
   - I/O and bus failures.
3. Number of switches to - quadruplex
   - triplex
   - duplex
   - simplex
4. Transient coverages in multiplex, duplex, simplex.
5. Diagnostability in duplex.

6. Proportion of catastrophic faults. These are the faults which cause a system failure even though there are 3 or more computers in the system when they occur. (As long as the Poisson hypothesis holds for fault rate (Section 4.3.1), this number should be 0).

7. Number of missed iterations.

Causes of system failures are recorded because the dominant system failure mode points to the area in the design where improvement would be significant.

Similarly the number of switches to duplex and simplex are recorded since these are less effective and less reliable modes. Furthermore, these results are useful when studying a non (fully) adaptive system. For example, if a TMR system cannot degrade to a simplex mode, all switches to simplex should be considered as failures. Thus, non adaptive configurations are just considered as a special case of adpative configurations. They don't need any special parameters. Coverages and diagnostability are determined since these are parameters to be used in analytical modeling.

6.4 STATE DIAGRAM

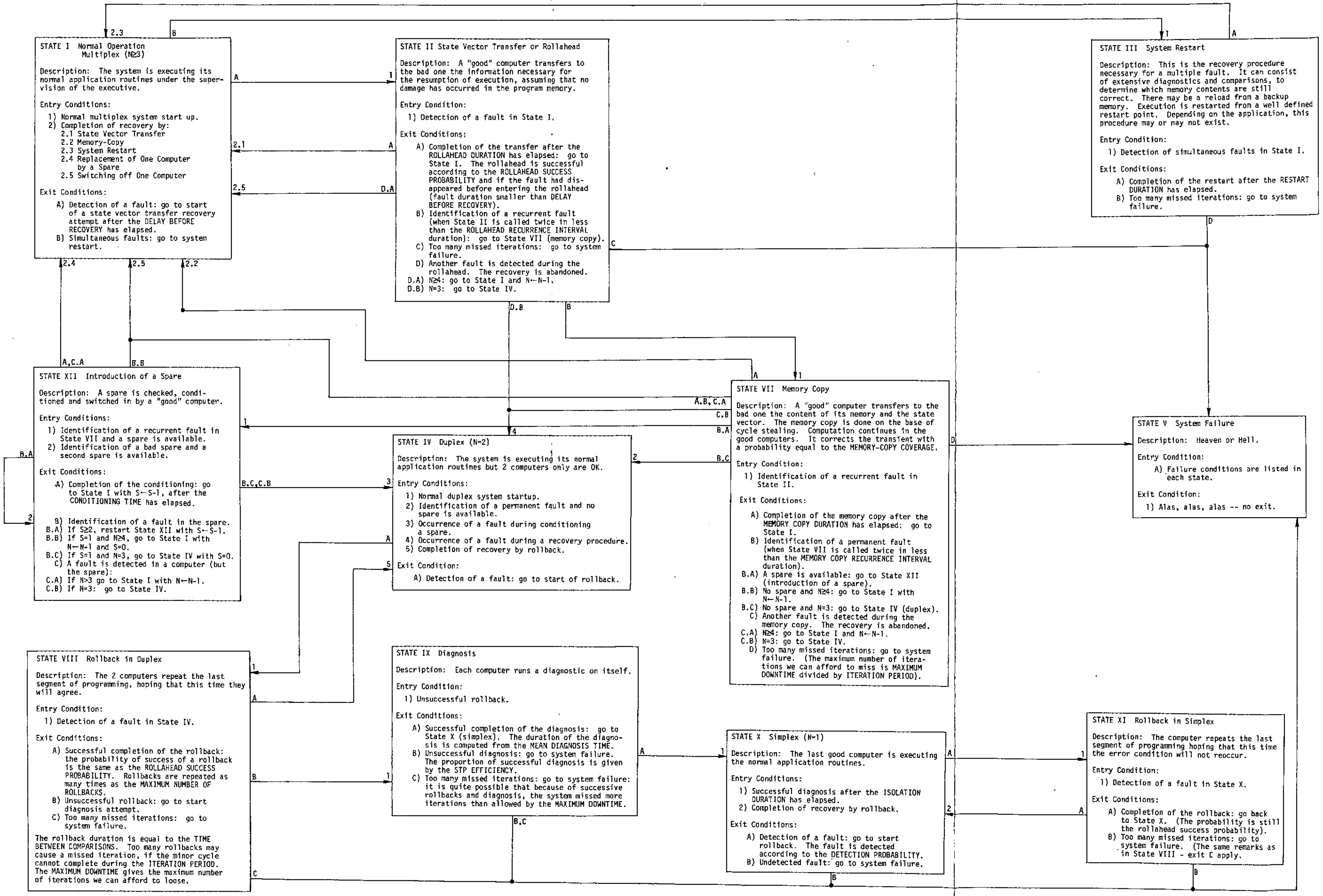
Figure 6.4-1 presents the detailed state diagram of an adaptive NMR configuration. The algorithms involved in States I, II, III, and VII do not vary for three or more active computers. Thus we avoid a proliferation of redundant states by maintaining a count in the simulation of the currently active computers.

6.4.1 Normal Operation (3 or more Units)

In the normal operation state with three or more computer units, the outputs of the computers are periodically compared. Disagreement of one or more computers constitutes fault detection and requires exit from this state.

As long as two computers are fault-free, the rollahead recovery procedure is used and, if it is not successful, the memory copy. If all computers disagree at the same time, a system restart is initiated.

THIS PAGE INTENTIONALLY LEFT BLANK



STATE I Normal Operation
Multiplex (N≥3)

Description: The system is executing its normal application routines under the supervision of the executive.

Entry Conditions:

1) Normal multiplex system start up.
2) Completion of recovery by:
2.1 State Vector Transfer
2.2 Memory-Copy
2.3 System Restart
2.4 Replacement of One Computer by a Spare
2.5 Switching off One Computer

Exit Conditions:

A) Detection of a fault: go to start of a state vector transfer recovery attempt after the DELAY BEFORE RECOVERY has elapsed.
B) Simultaneous faults: go to system restart.

STATE II State Vector Transfer or Rollahead

Description: A "good" computer transfers to the bad one the information necessary for the resumption of execution, assuming that no damage has occurred in the program memory.

Entry Condition:

1) Detection of a fault in State I.

Exit Conditions:

A) Completion of the transfer after the ROLLAHEAD DURATION has elapsed: go to State I. The rollahead is successful according to the ROLLAHEAD SUCCESS PROBABILITY and if the fault had disappeared before entering the rollahead (fault duration smaller than DELAY BEFORE RECOVERY).
B) Identification of a recurrent fault (when State II is called twice in less than the ROLLAHEAD RECURRENCE INTERVAL duration): go to State VII (memory copy).
C) Too many missed iterations: go to system failure.
D) Another fault is detected during the rollahead. The recovery is abandoned.
D.A) N≥4: go to State I and N←N-1.
D.B) N=3: go to State IV.

STATE III System Restart

Description: This is the recovery procedure necessary for a multiple fault. It can consist of extensive diagnostics and comparisons, to determine which memory contents are still correct. There may be a reload from a backup memory. Execution is restarted from a well defined restart point. Depending on the application, this procedure may or may not exist.

Entry Condition:

1) Detection of simultaneous faults in State I.

Exit Conditions:

A) Completion of the restart after the RESTART DURATION has elapsed.
B) Too many missed iterations: go to system failure.

STATE XII Introduction of a Spare

Description: A spare is checked, conditioned and switched in by a "good" computer.

Entry Conditions:

1) Identification of a recurrent fault in State VII and a spare is available.
2) Identification of a bad spare and a second spare is available.

Exit Conditions:

A) Completion of the conditioning: go to State I with S←S-1, after the CONDITIONING TIME has elapsed.

B) Identification of a fault in the spare.
B.A) If S≥2, restart State XII with S←S-1.
B.B) If S=1 and N≥4, go to State I with N←N-1 and S=0.
B.C) If S=1 and N=3, go to State IV with S=0.
C) A fault is detected in a computer (but the spare):
C.A) If N>3 go to State I with N←N-1.
C.B) If N=3: go to State IV.

STATE IV Duplex (N=2)

Description: The system is executing its normal application routines but 2 computers only are OK.

Entry Conditions:

1) Normal duplex system startup.
2) Identification of a permanent fault and no spare is available.
3) Occurrence of a fault during conditioning a spare.
4) Occurrence of a fault during a recovery procedure.
5) Completion of recovery by rollback.

Exit Condition:

A) Detection of a fault: go to start of rollback.

STATE VII Memory Copy

Description: A "good" computer transfers to the bad one the content of its memory and the state vector. The memory copy is done on the base of cycle stealing. Computation continues in the good computers. It corrects the transient with a probability equal to the MEMORY-COPY COVERAGE.

Entry Condition:

1) Identification of a recurrent fault in State II.

Exit Conditions:

A) Completion of the memory copy after the MEMORY COPY DURATION has elapsed: go to State I.
B) Identification of a permanent fault (when State VII is called twice in less than the MEMORY COPY RECURRENCE INTERVAL duration).
B.A) A spare is available: go to State XII (introduction of a spare).
B.B) No spare and N≥4: go to State I with N←N-1.
B.C) No spare and N=3: go to State IV (duplex).
C) Another fault is detected during the memory copy. The recovery is abandoned.
C.A) N≥4: go to State I and N←N-1.
C.B) N=3: go to State IV.
D) Too many missed iterations: go to system failure. (The maximum number of iterations we can afford to miss is MAXIMUM DOWNTIME divided by ITERATION PERIOD).

STATE V System Failure

Description: Heaven or Hell.

Entry Condition:

A) Failure conditions are listed in each state.

Exit Condition:

1) Alas, alas, alas -- no exit.

STATE VIII Rollback in Duplex

Description: The 2 computers repeat the last segment of programming, hoping that this time they will agree.

Entry Condition:

1) Detection of a fault in State IV.

Exit Conditions:

A) Successful completion of the rollback: the probability of success of a rollback is the same as the ROLLAHEAD SUCCESS PROBABILITY. Rollbacks are repeated as many times as the MAXIMUM NUMBER OF ROLLBACKS.
B) Unsuccessful rollback: go to start diagnosis attempt.
C) Too many missed iterations: go to system failure.

The rollback duration is equal to the TIME BETWEEN COMPARISONS. Too many rollbacks may cause a missed iteration, if the minor cycle cannot complete during the ITERATION PERIOD. The MAXIMUM DOWNTIME gives the maximum number of iterations we can afford to loose.

STATE IX Diagnosis

Description: Each computer runs a diagnostic on itself.

Entry Condition:

1) Unsuccessful rollback.

Exit Conditions:

A) Successful completion of the diagnosis: go to State X (simplex). The duration of the diagnosis is computed from the MEAN DIAGNOSIS TIME.
B) Unsuccessful diagnosis: go to system failure. The proportion of successful diagnosis is given by the STP EFFICIENCY.
C) Too many missed iterations: go to system failure: it is quite possible that because of successive rollbacks and diagnosis, the system missed more iterations than allowed by the MAXIMUM DOWNTIME.

STATE X Simplex (N=1)

Description: The last good computer is executing the normal application routines.

Entry Conditions:

1) Successful diagnosis after the ISOLATION DURATION has elapsed.
2) Completion of recovery by rollback.

Exit Conditions:

A) Detection of a fault: go to start rollback. The fault is detected according to the DETECTION PROBABILITY.
B) Undetected fault: go to system failure.

STATE XI Rollback in Simplex

Description: The computer repeats the last segment of programming hoping that this time the error condition will not reoccur.

Entry Condition:

1) Detection of a fault in State X.

Exit Conditions:

A) Completion of the rollback: go back to State X. (The probability is still the rollahead success probability).
B) Too many missed iterations: go to system failure. (The same remarks as in State VIII - exit C apply.

FIGURE 6.4-1 SIMULATOR DETAILED STATE DIAGRAM

6.4.2 Rollahead (or State Vector Transfer)

The rollahead state is entered to simulate the computer system's attempt to recover from a detected single fault. The state vector (consisting of program variables and all register contents) of one good computer is used to replace the non-agreeing computer's state vector. However all transient failures are not corrected by this procedure since a bad instruction cannot be restored. The approach taken in the simulation is to provide for the specification of a rollahead success probability. This probability can be formally defined as:

$$P_{suc} = \text{Pr} \left[\begin{array}{l}\text{fault is corrected given that a fault has occurred,} \\ \text{has been detected, and its physical cause has dis-} \\ \text{appeared when correction begins}\end{array}\right]$$

An analysis, which gives consideration to the type of memory (e.g. 2 1/2D, 3D, DRO, NDRO, etc.) and the consequences of memory faults, will yield an estimate of the rollahead success probability (or program integrity).

6.4.3 Memory Copy

This recovery procedure is entered after a specified number of rollaheads have been completed unsuccessfully. The memory contents of one good memory are transferred into the faulty memory. In order to avoid interruption of computation, the transfer is effected on the basis of cycle stealing. It ends with the updating of the state vector of the faulty computer.

Since, during a memory copy, normal application routines continue, it is possible that a new fault shows up. The following (conservative) assumption has been made in order to simplify the simulation. Upon detection of a second fault during a memory copy, the memory copy procedure is abandoned and the computer for which this memory copy was intended is discarded.

It is assumed that memory copy provides recovery from transient faults which have disappeared when the memory copy began with a probability equal to the memory copy efficacy.

6.4.4 System Restart

The system restart state is entered when all computers disagree upon comparison. The recovery procedure from this state may consist of a memory verification. Relevant memory locations are read, voted upon and restored. Extensive diagnosis may also be run. Finally, if a backup memory is available, reloading may be possible. Then the application program is reinitiated from the restart point.

After a successful system restart, the system returns to the normal operation state. However, since all computers stop their normal computation during a system restart, this recovery procedure is time critical.

Note that in a benign fault environment, the probability of having a system restart is quite small ($\simeq 1$ for 1 million faults). However, system restart is necessary if the fault environment is so harsh that bursts of faults can hit several computers at a time or if the probability of a short power failure is not negligible.

6.4.5 Introduction of a Spare

If a spare is available, it should be activated once a permanent fault has been recognized. As part of the activation process, the spare is checked and conditioned by one of the good computers. In the state diagram of Figure 6.4-1, spares are not available for the duplex and simplex simulation. This is thought to be compatible with the expected applications.

6.4.6 Normal Operation (2 Units)

The normal operation (2 units) state is entered upon the determination that a permanent fault exists in one of the three computers of the computer system. This state is quite similar to the normal operation (N units) state, except that the only available recovery procedure is program rollback.

6.4.7 Rollback

The rollback state is entered upon the detection of a fault when the computer system is in the normal operation (2 units) state. Rollback is the term used to describe repetition of the program segment executed just prior to the detected output disagreement. The state vector at the beginning of each program segment is maintained in order that the rollback procedure may be accomplished.

After the program segment has been repeated, the outputs of the two computers are compared; if the correction is successful, the computer system switches back to the normal operation (2 units) state. If the outputs differ, the system rolls back again; this unsuccessful recovery process continues a predetermined number of times before changing the computer system state to diagnosis.

Since both of the active computers remaining in the computer system must stop their normal computations during a rollback, this computer recovery procedure may be time-critical. However, if comparisons are frequent enough, a rollback should not last more than a few milliseconds.

6.4.8 Diagnosis

In triplex, voting provides a very easy and efficient way of isolating the faulty unit. Unfortunately, a disagreement upon comparison in duplex does not indicate which of the computers produced the wrong value. That is why the main recovery procedure in duplex is the rollback since there is no transfer of information from the good to the bad computer for such a procedure. But, if the rollback does not succeed, the bad computer must be isolated. For that purpose, self-tests are run. If they are successful, the faulty computer is isolated and the system switches to simplex. If unsuccessful the system is unable to decide which computer is faulty and the system fails. Diagnosis programs are obviously time critical. Note that it would be possible to include a memory copy which would take place once a diagnosis had been successful: the memory of the good computer would be copied into the bad one. However, this improvement is not so good as it would seem since many transients cannot be detected through diagnosis.

6.4.9 Normal Operation (Simplex)

In simplex operation, comparison is no longer available for detection of faults. We must rely mostly on the RETs to detect faults. CPU transients are difficult to detect. Some may be caught through go/no-go counters and storage protection. Memory faults are easier to detect. Parity check is especially useful. When a fault is detected a rollback is initiated. If the fault is not detected, a failure occurs.

6.4.10 Rollback in Simplex

This is the same procedure used in duplex. Since it is the only recovery algorithm available in simplex, it is repeated as long as it is not successful. If recovery from the fault cannot be effected, a system failure will occur when the system has been down too long.

6.4.11 System Failure

The system failure state is entered when the system is unable to run properly any longer, or when computation requirements have not been met for too long a period of time. Upon recognition of the condition of a system failure, the DRIVER program discontinues the simulation of a mission.

Causes of failures are:

1. Excessive time in rollahead, memory copy, or rollback: It should not happen since the system must be designed so that a recovery procedure does not endanger it. However it might happen that the continuous repetition of such procedures be fatal for the successful completion of the mission.

2. A too long system restart: A system restart is a very rarely called procedure. But it is long (a few seconds), and may not always be tolerable.

3. Diagnosis incomplete when available recovery time expires: Normally, diagnosis follows rollback. It is possible that these two recovery procedures sometime take too long.

4. Undetected faults in simplex.

5. A too long rollback in simplex: This happens when a permanent occurs or when a non-recoverable transient occurs.

6. EEM failures: In case of non dedicated EEMs the system fails when all EEMs fail or when all but one fail and the computers are unable to decide which is the good EEM.

7. Bus failures: The system fails when all buses fail or when all but one fail and the computers are unable to decide which is the good bus.

8. Actuator/sensor failures.

6.5 SIMULATOR IMPLEMENTATION

Because of the fault-driven nature of the simulator, the first activity in the simulation is the generation of a table of faults occuring in a given number of missions. The fault generator and the computer-system-state subroutines are described in the following subsections.

6.5.1 Fault Generation

6.5.1.1 Introduction

A major portion of the simulator is dedicated to the generation of faults according to mathematical algorithms which describe the occurrence of faults in the various components of the computer system. Two approaches to handling this problem were considered:

1. Generation of one fault at a time.
2. Generation of a fault table describing the faults which occur in the computer system between 0 and a time T.

The first approach is suitable if we consider only single faults and if we simply describe fault occurrences within the computer system, e.g., the fault-arrival rate in the system is λ and the probability that a fault is in the i^{th} part of the computer system is P_i. This procedure is described in LYON 62.

Since we must deal with transient failures also, we want to know how the computer system behaves in case of multiple faults. Furthermore, if the faults do not occur according to a Poisson law in all modules (burst of transient failures for example), the method described in LYON 62 is not readily applicable.

A more efficient and more general approach is to generate a fault table prior to simulation. This also makes the simulation program more functionally modular since, once the simulation has begun, we have only to scan the fault-table to determine when and where the next fault occurs.

6.5.1.2 Parameters

The parameters necessary to generate the fault table for a simulator run are a part of the parameters of simulation which are input by the simulator user for each simulator run.

6.5.1.2.1 Description of the Computer System

The computer system to be simulated is composed of n identical computers, each composed of m modules.

6.5.1.2.2 Description of the Fault Distributions

For each of the m modules, the distribution functions to be used in the generation of both permanent and transient faults must be indicated by the simulator user. Specific subroutines for the chosen distribution functions are then called and the parameters of the distribution are passed to these subroutines.

For permanent faults, only the Poisson distributions have been implemented. This is generally considered in the literature to be most realistic.

For transient faults, Poisson and burst distributions have been considered. Poisson distributions are considered because of their tractability and acceptance for the permanent fault case. Burst distributions are thought to be important because many transients likely are caused by components working near the limits of their tolerance specifications. As long as the conditions do not improve, faults will occur often in these components. A burst of transients is defined by its duration and the rate of transient occurrence during the burst. Bursts occur according to the burst rate.

6.5.1.2.3 Description of the Fault Duration

For each of the m modules, the distribution function of the transient failure durations to be used by the simulator programs must be indicated by the simulator user. Specific subroutines for the chosen distribution functions are called by the Driver and the subroutines receive the parameters of the distributions.

At the present time, the uniform and the exponential distributions have been implemented.

1. Uniform Distribution -- The transient failure duration is uniformly distributed between a minimum and a maximum duration.

2. Exponential Distribution -- The transient failure duration is exponentially distributed. The mean duration is $1/\gamma$.

6.5.1.3 Description of the Fault Table

The fault table consists of 300 records ordered according to the occurrence time of each fault. This table can contain up to 150 permanent faults and 150 transient faults. It has the following record format:

Occurrence Time	Duration	Module	Computer

Permanent failures are identified by a duration longer than the mission time.

6.5.1.4 General Organization of the Fault Generator

The first step consists of generating a table of permanent failures and a table of transient failures for each module in the computer system. Then these tables are merged into one sequentially-ordered (master) fault table. The general organization of the fault generator is presented in Figure 6.5-1.

6.5.1.5 Determination of the Occurrence Time of the Faults According to a Poisson Distribution Function

Faults occurring by a Poisson distribution process have a probability that one fault occurs during a small interval of time, dt, as follows:

$P_1 = \lambda dt$. (See PARZ 60).

The probability of no faults, P_0, occurring during the time interval dt is, $P_0 = 1-\lambda dt$, and the probability of more than one fault occurring is 0.

A Poisson distribution process has two very important properties:

1. It is memoryless: This means that the probability of a fault occurring between times t and t+dt is independent of fault occurrences before time t.

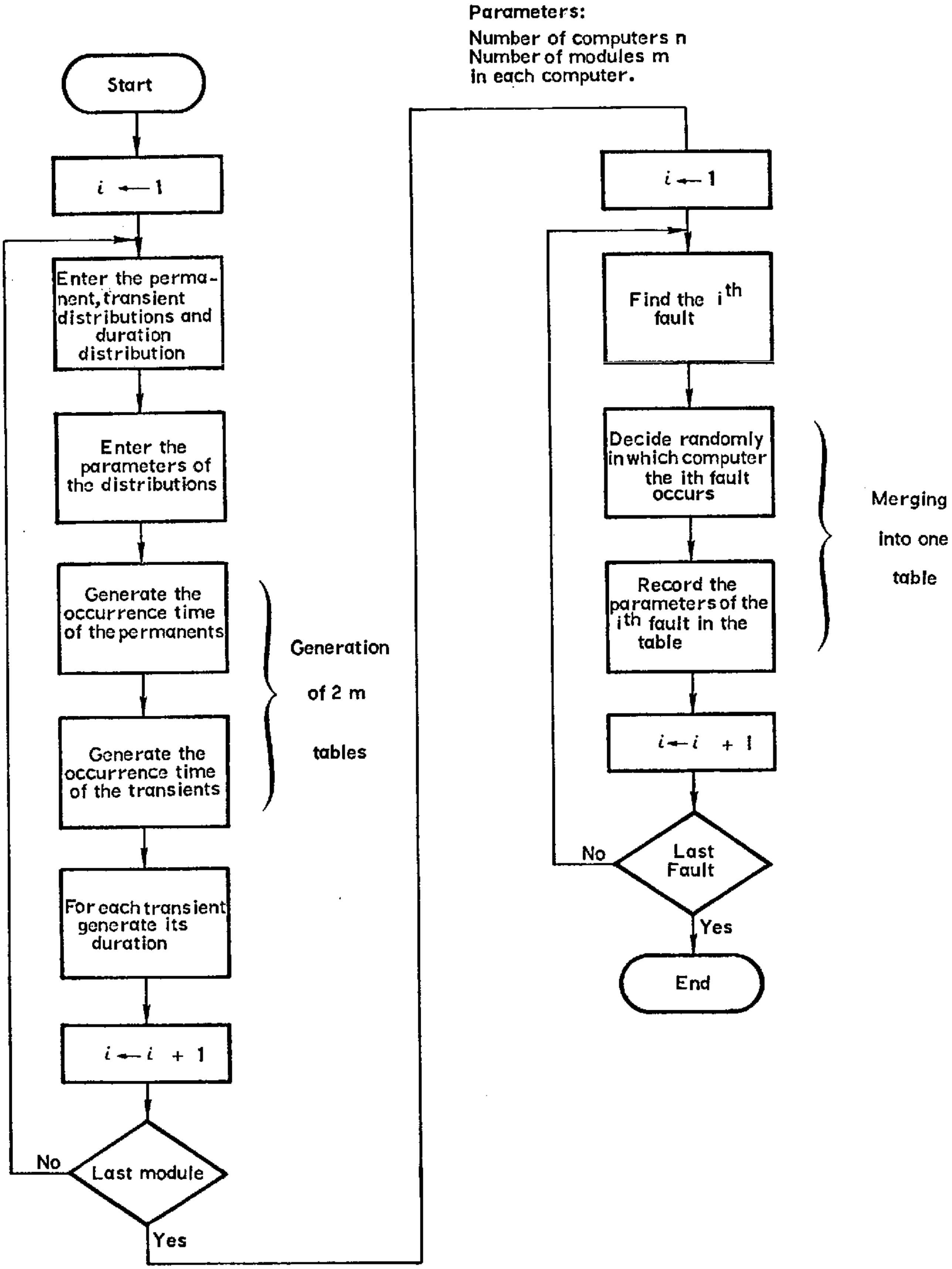

Parameters:
Number of computers n
Number of modules m
in each computer.
Start
$i \leftarrow 1$
Enter the permanent, transient distributions and duration distribution
Enter the parameters of the distributions
Generate the occurrence time of the permanents
Generate the occurrence time of the transients
Generation of 2 m tables
For each transient generate its duration
$i \leftarrow i + 1$
Last module
No
Yes
$i \leftarrow 1$
Find the i^{th} fault
Decide randomly in which computer the ith fault occurs
Record the parameters of the i^{th} fault in the table
Merging into one table
$i \leftarrow i + 1$
Last Fault
No
Yes
End

FIGURE 6.5-1 GENERAL ORGANIZATION OF THE FAULT GENERATOR

2. The probability density function for the random variable, T_τ, i.e. the interarrival time between two consecutive faults, is

$$f_{T_\tau}(t) = \lambda e^{-\lambda t}$$

Thus the probability distribution function of T_r is:

$$P\left[T_r \leq t\right] = \int_0^t f_{T_r}(u)\, du$$

$$= 1 - e^{-\lambda t}$$

Thus the probability of having no fault at time t is:

$$R(t) = e^{-\lambda t}$$

A difficulty arises at this point since the random number generator (function) available in the CYBERNET system produces outputs which are uniformly distributed on the interval $0 \leq U < 1$. The outputs of this generator can be converted using the approach described below. (HILL 70, SHRE 66).

We are concerned with the random variable T_r, the interarrival time between faults, whose distribution function is given above as

$$P[T_r \leq t] = 1 - e^{-\lambda t}$$

For the purposes of the simulation we wish to obtain values of t. We now note two important facts. First, $0 \leq P < 1$. Second, by algebraic manipulation it is possible to solve for t, e.g:

$$t = -\frac{1}{\lambda} \ell n \; (1-P)$$

Thus, for any value of P in the valid range, a value of t can be calculated. By generating values of P using the random number generator, which produces uniformly distributed numbers between zero and one, t can then be calculated.

A more formal description of the process follows. Using the random number generator which gives a number U uniformly distributed on the interval $0 \le U < 1$, we have to compute T_r which is exponentially distributed. That means that we have to find a function $f(U)$ such that:

$$T_r = f(U)$$

and $P[U \le u] = u$ (uniform distribution) $\Longrightarrow P[T_r \le t] = 1-e^{-\lambda t}$
(if $0 \le u < 1$)

If $T_r = f(U)$, we can define the inverse function $g(T_r)$ such that $U = g(T_r)$.

Thus, we have:

$$\begin{aligned} P[T_r \le t] &= 1-e^{-\lambda t} \\ &= P[f(U) \le t] \\ &= P[U \le g(t)] \\ &= g(t) \end{aligned}$$

The last equation is true since U is uniformly distributed on the interval, $0 \le U < 1$. Thus we know that the unknown function $f(U)$ is the inverse of the function $g(t) = 1-e^{-\lambda t}$

Hence:

$$u = g(t) = 1-e^{-\lambda t}$$

$$t = -\frac{1}{\lambda} \ell n\ (1-u) = f(u)$$

Since we have just found the function f, we can write

$$T_r = -\frac{1}{\lambda} \ell n\ (1-U)$$

But we can have a simpler expression: U is uniformly distributed on the interval, $0 \le U < 1$. Hence $1-U$ is also uniformly distributed on the same interval. This implies that the distribution of T_r does not change if we replace $1-U$ by U.

Finally, we have shown that if U is uniformly distributed on $0 \le U < 1$, then $T_r = -\frac{1}{\lambda} \ell n\ U$ is exponentially distributed, the parameter of the distribution being λ.

Using the random number generator provided by the CYBERNET system, we determine the different interarrival times and thus the occurrence times. The flowchart of the generation of the occurrence times of the faults in one module is presented in Figure 6.5-2.

6.5.1.6 Determination of the Duration

As stated earlier, both exponential and uniform distributions of transient fault duration are available in the simualtor. If the transient duration is exponentially distributed (parameter γ), we determine a duration D_T for each transient:

$D_T = -\frac{1}{\gamma} \ell n\ U$ using the same general procedure described for the occurrence time. If the duration is uniformly distributed on $0<D_T<D_{max}$, the duration D_T is $D_T = D_{max} \times U$.

6.5.1.7 Determination of the Occurrence Time of the Faults According to a Burst Distribution Function

The occurrence time and duration of the bursts is determined as in Sections 4.3.1.5 and 4.3.1.6. Then, for each burst, the occurrence time and duration of the transients are determined.

6.5.2 Normal Operation (3 or More Units)

The detailed diagram (Figure 6.4-1) shows that either single fault detection, or multiple fault detection may happen in this state. A multiple fault is detected when all computers disagree at the same time (or indicate an error condition). Note that in quintuplex for example, a fault which would hit only 2 computers at the same time is not considered as a multiple fault. This is due to the fact that 3 computers still agree and the good state of the system is known. Multiple faults necessitate special care since the good computers are not known. A system restart is to be entered. Single fault detection initiates a rollahead. Figure 6.5-3 is a flowchart of State I.

6.5.3 Rollahead

A general flowchart of the rollahead state is presented in Figure 6.5-4. A rollahead does not correct all faults. Memory damage cannot be recovered from through this procedure.

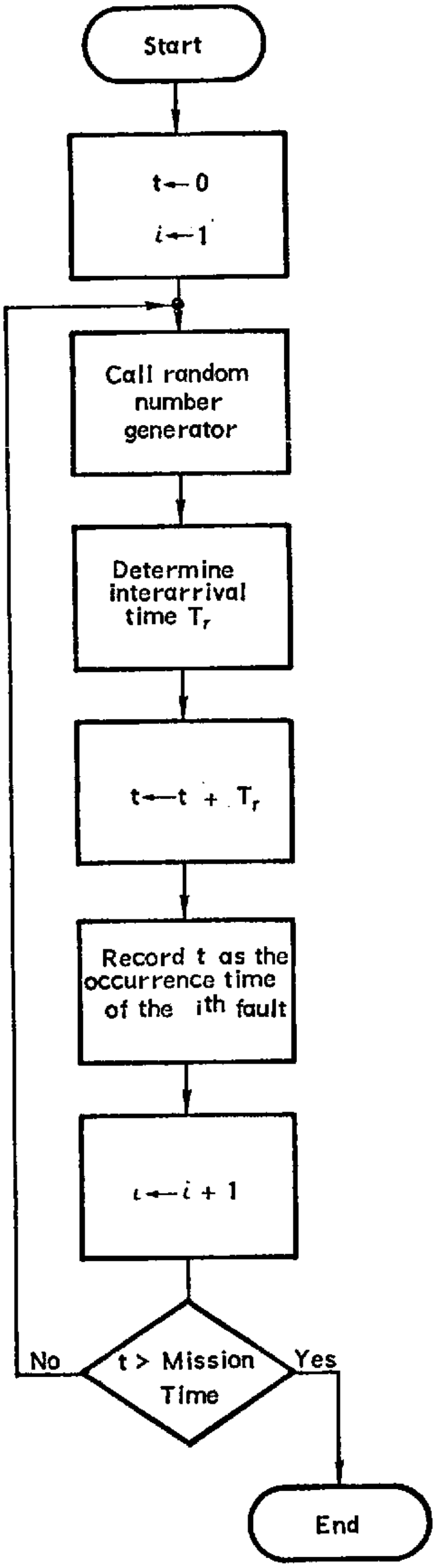

Start
t ← 0
i ← 1
Call random number generator
Determine interarrival time T_r
t ← t + T_r
Record t as the occurrence time of the i^{th} fault
i ← i + 1
t > Mission Time
No
Yes
End

FIGURE 6.5-2 GENERATION OF THE OCCURRENCE OF THE FAULTS IN ONE MODULE (POISSON DISTRIBUTION)

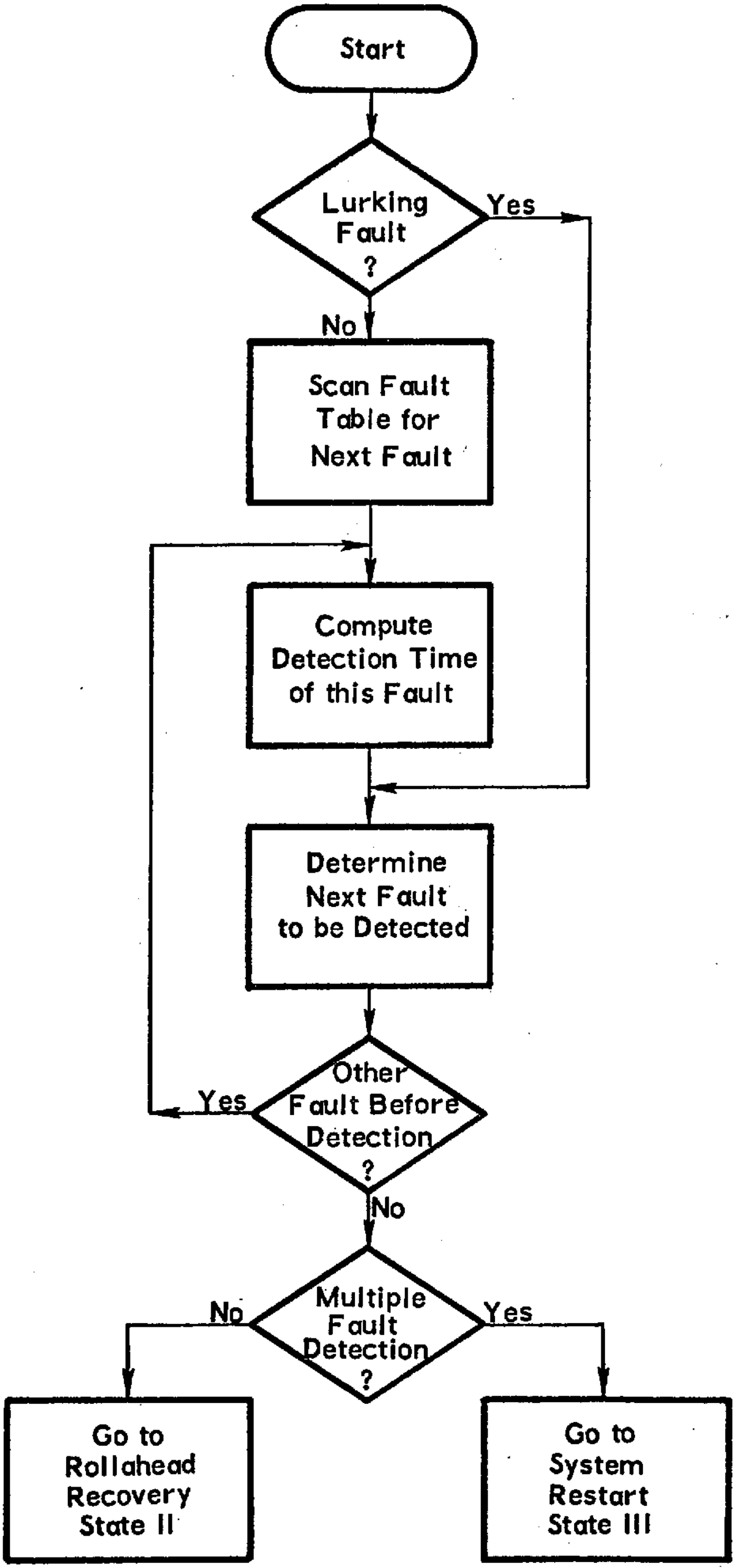

Start
Lurking Fault ?
Yes
No
Scan Fault Table for Next Fault
Compute Detection Time of this Fault
Determine Next Fault to be Detected
Other Fault Before Detection ?
Yes
No
Multiple Fault Detection ?
No
Yes
Go to Rollahead Recovery State II
Go to System Restart State III

FIGURE 6.5-3 NORMAL OPERATION STATE I

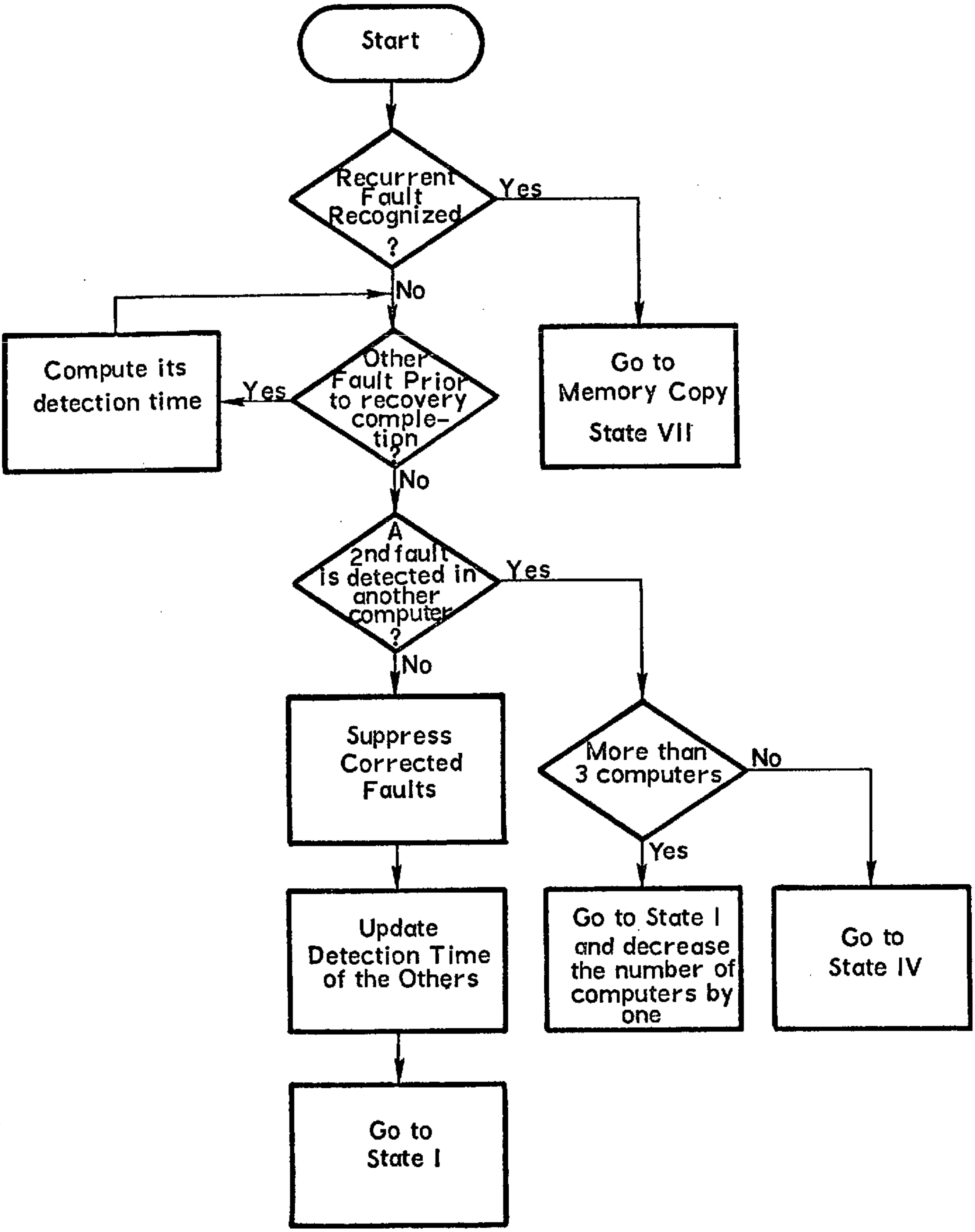

Start
Recurrent Fault Recognized ?
Yes
Go to Memory Copy State VII
No
Compute its detection time
Yes
Other Fault Prior to recovery comple-tion ?
No
A 2nd fault is detected in another computer ?
Yes
No
Suppress Corrected Faults
More than 3 computers
No
Yes
Update Detection Time of the Others
Go to State I and decrease the number of computers by one
Go to State IV
Go to State I

FIGURE 6.5-4 ROLLAHEAD STATE II FLOWCHART

The probability of success of rollahead is estimated by analyzing the memory organization. The main conclusions are:

1. With a DRO memory with protection bits, CPU and I/O faults do not cause memory damage. For these faults, the rollahead is always successful.

2. For memory transients, the analysis is more difficult. A first consideration is given to DRO and NDRO memories. Then, consequences of faults in the different circuits should be assessed. It must be determined which transients are likely to cause an instruction or constant to be destroyed. Thus, we can get an estimate of the success probability of rollahead. According to this estimate, each fault is marked as recoverable (or not) by rollahead. (It must not be forgotten that in any case permanents cannot be recovered from by such a procedure). The probability of success is much higher for an NDRO than for a DRO memory.

6.5.4 Other States

The subroutines describing the remaining states all have the same basic structure. For all of them, the fault table is scanned and then a decision is made as to which is the next state. Decisions are taken as described in the exit conditions shown on the detailed state diagram.

6.5.5 Introduction of the Scheduling Mechanisms

In Section 4.1 one of the criteria for correct execution of a set of programs is that the execution time of each program does not exceed a specified limit. Thus it is necessary to provide in the simulator a method for determination of the consequences of output delays and the number of missed iterations due to execution of recovery procedures.

Scheduling mechanisms are described in Section 3.3. Section 3.5 describes how recovery procedures fit in the different schemes. We list below the fundamental remarks - as far as simulation is concerned - of Section 3.5.

1. Dichotomy of scheduling mechanisms into 2 classes: synchronous type and asynchronous type mechanisms.

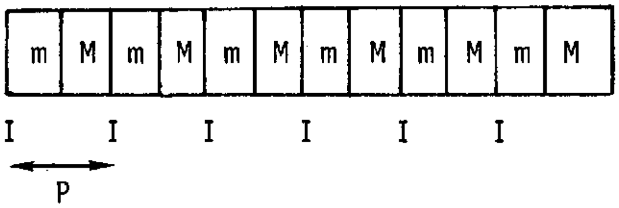
2. The rollback structure of minor cycle computations corresponds with the scheduling of the computations for any of the scheduling mechanisms.
3. In the case of a synchronous mechanism, each segment of a major cycle computation is similar to the minor cycle computations in that the rollback structure and the scheduling of the segment correspond.
4. With an asynchronous type, things are much more complicated. If comparisons take place only when an output occurs, the entire computations must be repeated if an error occurs in a major cycle. Furthermore, rollback may also be necessary for the minor cycle computation taking place during this major cycle.

The main point of the simulation is to evaluate the probability that a mission is successfully completed. One instance of failure is when, because of recovery procedures, all computations necessary to the success of the mission cannot be achieved. It can be assumed that for a specific type of mission there is a maximum number of consecutively missed minor cycles. For example, if the aircraft is unstable, this number may be no more than 1. For some other type of mission, it may be a hundred. Thus, our first goal will be to count the missed minor cycles, and record a failure whenever too many consecutive minor cycles are lost. Note that because of the second remark, the simulation will be very similar for both asynchronous and synchronous cases.

The main difference between these two cases is for major cycles. A rollback, in an asynchronous system may considerably delay the completion of a major cycle computation and may also cause more than one rollback.

We shall first look at the simpler case, i.e. synchronous scheduling.

6.5.5.1 Synchronous Scheduling



m: Minor Cycle Computations

M: Major Cycle Computations

I: Minor Cycle Initiation

P: Minor Cycle Period

After the minor cycle processing is completed, the remaining time before the next RTI is used for major cycle processing.

6.5.5.2 Detection of Faults

Faults are detected when comparison takes place. A fault in the CPU is detected on the comparison following its occurrence. Things are different for memory fault. If a fault hits a minor cycle program, detection will occur in the current minor-cycle period. But if it hits a major cycle program, detection occurs during the major cycle following its occurrence. Thus, it is quite possible to have an undetected fault for a while. In any case, having frequent comparison points will make detection faster.

Some faults are detected earlier than the comparison following their occurrence. These are the faults detected by RETs. For example, a bad memory word may cause a parity error. Let's examine the consequence of this feature. If the recovery procedure is the rollahead, the error interrupt is left pending since the state vector transfer can be initiated only at well-defined points. It is at these points that comparisons take place. When the recovery procedure is the rollback, the error interrupt can be immediately taken into account and the rollback can be initiated.

6.5.5.3 Iteration Losses

Our goal is to determine if a recovery procedure has lasted too long. A subroutine determines the number of consecutively missed iterations. The main difficulty is that it is not enough to determine if recovery from a fault took longer than a specified time. We must be aware that another recovery in the same cycle might have decreased the time available for the second recovery.

6.5.5.4 Asynchronous Scheduling

As explained earlier and in Section 3.5.1, a fault occurring in a major cycle may cause more than one rollback/rollahead. This happens when the major cycle routine where the fault occurs is interrupted by a minor cycle routine. This is simulated in the following way. The interrupt rate and the average length of a program segment are known. Thus it is possible to compute the probability of having an interrupt coming between fault occurrence and the end of the program segment, when the comparison and detection takes place. If an interrupt has come in between, we assume that two recovery procedures take place.

The recovery procedure is assigned the highest interrupt priority. If any other interrupt comes during the recovery, it is ignored, thus causing a missed iteration.

6.5.6 EEM Faults

The External Electronics Module is the additional hardware in charge of the voting and the recovery initiation. It is subject to faults and consequence of faults in the EEM must be assessed. There are roughly two kinds of organization: dedicated and non-dedicated EEMs.

6.5.6.1 Dedicated EEMs

An EEM is associated with each computer. A fault in the EEM causes the computer associated with it to fail. Thus the failure rate of the EEM can be added to the failure rate of each computer.

A fault in the EEM may also cause in some configurations the loss of the corresponding bus. Analysis of the EEM design should yield the probability that such a fault happens.

6.5.6.2 Non-Dedicated EEMs

In this case, EEMs are not directly associated with computers. The computers vote on all EEMs or at any time, a primary EEM is chosen and switched to all computers. In case of failure, another one is chosen. In both cases, failure of an EEM does not cause a computer to fail. As long as one computer and one EEM are still good, the system can continue to run.

Faults in an EEM do not cause a recovery procedure. These are masked by voting. If there are only 2 EEMs, self-checking properties are used to determine which EEM has not failed. Thus the probability of fault detection in an EEM must be estimated.

6.5.7 Input-Output Faults

There are many possible I/O configurations (see Section 2). In order to provide the user with a reasonable number of different possibilities, we have modeled the two principal types of I/O configurations: these are the dedicated, and non-dedicated bus configurations. It is expected that these will provide useful approximations to systems employing variations of these approaches.

6.5.7.1 Dedicated Buses

This type of configuration is sketched on Figure 2.1-3. We list below the assumptions made for modeling this configuration:

- When a computer fails, the bus and the sensors/actuators associated with this computer cannot work any longer and thus the simulator program considers them as if they had failed.
- As long as two or more identical sensors have not failed, the system is not endangered.
- When only one good sensor of a redundant set is left, the system must be able to recognize the good sensor. This is possible through use of reasonableness tests. For each sensor, a coverage parameter must be estimated by the designer. This coverage is the probability that if all but one identical sensors are faulty, the system is able to recognize the good one.
- Failure of a complete set of identical sensors is considered as a system failure.
- Actuators are modeled in the same way as sensors. This is a valid assumption if actuators can be partitioned, each part being associated with one bus.

The condition of each of the devices in the system is represented by a Boolean variable, with a one indicating a healthy device, and zero indicating a failed device. The device-condition set is summarized as a Boolean matrix, M(B,S), where B is the number of busses and S is the number of sets of redundant devices. In the case of dedicated busses, when computer i fails, all devices connected to computer i's bus have been forced into what is, in effect, a failed state since it is no longer possible to communicate with them. For this case, M(i,j) is set to zero for all j.

Figure 6.5-5 is a sketch of such an organization. All devices of set 2 are similar. When device 3 of set 2 fails, the device-condition matrix M becomes

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

If subsequent to this, computer 2 fails, then in effect bus 2 and all devices connected to it are unusable. Thus M becomes

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

It appears that the good computers may have difficulties in deciding which of the devices in set number 2 is still good. If there is no way of deciding which is good, the coverage for the second set is input as 0 and a system failure is the consequence of the failure of computer 2. If it is always possible to decide which is good (totally self-checking device), then there is no system failure in this case. Intermediate cases are possible. The coverage is then a number between 0 and 1.

At the end of the simulated mission, each column of the matrix is scanned. If a column has no "1", it is a system failure. If a column j has only one "1", a random number $0 \leq x < 1$ is generated and compared with the coverage of the j^{th} device. If the random number is the bigger, it is a system failure condition.

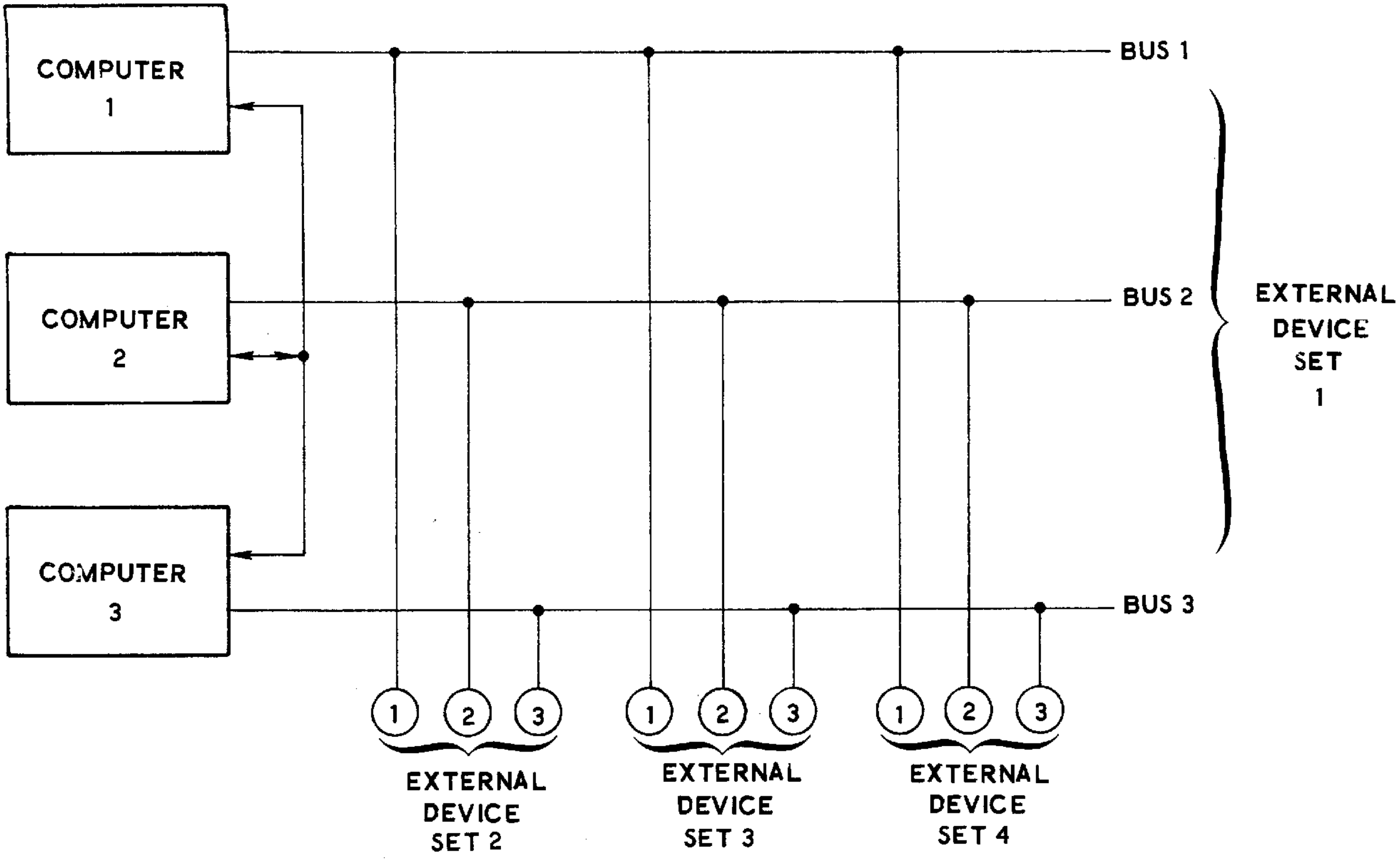


FIGURE 6.5-5 EXAMPLE OF DEDICATED BUS CONFIGURATION

6.5.7.2 Non-Dedicated Buses

The modeling is the same as in the previous case except that the failure of one computer does not cause a bus to fail. Thus, it would seem that this system is always more reliable than a dedicated bus configuration. However this is not true since the reliability of the voter/switch must be taken into account. It may be the EEMs which perform this voting function.

Whether the busses are dedicated or not, external device set 1 is by definition the bus: if a bus fails, all devices connected to this bus fail.

6.6 TESTING

6.6.1 Fault Generator

This is the easiest part to test. Given a definite distribution its mean and variance can be computed. They can also be computed from a sample obtained from the fault generator. Comparison of the two sets of results and taking into account the size of the sample permit to validate (or not) the generator.

6.6.2 Simulator

Validation of the simulator is a more difficult task and as a matter of fact, could only be completed when comparisons with experimental results could be achieved. Obviously, this is not possible and some alternate route has to be found.

First of all we test that the simulator does what we can expect of it in many different cases where the faults are known. For these tests, the faults are generated by hand so that the different paths of the program are exercised.

After being sure that the program does what the programmer expects, the results have still to be tested. The general case cannot be tested. However, by simulating simple configurations where some parameters are chosen such that they do not affect the outputs, results are obtained which can be compared with results of the modeling.

6.7 SAMPLE RUN

The output of the simulator for the simulation of a software TMR configuration without memory copy is presented in Figure 6.7-1. The remarks below refer to some of the parameters.

1. In this run, we have not studied sensor reliability. However, the program requires at least that 2 "sensors" be indicated. By definition "sensor" 1 is always the bus. In this case, sensor 2 never fails and thus does not influence the simulation.

2. Here we have chosen a very short fault duration (1μs). Thus, it is not useful to wait for the dissipation of the fault when initiating a recovery procedure, since faults are detected only at the comparison every 5 milliseconds.

3. These parameters are irrelevant in this case since there is no memory copy.

4. The restart duration is chosen longer than the maximum down time (30 ms). Since computation is stopped during a system restart, a restart implies here a failure condition.

5. Here we mean the proportion of memory which is affected by minor cycle programs. When memory damage has occurred, detection will be 100 times slower if the fault hit a major cycle program than if it hit a minor cycle. In this case faults are four times more likely to hit a major than a minor cycle program. The choice of .2 is arbitrary for this case. It is not a critical parameter since even if 3 seconds elapses before detection of the fault, the probability of having another fault in the mean time is very low.

6. Coverage means here the probability that the system recovers from a transient without discarding a computer.

7. From the size of the sample, we can conclude that

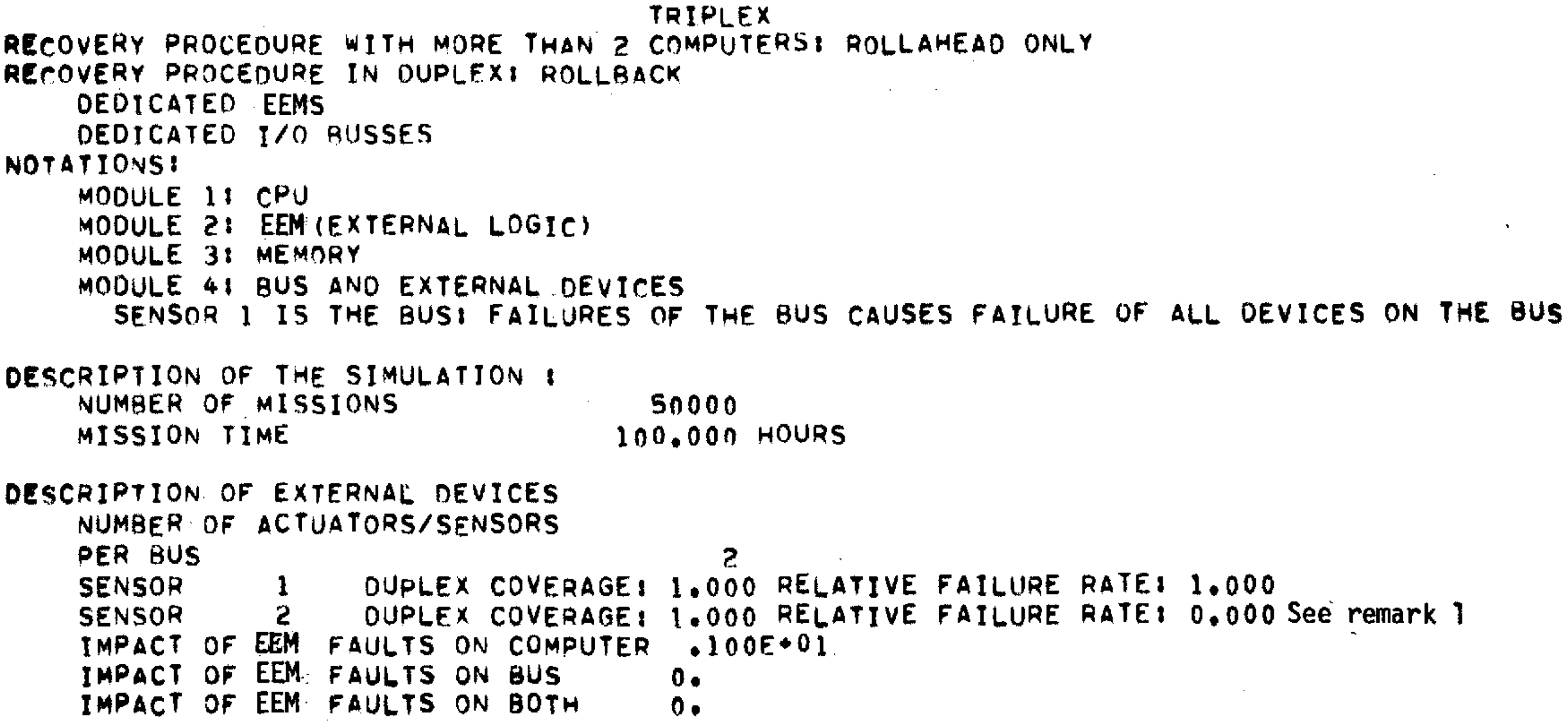
° Multiplex coverage = 75% (± 1%)

° Duplex coverage = 73% (± 3%)

° Diagnostability = 89% (± 2%)

° Failure Probability After 100 Hours $= \frac{149}{50000} = 3\ (10)^{-3} \pm .4\ (10)^{-3}$

The number of transients occurring in simplex is the difference between the total number of transients and those occurring in multiplex and duplex (recovered or not). In this case, this different yields 10175 -(6994+2319+615+223) = 34. Thus the value for the simplex coverage is not very significant since the size of the sample (34) is too small.



```
                             TRIPLEX
RECOVERY PROCEDURE WITH MORE THAN 2 COMPUTERS: ROLLAHEAD ONLY
RECOVERY PROCEDURE IN DUPLEX: ROLLBACK
    DEDICATED  EEMS
    DEDICATED I/O BUSSES
NOTATIONS:
    MODULE 1: CPU
    MODULE 2: EEM(EXTERNAL LOGIC)
    MODULE 3: MEMORY
    MODULE 4: BUS AND EXTERNAL DEVICES
      SENSOR 1 IS THE BUS: FAILURES OF THE BUS CAUSES FAILURE OF ALL DEVICES ON THE BUS

DESCRIPTION OF THE SIMULATION :
    NUMBER OF MISSIONS                    50000
    MISSION TIME                        100.000 HOURS

DESCRIPTION OF EXTERNAL DEVICES
    NUMBER OF ACTUATORS/SENSORS
    PER BUS                                   2
    SENSOR     1    DUPLEX COVERAGE: 1.000 RELATIVE FAILURE RATE: 1.000
    SENSOR     2    DUPLEX COVERAGE: 1.000 RELATIVE FAILURE RATE: 0.000 See remark 1
    IMPACT OF EEM  FAULTS ON COMPUTER  .100E+01
    IMPACT OF EEM  FAULTS ON BUS       0.
    IMPACT OF EEM  FAULTS ON BOTH      0.
```

FIGURE 6.7-1   SOFTWARE TMR WITHOUT MEMORY COPY

```
DESCRIPTION OF THE RECOVERY CHARACTERISTICS
    DELAY BEFORE RECOVERY                  0.000 MILLISECONDS
    ROLLAHEAD DURATION                      .100 MILLISECONDS
    MEMORY-COPY DURATION                2000.000 MILLISECONDS
    RECURRENCE INTERVALS
         ROLLAHEAD                                 3000. MILLISECONDS
         MEMORY-COPY                               3000. MILLISECONDS
    RESTART DURATION                    1000.000 MILLISECONDS
    MEMORY-COPY EFFICACY                    .9999
    PROGRAM SURVIVABILITY                   .300
    MAXIMUM NUMBER OF ROLLBACKS               1
    MEAN DIAGNOSIS TIME                   15.000 MILLISECONDS
    DETECTION PROBABILITY IN CPU            .050
    DETECTION PROBABILITY IN MEMORY         .450
    STP EFFICIENCY                          .980
    ISOLATION DURATION                     0.000
    NUMBER OF SPARES                          0


DESCRIPTION OF THE SOFTWARE ≠
    ITERATION PERIOD.                     30.000 MILLISECONDS
    MINOR CYCLE DURATION                   5.000 MILLISECONDS
    MAJOR CYCLE DURATION.                    100 ITERATIONS
    TIME BETWEEN COMPARISONS               5.000 MILLISECONDS
    MAXIMUM DOWN TIME.                    30.000 MILLISECONDS
    SIZE OF MINOR CYCLE PROGRAM             .200


DESCRIPTION OF THE FAULT ENVIRONMENT
    MODULE  1 : PERMANENT RATE           .45E-03 PER HOUR
                TRANSIENT RATE           .45E-03 PER HOUR
                TRANSIENT DURATION       .10E-02 MILLISECONDS (EXPONENTIAL)
    MODULE  2 : PERMANENT RATE         0.        PER HOUR
                TRANSIENT RATE         0.        PER HOUR
                TRANSIENT DURATION     0.        MILLISECONDS (EXPONENTIAL)
    MODULE  3 : PERMANENT RATE           .25E-03 PER HOUR
                TRANSIENT RATE           .25E-03 PER HOUR
                TRANSIENT DURATION       .10E-02 MILLISECONDS (EXPONENTIAL)
    MODULE  4 : PERMANENT RATE           .60E-05 PER HOUR
                TRANSIENT RATE         0.        PER HOUR
                TRANSIENT DURATION     0.        MILLISECONDS (EXPONENTIAL)
```

FIGURE 6.7-1 SOFTWARE TMR WITHOUT MEMORY COPY (Cont'd)

```
                                RESULTS
NUMBER OF FAULTS           20402
NUMBER OF TRANSIENTS       10175
NUMBER OF USED SPARES          0
NUMBER OF QUADRUPLEX           0
NUMBER OF TRIPLEX              0
NUMBER OF DUPLEX           11613
NUMBER OF SIMPLEX            922
NUMBER OF ROLLAHEADS       18607
NUMBER OF MEMORY-COPIES        0
NUMBER OF SYSTEM-RESTARTS      0
NUMBER OF ROLLBACKS         1647
NUMBER OF FAILURES           149
    NO. OF TRANSIENTS RECOVERED FROM IN MULTIPLEX      6994
    NO. OF TRANSIENTS NOT RECOVERED FROM IN MULTIPLEX  2319
    NO. OF TRANSIENTS RECOVERED FROM IN DUPLEX          615
    NO. OF TRANSIENTS NOT RECOVERED FROM IN DUPLEX      223
CAUSES OF FAILURES: DUPLEX FAILURES      110
                    EEM FAILURES           0
                    I/O FAILURES           1
                    EXCESSIVE DOWNTIME     0
                    SIMPLEX FAILURES      38
PROPORTION OF MISSED ITERATIONS  .6303E-08
LONGEST SERIES OF MISSED ITERATIONS    2

(BELOW, A COVERAGE HAS A VALUE OF -1 IF IT WAS NOT   POSSIBLE TO COMPUTE IT,I.E.
MULTIPLEX COVERAGE:      .751E+00    WHEN NO TRANSIENTS OCCUR IN  ONE OF   THE TWO MODES)
DUPLEX COVERAGE:         .734E+00
SIMPLEX COVERAGE:        .833E-01
CATASTROPHIC FAULTS:    0.
DIAGNOSTIBILITY:         .893E+00
```

FIGURE 6.7-1   SOFTWARE TMR WITHOUT MEMORY COPY (Cont'd)

THIS PAGE INTENTIONALLY LEFT BLANK

7.0 PARAMETERS

Before either the analytic model or the simulator can be used for the study of redundant computer configurations, values for their input parameters must be determined. The simulator-input values are obtained by means of an analysis of the computer and configuration under study. The simulator can then be used to obtain estimates of the parameters required by the analytic model.

7.1 SIMULATOR

The simulator can be used to estimate the parameters required by the mathematical model. It requires about forty inputs that are a function of the system configuration, the application, the fault environment and the computer's reliability and speed. Here we will consider the STP efficiency, the program integrity and the BITE efficiency. The other parameters are discussed in Section 6.3.

7.1.1 STP Efficiency

The Self-Test Program efficiency is the probability that the STP returns a correct fault indication, once a permanent (or leaky transient) fault has been detected. This is a fundamental parameter for (residual) duplex systems since it gives the probability of choosing the good computer for adaptation to simplex, once a fault has been detected through comparison and not corrected by the rollback. The STP efficiency comprises not only the proportion of faults detected with the diagnostic routine, but also the proportion of faults detected through BITE features. Because of this, some faults will be detected immediately, and others will be detected several milliseconds after the diagnostic program is initiated. If the resulting time loss is critical, the mission could fail. Thus we associate with the STP efficiency its maximum execution time -- i.e. the time it takes to execute the entire diagnostic program.

7.1.1.1 STP Requirements

For it to be effective, the STP in conjunction with BITE should verify proper operation of the following modules.

1. Memory - program memory tested by sum check if main store parity not available. Data areas tested by write/read verification with test data.
2. CPU - All instructions should be executed in a predescribed sequence with required variations and exhaustive data patterns to insure that all instructions are operating properly. All addressing schemes and registers should also be tested.
3. I/O Test - The I/O should be tested using I/O wrap checks to insure all I/O functions are operating properly.

BITE should include features such as time-out counters, I/O parity, power monitoring circuits and storage protection.

7.1.1.2 Efficiency Estimation

The STP efficiency analysis procedure consists of several steps: 1) partition the computer into several independent modules as described above, and obtain reliability data and circuit documentation for the components of each module, 2) determine which sections have no effect on computer operation (such as unused I/O channels, the AGE interface or elapsed time counter), 3) determine the failure modes (such as nand gate stuck on 1) for each circuit, and its detectability based on the STP program and BITE, 4) the STP efficiency can then be determined as follows:

Let λ_{ij} = occurrence rate of j^{th} failure mode of i^{th} circuit

β_{ij} = detectability of j^{th} failure mode of i^{th} circuit

n_i = quantity of i^{th} component

Then the STP program efficiency is given by:

$$\text{STP efficiency} = \frac{\sum_i n_i \sum_j \beta_{ij} \lambda_{ij}}{\sum_i n_i \sum_j \lambda_{ij}} \qquad \begin{array}{l} \text{Total detectable failure rate} \\ \\ \text{Total failure rate} \end{array}$$

The maximum diagnosis time can be obtained from a sizing and timing analysis of the STP program.

7.1.1.3 Typical Computers

Most computer manufacturers supply an STP program with an efficiency of 95% (manufacturer supplied estimate) and diagnosis time of 10-30 milliseconds.

7.1.2 Program Integrity

The Program Integrity (PI) is the probability that a transient in the memory will not result in any modification to the program. The rollahead/rollback procedures correct any transient that doesn't destroy the program or last too long. The program survivability parameter is used by the simulator in conjunction with the transient duration to estimate the rollahead/rollback success probabilities, which in turn are used indirectly to estimate the transient leakage.

7.1.2.1 CPU Faults

If the memory is not protected, then there is a small chance that a CPU transient will result in a program modification, for it could cause indexing errors resulting in incorrect address computations. Thus portions of critical program segments (such as the landing module) not currently being executed could be destroyed resulting in a lurking fault. This could cause a system failure at a later time in the mission.

However, many contemporary aerospace computers have storage protection capability. This facility prevents the CPU from modifying the contents of any protected storage locations.* Thus the chance that a CPU fault will result in program modification is virtually zero, and the program survivability to a CPU transient is essentially 100%.

The simulator currently assumes the existence of storage protection in the candidate computer.

7.1.2.2 Memory

The program integrity for a particular computer memory is dependent on its type and organization. A read-only memory offers the best protection, as it provides a PI of 100%. The NDRO plated wire memory is not

*Except under special conditions defined by the computer manufacturer.

quite as good, but has a much better PI than a DRO core memory. The DRO core memory is particularly bad since incorrect data obtained because of a fault occurring during a read cycle is written back into memory during the restore cycle resulting in non-intentional modification to the memory (program). Trade off data for DRO and NDRO memories is discussed in Section 9.

7.1.2.3 PI Estimation

The program integrity is defined by

$\mathrm{PI} \triangleq \mathrm{Pr}$ {The program memory contents is not modified, given that a transient failure has occurred in memory}

The program integrity can be estimated using a top-down procedure consisting of the following steps: (1) partition the memory into its functional components; (2) estimate the relative transient failure rate $({}^{\tau}i/\tau)$ for each component; (3) estimate the probability (PI_i) that when a transient occurs in the i^{th} memory component, no program word will be damaged; and (4) calculate the program integrity using this formula

$$\mathrm{PI} = \sum_i [(\mathrm{PI}_i)({}^{\tau}i/\tau)] \qquad \text{(Equation 1)}$$

where

$\mathrm{PI}_i \triangleq \mathrm{Pr}$ {Program memory is not modified when a transient occurs in the i^{th} component in memory}

$\tau_i \triangleq$ Transient failure rate for the ith memory component

$\tau = \sum_i \tau_i \triangleq$ Memory transient failure rate

The above formula is derived using the total law of probability which states (PARZ 60)

if $\{A_i\}$ is a set of mutually exclusive events

and $B \subset \bigcup_i A_i \equiv B \subset [A_1 \cup A_2 \cup \ldots \cup A_n]$

then $\mathrm{Pr}[B] = \sum_i \mathrm{Pr}\,[B|A_i]\ \mathrm{Pr}\,[A_i]$

From the above theorem, it follows that if $\{A_i\}$ is a set of mutually exclusive events and $(B \cap C) \subset \bigcup_i A_i$

then $$\Pr\,[B|C] = \sum_i \Pr\,[B|(C \cap A_i)]\ \Pr[A_i|C] \qquad \text{(Equation 2)}$$

If we let

B = event that a program word is modified

C = event that a transient fault occurs in memory

and A_i = event that a transient fault occurs in the i^{th} memory component we have

$PI = \Pr\,[B|C] = \Pr$ {Program memory is not modified given that a transient has occurred in memory}

$PI_i = \Pr\,[B|(C \cap A_i)] = \Pr$ {Program memory is not modified given that a transient has occurred in the i^{th} component}

$^{\tau}i/\tau = \Pr\,[A_i|C] = \Pr$ {Transient occurs in the i^{th} component given that a transient has occurred in memory}*

Equation 1 can be obtained from equation 2, by substitution, as is shown below:

$$\underbrace{\Pr\,[B|C]}_{PI} = \sum_i \underbrace{\Pr\,[B|(C \cap A_i)]}_{[(\,PI_i)} \ \underbrace{\Pr\,[A_i|C]}_{\times \ (^{\tau}i/\tau)]}$$

The program integrity can be determined systematically to whatever level of detail is required. This is because the PI determination procedure is recursive, that is, the i^{th} element can be partitioned into its components or failure modes and an analogous procedure used for determining PI_i from the PI_{ij}'s.

*To obtain this result, we assumed that the transient inter-arrival times for the memory and its components are exponentially-distributed random variables with means of $1/\tau$ and $1/\tau_i$, respectively.

$$PI_i = \sum_j (PI_{ij})(\tau_{ij}/\tau_i)$$

where PI_{ij} = Pr {Memory word damaged given that a transient has occurred in the jth subcomponent of the ith component}

τ_{ij} = Transient failure rate of the j^{th} subcomponent of the ith component.

The PI_{ij} can be determined by further partitioning, or by using engineering judgment to estimate the effect of the transient. In the latter case, we taken into account any masking effects. For example, only a small portion of the memory is used during a read/restore cycle, so transients occurring in certain components (such as address drivers) will only cause damage to a program word when the component is used during the duration of the transient. In this case, PI_{ij} is one minus the probability that the faulty component will be used during the transient duration and its use will result in damage to a program word.

7.1.3 BITE Efficiency

The BITE efficiency is the probability that the built-in-test equipment will detect the occurrence of a transient or permanent fault without the aid of a diagnostic program. This parameter is used by the simulator to determine the effect of rollback in simplex, and uncovered transients in duplex. It has only a negligible effect on configurations of 3 or more operating computers. This parameter is needed separately for both the CPU and the memory.

7.1.3.1 CPU BITE Efficiency

The CPU BITE efficiency is about 5% for most typical aerospace computers (excluding the power supply). In order to obtain a more accurate figure, a detailed analysis of the organization and data flow of the CPU is necessary. This parameter only has a third order effect on RCS survivability, thus a detailed estimate is unnecessary.

7.1.3.2 Memory BITE Efficiency

The memory BITE efficiency can be estimated by a procedure similar to the STP estimation procedure. Briefly, the following steps are necessary:

1. Partition the memory into components and obtain the failure rate (λ_i) and quantity (n_i) of each.

2. Determine the important failure modes of each component and its probability of occurrence (λ_{ij}).

3. Determine which failure modes can give a BITE indication (such as parity). Let $\beta_{ij} = 1$ if the j^th^ failure mode of the i^th^ component gives a BITE indication and 0 if it doesn't.

4. The memory BITE efficiency is then given by the following formula:

$$\text{Efficiency} = \frac{\sum_i n_i \sum_j \beta_{ij} \lambda_{ij}}{\sum_i n_i \lambda_i}$$

7.2 ANALYTIC MODEL

An iterative relationship developed from the mathematical model is used to evaluate reconfigurable computer systems employing transient recovery. It requires values for the following parameters:

1. The effective computer failure rate.

2. The recoverability for 2, 3 and more operating computers.

3. The transient leakage for 1, 2, 3 and more operating computers.

7.2.1 Computer Effective Failure Rate

The computer effective permanent failure rate is the value of λ used in the analytic model for system evaluation purposes. This failure rate may be different from that solely attributable to the computer. This comes about because some HASW systems involve the use of computer-dedicated EEMs such that an EEM failure prevents the proper operation of its associated computer.

Obtaining the computer effective permanent failure rate is accomplished using these steps. First, an estimate is obtained of the computer failure rate. This may be obtained from the manufacturer or may be estimated independently by the evaluator. Second, a rough design of the EEM is prepared.* Next, the EEM

*Assuming an actual design is not available.

failure rate is estimated.. Then, the EEM design is analyzed and the portions identified that, when failed, impair the computer's operation. The failure rates of these portions are then added to the failure rate of the computer to yield the effective failure rate. If, as is the case for some configurations, other EEM failure impair bus operation, these failures are added to obtain a bus effective failure rate.

7.2.2 Recoverability

This parameter is a measure of a redundant (N) computer system's ability to recover to a less redundant (N-1) computer system from permanent or leaky transient faults. The recoverability for 3 or more operating computers is very close to 1.0 for a well designed system since a faulty computer is immediately updated during the comparison. For a (residual) duplex system, a self test program must be invoked to isolate the faulty computer, so the recoverability for duplex is dependent upon the STP efficiency.

The duplex recoverability is obtained as one of the outputs from the simulator, which determines it by forming the ratio of the number of residual simplex systems to the number of permanents and leaky transients in duplex.

7.2.3 Transient Leakage

The transient leakage represents the probability that a computer will not completely recover from a transient, i.e. the system will mistake the transient for a permanent fault and adapt from N to N-1 operating computers. The simulator estimates the transient leakage for 1, 2 and 3 operating computers by determining the ratio of uncovered to covered transients in each case. The transient leakage is dependent on the memory type, the transient duration, and the transient recovery algorithms used.

8.0 COMPLEMENTARY ANALYTIC-SIMULATIVE TECHNIQUE

8.1 OVERALL STRUCTURE

The analytic modeling approach described in Section 5 and the simulation technique described in Section 6 each has its strengths and limitations. However when these two system evaluation approaches are combined, and supplemented by some engineering analysis, a very powerful technique results. The combination is illustrated in Figure 8.1-1.

This Complementary Analytic-Simulative Technique (CAST) evolved as it became evident that neither analysis nor simulation alone could satisfy all the RCS evaluation requirements. Analytic modeling provides flexibility and rapid, economical data-generation. However the solutions for some configurations are very cumbersome and in certain cases the mathematical model formulated is intractable. Simulation permits computer system details to be included easily, but data generation is slow and expensive. CAST permits the user to obtain the best features of both analytic modeling and simulation.

8.2 RCS ENGINEERING ANALYSIS

The RCS engineering analysis is performed to provide six categories of information to the analytic modeling and the simulation. These information categories are:

1. Configuration Particulars
2. Fault Environment
3. System Failure Criteria
4. Software Structure
5. Recovery Features
6. Test Features

The configuration particulars are: the computer system type, e.g. adaptive or non-adaptive, etc.; the maximum number of machines; and the external hardware utilized.

The items provided under the fault-environment category are: the permanent-fault occurrence rate; the transient-fault occurrence rate; transient duration; and occurrence rate for bursts of faults.

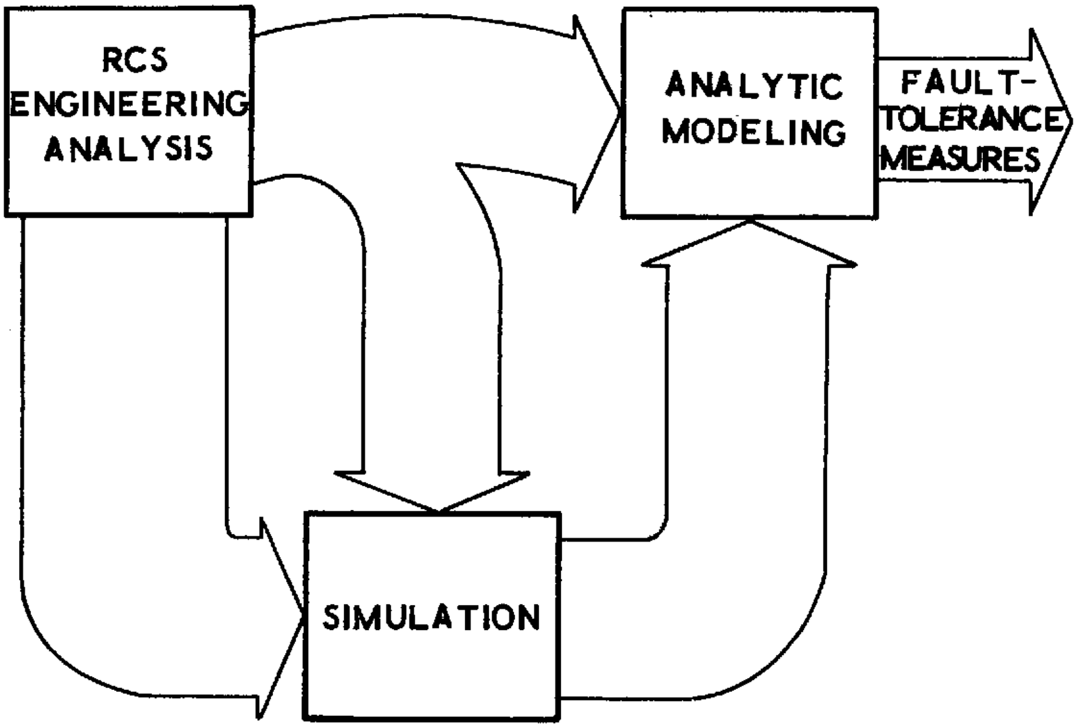


FIGURE 8.1-1 FAULT-TOLERANCE MEASURES CAN BE PRODUCED THROUGH A COMBINATION OF ENGINEERING ANALYSIS, SIMULATION, AND ANALYTIC MODELING

There are three system failure criteria that may be applied. These are: missed iterations; output not delivered in time; and/or a critical computation missed.

The software structure information that is provided as a result of the RCS engineering analysis includes the type of scheduling mechanism employed in the executive, e.g. synchronous or asynchronous; and the general sequence of the applications program segments.

Recovery features deal principally with the specification of which recovery algorithms should be used and in what sequence. The six basic possibilities are: rollahead; memory copy; rollback; system restart; system adaptation; and spare introduction.

The final category of information produced by the RCS engineering analysis is that of test features. This category includes information about: self-test programs (e.g. effectiveness and maximum diagnosis time); the use of error detecting, error-correcting codes; the use and effectiveness of built-in test equipment; output results comparison; voting of output results; and finally reasonableness tests.

8.3 SIMULATION

The results produced by the simulator developed have been described in detail in Section 6.3. The reader is merely reminded here that the following items are available as simulator outputs:

1. Permanent-fault coverage
2. Transient-fault coverage
3. Detectability
4. Diagnostability
5. Recoverability

8.4 ANALYTIC MODELING

The analytic modeling provides the following measures of fault-tolerance:

1. Computer system survivability (or failure probability)
2. Computer system reliability

Figure 8.4-1 is a summary diagram of CAST showing what is produced by each of the three aspects of the technique.

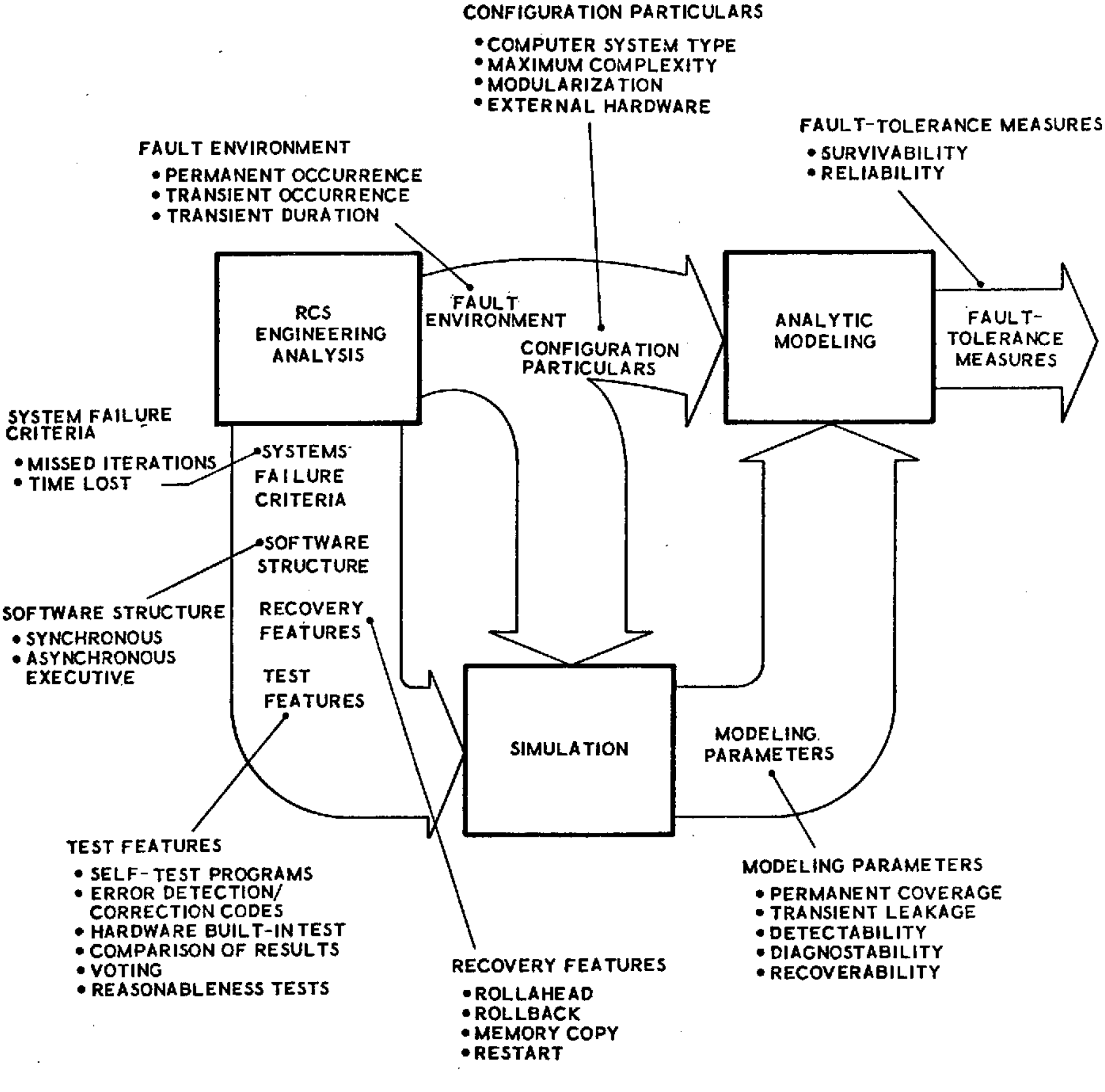


FIGURE 8.4-1 CAST SUMMARY DIAGRAM

THIS PAGE INTENTIONALLY LEFT BLANK

9.0 CONFIGURATION ANALYSES AND TRADE-OFF STUDIES

9.1 GENERAL

The analyses and trade-off studies described below are presented to show the merits of the various reconfigurable computer system organization concepts and the effects of using the various reliability enhancement techniques. Before presenting these analyses and trade-offs it is appropriate to discuss the ground rules governing these studies.

The general application class for the RCS studied is that of a machine that provides at least the capability sufficient to handle an all-digital, fly-by-wire control system for a passenger-carrying airplane. The decision was made to consider a machine that has a memory capacity of 16K, 32-bit words.* The general class of aerospace computers appropriate for this application all supply at least that capability in one ATR enclosure. It is also possible to satisfy the requirements of the four function classes of attitude and flight-path control, area nagivation, communications, and air-traffic control (with the exception of display) defined in RATN 73, in just over 12K words, thus leaving 4K for the executive. Hence, we are not faced with the problem of excessive memory requirements.

The configurations presented are not oriented toward particular individual aerospace computers, but rather toward classes of machines. Thus the results should not be construed as favoring a particular design.

One difficulty encountered in evaluating a computer configuration is the treatment of sensor and/or actuator failures. This difficulty arises because of the diversity of devices, the particular devices employed, the total number of devices, and the individual device redundancies. Because of these complexities, it was decided to include a seminal capability in CAST (principally in the simulator portion) to model I/O devices, and to set I/O device failure rates to zero for these studies.

* Appendix B contains descriptions of four representative machines.

9.2 PARAMETERS USED FOR EVALUATION

Evaluation studies have been made for three classes of configurations: a mostly-software configuration, a hardware-aided-software configuration and a mostly-hardware configuration. The first set of configurations uses a 16K memory and a computer with developed I/O capabilities. The second and third ones use only a 16K memory, but the computer has less I/O capabilities.

When studying the influence of some parameters, we specify the configuration and the varying parameters. Non-varying parameters are those indicated in the table corresponding to the configuration.

Table 9.2-I is for mostly-software configurations. Table 9.2-II is for hardware-aided-software configurations.

9.2.1 Mostly-Software Configurations (Table 9.2-I)

9.2.1.1 Physical Parameters

9.2.1.1.1 Design Decisions

The number of computers will always be specified. A software configuration has no EEM. Each bus is supposed to be dedicated to a computer, except otherwise specified.

Sensor redundancy is studied in Section 9.3.5.4. Otherwise, we always separate them from our study.

9.2.1.1.2 Failure Characteristics

The failure rate (per million hours) comes from our evaluation of a typical computer with high I/O capabilities*. The intercommunications module failure rate is included into the CPU failure rate. This is due to the fact that the results of such faults are bad outputs, as for CPU faults. The bus has a typical failure rate of 6 per million hours.

Transient rates are supposed equal to permanent rates and the transient duration is set to 1 μs. There is little data confirming these assumptions. Section 9.3.6 shows a great sensitivity of the failure probability to the transient rate and duration. This demonstrates the important need of a better knowledge of the transient environment.

* A detailed failure-rate determination for one of the four representative machines is presented in Appendix C.

TABLE 9.2-1 LIST OF INPUT PARAMETERS FOR MOSTLY SOFTWARE CONFIGURATIONS

1) PHYSICAL PARAMETERS

a) Design Decisions

Number of Computers	
Dedicated/Non Dedicated EEMs	–
Number of EEMs	–
Dedicated/Non Dedicated Buses	D
Number of Buses	–
Number of External Devices Per Bus	0
Number of Spare Computers	0

b) Failure Characteristics

	λ[3]	μ[3]	τ[3]	$1/\gamma$[3]
CPU (I)	450	NA[4]	450	1μs
Memory (II)	200	NA[4]	200	1μs
EEM[12] (III)	–	NA[4]	–	–
Bus and External Devices (IV)	6	NA[4]	0	–
Impact of EEM Failure on Bus				–
$\frac{\text{Number of Faults in the Bus}}{\lambda_{IV}}$				1
$\frac{\text{Number of Faults in Each External Device}}{\lambda_{IV}}$				0

2) SOFTWARE CHARACTERISTICS

Synchronous/Asynchronous Scheduling Mechanism	S
Iteration Period	30 ms
Minor Cycle Duration	5 ms
Major Cycle Duration (In terms of iterations)	100
Time Between Comparisons	5 ms
Maximum Down Time	30 ms
Relative Size of Minor Cycle Program	.2
Interrupt Rate	–

3) PARAMETERS AFFECTING FAULT DETECTION AND ISOLATION IN THE COMPUTERS

a) Detection Efficiency

Number of Computers	3 or more	2	1
Comparisons[7]	100%	100%	0%
CPU BITE	5%	5%	5%
Memory BITE	45%	45%	45%

b) Isolation Efficiency[9]

Number of Computers	3 or more	2
Comparisons[7]	100%[10]	0%[10]
CPU BITE	5%	5%
Memory BITE	45%	45%
Diagnosis[8] (STP)	98%	98%

4) PARAMETERS AFFECTING FAULT DETECTION AND ISOLATION IN THE EXTERNAL HARDWARE

a) EEM[11]

Number of EEMs	3 or more	2	1
Detection Efficiency	100%[10]	100%[10]	0%[10]
Isolation Efficiency	100%[10]	–	N-A.[14]

b) Bus and External Devices

- Bus

Number of Buses	3 or more	2	1
Detection Efficiency	100%[10]	100%[10]	0%[10]
Isolation Efficiency	100%[10]	0	N-A.[14]

- For Each External Device

Number of Redundant Devices	3 or more	2	1
Detection Efficiency	100%[10]	100%[10]	0%[10]
Isolation Efficiency	100%[10]	–	N.A.[14]

5) TRANSIENT FAULT RECOVERY PARAMETERS

	Efficiency for CPU Fault	Efficiency for Memory Fault	Duration	Maximum Number of Trials	Time Limit	In Use
Rollahead	100%[10]	.30	.1 ms	NA[14]	3000 ms	Yes[10]
Rollback			5 ms	1	NA[14]	Yes[10]
Memory Copy	NA[14]	.9999	2000 ms	NA[14]	3000 ms	Yes
System Restart Duration	100%[10]	100%[10]	1000 ms	NA[14]	NA[14]	Yes[10]

6) PERMANENT FAULT RECOVERY PARAMETERS

Always done by switching off or ignoring the faulty computer, once the permanent has been recognized. If a spare is available, it is switched in.

Number of Computers	3 or more	2	1
Efficiency	100%[10]	98%	0%[10]
Duration	0[10]	$0 \leq d \leq 30$ ms	NA[11]

FOOTNOTES

(1) Implicit parameter when each EEM/bus is dedicated to one computer.

(2) The bus by itself is considered as an external device.

(3) λ: permanent fault rate; μ: dormant fault rate.
τ: transient fault rate; $1/\gamma$: mean transient duration.

(4) Non applicable when there is no spare.

(5) In this example, it is assumed that the four external devices have the same fault rate.

(6) Non applicable when synchronous scheduling.

(7) These parameters are implicit. However, the 100% efficiency is reached only after the whole memory has been exercised, i.e. after a full major cycle. Thus BITE feature may speed up detection.

(8) This parameter is valid only for permanent faults. Diagnosis is no help with transient faults.

(9) Isolation efficiency is 100%, once the fault has been detected in simplex.

(10) Implicit parameters.

(11) These parameters are not applicable with dedicated EEM: in this case, faults in the EEM are considered as equivalent to CPU faults.

(12) These parameters are 0 with a software TMR since there is no EEM.

(13) Rollahead is used for 3 or more computers.
Rollback is used for 2 or 1 computers.

(14) Non applicable.

(15) Rollahead is preceded by an imposed delay to allow the transient to dissipate.

(16) Irrelevant when EEM are dedicated.

9.2.1.2 Software Characteristics

Except in Section 9.3.7, these parameters are constand and valid for all configurations.

We have a synchronous scheduling mechanism. The sample period is typical of a fly-by-wire and is 30 ms long. The minor cycle duration which represents the duration of the computation repeated every period is 5 milliseconds long. A major cycle is a hundred periods long, i.e., 3 seconds long. Comparisons take place every 5 ms. We suppose that the maximum down time is 30 milliseconds long. This means that two consecutive iteration misses cause a system failure.

9.2.1.3 Parameters Affecting Fault Tolerance in the Computers

Ways of evaluating these parameters are explained in Section 7. Diagnosis efficiency is given by the manufacturer. The values listed here are typical of a computer with high I/O capabilities.

9.2.1.4 Parameters Affecting Fault Tolerance in the External Hardware

As stated in 9.2.1.1.1, there are no EEM and no external devices in this configuration.

We suppose that the system needs two busses to perform satisfactorily. This corresponds to an isolation efficiency of 0% when only two busses are left.

9.2.1.5 Transient Fault Recovery Parameters

We suppose that 70% of a DRO memory transient fault affects the content of a memory word, thus impairing rollahead and rollback efficiency. A rollahead lasts .1 ms and a rollback 5 ms (time between comparison).

The memory copy has an efficacy of .9999. This corresponds to the very low probability that the little routine bootstrapping the memory copy recovery be itself destroyed. A memory copy lasts 2 seconds which roughly corresponds to copying one word every 100 microseconds.

We allow for only 1 rollback. This means that if the rollback is unsuccessful, adaptation to simplex is attempted. This is reasonable since we supposed a transient duration of 1 microsecond.

The recurrence intervals are 3 seconds long, or one major cycle. We suppose that during one major cycle the whole program memory is exercised. Thus, it is unlikely to mistake a permanent for a transient. This would happen if the permanent was redetected after more than 3 seconds, following an unsuccessful recovery attempt.

9.2.1.6 Permanent Fault Recovery Parameters

The manufacturer gives the efficiency and duration of the diagnostic routine.

9.2.1.7 Modeling Parameters

Simulation of the mostly-software configurations results in the following modeling parameter values:

Parameter	Value
ℓ_1	0.87
ℓ_2	0.37
ℓ_3, ℓ_4, ℓ_5	10^{-4}
v_2	0.94
w_2	0.8
w_3, w_4, w_5	1

A total of more than ten million missions were simulated during the simulation runs performed on this contract. When the state of the configuration consisted of three or more working computers there were no faults that directly resulted in system failure.

9.2.2 Hardware-Aided-Software Configurations (Table 9.2-II)

Most of the considerations of Section 9.2.1 are still valid except that now we consider a computer with minimum I/O capabilities.

9.2.2.1 Physical Parameters

9.2.2.1.1 Design Decisions

The EEM's and the busses in a HASW system may or may not be dedicated.

9.2.2.1.2 Failure Characteristics

The computer has a lower failure rate than for a mostly-software configuration. The failure rate of the EEMs partially makes the difference. The complexity of an EEM should be less than the complexity of the intercommunication module of the computer used in Section 9.2.1.

We suppose that in 60% of the cases, the loss of the EEM will cause the loss of the bus since the EEM outputs wrong data.

9.2.2.2 Software Characteristics

See Section 9.2.1.2.

9.2.2.3 Parameters Affecting Fault Tolerance in the Computers

The main difference from the software case is that we assume a machine with memory parity which raises the memory BITE efficiency to 80 percent.

9.2.2.4 Parameters Affecting Fault Tolerance in the External Hardware

In the case of non-dedicated EEMs, we assume that hardware configurations require two EEMs to be fault free before system failure. This corresponds to an isolation efficiency of 0% when two EEMs are left.

9.2.2.5 Fault Recovery Parameters

The same remarks as in 9.2.1.5 and 9.2.1.6 are still valid.

9.2.2.6 Modeling Parameters

Simulation of the hardware-aided software configurations results in the following modeling parameter values:

Parameter	Value
ℓ_1	0.87
ℓ_2	0.45
ℓ_3, ℓ_4, ℓ_5	10^{-4}
v_2	0.90
w_2	0.90
w_3, w_4, w_5	1

TABLE 9.2-II LIST OF INPUT PARAMETERS FOR HARDWARE CONFIGURATIONS

1) PHYSICAL PARAMETERS

a) Design Decisions

Number of Computers	
Dedicated/Non Dedicated EEMs	
Number of EEMs	
Dedicated/Non Dedicated Buses	
Number of Buses	
Number of External Devices Per Bus	0
Number of Spare Computers	0

b) Failure Characteristics

		λ(3)	μ(3)	τ(3)	$1/\gamma$(3)
CPU	(I)	200	NA(4)	200	1us
Memory	(II)	250	NA(4)	250	1us
EEM(12)	(III)	100	NA(4)	100	1us
Bus and External Devices	(IV)	6	NA(4)	0	–
Impact of EEM Failure on Bus					60%
Number of Faults in the Bus / λ_{IV}					100%
Number of Faults in Each External Device / λ_{IV}					–

2) SOFTWARE CHARACTERISTICS

Synchronous/Asynchronous Scheduling Mechanism	S
Iteration Period	30 ms
Minor Cycle Duration	5 ms
Major Cycle Duration (In terms of iterations)	100
Time Between Comparisons	5 ms
Maximum Down Time	30 ms
Relative Size of Minor Cycle Program	.2
Interrupt Rate	

3) PARAMETERS AFFECTING FAULT DETECTION AND ISOLATION IN THE COMPUTERS

a) Detection Efficiency

Number of Computers	3 or more	2	1
Comparisons(7)	100%	100%	0%
CPU BITE	5%	5%	5%
Memory BITE	80%	80%	80%

b) Isolation Efficiency(9)

Number of Computers	3 or more	2
Comparisons(7)	100%(10)	0%(10)
CPU BITE	5%	5%
Memory BITE	80%	80%
Diagnosis(8) (STP)	97.5%	97.5%

4) PARAMETERS AFFECTING FAULT DETECTION AND ISOLATION IN THE EXTERNAL HARDWARE

a) EEM(11)

Number of EEMs	3 or more	2	1
Detection Efficiency	100%(10)	100%(10)	0%(10)
Isolation Efficiency	100%(10)	0	N-A.(14)

b) Bus and External Devices

- Bus

Number of Buses	3 or more	2	1
Detection Efficiency	100%(10)	100%(10)	0%(10)
Isolation Efficiency	100%(10)	0	N-A.(14)

- For Each External Device

Number of Redundant Devices	3 or more	2	1
Detection Efficiency	100%(10)	100%(10)	0%(10)
Isolation Efficiency	100%(10)	–	N.A.(14)

5) TRANSIENT FAULT RECOVERY PARAMETERS

	Efficiency for CPU Fault	Efficiency for Memory Fault	Duration	Maximum Number of Trials	Time Limit	In Use
Rollahead	100%(10)	.30	.1 ms	NA(14)	3000 ms	Yes(10)
Rollback			5 ms		NA(14)	Yes(10)
Memory Copy	NA(14)	.9999	2000 ms	NA(14)	3000 ms	Yes
System Restart Duration	100%(10)	100%(10)	1000 ms	NA(14)	NA(14)	Yes(10)

6) PERMANENT FAULT RECOVERY PARAMETERS

Always done by switching off or ignoring the faulty computer, once the permanent has been recognized. If a spare is available, it is switched in.

Number of Computers	3 or more	2	1
Efficiency	100%(10)	97.5%	0%(10)
Duration	0(10)	0≤d≤10 ms	NA(11)

FOOTNOTES

(1) Implicit parameter when each EEM/bus is dedicated to one computer.

(2) The bus by itself is considered as an external device.

(3) λ: permanent fault rate; μ : dormant fault rate.
τ: transient fault rate; $1/\gamma$: mean transient duration.

(4) Non applicable when there is no spare.

(5) In this example, it is assumed that the four external devices have the same fault rate.

(6) Non applicable when synchronous scheduling.

(7) These parameters are implicit. However, the 100% efficiency is reached only after the whole memory has been exercised, i.e. after a full major cycle. Thus BITE feature may speed up detection.

(8) This parameter is valid only for permanent faults. Diagnosis is no help with transient faults.

(9) Isolation efficiency is 100%, once the fault has been detected in simplex.

(10) Implicit parameters.

(11) These parameters are not applicable with dedicated EEM: in this case, faults in the EEM are considered as equivalent to CPU faults.

(12) These parameters are 0 with a software TMR since there is no EEM.

(13) Rollahead is used for 3 or more computers.
Rollback is used for 2 or 1 computers.

(14) Non applicable.

(15) Rollahead is preceded by an imposed delay to allow the transient to dissipat

(16) Irrelevant when EEM are dedicated.

A total of more than ten million missions were simulated during the simulation runs performed on this contract. When the state of the configuration consisted of three or more working computers, there were no faults that directly resulted in system failure.

9.2.3 Mostly-Hardware Configurations

These configurations use the same computers as those described in Section 9.2.2. Thus, most of the parameters are the same. However, the complexity of the EEM increases and the failure rate of the EEMs has been estimated in this case to be 200 per million hours.

9.3 GENERATION OF RESULTS

9.3.1 Assessed Configurations

The configurations assessed are mostly-software, hardware-aided-software and mostly-hardware with 3, 4, and 5 computers plus a duplex configuration.

The computers are all adaptive in that whenever two computers are remaining, the system enters a residual duplex mode as described in Section 2.2.2.

Hardware configurations require two EEMs to be fault free before system failure.

Two busses are required to be fault free for hardware and software configurations. The case of adaptation of busses from duplex to simplex when detection coding and I/O wrap is used is also considered.

9.3.1.1 Quintuplex Configurations

Table 9.3-I is a summary of quintuplex configuration assessments. The entries are ordered in increasing 10-hour failure probability. Under "configuration type," HASW represents hardware-aided-software, MSW represents mostly-software, and MHW represents mostly-hardware. The number under "computers," "EEMs," and "busses" indicate the redundancy of that device. The two dedication columns indicate whether members of the neighboring columns are dedicated or not (DED is dedicated while N/D is non-dedicated). The table represents 14 different configurations.

Figures 9.3-1 and 9.3-2 are plots of the failure probabilities versus mission time and extended mission time, respectively, for the configurations in Table 9.3-I. The curves are identified by numbers corresponding to the "number" column of Table 9.3-I.

Figure 9.3-1 shows failure probability as the ordinate on a logarithmic scale between 10^{-12} and 10^{-5}. The abscissa is an equal increment scale of mission time from 1 to 25 hours. Figure 9.3-2 is on log-log scale with failure probability shown from 10^{-10} to 10^{-2} and mission time from 1 to 1000 hours.

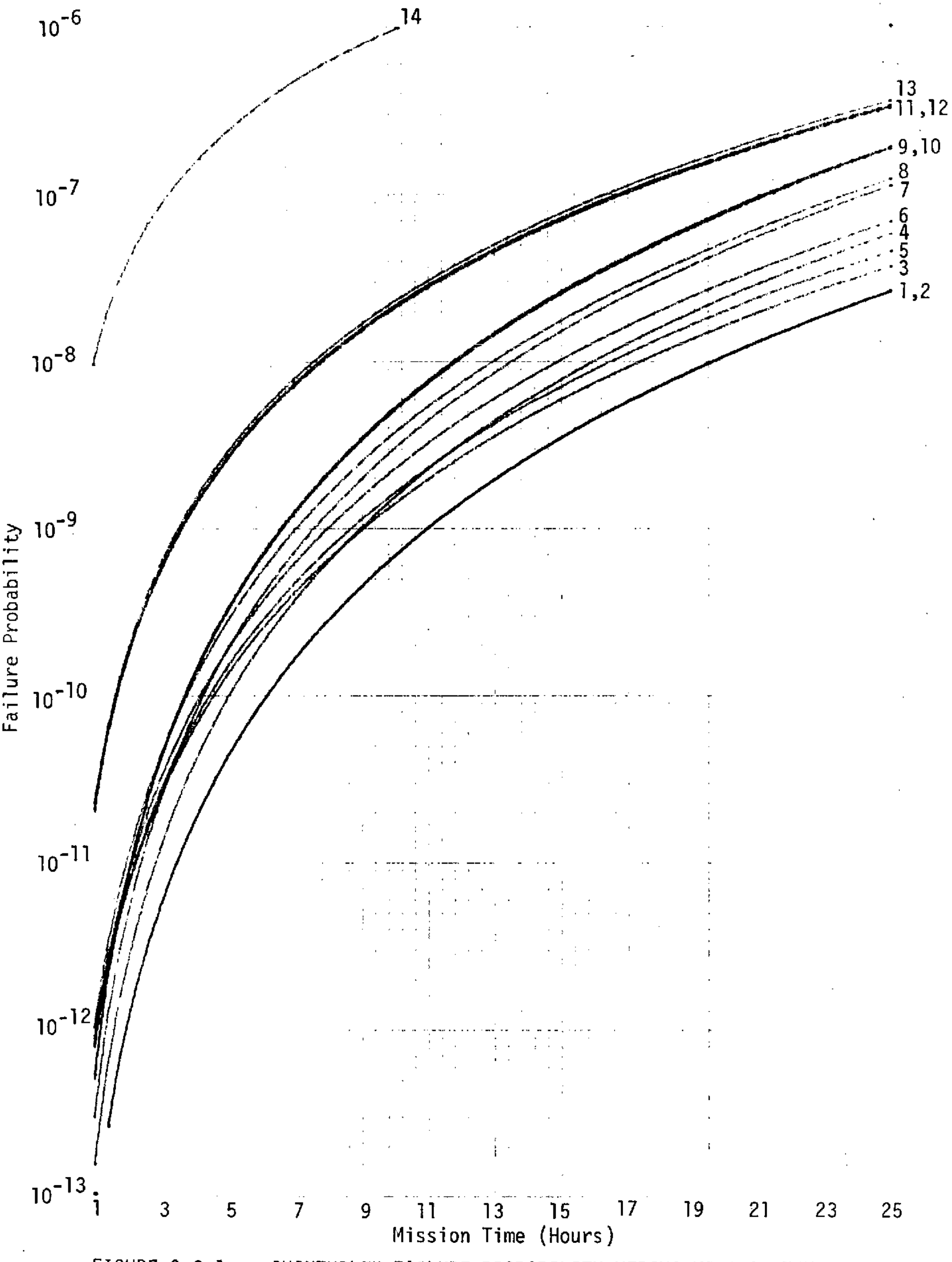


FIGURE 9.3-1 QUINTUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES

TABLE 9.3-I SUMMARY OF QUINTUPLEX CONFIGURATION ASSESSMENTS

NUMBER	CONFIGURATION TYPE	COMPUTERS	DEDICATION	EEMs	DEDICATION	BUSSES	10 HOUR FAILURE PROBABILITY
1	HASW	5	N/D	5	N/D	5	$6.65(10)^{-10}$
2	HASW	5	N/D	5	DED	5	$6.71(10)^{-10}$
3	HASW	5	N/D	5	N/D	4	$1.37(10)^{-9}$
4	HASW	5	DED	5	N/D	5	$1.50(10)^{-9}$
5	MHW	5	N/D	5	N/D	4	$1.59(10)^{-9}$
6	HASW	5	DED	5	DED	4	$2.20(10)^{-9}$
7	MHW	5	DED	5	DED or N/D	5	$2.90(10)^{-9}$
8	MHW	5	DED	5	N/D	4	$3.60(10)^{-9}$
9	MSW	5	N/D	-	-	5	$4.90(10)^{-9}$
10	MSW	5	DED	-	-	5	$5.06(10)^{-9}$
11	HASW	5	N/D	4	N/D	5	$2.02(10)^{-8}$
12	HASW	5	N/D	4	N/D	4	$2.09(10)^{-8}$
13	HASW	5	N/D	4	DED	4	$2.23(10)^{-8}$
14	HASW	5	N/D	4	N/D	3	$9.60(10)^{-7}$

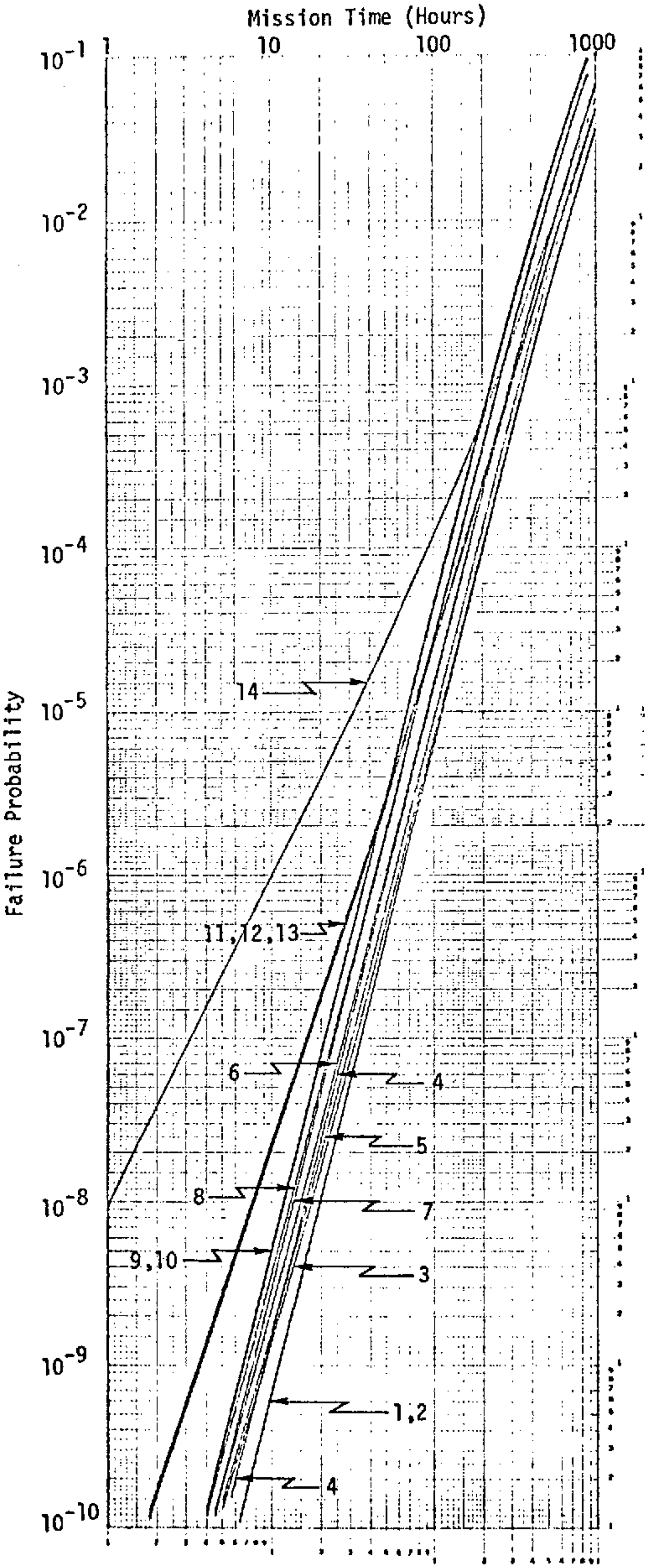


FIGURE 9.3-2

QUINTUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES

Figure 9.3-3 shows the effect of adaptive busses in quintuplex configurations. Adaptive busses are achieved by error-detecting coding on the busses and the use of I/O wrap. When down to duplex busses, codes on I/O wrap can identify a faulty bus. All curves are HASW. Curves 1 and 2 represent dedicated EEMs with either 3 or 4 busses. Curve 3 represents a dedicated EEM with 5 busses. Curves 4, 5, and 6 represent 5 non-dedicated EEMs with 3 busses, 4 busses, and 5 busses, respectively.

9.3.1.2 Quadruplex Configurations

Table 9.3-II is a summary of the quadruplex configurations. The entries are ordered by increasing failure probability. An explanation of the columns of the table is given in Section 9.3.1.1 for quintuplex configurations.

Figures 9.3-4 and 9.3-5 are plots of the failure probabilities versus mission time and extended mission time, respectively, for the configurations in Table 9.3-II. The curves are identified by the numbers in the "number" column of Table 9.3-II.

Figure 9.3-6 shows the effect of adaptive busses in quadruplex configurations. All curves are HASW. Curves 1 and 3 represent dedicated EEMs with 3 and 4 busses, respectively. Curves 2 and 4 represent non-dedicated EEMs with 3 and 4 busses, respectively.

9.3.1.3 Triplex Configurations

Table 9.3-III is a summary of the triplex configurations. Duplex and TMR configurations are shown as well. The entries are ordered in increasing failure probability. An explanation of the columns of the table is given in Section 9.3.1.1. The failure probability for configuration number 6, the MSW with dedicated busses, is verified by simulation. Some entries represent more than one configuration with the same failure probability.

Figures 9.3-7 and 9.3-8 are plots of the failure probabilities versus mission time and extended mission time, respectively, for the configurations listed in Table 9.3-III with the exception of TMR and duplex. TMR and duplex curves will be shown in Section 9.3.2. The curves are identified by the number in the "number" column of Table 9.3-III.

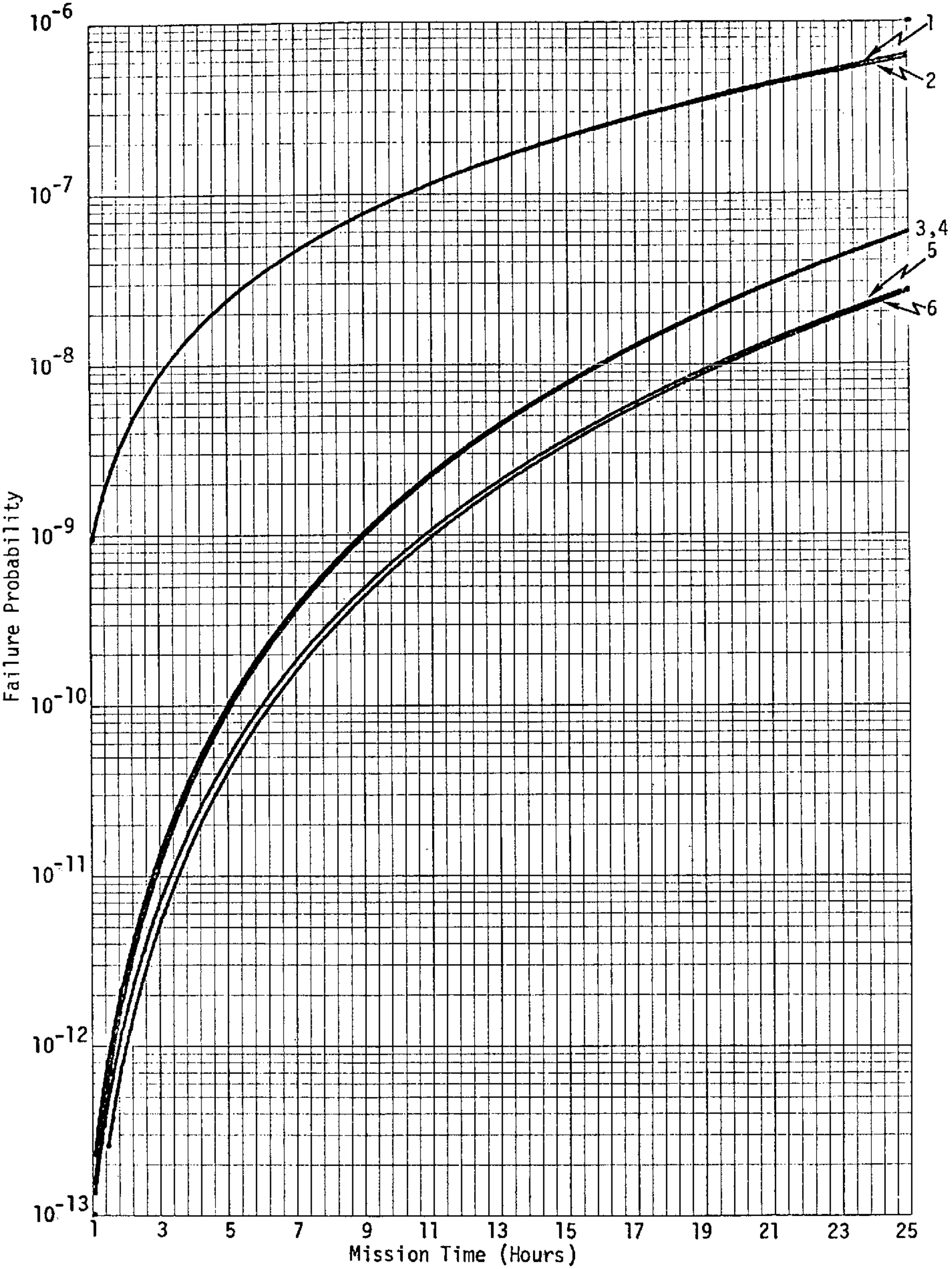


FIGURE 9.3-3 QUINTUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR ADAPTIVE BUSSES

TABLE 9.3-II SUMMARY OF QUADRUPLEX CONFIGURATION ASSESSMENTS

NUMBER	CONFIGURATION TYPE	COMPUTERS	DEDICATION	EEMs	DEDICATION	BUSSES	10 HOUR FAILURE PROBABILITY
1	HASW	4	N/D	5	N/D	4	$9.67(10)^{-8}$
2	HASW	4	N/D	4	DED or N/D	4	$1.08(10)^{-7}$
3	MHW	4	N/D	4	DED or N/D	4	$1.45(10)^{-7}$
4	HASW	4	DED	4	DED or N/D	4	$1.71(10)^{-7}$
5	MHW	4	DED	4	DED or N/D	4	$2.75(10)^{-7}$
6	MSW	4	N/D	-	-	4	$4.56(10)^{-7}$
7	MSW	4	DED	-	-	4	$4.68(10)^{-7}$
8	HASW	4	N/D	5	N/D	4	$8.76(10)^{-7}$
9	HASW	4	N/D	4	N/D	4	$8.87(10)^{-7}$
10	HASW	4	DED	4	N/D	3	$9.49(10)^{-7}$
11	HASW	4	N/D	3	DED	3	$6.48(10)^{-6}$
12	HASW	4	N/D	3	N/D	3	$6.74(10)^{-6}$

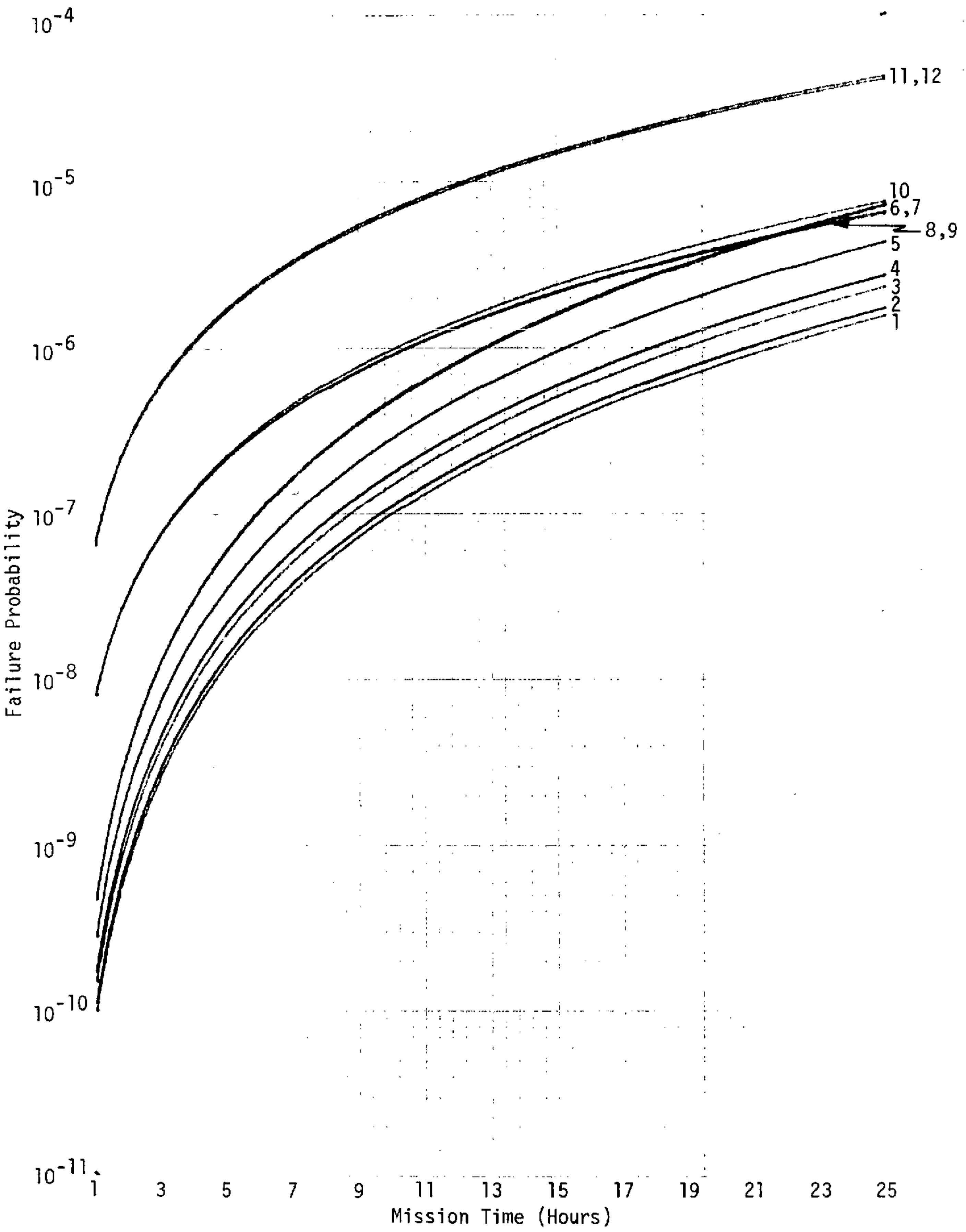


FIGURE 9.3-4 QUADRUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES

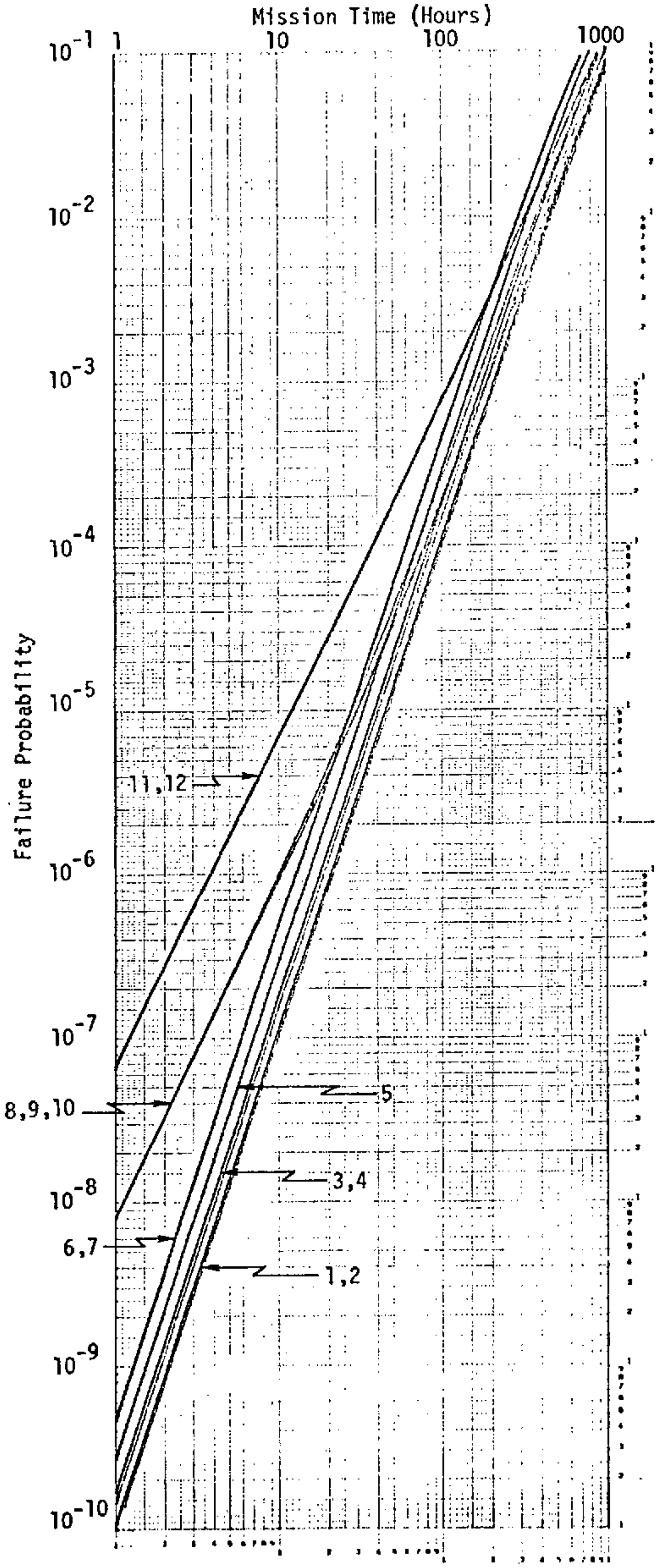


FIGURE 9.3-5

QUADRUPLEX FAILURE PROBABILITY VERSUS EXTENDED MISSION TIME FOR NON-ADAPTIVE BUSSES

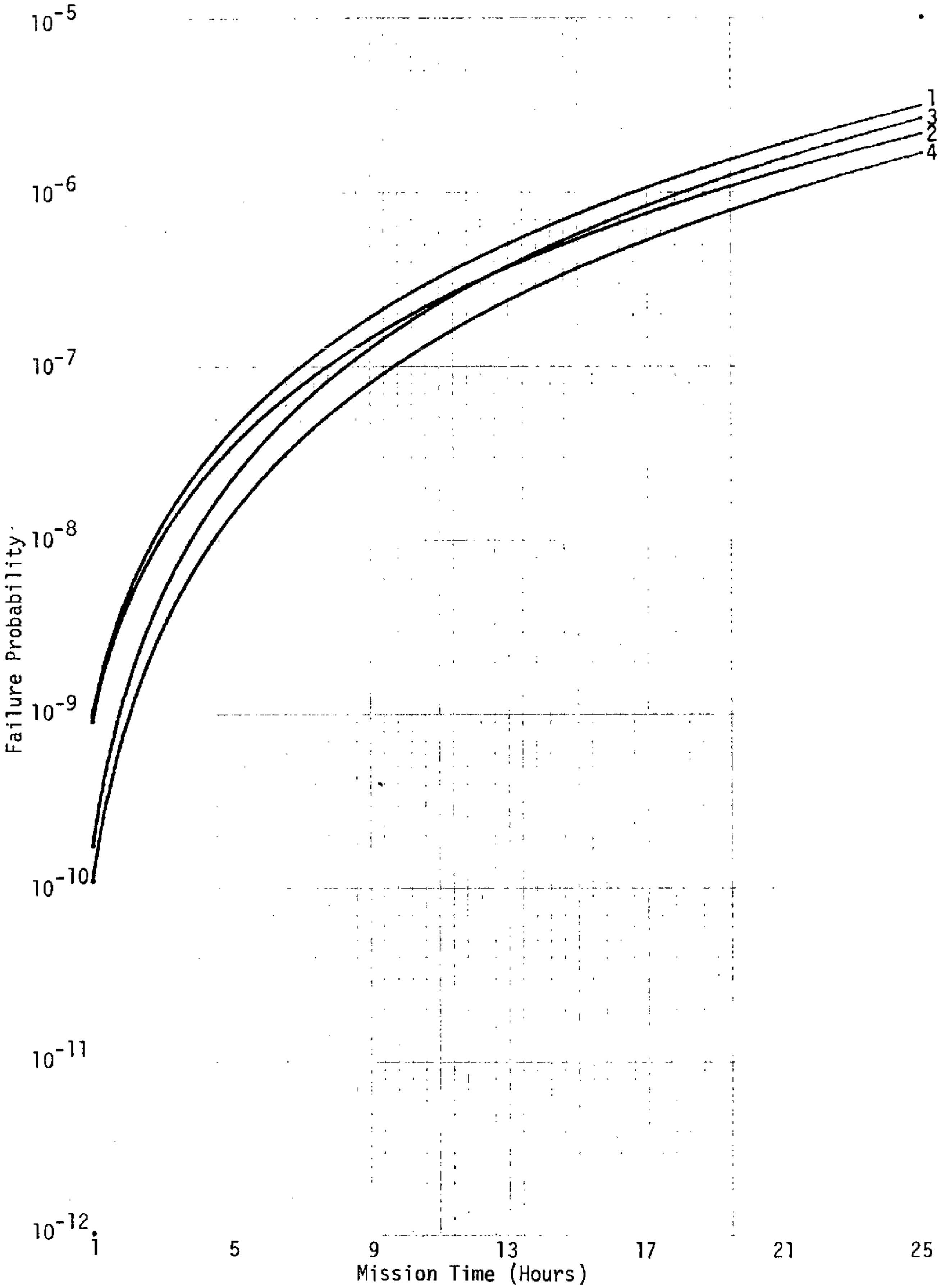


FIGURE 9.3-6 QUADRUPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR ADAPTIVE BUSSES

TABLE 9.3-III SUMMARY OF TRIPLEX CONFIGURATION ASSESSMENTS

NUMBER	CONFIGURATION TYPE	COMPUTERS	DEDICATION	EEMs	DEDICATION	BUSSES	10 HOUR FAILURE PROBABILITY
1	HASW	3	N/D	4	N/D or DED	3 or 4	$1.48(10)^{-5}$
2	HASW	3	N/D	3	N/D or DED	3	$1.86(10)^{-5}$
3	HASW	3	DED	3	N/D or DED	3	$2.05(10)^{-5}$
4	MHW	3	N/D	3	N/D or DED	3	$2.71(10)^{-5}$
5	MHW	3	DED	3	N/D or DED	3	$2.84(10)^{-5}$
6	MSW	3	N/D	-	-	3 or 4	$4.4(10)^{-5}$
7	MSW	3	DED	-	-	3	$4.5(10)^{-5}$
8	HASW Non/Adaptive	3	N/D	3	N/D	3	$1.3(10)^{-4}$
9	Duplex	2					$6(10)^{-4}$

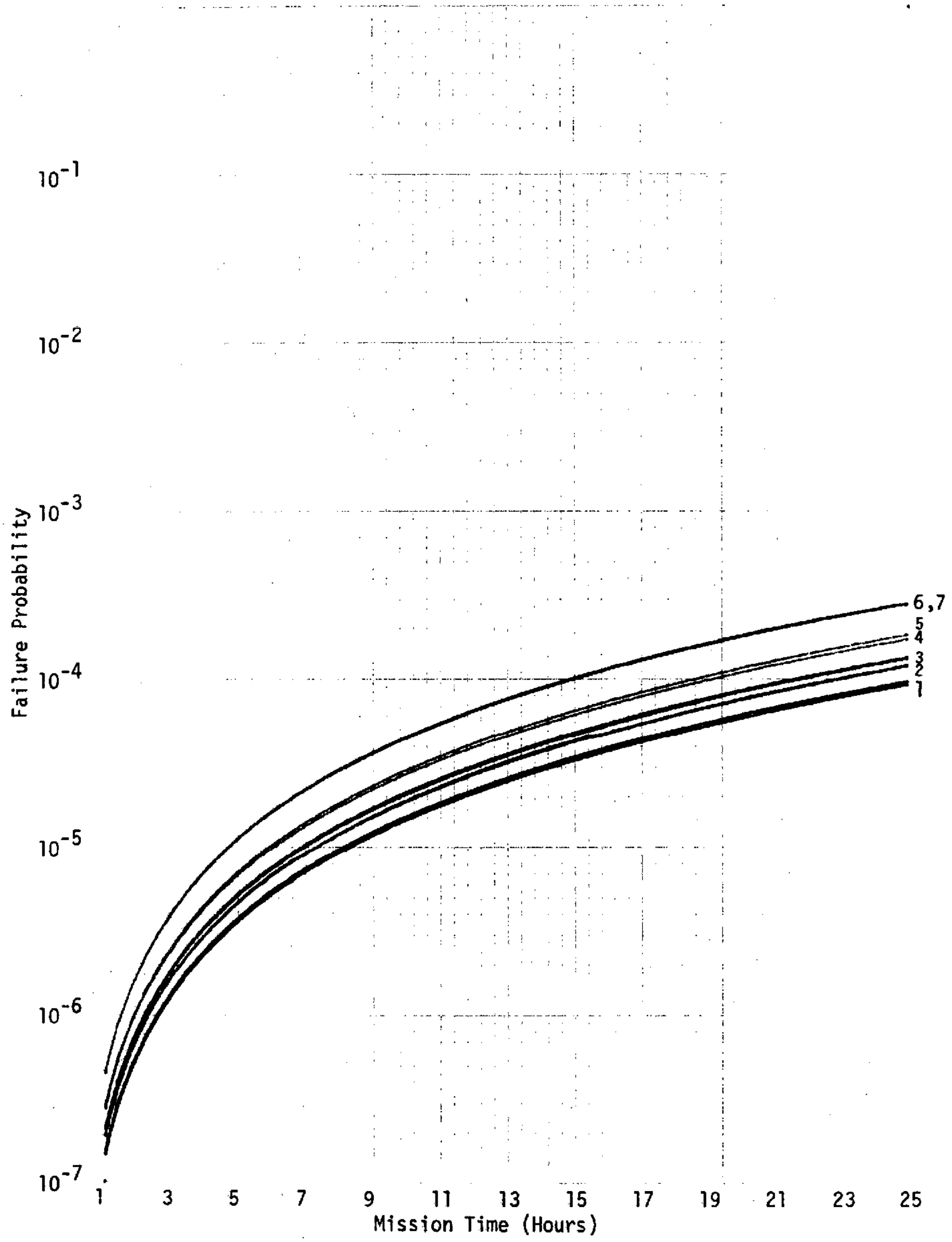


FIGURE 9.3-7 TRIPLEX FAILURE PROBABILITY VERSUS MISSION TIME FOR NON-ADAPTIVE BUSSES

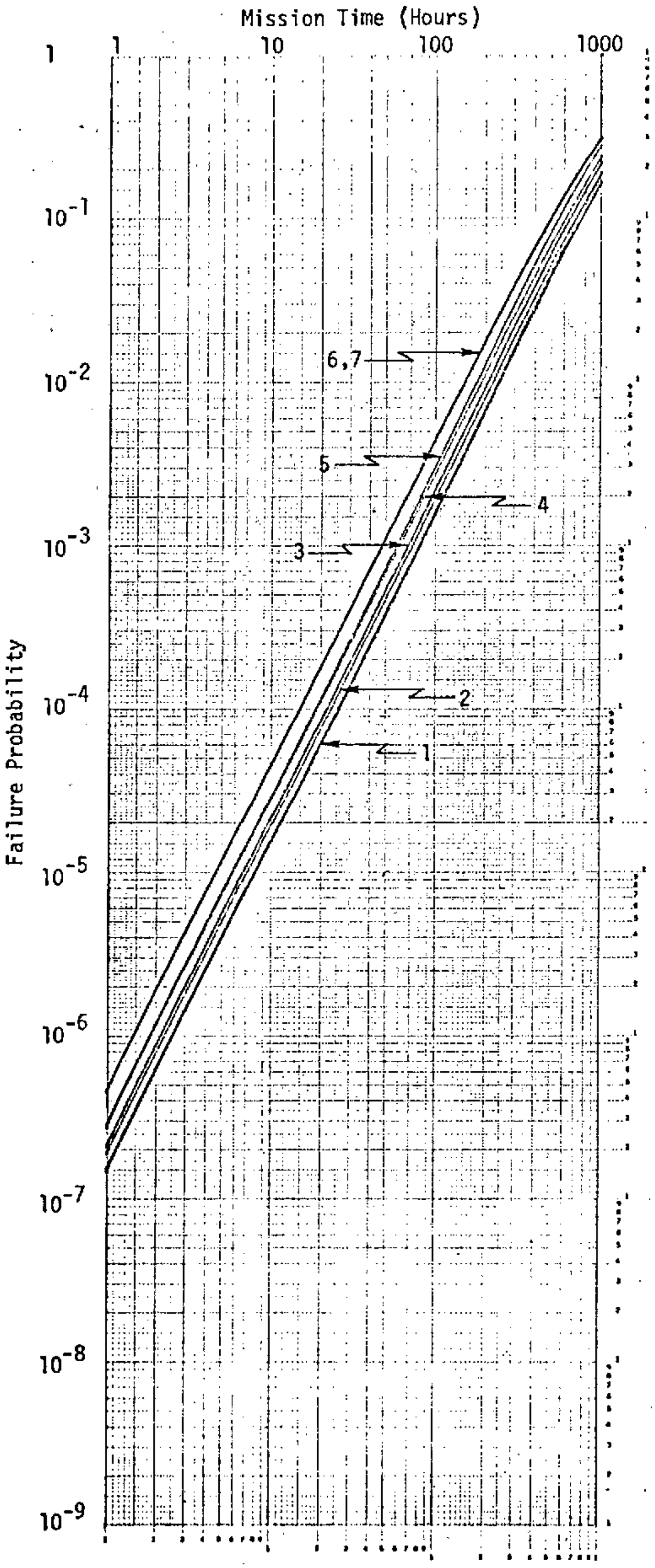


FIGURE 9.3-8

TRIPLEX FAILURE PROBABILITY VERSUS EXTENDED MISSION TIME FOR NON-ADAPTIVE BUSSES

In the triplex case, adding adaptive busses does not significantly decrease the failure probability, since most system failures are in the computers and EEMs.

9.3.2 Effect of Redundancy

Figure 9.3-9 shows that added redundancy yields a large improvement in failure probability. Here failure probability is shown versus extended mission time. The curves represent configurations as shown below:

1. Quintuplex with five EEMs and four busses.
2. Quintuplex with four EEMs and four busses.
3. Quadruplex with five EEMs and four busses.
4. Quadruplex with four EEMs and four busses.
5. Triplex with four EEMs and three busses.
6. Triplex with three EEMs and three busses.
7. TMR with three EEMs and three busses.
8. Duplex

Computers, EEMs, and busses are non-dedicated.

The results are summarized as shown below:

Configuration	10 Hour Failure Probability Improvement Ratio Over Simplex
Duplex	4.3
TMR	24
Triplex	160
Quadruplex	$2.4(10)^4$
Quintuplex (4EEM)	$1.2(10)^5$
Quintuplex (5EEM)	$1.7(10)^6$

It is interesting that each increment of redundancy gives about two orders of magnitude improvement in failure probability.

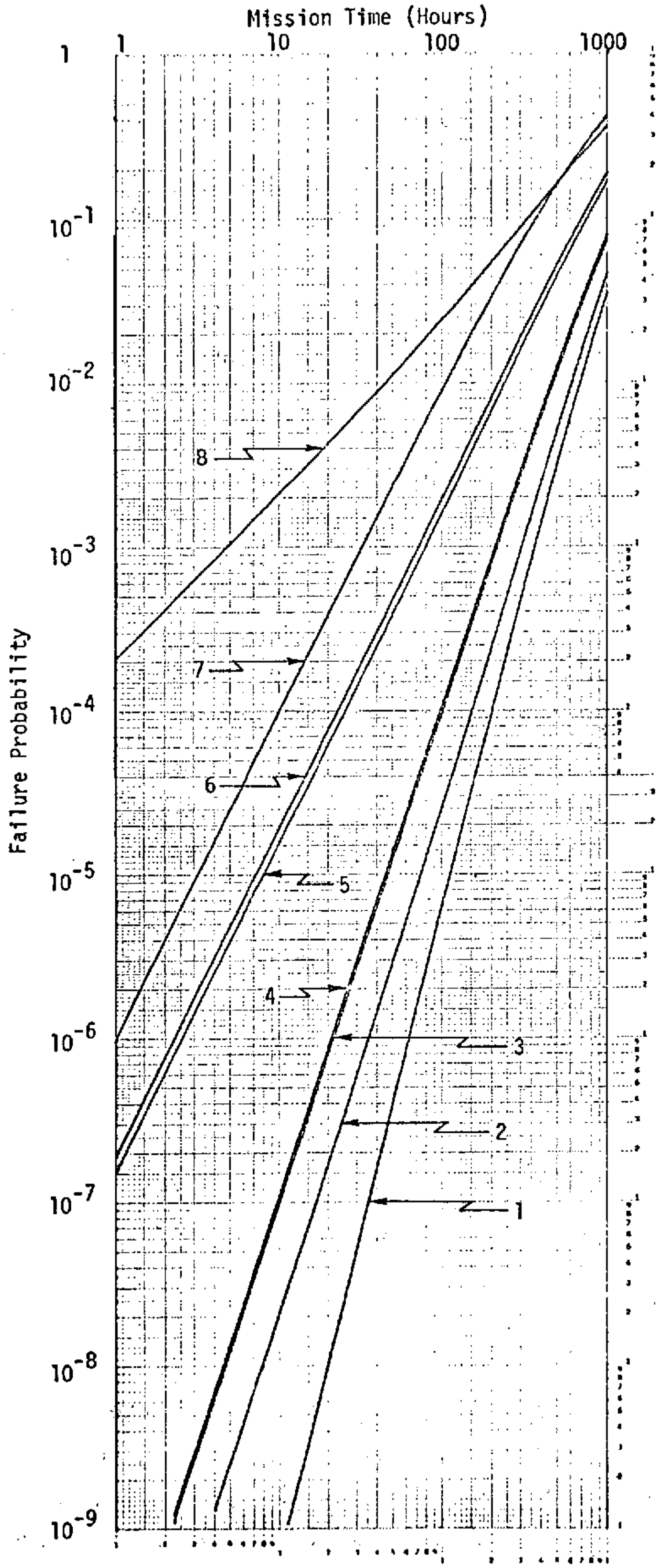


FIGURE 9.3-9

PROBABILITY VERSUS EXTENDED MISSION TIME FOR 5, 4, 3 AND 2 COMPUTERS

9.3.3 Effect of Non-Unity Recoverability

A recoverability (w_i) of 1 has been assumed in the generated results. Errors in recovery algorithms or single point failures could cause a recoverability of less than 1.

Figure 9.3-10 shows how recoverability can affect failure probability in a plot over an extended mission time. The curves are identified by numbers on the figure as follows:

1. Triplex with $w_i = .99$
2. Quadruplex with $w_i = .99$
3. Quintuplex with $w_i = .99$
4. Triplex with $w_i = .999$
5. Quadruplex with $w_i = .999$
6. Quintuplex with $w_i = .999$
7. Triplex with $w_i = .9999$
8. Quadruplex with $w_i = .9999$
9. Quintuplex with $w_i = .9999$
10. Triplex with $w_i = 1$
11. Quadruplex with $w_i = .99999$
12. Quintuplex with $w_i = .99999$
13. Quadruplex with $w_i = .999999$
14. Quintuplex with $w_i = .999999$
15. Quadruplex with $w_i = .9999999$
16. Quintuplex with $w_i = .9999999$

where $i = 3, 4, 5$. Here we exclude busses and dedicate the EEMs.

The figure shows that short mission times are affected more by non-unity recoverability. Also the survivability gained by added redundancy can be overshadowed by recoverability. When recoverability is the greatest contributor to failure, then redundancy becomes a disadvantage as can be seen from the initial mission hours of curves 1, 2 and 3.

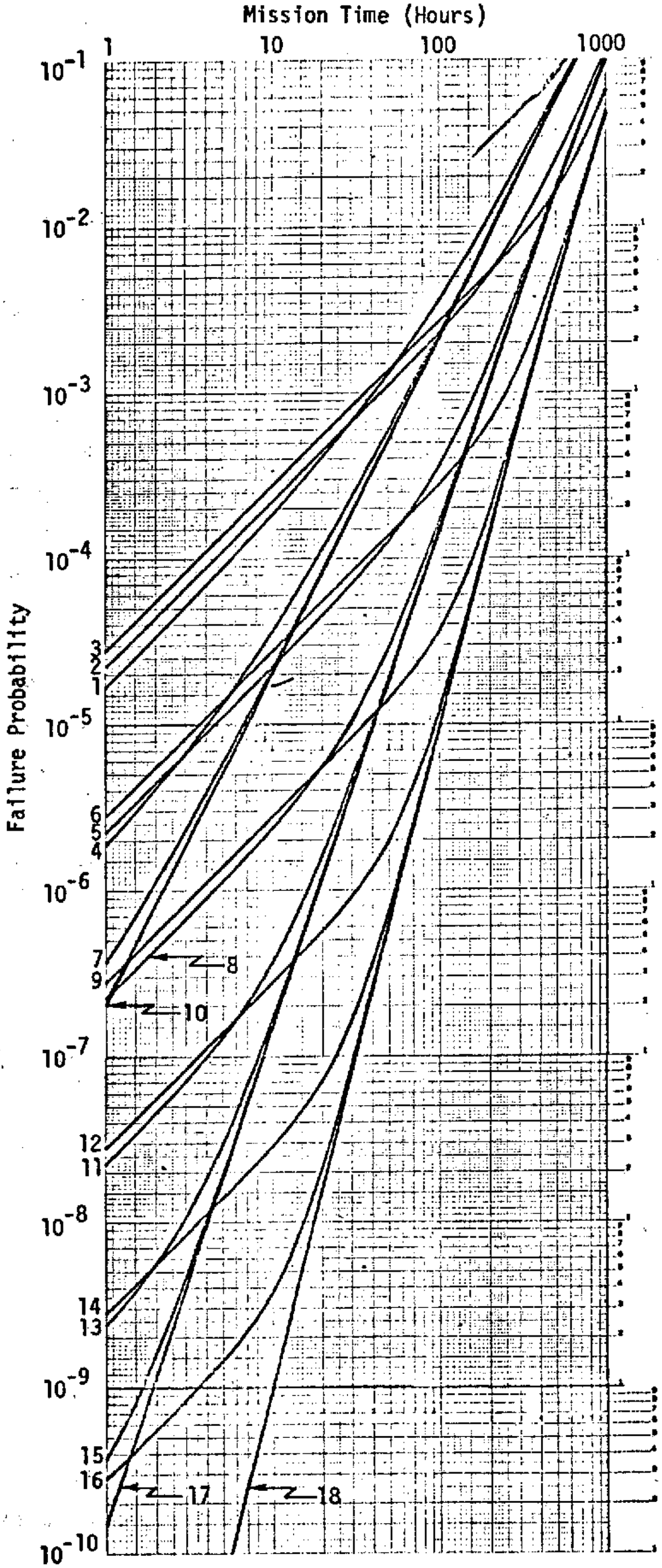


FIGURE 9.3-10

EFFECTS OF NON-UNITY RECOVERABILITY ON FAILURE PROBABILITY FOR EXTENDED MISSION TIMES

9.3.4 Effects of Adaptivity

Adaptive configurations can adjust to a new fault tolerant scheme when units have been recorded as faulty. More adaptable systems admit more faults than less adaptable systems, increasing their discrete fault tolerance. (Discrete fault tolerance also depends on the level of redundancy.) Adaptability allows an improvement in failure probability while non-adaptability causes a degradation at the same level of redundancy.

Table 9.3-IV summarizes the effects of adaptability. Quintuplex, quadruplex, and triplex imply adaptive cases where voting in the prime method of fault detection and diagnosis with three or more fault free computers. A residual duplex mode is entered when two computers remain. Two out of N and TMR configurations have no residual duplex, while QMR is a non-adaptive 3 out of 5 vote.

It is interesting that QMR has five computers and a greater failure probability than the four computer configurations. This is because it has a discrete fault tolerance no greater than quadruplex and more computers to have faults.

Figure 9.3-11 shows the effects of adaptivity on failure probability over extended mission times.

9.3.5 Effects of RETs

The method to study the RETs consists in identifying the input parameters which are affected by the introduction of each RET. Then simulation runs are made with parameters representing the presence and the absence of these RETs. When possible, an alternative method consists in computing the impact of modifying one parameter in one run, thus avoiding one simulation run. From these runs, we get different sets of parameters which are used for analytical modeling.

9.3.5.1 DRO Versus NDRO

Three cases of hardware aided software NMR have been studied. The three systems are identical except for the memory and one recovery algorithm. The first one has DRO memory and no memory copy. This is why many transients in multiplex cannot be corrected. The second one does include a memory copy.

TABLE 9.3-IV SUMMARY OF THE EFFECTS OF ADAPTABILITY

Configuration		10 Hour Failure Probability	Failure Probability Degradation Ratio (With Respect To The Adaptive Case)	Discrete Fault Tolerance
5 Computers	Quintuplex	$1(10)^{-9}$	1	4
	2 out of 5	$6(10)^{-9}$	6	3
	QMR (3 out of 5)	$1.5(10)^{-6}$	1500	2
4 Computers	Quadruplex	$1.4(10)^{-7}$	1	3
	2 out of 4	$7(10)^{-7}$	5	2
3 Computers	Triplex	$2(10)^{-5}$	1	2
	TMR	$1.3(1)^{-4}$	6.5	1

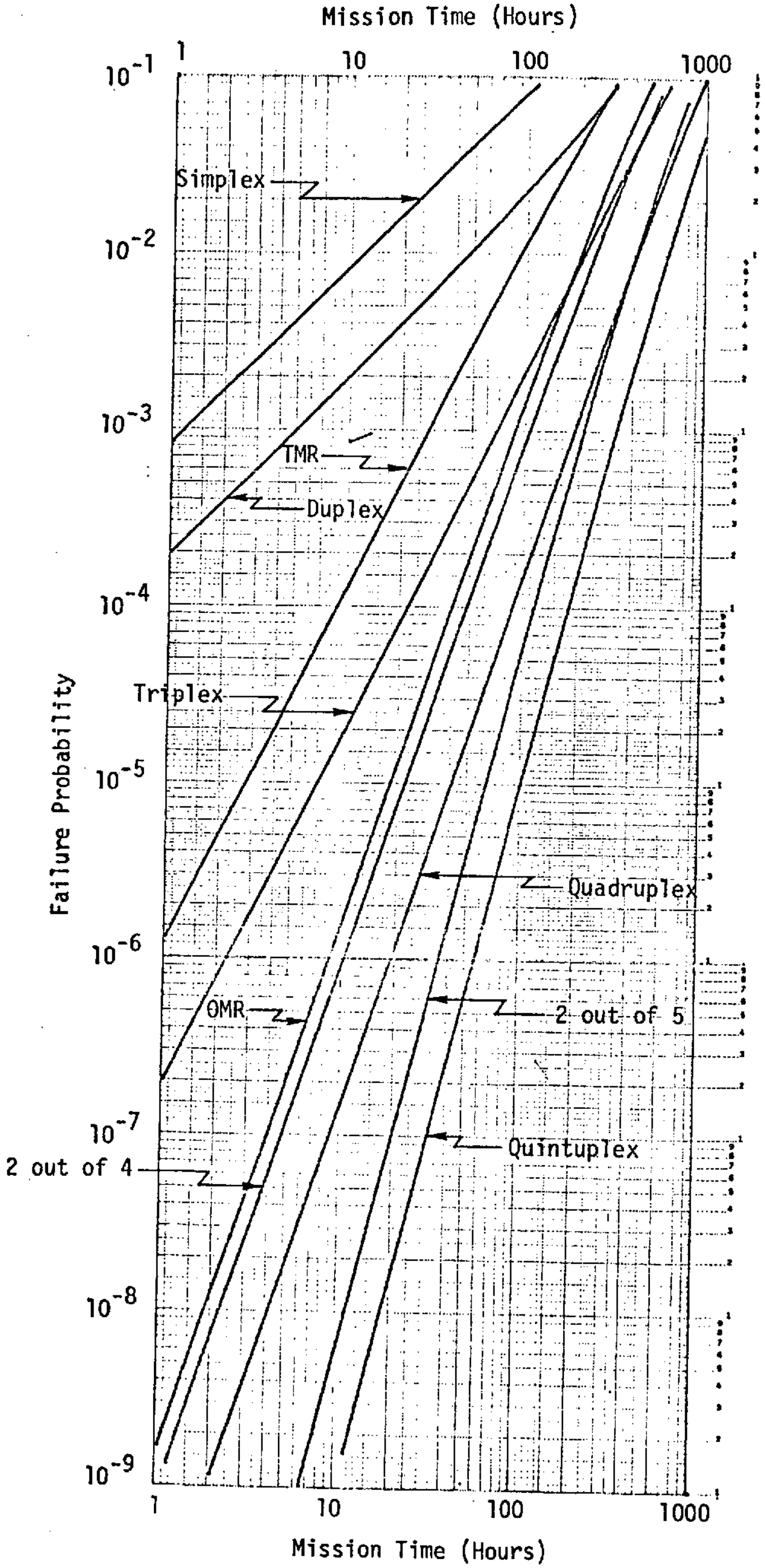


FIGURE 9.3-11

EFFECT OF ADAPTABILITY ON FAILURE PROBABILITY FOR EXTENDED MISSION TIMES

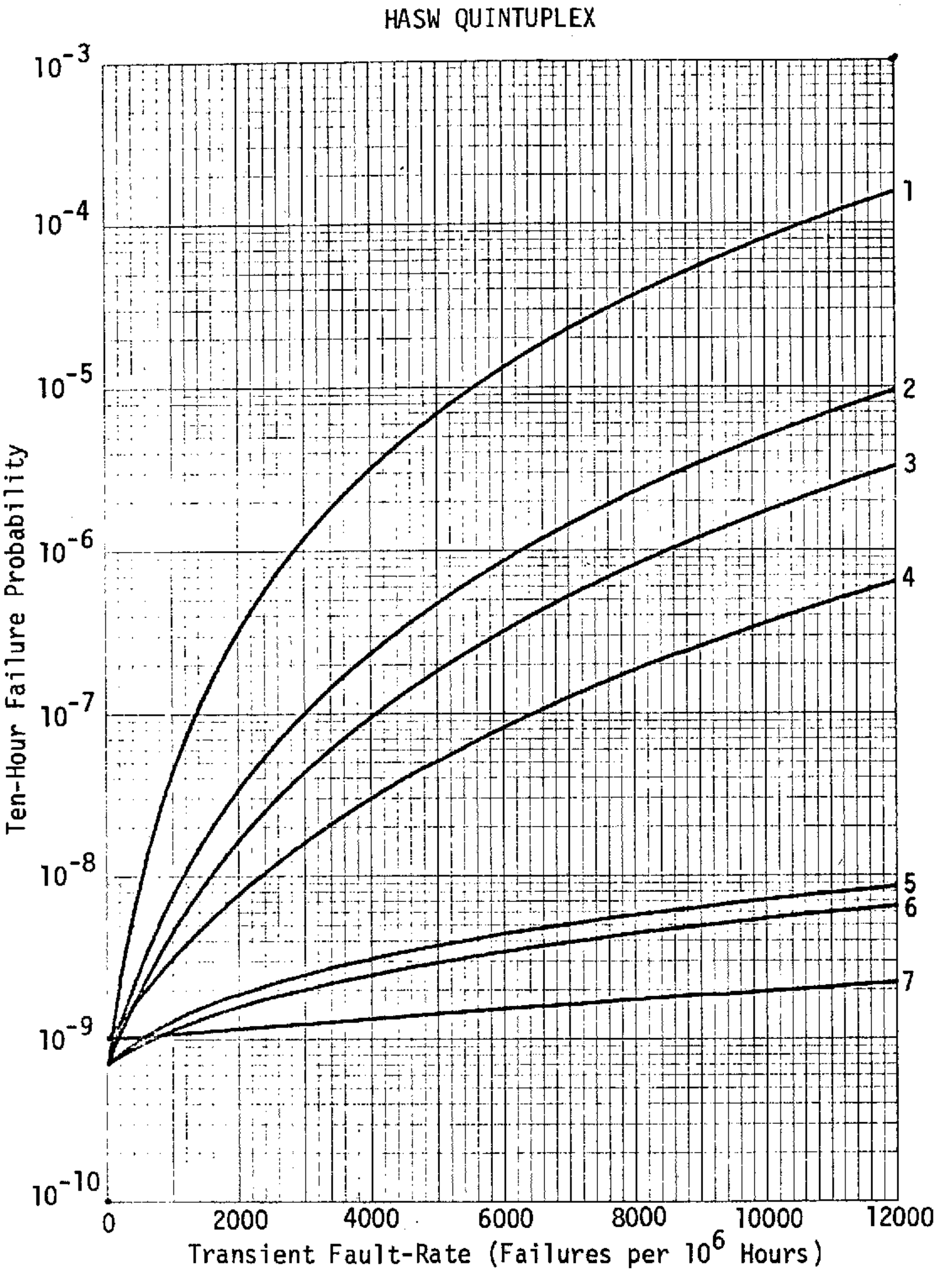
Most transients are corrected and the leakage is very low. Finally the third system has a NDRO memory, and of course, no memory copy. We assume that the technology involved in NDRO memories implies slightly higher failure rate and that transients changing a memory content accounts for 1% of all memory transient. Furthermore runs are made with 2 different values for the transient duration: 1 microsecond and 1 millisecond. Results of the simulation are indicated in Table 9.3-V.

Fault Duration	DRO (λ=250 10^{-6}) No Memory Copy			DRO (λ=250 10^{-6}) Memory Copy			NDRO (λ=300 10^{-6}) No Memory Copy		
	ℓ_1	ℓ_2	$\ell_3=\ell_4=\ell_5$	ℓ_1	ℓ_2	$\ell_3=\ell_4=\ell_5$	ℓ_1	ℓ_2	$\ell_3=\ell_4=\ell_5$
1μs	.87	.33	.33	.87	.33	10^{-4}	.51	10^{-2}	10^{-2}
1ms	.87	.45	.45	.87	.45	10^{-4}	.51	.20	.20

TABLE 9.3-V LEAKAGE COEFFICIENTS

It appears that using a NDRO improves dramatically the leakages in duplex and simplex. However it decreases slightly the leakage in multiplex (due to the absence of memory copy) and also increases the failure rate of the memory. Thus, we need using the analytical model to draw more precise conclusions.

Figures 9.3-12 - 9.3-14 illustrate the results obtained for duplex, triplex, quadruplex and quintuplex. These curves plot the 10-hour failure probability versus the transient fault rate for different types of memory and various recovery algorithms and fault durations. In this section we are interested only in the influence of the type of memory. We examine the curves whose last digits are either 6 or 7. It appears that an NDRO memory improves the system survivability in most of the cases. It does not if the transient rate is very low. This may also be verified by comparing curves 2 and 4. This



1. DRO; No Transient Coverage
2. DRO; No Memory Copy Transient Duration, 1 ms
3. DRO; No Memory Copy Transient Duration, 1 μs
4. NDRO; Transient Duration, 1 ms
5. DRO; Memory Copy Transient Duration, 1 ms
6. DRO; Memory Copy Transient Duration, 1 μs
7. NDRO; Transient Duration, 1 μs

For DRO memory, computer permanent fault-rate is 550 failures per million hours.

For NDRO memory, computer permanent fault-rate is 600 failures per million hours.

FIGURE 9.3-12 10-HOUR FAILURE PROBABILITY VERSUS TRANSIENT FAULT RATE

HASW QUINTUPLEX

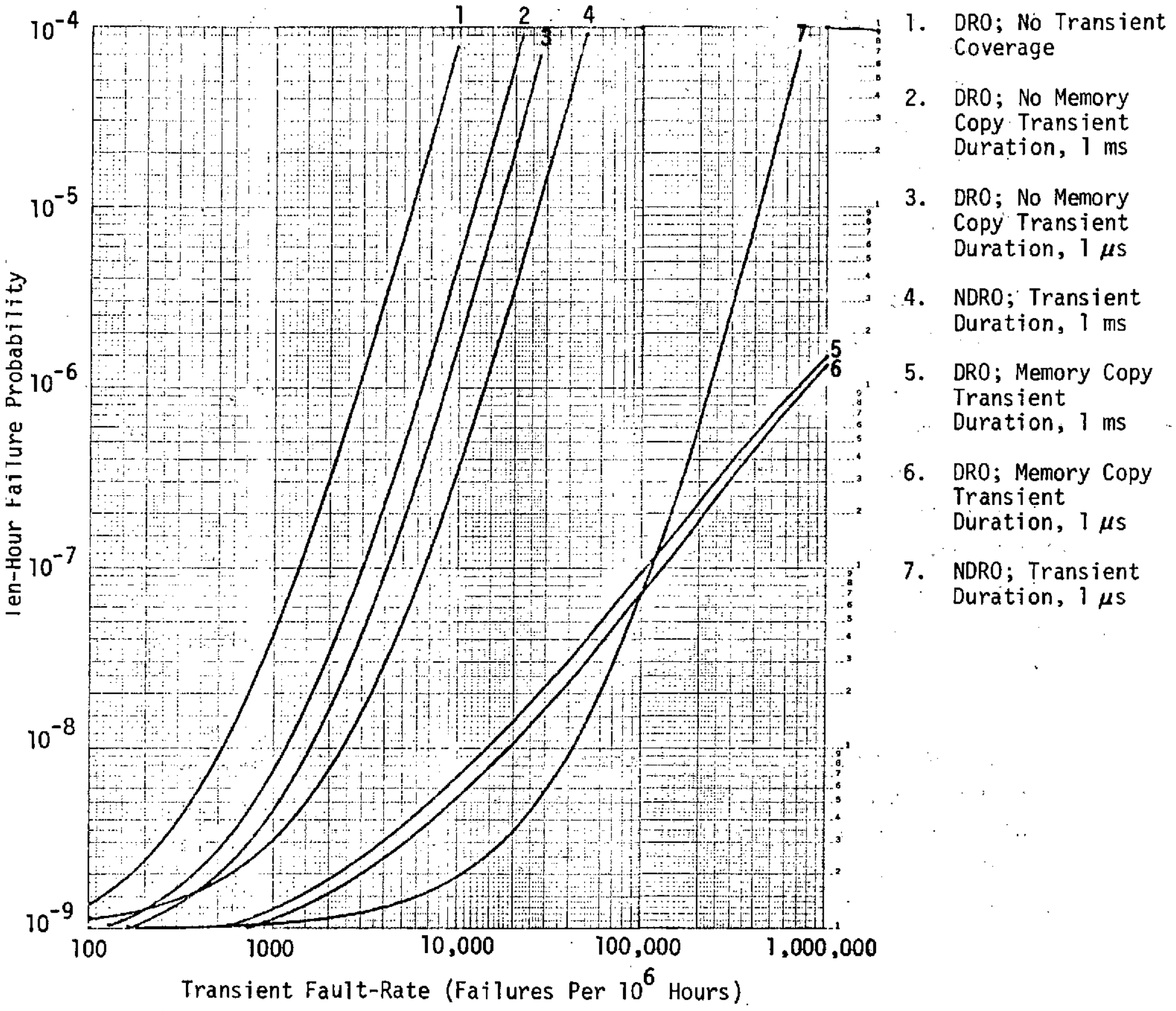


FIGURE 9.3-13 10-HOUR FAILURE PROBABILITY VERSUS TRANSIENT FAULT RATE

is due to the fact that the main advantage of NDRO is to enhance the probafility of recovery from a transient fault in duplex. It can also be noted that the improvement is more dramatic in triplex than in quintuplex because a triplex system has a higher probability to degrade to duplex. The other curves are described in Section 9.3.6.

9.3.5.2 Effects of BITE

Built-in-test equipments permit detection and isolation of a fault in a computer without using such devices as voters and comparators nor a diagnosis routine.

Even though it may decrease slightly the time between occurrence of a fault and its detection, it has no measurable effect in a multiplex system.

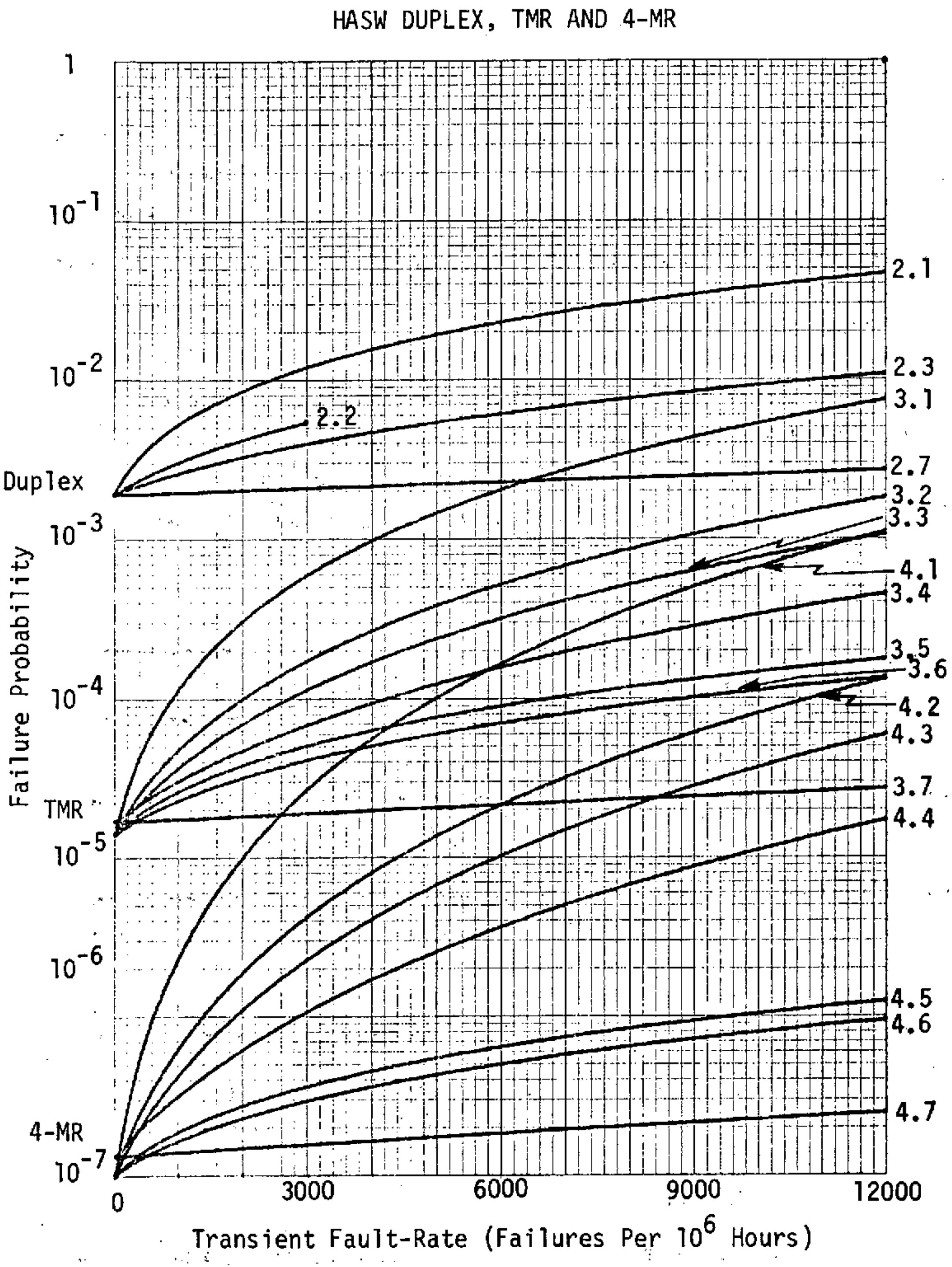
In duplex BITE are essential. They make possible to isolate many transient faults which could not be isolated through diagnosis, since the transient would have disappeared when the diagnosis is run (improvement of diagnostibility v_2).

In simplex BITE are also essential, since they provide the only way to detect transients (improvement of leakage ℓ_1).

Effects of BITE are given for two software adaptive TMR configurations. We see that improving duplex and simplex does improve an adaptive TMR. This improvement would be less significant with quadruplex and especially quintuplex since it is unlikely that such configurations degrades down to a simplex. Table 9.3-VI shows the improvements due to BITE for a software adaptive TMR. Similar results could be obtained with a hardware configuration.

	v_2	$1-\ell_1$	Failure Probability After 100 Hours
BITE	90%	8%	$19x10^{-4}$
No BITE	71%	0%	$55x10^{-4}$

TABLE 9.3-VI EFFECTS OF BITE



The first digit on each curve refers to the number of computers.

The second digit refers to implementation details and fault duration as explained below.

1. DRO; No Transient Coverage
2. DRO; No Memory Copy Transient Duration, 1 ms
3. DRO; No Memory Copy Transient Duration, 1 μs
4. NDRO; Transient Duration, 1 ms
5. DRO; Memory Copy Transient Duration, 1 ms
6. DRO; Memory Copy Transient Duration, 1 μs
7. NDRO; Transient Duration, 1 μs

FIGURE 9.3-14 10-HOUR FAILURE PROBABILITY VERSUS TRANSIENT FAULT RATE

9.3.5.3 Effects of Diagnostics

Diagnostics are not useful in multiplex since voting provides isolation of a fault. In duplex, they are essential since they provide isolation of a fault, once it has been detected through comparison. In simplex, they can be used if they are run periodically. However they will catch very few transients. Their only usefulness is that they can provide a warning that the system has failed.

Effects of diagnostics are assessed by simulating a configuration with diagnostics. Counting the number of times a diagnostic routine is called and dividing by 2 provides the number of failures due to the absence of diagnostics. Table 9.3-VII gives the results for two adaptive software N-plex configurations.

	Quadruplex	Triplex		
	F(100)	F(10)	F(100)	Diagnostibility v_2
Diagnostics	$18x10^{-5}$	$15x10^{-6}$	$19x10^{-4}$	90%
No Diagnostics (But BITE are present)	$56x10^{-5}$	$58x10^{-6}$	$44x10^{-4}$	61%

TABLE 9.3-VII FAILURE PROBABILITIES AFTER 10 AND 100 HOURS FOR QUADRUPLEX AND TRIPLEX WITH AND WITHOUT DIAGNOSTICS

9.3.5.4 Codes and I/O Wraparound

In order to include error correcting/detecting codes in the candidate computers, extensive hardware modification is required. This is not within the realm of "off-the-shelf computers" and will not be considered here. Single error detecting codes in memory (parity) are built into some of the candidate computers and are considered as a part of BITE. Codes and I/O wraparound checks are useful for error detection and fault masking in the I/O system.

Single error correcting/double error detecting codes allows one fault to occur before the hardware becomes inoperative. The reliability model to be used for the area covered by the code is

$$R = e^{-\lambda T} (1+c\lambda T)$$

where λ = failure rate of the bus and the code generating and decoding circuitry.

In the configurations we have assessed three or more redundant serial busses were postulated. Section 9.3.1 showed the effect of adding detecting codes and I/O wrap to the bus (allows adaptability). Error correcting codes on serial busses are not practical because bus faults will probably affect more than one bit.

To show the effect of error correcting codes on busses, we use a four-bit, byte-serial bus as a strawman. We compare a simplex error-correcting bus with a duplex error-detecting bus. The simplex bus requires 7 wires plus coding/decoding circuitry which yields a failure rate of 7 x 6 failures per 10^6 hours = 42. The duplex bus requires 4 wires per bus plus detection/diagnosis circuitry for a failure rate of 4 x 6 failures per 10^6 hours = 24 per bus.

Figure 9.3-15 shows coded simplex versus duplex for various values of coverage. Coded simplex is slightly better with non-unity coverage because of the lower total failure rate. A TMR bus is included with duplex coverage values of 0 and .9 with λ = 56.

9.3.5.5 Reasonableness Tests and Sensor Redundancy Management

Sensors may be either self-checking error-indicating or non self-checking. Multiple non-checking sensors require at least three to resolve a faulty sensor when comparison is used for error detection. Selection of the median sensor value is a good method of generating a common input for all computers. Median selection masks the effect of a sensor that has deviated by a large amount from the true value while averaging weighs the effect of all sensors equally. Any sensor that deviates more than a fixed amount from the median is indicated as faulty. Deferring final judgment on whether a sensor has permanently failed until fault indications appear two or more times in a row reduces the number of sensor transients recorded as permanent.

Reasonableness tests can provide a method of isolating a faulty sensor when one or two copies are in use. The tests check if the difference in successive sensor values do not exceed a specified limit. In some cases, a total system analysis is required to resolve a faulty sensor.

Truncation of least significant bits is not a good method of resolving sensor values. It is possible for two binary values to differ by one arithmetically and to have no bit positions that are equal.

In order to provide a feeling of the usefulness of the reasonableness tests, the following simulation runs have been made. The system is a software TMR (Table 9.2-I) where we include a sensor whose failure rate is 650 faults per mission hours (equal to the computer failure rate). In the first run we suppose there is no way of deciding which sensor is good if there are only two of them working and they disagree. In the second run, in 80% of the cases, the system is able to decide which sensor is good. Finally, we also compare with results obtained when sensors are supposed to be perfect. The results are listed in Table 9.3-VIII.

TABLE 9.3-VIII EFFECTS OF REASONABLENESS TESTS

	Failure Probability After 10 Hours
Sensor No Reasonableness Test	100×10^{-6}
Sensor Reasonableness Test	42×10^{-6}
Perfect Sensors	15×10^{-6}

A sensor may be dedicated to one input bus or be available to all busses. If sensors are dedicated to busses and busses to computers, then the loss of one computer causes the loss of all sensors on the bus associated with the computer. The less dedication there is, the better it is, as can

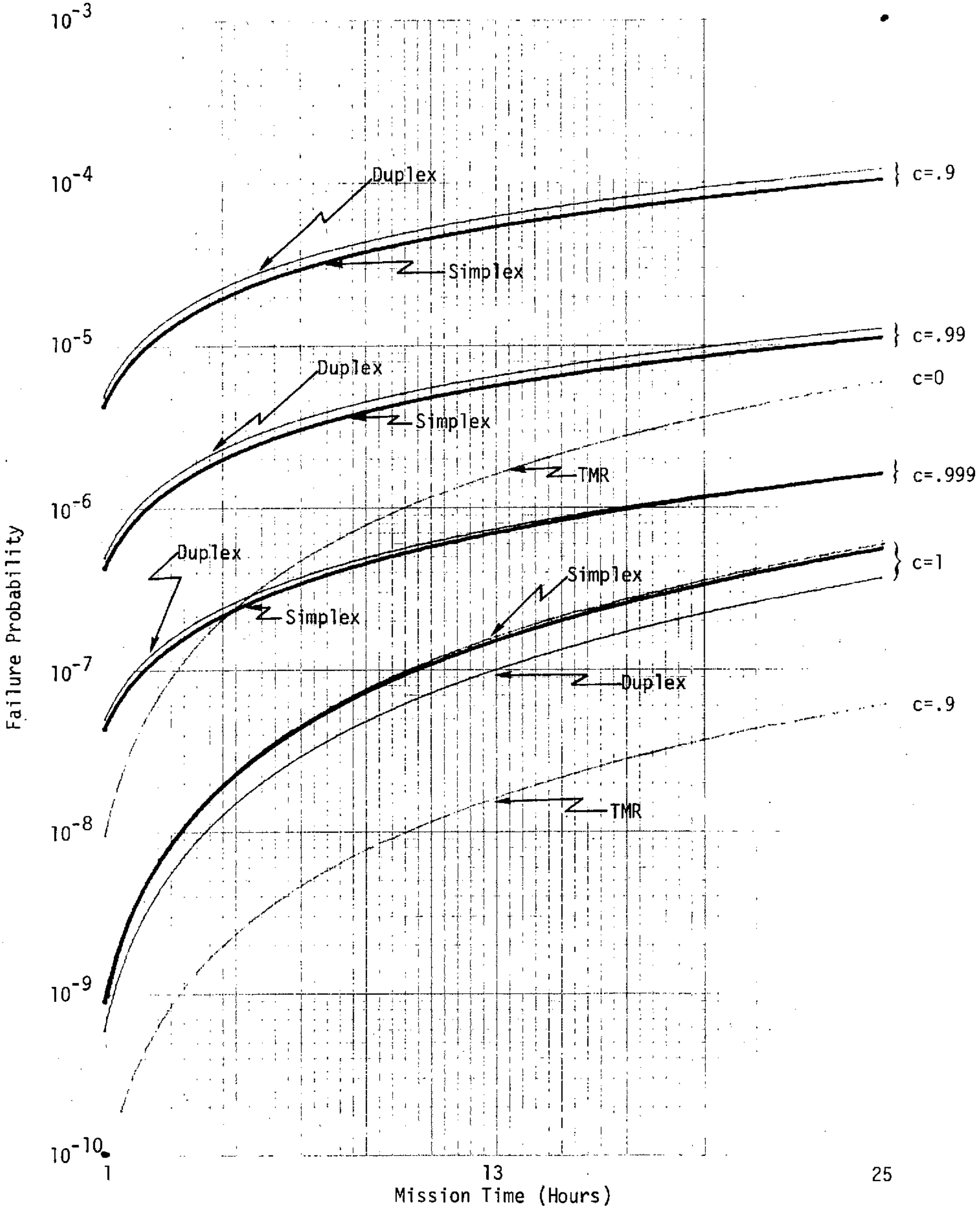


FIGURE 9.3-15 COMPARISON OF CODED SIMPLEX, DUPLEX AND TMR BUSSES

be seen from Table 9.3-IX. We have made two simulation runs. In the first run, sensors are dedicated to busses and busses to computers. In the second run, sensors are non dedicated. In both cases, reasonableness are present.

TABLE 9.3-IX EFFECTS OF NON-DEDICATED SENSORS

	Failure Probability After 10 Hours
Dedicated Sensors	94×10^{-6}
Non-Dedicated Sensors	42×10^{-6}
Perfect Sensors	15×10^{-6}

9.3.5.6 Voters, Adaptive Voters, and Comparators

Voters have the capability of polling the output of N elements and "voting" a consensus output when M elements agree ($N \geq M$). Adaptive voters differ from voters in that N and M may vary during a mission. Adaptive voters give rise to adaptive configurations while voters are used for "NMR" configurations. One type of voter is required for configurations of three or more computers in either a hardware or software implementation. A variation in the type of voter used implies a change of configuration. Voting is evaluated in Section 9.3.4.

Comparators are used in duplex configurations and may have either a hardware or software implementation. Hardware comparators may or may not have a self-checking feature. The self-checking feature signals the computers when the comparator fails and allows a switch to simplex.

If self-checking comparators fail, the duplex system will recognize this failure and degrade to simplex since error detection by comparison is no longer possible.

If non-self-checking comparators fail, an erroneous disgareement signal will be generated. A rollback will be attempted and will apparently fail. Diagnosis will be inconclusive, since no computer is faulty. The monitor will select a computer for simplex by a coin flip, and of course, will select a fault free one. Therefore, self-checking comparators only allow the system to recognize a comparator failure without going through the trauma of an inconclusive diagnosis.

9.3.5.7 Dedicated/Non-Dedicated I/O Units

The impact of dedicating the not dedicating I/O units is shown in Section 9.3.1. Several configurations where computers, EEM's, and I/O busses are and are not dedicated are evaluated. The results show that it is best to not dedicate computers to EEM's, but failure probability improvement is achieved by dedicating EEM's to busses.

9.3.5.8 Independent Hardware Monitor

The independent hardware monitor (IHM) may have one of three purposes:

1. As a laboratory tool to test system performance,
2. As a fault recorder to aid in ground maintenance, and
3. As an error checking device to signal a system error.

The first two purposes, although useful, are not in the realm of fault tolerance. It is important, however, that the system has no faults prior to a mission.

In fault-tolerant computer design, the designer identifies all the possible faults that can occur in the system and labels it the fault set. A fault-tolerant design then protects the system from faults within the fault set. Overlooked faults and hardware and software design errors that have not been uncovered during checkout can cause a system failure. These causes of system failure cannot be exactly quantified because they would be corrected if they are identified. For modeling purposes, the probability of such an event is taken to be much smaller than the overall failure probability during the mission.

The IHM can detect some system faults by reasonableness tests. The tests would verify that the difference between the present outputs and the previous outputs does not exceed a specified limit. The effectiveness of the tests depends on the application, and we estimate them to be between 25 and 75 percent effective. The IHM can also serve as a limit detector on the actuators.

If the failure is due to a software error, redundant algorithms could be used for critical programs. The only recourse from a failure of the alternate computations is a system restart.

9.3.6 Effects of Transients

In order to enhance the reliability of the system, transient recovery should be provided. If not, transient faults would have the same effect as permanent and would cause the loss of a computer. Furthermore, transients are very difficult to diagnose and thus lack of provision for transient recovery in duplex would decrease the diagnostibility.

Once transient recovery has been decided, the algorithms have still to be chosen. Here, it will be seen that the distribution and the duration of transients are important and should be known if best results are to be obtained.

9.3.6.1 Introduction of Transient Recovery

In Figures 9.3-12 - 9.3-14, curves whose last digit is a 1 represent the survivability of HASW system without any kind of transient recovery (except adaptability). It can readily be seen that introducing a transient recovery algorithm (even a bad one) always improves dramatically the survivability. This improvement is even more than it appears on these plots since the curves without transient recovery do not take into account the decrease in duplex diagnostibility.

9.3.6.2 Transient Recovery Algorithms

9.3.6.2.1 Duplex

The recovery procedure in duplex is the rollback. On Figure 9.3-14 curves 2.2 (interrupted by the crash of the system where the plotting was done), 2.3 and 2.7 illustrate improvement due to rollback. Curve 2.7 corresponds to a NDRO memory where rollback is very efficient since transients causing program memory damage are very rare.

9.3.6.2.2 Multiplex

For the cases with 3 or more computers, rollback is replaced by rollahead. Thus computation is not interrupted. Comparing, for example, curves 3.1 and 3.2 or 3.3 of Figure 9.3-14 shows the improvement due to introduction of recovery algorithm in an adaptive TMR. However, rollahead does not correct those transients which damage the memory. That is why the memory copy is introduced (curves 3.5 and 3.6). Another solution consists in replacing the DRO memory by a NDRO memory (curve 3.7).

9.3.6.3 Influence of Transient Duration

The plots of Figures 9.3-12 - 9.3-14 illustrate what happens when the average transient duration is 1 micro- and 1 millisecond. Results are always worse in the case of a long transient. This is due to the fact that the recovery begins when the transient is still active, thus causing the recovery not to be successful. In order to avoid that, a delay can be introduced between detection and start of recovery (see Section 5.4.7).

9.3.6.4 Influence of Bursts of Transients

Up to now, it was always assumed that transients arrived isolated in time, according to the transient fault rate. When transient faults arrive in bursts, it is supposed that during a short period (of the order of a second) many transients hit the same unit. This corresponds to a component who would work for a while at the limits of its tolerance specifications.

We have made one simulation run supposing that the memory of the computers could be hit by bursts. We suppose that there are 80 bursts per million hours. A burst in average lasts half a second and during a burst the transient fault rate is 5 per second. Thus, on the average, there are 200 transients per million hours in the memory. It can be seen that this is a mild burst environment. Other inputs were the same as for the software TMR of Table 9.2-I. No other transients hit the memories.

Results are intriguing: the system degrades to duplex 5 percent more often than without burst. This is due to the fact that many bursts are mistaken as permanents. The number of times a diagnostic is called is increased by 8 percent. Thus a non-adaptive TMR system in a burst environment would have an 8 percent larger system failure probability than the same system without bursts. However, the diagnostibility is improved by the bursts. This is due to the fact that a diagnostic routine is likely to return a fault indication. This increase in diagnostibility makes the adaptive TMR in a burst environment more reliable than in a Poisson environment.

A way to decrease the probability of mistaking a transient for a permanent would be to decrease the "Recurrence Interval." The recurrence interval (3s) is used in the following way: if a fault is redetected less than 3 seconds after its recovery attempt, it is assumed that the fault is permanent. Decreasing the recurrence interval would obviously help for burst. Decreasing the recurrence interval too much would cause the system to continue to attempt transient recovery on a permanent fault.

The wide variations in the results of this section show that a better knowledge of the transient environment is necessary in order to optimize transient recovery. A second conclusion is that NDRO is an excellent protection against transient damage.

9.3.7 Scheduling Effects

Fault recovery is not an instantaneous action. Thus, a fault recovery may cause the system to miss some iteration(s). If it is not catastrophic to miss a few consecutive iterations, then all properly designed systems will tolerate fault recovery without problems. However, if for the safety of the flight, it is dangerous to miss more than one iteration, then the scheduling may be an important factor in the survivability.

We have studied 3 cases of software TMR: 2 synchronous schedulings and one asynchronous. In all cases, we suppose that the basic iteration period is 30 ms and that the major cycle lasts 100 periods.

The first synchronous case corresponds to a light load and a fast comparison: the minor cycle lasts 5 ms and comparisons also take place every 5 ms.

The second case corresponds to a heavy load and a slow comparison: the minor cycle lasts 15 ms and comparison also takes place every 15 ms.

The third case is an asynchronous scheduling: major cycle tasks can be interrupted by minor cycle tasks which last an average of 5 ms. Comparisons also occur every 5 ms. The major difference between asynchronous and synchronous scheduling is that since a minor cycle task can interrupt a major cycle task, a fault may cause damage in more than one program and thus be detected more than once.

In all cases, it is assumed that a system failure occurs when 2 or more iterations are successively missed. Results are given in Table 9.3-X.

	Triplex Leakage ℓ_3	Duplex Leakage ℓ_2	Diagnostibility v_2	Failure Probability After 100 Hours
Synchronous Light Load	10^{-4}	21.5%	90%	19×10^{-4}
Synchronous Heavy Load	10^{-4}	21.5%	73%	41×10^{-4}
Asynchronous	10^{-3}	21.5%	87%	21×10^{-4}

TABLE 9.3-X EFFECTS OF SCHEDULING

A heavy load is not a problem in triplex (or 4-plex and 5-plex) since recovery does not interrupt significantly the normal flow of computation. However, in duplex the situation is quite different. Since comparisons take place every 15 ms, rollback duration is 15 ms long. If the rollback is successful, the mission is not endangered. Thus the leakage ℓ_2 does not vary. But if the rollback is unsuccessful, diagnostics have to be run to allow simplex operation. It happens rather often that there is not enough time to run these diagnostics. The diagnostibility decreases and the failure probability increases significantly.

Asynchronous scheduling may cause a few transients to be mistaken as permanents since they damage a few programs and are detected more than once. However, the increase of the leakage ℓ_3 is not having any significant consequences: it is as if the permanent fault rate was increased by one thousandth. The diagnostibility is slightly less than with the synchronous case because the asynchronous organization makes it slightly more likely to miss more than one iteration during the sequence rollback-diagnostics.

In conclusion, it appears that a heavy computational load is to be avoided. If it cannot be avoided, 5 plex and 4 plex are better than triplex since it is less likely to degrade to duplex with these systems. Asynchronous scheduling makes little difference from synchronous scheduling. This does not take into account the higher complexity of an asynchronous system which may cause some extra failures not taken into account by the simulation.

9.4 CONCLUSIONS

The conclusions reported below were obtained by use of CAST. They are based on a ten-hour flight and failure rates thought to be applicable to the off-the-shelf avionics computers studied. The reconfigurable computer systems were assumed to be composed of as many as five machines.

As shown in Figure 9.3-11, the greatest improvement in system survivability is obtained by increased redundancy. Each increment of redundancy decreases the 10-hour failure probability by approximately two orders of magnitude. The greatest failure probability decrease occurs when changing from triplex to quadruplex, e.g., a 200-fold improvement. Increasing redundancy also increases cost in terms of power, weight, and volume not only due to the added units but due also to the increased complexity of intercommunications modules, external electronics modules, and bus switches.

Increasing redundancy has diminishing returns if there are errors in permanent-recovery algorithm design. This error penalty becomes more severe with added redundancy as was shown in Section 9.3.3. Using simpler recovery algorithms, i.e., those involving less RCS adaptivity, is a possible way of ensuring error-free recovery. However, the increase in failure probability for air-transport-type missions due to decreased adaptivity (e.g., not adapting the system down to one computer is less than that caused by decreased redundancy or recoverability.

Since redundancy has such a large effect on failure probability, external hardware should have an equivalent redundancy to prevent external failures from depressing the overall survivability.

The techniques reported here devote much attention to the modeling of transient faults. The results show that a knowledge of the transient environment results in effective transient recovery features. Underestimating transient duration results in many transients being recorded as permanent, while overestimating transient duration leaves the system unduly vulnerable to further faults.

Finally, subject to the qualifications and assumptions described in the first paragraph of this subsection, configuration assessment has shown that hardware-aided software configurations provide a lower probability of failure than mostly-hardware or mostly-software configurations.

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

AVIZ 72 Avizienis, A., "The Methodology of Fault-Tolerant Computing," Proc. First USA-Japan Computer Conference, 1972.

BOUR 69 Bouricius, W.G., et al., "Reliability Modeling Techniques For Self-Repairing Computer Systems," Proc. ACM 1969 Ann. Conf.

BOUR 71 Bouricius, W.G., et.al., "Reliability Modeling for Fault-Tolerant Computers", IEEE Transactions on Computers, Vol. C-20, No. 11, November 1971.

DAVE 58 Davenport, W.B., and Root, W.L., *Introduction to Random Signals and Noise,* McGraw-Hill, New York, 1958.

DENN 67 Dennery and Krzywiki, *Mathematics for Physicists*, Harper, 1967.

HILL 70 F.S. Hillier, G.J. Lieberman, *Introduction to Operations Research*, pp. 447-450, Holden-Day, Inc., San Francisco, 1970.

KRUU 63 Kruus, J., "Upper Bounds for the Mean Life of Self-Repairing Systems," Report R-172 Coordinated Science Laboratory, University of Illinois, July 1963.

LIPS 68 Lipschutz, *Linear Algebra*, Schaum Outline (McGraw-Hill), 1968.

LYON 62 Lyons & Vanderkulk, "The Use of TMR to Improve Computer Reliability," IBM Journal, April 1962.

PARZ 60 Parzen, E., *Modern Probability Theory and Its Applications*, pp. 251-263, Wiley & Sons, 1960.

RATN 73 Ratner, R.S., et.al., "Design of a Fault-Tolerant Airborne Digital Computer, Volume II - Computational Requirements and Technology," NASA Contract NAS1-10920, Stanford Research Institute, October, 1973.

ROHR 73 Rohr, John A., "System Software for a Fault-Tolerant Digital Computer," Ph.D. Thesis, University of Illinois, 1973.

SHRE 66 Shreider, Y.A., *The Monte Carlo Method*, Pergamon Press, New York, 1966.

TSOU 73 Tsou, Ed., Daly, H.S., Swearingen, C.N., "Highly Reliable Processor System for Space Application," AIAA Computer Network Systems Conference, Huntsville, Alabama, April, 1973.

ULTR 74 "Fault-Tolerant Avionics Systems Architectures Study," Ultrasystems, Inc., April, 1974, AFAL Contract F33615-73-C-1163.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendices A, B, and C contain proprietary data from various computer manufacturers. Thus these appendices have been distributed only to Government representatives at Langley Research Center.