Order Number 9513392

# Design and evaluation of real-time fault-tolerant control systems

Kim, Hagbae, Ph.D.

The University of Michigan, 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

# DESIGN AND EVALUATION OF REAL-TIME FAULT-TOLERANT CONTROL SYSTEMS

by

**Hagbae Kim**

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in The University of Michigan
1994

Doctoral Committee:
Professor Kang G. Shin, Chair
Professor Semyon M. Meerkov
Professor Yoram Koren
Assistant Professor Farnam Jahanian

To My Late Grandmother, My Parents, and Hyunjung

# ACKNOWLEDGEMENTS

I am most grateful to my advisor, Professor Kang G. Shin, for his guidance and constant encouragement throughout my graduate school years, which made my Ph.D. process so much enjoyable and will go on leading me in the future. He introduced the field of real-time fault-tolerant control systems to me four years ago, and has always been eager to discuss the dissertation topics with invaluable technical and professional advice.

Many thanks also go to my committee members for their important comments and suggestions. In particular, Professor Meerkov helped me successfully finish my first directed study.

This dissertation work has been supported in part by the NASA under Grant NAG-1-1120 and by the Office of Naval Research under Grant N00014-91-J-1115. I would also like to note that the work in the fault-tolerant latency of Chapter 4 and in the reconfiguration latency of Chapter 5 has benefited from the quantitative data provided by Chuck Roark and his colleagues of the TI Microelectronic Center, Plano, Texas.

I thank all fellow students of the Real-Time Computing Laboratory and especially my officemates for their friendship and insightful discussions. I also want to thank many Korean colleagues on the campus by whom my life in Ann Arbor could be decorated with many unforgettable memories.

Although they are in the distance across the Pacific, my parents, brother, and sisters have bestowed a constant love and encouragement on me. This thesis could not have been finished without their support. Last but not least, I would like to express my gratitude and love to my wife, Hyunjung.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF APPENDICES

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation

The use of digital computers for such applications as aircraft, automated factories and nuclear reactors, has become commonplace due to the availability of inexpensive, powerful computers and the increasing need to control more sophisticated processes.

A traditional control system consists of simple components, such as an electrical, mechanical, or hydraulic hard-wired controller, in the feedback loop of the controlled process. These controllers have the virtue of simplicity in designing and implementing them, and evaluating their performances. By contrast, recent and future developments — which are likely to be intrinsically unstable to achieve other purposes — such as a highly fuel-efficient "fly-by-wire" aircraft demand the controllers to react fast enough, for example, to maintain stability. Digital computers are thus playing a more important role in maintaining safety-critical functions while improving control system performance significantly.

Computers for controlling life-threatening processes must be highly reliable to avoid any catastrophe. For example, the computer used in commercial aircraft requires a failure probability lower than $10^{-10}$ per hour. To reduce the probability of failure, early systems focused on fault-avoidance through rigorous quality control, where component-failure rates were reduced by careful design and manufacturing. But this method alone cannot always meet system design goals because of economic and technical problems. To guarantee the safe operation of these processes in the presence of computer-component failure(s), some form of fault-tolerance should be provided with the controller computer by using two kinds of redundancy: space and time. Fault-tolerant controller computers are far more complex and difficult to design and validate by resolving such issues as control-task scheduling and allocation, optimal queue control, and optimal redundancy management than non-fault-

tolerant counterparts due to the stringent reliability required.

In a real-time control system, the environment (or controlled process) in which a controller computer operates is an active component of the system, and a system failure may occur not only because of the slow execution of critical tasks but also because of massive hardware or software component failures. The design and evaluation of a fault-tolerant controller computer in a highly-reliable real-time system is even more complicated due to (i) its interaction with the controlled process (the operating environment) and the difficulty in expressing the needs of the controlled process and (ii) the difficulty in predicting the execution time of control tasks that subsumes the time required to recover from an error in case of component failure(s). One has to determine the strict timing requirements of the controlled process, and specify and evaluate the controller-computer's performance in the context of the controlled process.

Thus, derivation of the timing information from a controlled process is essential to the design of a fault-tolerant controller computer. The type and degree of redundancy should be determined to minimize a certain cost based on this information. In other words, the need of reliable and safe operation of the controlled process synergistically interacting with the controller computer must be formally specified in a form that is understandable to the controller computer. This specification captures the interplay between a controlled process and its controller computer.

## 1.2 Research Objectives

In order to formally specify the timing information about a controlled process, we must analyze the controlled process. Specifically, we will derive the hard deadline of a real-time system, which represents system inertia/resilience against a *dynamic failure*. The deadline will henceforth be called the *Control System Deadline* (CSD). The computation time delay or erroneous input disturbance due to controller-computer failures or environmental disruption such as an ElectroMagnetic Interference (EMI) may lead to a dynamic failure by either violating the necessary conditions for system stability or causing the system to leave the region of the state space defined by the system specification (i.e., the *allowed state space*), if the active duration of this abnormality exceeds the CSD. The control system deadline of a critical-control task was usually assumed to be given *a priori*. This presupposed the existence of a precise definition of the hard deadline and a method to derive it, which, however,

have not been addressed in detail. We thus propose a heuristic algorithm to iteratively compute the CSD as a function of the system state and time by examining system stability or testing the given (or derived) constraints of control inputs and states.

To provide information meaningful to the design and evaluation of fault-tolerance strategies in a controller computer, we evaluate the *Fault-Tolerance Latency* (FTL), defined as the cumulative time taken by all sequential steps to recover from an error or failure, while considering various fault-tolerance mechanisms. To meet timing constraints or avoid a dynamic failure, the latency of any fault-handling policy — which consists of several stages such as error detection, fault location and recovery — must not be larger than the *Application Required Latency* (ARL), which depends upon the controlled process under consideration and its operating environment. Note that one can derive the ARL using the hard-deadline information, because the CSD is equal to the sum of ARL and the minimum time to executethe remaining control task to generate a correct control input.

We also pay special attention to the *reconfiguration latency*, defined as the time taken for the whole reconfiguration process, because reconfiguration is generally the most time-consuming stage of a fault- or error-handling process and thus makes the greatest impact upon the FTL.

Using the timing information derived from a real-time control system, we develop a cost-effective design strategy for a fault-tolerant controller computer by trading off between space and time redundancy. Among many fault-tolerance schemes, we have optimized the design of a *Triple Modular Redundant* (TMR) controller computer by applying time redundancy based on the information of the control system deadline, because the TMR structure has been one of the most popular and practical fault-tolerance schemes using spatial redundancy. Since more than 90% of faults are known to be transient [66] (especially external faults, because disruptive environmental conditions are temporary and may cause functional error modes without actual component damage [5, 29]) and system reconfiguration is expensive both in time and hardware [31], retry or re-execution on the same hardware without any reconfiguration is effective in recovering from a TMR failure[1] or a masked error.

First, we propose an "optimal" instruction-retry policy minimizing the probability of control-system failure that may occur due to either multiple consecutive TMR failures whose inter-arrival times exceed the CSD or the exhaustion of spares as a result of frequent system reconfigurations, upon detection of a masked (by the TMR) error as well as a TMR failure in a controller computer.

[1] Failure to establish a majority among the processing modules in TMR system.

Secondly, we propose an adaptive recovery method for TMR failures by "optimally" choosing either Re-execution of the task on the Same HardWare (RSHW) or Replace the faulty HardWare, reload, and Restart (RHWR) based on an estimated cost. The cost, the mean execution time of a given task, is computed by fault-state likelihoods, which are updated with the voting results using the Bayes theorem.

The TMR structure in masking the effects of a single faulty module is effective only under the assumption that multiple faults are unlikely to occur (near) simultaneously in different modules. Note that real-time control systems often perform safety-critical missions under harsh environments with EMI such as lightning, HIRF (high intensity radio frequency fields), or NEMP (nuclear electromagnetic pulses), and the controller computers using spatial redundancy usually share the same clock and power sources (and also the same operating environment). In this case, the effects of near-coincident faults that will disable the TMR's masking capability can no longer be ignored [32]. We therefore propose a method to eliminate or alleviate the effects of (near) coincident faults by sequencing tasks on different modules so as to maximize the number of tasks that do not suffer from TMR failures.

In summary, the objectives of this dissertation are to:

- Derive the control system deadlines of controlled processes,
- Evaluate the fault-tolerance latency of controller computers and the reconfiguration latency,
- Design an optimal instruction-retry policy for TMR controller computers,
- Propose an adaptive recovery method using task re-execution in TMR controller computers, and
- Develop an optimal task-sequencing strategy in TMR controller computers.

## 1.3 Outline of the Dissertation

The dissertation is organized as follows. In Chapter 2, we describe basic concepts and terminology in the analysis and design of real-time fault-tolerant control systems. We also provide background information both on the derivation of the CSD and the FTL and on the optimal design of fault-tolerant computers.

Chapter 3 presents a method for deriving the CSD from the dynamic characteristics of the controlled process, their fault behaviors, and the properties of the control algorithms

employed. In this chapter, we compute the CSD by iteratively examining the necessary conditions for (asymptotic) system stability or the residence of system states in the allowed state space with the equations of dynamics modified to account for the effects of controller-computer failures. We also demonstrate the application of the CSD information with a design example that optimizes time-redundancy recovery methods such as retry or rollback and an evaluation example that assesses the system reliability.

In Chapter 4, we evaluate the FTL while considering various fault-tolerance mechanisms and use the evaluated FTL to check if a fault-handling policy can meet the timing constraint, FTL $\leq$ ARL, for a given real-time application. We investigate all possible fault-handling scenarios and express FTL with several random and deterministic variables that model the fault behaviors and/or the capability and performance of fault-handling mechanisms. We also present a simple example to demonstrate the application of the evaluated FTL in real-time systems, where an appropriate fault-handling policy is selected to meet the timing requirement with the minimum degree of spatial redundancy.

Chapter 5 focuses on the evaluation of the reconfiguration latency, which is generally the largest portion of the FTL. To do this, we first classify the reconfiguration techniques into four types: reconfigurable duplication, reconfigurable N-Modular Redundancy(NMR), backup sparing, and graceful degradation. For each type of reconfiguration, we analytically evaluate the reconfiguration latency, defining several parameters to describe the task size, the individual processor capabilities, and the underlying reconfiguration strategy and system architecture.

In Chapter 6, we propose an optimal instruction-retry policy in a TMR controller computer. An "optimal" instruction-retry period is derived here by minimizing the probability of dynamic failure upon detection of a masked (by the TMR) error as well as a TMR failure. We also derive the minimum number of spares needed to reduce the probability of dynamic failure below a pre-specified level for a given mission by using the derived optimal retry period.

Chapter 7 develops an adaptive recovery method for TMR failures by optimally choosing either RSHW or RHWR based on the estimation of the costs involved. We derive and compare the expected costs of RSHW with that of RHWR by using the likelihoods of all possible states in the TMR system, which are updated by the Bayes theorem with each voting result. In this chapter, we also present our simulation results to show that the proposed method outperforms the conventional reconfiguration method using only RHWR under various conditions.

In Chapter 8, we derive the probability mass function of the number of TMR failures for a *random sequence* as well as the *conventional sequence* of tasks on the three modules. (In the conventional task sequencing, all three copies of a task are executed at the same time on the three modules.) Then, we develop an *optimal sequence* of tasks by minimizing the mean number of TMR failures. This chapter illustrates several examples showing significant improvements in reducing TMR failures with the optimal task sequencing.

The dissertation concludes with Chapter 9, which summarizes the contributions of this dissertation and discusses future directions.

# CHAPTER 2

# PRELIMINARIES

In this chapter, we describe the attributes characterizing a real-time control system with the basic concepts of real-time computing and fault-tolerance. We also give a summary of the background involving extensive literature in the area of analysis and design of real-time control systems and our approach for : (i) the derivation of the CSD, (ii) the evaluation of the FTL, and (iii) the optimal design of fault-tolerant computers, especially using the TMR structure.

## 2.1 Real-Time Control Systems

Digital computers are commonly used in real-time control systems due mainly to their improved performance and reliability in dealing with increasingly complex controlled processes. A digital computer, called the *controller computer*, in the feedback loop of such a control system calculates the control input by executing a sequence of instructions, thereby introducing an unavoidable delay — called the computation-time delay — to the *controlled process*. This calculation introduces an extra delay in addition to the system delay commonly seen in the control literature. Computation-time is an important component of the delay in the feedback loop, which also includes contributions related to measurement or sensing, A/D and D/A conversion, and actuation. However, these other delay contributions are usually constant, and thus easy to model. Due to data-dependent loops and conditional branches, and contentions in resource sharing during the execution of programs that implement control algorithms, the computation-time delay is a continuous random variable which is usually much smaller than one sampling period, $T_s$, if no failure occurs to the controller computer.

When a component failure or environmental disruption, like EMI, occurs, the time spent

for error detection, fault location, and recovery (i.e., the FTL) must be added to the execution time of a control program, thus increasing the computation-time delay significantly. This increase in task execution time seriously degrades the system performance and may even lead to a dynamic failure.

Some performance functional may be available to measure the effects of the computation-time delay, which are characterized by the controlled process, for example, fuel, time, or some other physical parameters associated with the trajectory of the system (like a vehicle) as it travels from one point to another. It is not easy to quantify the performance of the controlled process in units which are physically meaningful in the context of application. In [59], this performance measure was represented by the additional cost of running the controlled process that is accrued because the computer has a non-zero response time.

We focus on *hard* real-time tasks having control system deadlines that must be met to ensure system stability and/or the system residence in its safe region (allowed state-space). In other words, the controller computer should perform its intended functions to provide correct control inputs within a stringent timing limit in such life-critical controlled processes as aircraft and spacecraft, power systems, life support systems, and nuclear reactors. This is one of the distinguishing characteristics of real-time computing, which has emerged as an important discipline in computer science and engineering. Real-time control is an important application of real-time computing, where the computer system is required to close feedback control loops.

To guarantee the safe and reliable operation of the computer in the presence of faults,[1] the controller computer should be equipped with some form of fault-tolerance. Since the controller computers must provide predictable performance even in the presence of faults, fault-tolerant controller-computers should be designed in the context of the timing requirements of the controlled process (to be specified in Chapter 3). Note that fault-tolerance is achieved by making a tradeoff between time and spatial redundancy. Although most design criteria concentrate on spatial optimization while treating time as a cheap resource in non-real time applications, time becomes so invaluable as to trade space in a real-time environment. This tradeoff between time and spatial redundancy will be covered in Chapters 6 and 7.

[1] According to the origins of faults with respect to the system boundary, faults are classified into (i) *internal faults*, which represent the malfunctioning or damaged parts of a system that induce errors due to physical defects during manufacture or due to component aging, and (ii) *external faults*, which result from environmental interferences or disruptions, such as electromagnetic perturbation, radiation, temperature, or vibration. External faults are likely to be near-coincident and their effects are treated in Chapter 8.

## 2.2 Background

**Control System Deadline:** Failures in actuators, sensors, mechanical parts, A/D and D/A converters may also induce a system failure. In [45], a mathematical framework was presented to describe the interactions between mechanisms to detect or isolate failures of components (actuators, sensors, or computers) failures and the reconfiguration of control algorithms. The author of [67] focused on the design of fault-tolerant control systems to enhance system reliability. In contrast to the work cited above, a goal of this dissertation is to analyze the interplay between a controlled process and its fault-tolerant controller computer, and to derive the deadline information of the controlled process that is useful for the design and evaluation of the controller computer.

Several researchers attempted to analyze the effects of computation-time delay on the performance or stability of a control system. The authors of [75] considered the qualitative effect of feedback delay on a multivariable, computer-controlled linear system by proposing an algorithm to compensate for the delay effect. The sufficient (necessary) conditions of stability with a feedback delay and the delay effects on quadratic performance indices were presented for a linear control system [20] (for a nonlinear robot control system [55]). In [52], a more detailed analysis of the stability of a digital control system with a feedback delay was carried out by modifying the state difference equation. However, all of these analyses are based on the assumption that the feedback delay is fixed or constant. Although the stability problem with a variable-feedback delay was investigated in [22], it was still based on a regular and periodic (thus deterministic) pattern of delays. In [6], a control system with a random time-varying delay in the feedback loop was modeled with a stochastic-delay differential equation, and sufficient conditions were derived for the almost-sure sample stability under which almost every possible differential equation for an ensemble of stochastic systems has a stable solution. However, it did not show any explicit relation between the performance (or stability) and the magnitude of delay, but, instead, gave a condition of the coefficients of the state equations and the average rate-of-change of delay for sample stability. Furthermore, this work assumed a delay to be bounded by the 'worst-case' inter-sample period. In [59], the control system deadline (hard deadline) in controlling the elevator deflection of the aircraft landing problem was obtained numerically by using the concept of allowed state space.

The problem associated with disturbances to the control input was treated by analyzing the observability of a linear system with a dynamic feedback controller under unknown disturbances in the control input [24] and experimentally testing the functional error modes

of computer-based control systems in a "harsh" operating environment [5].

**Fault-Tolerance Latency and Reconfiguration Latency:** Most work on fault-tolerance uses simple models for FTL, which was also represented in [66] as the sum of Mean Time To Detection (MTTD) and Mean Time To Repair (MTTR). Reliability or dependability models assumed the recovery time to have a certain probability distribution; if the recovery time is distributed exponentially, the transition from error state to normal state is represented by the mean rate in a Markov model [15, 19, 68]. In [47], recovery procedures were represented by instantaneous probabilities which measure the effectiveness of fault- or error-handling mechanisms while ignoring the recovery time due to the stiffness existing between fault occurrence and recovery. The authors of [18] derived the distribution of system-recovery times using a truncated normal distribution and a displaced exponential distribution, which captures general short periods of normal recovery and special long durations of rare abnormal recovery. This work was based on the recovery time data collected from various experimental sources.

Note, however, that all of the foregoing approaches do not treat the recovery as consisting of several sequential stages such as fault detection or isolation, system reconfiguration, and recovery of contaminated computations, but instead derive a simple expression for the recovery-time distribution.

In [16, 36], the experimental data/statistical methods (i.e., sampling and parameter-estimation methods) for characterizing the times of fault detection, system reconfiguration, and computation recovery were discussed based on hardware fault injections in the Fault-Tolerant Multiple Processor (FTMP). In [4, 53], the recovery times were estimated for a pooled-spare and $N$-modular redundant systems. The effects of various fault-tolerance features on FTL were described there. However, the results were given in a specific application context using spatial redundancy only, and assumed that the time required for each stage of fault or error recovery is approximated to be in a deterministic range.

For the problem of evaluating the reconfiguration latency, although many researchers have proposed reconfiguration algorithms for various system architectures [3, 10, 37, 50, 71], little work has been done to evaluate the actual times taken for those reconfiguration processes. In [36], the reconfiguration times were measured with detection and isolation times via fault-injection experiments on FTMP, and the average reconfiguration time was experimentally measured to be 82 milliseconds for switching tasks and setting up interconnection routing. In [53], the reconfiguration latencies ranging approximately 40–90 milliseconds

were evaluated for specific load-module sizes of several demonstration applications in a pooled-spares system implemented in the Dynamic Reconfiguration Demonstration System (DRDS) Phase 1 of the DRDS Program.

**Fault-Toterance, TMR Systems, and Time Redundancy:** Fault tolerance is generally accomplished by using redundancy in hardware, software, time, or a combination thereof. There are three basic types of redundancy in hardware and software: static, dynamic, and hybrid. Static redundancy masks faults by taking a majority of the results from replicated tasks [44]. Dynamic redundancy takes a two-step procedure for detection of, and recovery from, faults [7]. The effectiveness of this method relies on selecting a suitable number of spares, a fault-detection scheme, and a switching operation. Hybrid redundancy is a combination of static and dynamic redundancy [11]. A core based on static hardware redundancy, and several spares are provided to tolerate faults. Such redundant systems could provide very high reliability depending on the number of spares used, under the assumption of perfect coverage and switching operation. However, new faults may occur during the detection of existing faults, and the switching operation becomes very complex as the number of spares increases. In order to reduce the complexity of switching operation and enhance reliability at low cost, self-purging [42] and shift-out [12] schemes were developed, where faulty modules were removed but not replaced by standby spares. In these schemes, the additional operation required to select nonfaulty spare(s) is not needed, thus making the switching operation simpler. But it is difficult to implement either a threshold voter or a shift-out checking unit which requires comparators, detectors, and collectors.

Triple Modular Redundancy (TMR) has been one of the most popular fault-tolerance schemes using spatial redundancy. There are several papers to design or analyze TMR systems [2, 9, 23, 64, 73], which is applied for design of a controller computer in Chapters 6–8.

In FTMP [23], computations are performed by triads which are triplicated processors/memories connected by redundant common serial buses, and its quad-redundant clocks use bit-by-bit voting in hardware on all transactions over these buses. C.vmp [64] is a TMR system trading performance away for reliability, by switching between TMR mode and independent modes under program control, where each bus transaction is voted on in a bit parallel fashion by bi-directional voters for voting mode. The probability of system crash due to multiple channel faults is shown in [63] to be insignificant even when the outputs of computing modules are infrequently voted on, if the system is free of latent faults. In FTP

[63], votes are taken only on certain bus transactions by the programmed vote instructions. Increasing the voting frequency improves the fault-detection capability, reduces the time wasted on continuing the execution with contaminated intermediate results, but increases the time overhead of voting: for example, the bus cycle time on C.vmp [64] was extended from 27% to 67% with an average penalty of 40% on the more prevalent memory access for the propagation and synchronization delays of the bi-directional voting between the processes and memories. In [73], an optimal TMR structure for resynchronization to recover from a transient fault was shown to improve significantly the mission time of a small system in spite of the effects of unreliable voter circuits.

In a recent paper [9], a modular TMR multiprocessor was designed to increase reliability and availability by using a retry mechanism to recover from a transient fault and switching between TMR and dual-processor modes to separate a permanent fault. A simple multiple retry policy, which was also a criterion used to detect a permanent fault, was employed there. This can tolerate multiple faults only by treating them as a sequence of single faults with repair between fault occurrences, thus requiring frequent votes for effective fault detection. A TMR failure caused by near-coincident faults on different modules (two or more faults occurring within a short time period) must also be detected and recovered in spite of its rare occurrences. The effect of dependent faults inducing a TMR failure was eliminated by periodic resynchronization with an optimal time interval in [25]. However, the fault models of [25] and [73] did not include permanent faults for which resynchronization is no longer effective.

In addition to spatial redundancy with fault masking or reconfiguration, time redundancy can be applied effectively to recover from transient faults, which will be treated in Chapters 6 and 7. Such recovery techniques are classified into instruction retry [40], program rollback [62], program reload and restart with module replacement. Several papers attempted to develop an optimal recovery policy using time redundancy mainly for single-module (simplex) systems. Instruction retries and program rollbacks were analyzed with optimal parameters (number of retries and intercheckpoint interval) of each procedure in [34]. In [7], an optimal module switching policy was proposed to maximize application-oriented availability with *a priori* specified retry period in an ad hoc manner. In another paper [40], the maximum allowable retry period was derived to minimize mean task completion time, while classifying the type of each fault.

# CHAPTER 3

# DERIVATION AND USE OF CONTROL SYSTEM DEADLINES

## 3.1 Introduction

This chapter presents a method for deriving the CSD of a real-time system, which is defined as the maximum tolerable duration of a controller-computer failure by using the dynamic equation of the controlled process, the information about failure occurrence rates and durations characterizing the environment and fault-tolerance features, and well-defined control algorithms. The dynamic equations are modified to account for the presence of some controller-computer failures that cause computation-time delays. Using these modified equations, we compute the CSD iteratively by examining the necessary conditions for (asymptotic) system stability or the residence of system states in the allowed state space.

We also extend this method to cover the case of nonzero *error latency* [60] owing to an imperfect error detection coverage. There may be control input *disturbances* due to erroneous computation during the error latency in addition to the computation-time delay upon occurrence of component-failure/environmental-interference.

For nonlinear time-invariant control systems, we first linearize the nonlinear dynamics around an operating point and then use well-developed linear systems methods to derive an optimally stabilizing control input and examine system stability (Lyapunov's indirect method) in the presence of additional feedback delays.

To demonstrate the application of the derived CSD information, we also include (i) a design example that optimizes time-redundancy recovery such as retry or rollback, and (ii) an evaluation example that assesses system reliability by using a Markov chain model augmented with new states representing system inertia.

The rest of this chapter is organized as follows. In Section 3.2, we address the generic

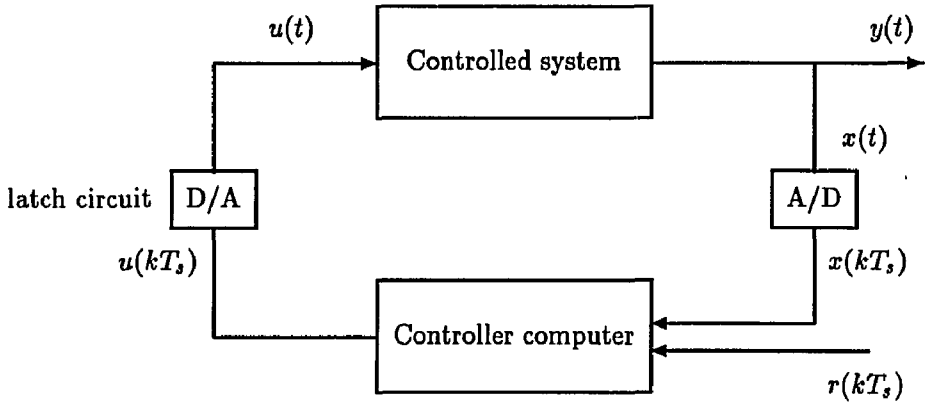



**Figure 3.1:** A computer-controlled system.

problem for qualitative analysis of computation-time delay and/or input disturbance effects, and review the basic definition of the CSD in real-time digital control systems. Section 3.3 presents a method for modifying the state difference equation in the presence of the computation-time delay only, and then analyzes system stability by examining the pole positions of the modified state difference equation. In Section 3.4, we extend the method proposed in Section 3.3 to the case of covering the effects of input disturbances. Control system deadlines are derived there stochastically as well as deterministically. In Section 3.5 the CSD associated with the one-shot event model is derived, when even one occurrence of a computer failure for a long period may drive the controlled process out of its allowed state space, i.e., a dynamic failure occurs. Section 3.6 treats the CSD of nonlinear time-invariant systems, without considering control input disturbances. The linearization method is applied for this analysis. In Section 3.7, several simple control systems are examined to demonstrate our approach. Section 3.8 deals with the application of the CSD information by presenting a design example that optimizes time redundancy such as retry or rollback, but also an evaluation example that assesses system reliability by using the derived deadline information. The chapter concludes with Section 3.9.

## 3.2 Generic Problem of Controller-Computer Failures

As shown in Fig. 3.1, a linear time-invariant controlled process is generally represented by the vector difference equation as shown in Eq. (3.1) and is equipped with a well-designed controller that stabilizes the overall control system and optimizes the given control objective:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \tag{3.1}$$

where $k$ is the time index, one unit of time represents the sampling interval $T_s$, and $\mathbf{x} \in \mathcal{R}^n$ and $\mathbf{u} \in \mathcal{R}^\ell$ are the state and input vectors, respectively. The coefficient matrices, $\mathbf{A} \in \mathcal{R}^{n \times n}$ and $\mathbf{B} \in \mathcal{R}^{n \times \ell}$, are obtained from those of the corresponding continuous-time model:

$$\mathbf{A} = e^{\mathbf{A}_c T_s}, \quad \mathbf{B} = \int_0^{T_s} e^{\mathbf{A}_c (T_s - s)} \mathbf{B}_c \, ds, \tag{3.2}$$

where $\mathbf{A}_c$ and $\mathbf{B}_c$ are the corresponding coefficient matrices of the continuous-time model. We observe the behaviors of a discretized-system by investigating all phenomena only at each sampling time, because a controller computer is required only to response periodically at a sampling time and because our main concern is to analyze the effects of the controller computer on the controlled process. Note that although some abnormal behaviors possible may occur during the inter-sampling interval we assume to use a discretized-system involving those effects. When the computer generates its output during the inter-sampling interval, its effects can be represented using a different conversion process from a continuous-time domain to a discrete-time one (like Eq. (3.3). The (digital) controller computer reads sensor values, compares them with desired values, and calculates the control input once every $T_s$ seconds according to a programmed control strategy. The control input, which is held constant within each sampling interval by a latch circuit, is applied to the continuous-time controlled process.

Eq. (3.1) must include the delays associated with the measurement or sensing, A/D and D/A conversion, and the execution of control algorithms. The sum of these delays is usually much smaller than one sampling period, $T_s$, in the absence of computer failure(s) or external interferences like an EMI, which was called the *delay* problem in [55]. For the delay problem where the magnitude of delay, $\Delta$, is smaller than $T_s$, one can change the state equation (3.1) to:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}_1 \mathbf{u}(k) + \mathbf{B}_2 \mathbf{u}(k-1) \tag{3.3}$$

where $\mathbf{B}_1 = \int_\Delta^{T_s} e^{\mathbf{A}_c (T_s - s)} \mathbf{B}_c ds$ and $\mathbf{B}_2 = \int_0^\Delta e^{\mathbf{A}_c (T_s - s)} \mathbf{B}_c ds$.

As mentioned earlier, digital computers are highly susceptible to transient EMI such as lightening, high intensity radio frequency fields (HIRF), and nuclear electromagnetic pulses (NEMP). The main problem caused by EMI is functional error modes — or computer failures — often without component damages. When a computer failure occurs and it is detected upon its occurrence (i.e., with a zero *error latency* measured from the occurrence to the detection of an error [60]), a certain recovery process is invoked.[1] Then, the

[1] A fault-tolerance mechanism consists of error detection, fault location, system reconfiguration, and

computation-time delay resulting from the corresponding recovery actions may be large relative to $T_s$. This was termed the *loss* problem in [55]. Note that the detection schemes are, in reality, imperfect with a detection probability smaller than one, resulting in a non-zero error latency. During this error latency, the control input will be updated erroneously until the fault inducing the computer failure disappears, or the computer failure is detected and handled properly.

Suppose a computer failure occurs at time $k_0$, it is detected $n_1$ sampling intervals after its occurrence, and the subsequent recovery takes $n_2$ sampling intervals during which the control input will be held constant at $\mathbf{u}(k_0+n_1+1)$ by the D/A converter and latch circuits. The control inputs during this period are formally represented as:

$$\underbrace{\mathbf{u}(k_0+1)\mathbf{I}_\Delta, \mathbf{u}(k_0+2)\mathbf{I}_\Delta, \cdots, \mathbf{u}(k_0+n_1)\mathbf{I}_\Delta}_{n_1}, \cdots,$$
$$\underbrace{\mathbf{u}(k_0+n_1+1), \mathbf{u}(k_0+n_1+1), \cdots, \mathbf{u}(k_0+n_1+1)}_{n_2}, \mathbf{u}(k_0+n_1+n_2+1), \cdots,$$

where $\mathbf{I}_\Delta$ is a diagonal matrix with $Diag[\mathbf{I}_\Delta]_i = 1 + \Delta u_i$ and $\Delta u_i$ is a random sequence which is modeled by the output of a dynamic system with a white-noise input. Since faults/interferences occur randomly during the mission lifetime, their occurrences are considered stochastic perturbations to the controlled process, which can be modeled depending on the fault characteristics. When the environment is assumed to be stochastically stationary, the occurrence and duration of computer failure(s), and the magnitude of disturbances in the control input can be represented by several probability density functions. The relative frequency of disturbance and delay due to such computer failure(s) depends upon the *coverage* (the probability of detecting an error induced by an arbitrary fault), which is determined by failure-detection mechanisms [60].

Stationary occurrences of controller-computer failures/interferences not only degrade the performance of the controlled process but may also lead to loss of system stability if their active duration exceeds the CSD. Even one occurrence of this abnormality with a long active duration in the one-shot event model may make the system leave its allowed state space [56].

The CSD of the stationary model is defined as the maximum duration of the controller computer's failure without losing system stability [56]. More formally, we have the following definition.

---

recovery. General recovery processes are retry, rollback, and reconfiguration, each of which takes a finite time. See [66] for a more detailed account of fault-tolerance mechanisms.

**Definition 1:** Let $\mathbf{x}_e$ denote an equilibrium state of the system represented by Eq. (3.1). Then, $\mathbf{x}_e$ is said to be *stable* at time $k_0$ if for each $\epsilon > 0$ there exists a $\delta(\epsilon, k_0) > 0$ such that

$$\|\mathbf{x}(k_0) - \mathbf{x}_e\| \leq \delta \Longrightarrow \|\mathbf{x}(k) - \mathbf{x}_e\| \leq \epsilon, \ \ \forall k \geq k_0. \tag{3.4}$$

The equilibrium state $\mathbf{x}_e$ is said to be *asymptotically stable* at time $k_0$, if it is stable at time $k_0$, and there exists a $\delta_1(k_0) > 0$

$$\|\mathbf{x}(k_0) - \mathbf{x}_e\| \leq \delta_1(k_0) \Longrightarrow \|\mathbf{x}(k) - \mathbf{x}_e\| \longrightarrow 0 \quad \text{as } k \longrightarrow \infty. \tag{3.5}$$

In linear time-invariant systems, stability can be checked simply by using the pole positions of the controlled process in the presence of random computer failures. Using this information one can derive the CSD stochastically or deterministically with the sample(s) and the ensemble average of the controlled process:

$$D(N) = \inf_{C_{env}} \sup\{N \ : \ \|\lambda(N)\| < 1\}, \tag{3.6}$$

where $\lambda(N)$ is the eigenvalue of the controlled process in the presence of computer failures of the maximum duration $NT_s$ and $C_{env}$ represents all the environmental characteristics that cause controller-computer failures.

Consider a state trajectory evolved from time $k_0$ to $k_f$. Let $\mathbf{X}_A(k)$ and $\mathbf{U}_A$ be the allowed state space at time $k$ and the admissible input space, respectively. If a computer failure, which occurred at $k_1$ $(k_0 \leq k_1 \leq k_f)$ and was detected $N_1T_s$ later, is recovered within $N_2T_s$, where $N = N_1 + N_2$, $0 \leq N_1, N_2 \leq N$, then the control input during these $N$ sampling periods is:

$$\mathbf{u}_a^N(k) = \mathbf{u}(k)\mathbf{I}_\Delta \Pi_{k_1}(N_1) + \mathbf{u}(k_1 + N_1)\Pi_{k_1+N_1}(N_2), \ \ k_1 \leq k \leq k_1 + N$$

where $\Pi_m(n)$ is a rectangular function from $m$ to $m+n$, i.e., $\Pi_m(n) = \xi(k-m) - \xi(k-m-n)$ where $\xi$ is the unit step function. Then, the CSD during $[k_0T_s, \ k_fT_s]$ is also defined as:

$$D(N, \mathbf{x}(k_0)) = \inf_{\mathbf{u}_a^N(k) \in \mathbf{U}_A} \sup\{N \ : \ \phi(k, k_0, \mathbf{x}(k_0), \mathbf{u}_a^N(k)) \in \mathbf{X}_A(k), \ k_0 \leq k \leq k_f\}, \tag{3.7}$$

where the state trajectory is assumed to evolve as:

$$\mathbf{x}(k) = \phi(k, k_0, \mathbf{x}(k_0), \mathbf{u}_a^N(k)). \tag{3.8}$$

## 3.3 CSD of Stationary Model Due to Feedback Delays

In our model, the controlled processes represented by Eq. (3.1) are usually unstable in the absence of feedback control, i.e., the real part of at least one eigenvalue of $A$ is greater

than one. For example, the aircraft designer may push his design to the edge of instability to improve the fuel-efficiency of a future aircraft, where the fast, accurate, and consistent control is required [59]. The state-feedback control input that stabilizes such unstable systems can be calculated by using the observed (or estimated) states according to their own control objectives such as time-optimal control with an energy constraint, optimal state-tracking, and optimal linear regulator [39]. Suppose the feedback control input is computed by $\mathbf{u}(k) = -\mathbf{F}\mathbf{x}(k)$, where the feedback matrix $\mathbf{F}$ depends upon the control objective used. We want to derive necessary conditions, under which it will remain (asymptotically) stable even in the presence of random failures to the controller computer. These conditions require the knowledge of the system dynamic equations, the control algorithm to be used, and the environmental characteristics.

Consider a simple controlled process represented by a linear time-invariant differential equation, which can be converted to a discrete-time problem by using Eq. (3.2) and a quadratic performance index:

$$J = \frac{1}{2}\left(\sum_{k=0}^{k_f-1}\left[\mathbf{x}^T(k)\mathbf{Q}\mathbf{x}(k) + \mathbf{u}^T(k)\mathbf{R}\mathbf{u}(k)\right] + \mathbf{x}^T(k_f)\mathbf{Q}\mathbf{x}(k_f)\right), \tag{3.9}$$

where the matrix $\mathbf{Q} \in \mathcal{R}^{n\times n}$ and $\mathbf{R} \in \mathcal{R}^{\ell\times\ell}$ are positive semidefinite and positive definite, respectively, and are determined by the control objective of interest. The optimal control input is calculated by minimizing $J$ while maintaining system stability, that is, $\mathbf{u}(k) = -\mathbf{F}\mathbf{x}(k)$, where the state feedback gain matrix $\mathbf{F}$ is obtained by solving a discrete Riccati equation [39].

Let the control input have been updated at $t = mNT_s$. If the control input had not been updated for $i$ $(0 \le i \le N)$ sampling periods since $t = mNT_s$ due to a long computation-time delay, the corresponding state equations for the group of intervals during which the system failed to update the control input become:

$$\begin{aligned}
\mathbf{x}(mN+1) &= \mathbf{A}\mathbf{x}(mN) + \mathbf{B}\mathbf{u}(mN) \\
\mathbf{x}(mN+2) &= \mathbf{A}\mathbf{x}(mN+1) + \mathbf{B}\mathbf{u}(mN) \\
&= \mathbf{A}^2\mathbf{x}(mN) + (\mathbf{A}+\mathbf{I})\mathbf{B}\mathbf{u}(mN) \\
&\vdots \\
\mathbf{x}(mN+i) &= \mathbf{A}^i\mathbf{x}(mN) + \sum_{j=0}^{i-1}\mathbf{A}^j\mathbf{B}\mathbf{u}(mN) \\
\mathbf{x}(mN+i+1) &= \mathbf{A}^{i+1}\mathbf{x}(mN) + \sum_{j=1}^{i}\mathbf{A}^j\mathbf{B}\mathbf{u}(mN) + \mathbf{B}\mathbf{u}(mN+i)
\end{aligned}$$

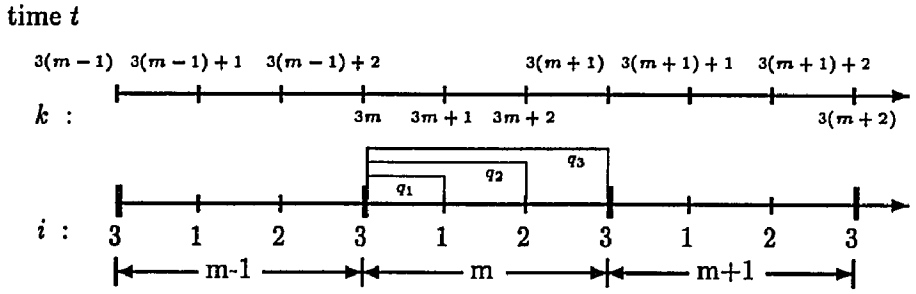



**Figure 3.2:** Time index for argumented vectors representing the stationary occurrence of computer failures (delays) when $N = 3$.

$$\vdots$$

$$\mathbf{x}((m+1)N) = \mathbf{A}^N\mathbf{x}(mN) + \sum_{j=N-i}^{N-1} \mathbf{A}^j\mathbf{B}\mathbf{u}(mN) + \sum_{j=0}^{N-i-1} \mathbf{A}^j\mathbf{B}\mathbf{u}(mN+N-j-1), \tag{3.10}$$

where $m$ is the time index for the groups of $N$ sampling intervals each. Let $\mathbf{X}(m) = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T \equiv [\mathbf{x}(mN+1), \mathbf{x}(mN+2), \ldots, \mathbf{x}((m+1)N)]^T$ and $\mathbf{U}(m) = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]^T \equiv [\mathbf{u}(mN+1), \mathbf{u}(mN+2), \ldots, \mathbf{u}((m+1)N)]^T$. That is, $\mathbf{X}(m)$ and $\mathbf{U}(m)$ are respectively the augmented state and control vectors at the group of sampling intervals during which the system failed to update the control inputs (see Fig. 3.2). When the delay is equal to $i$ sampling periods, the following augmented state equations result:

$$\mathbf{X}(m+1) = \mathbf{A}_D\mathbf{X}(m) + \mathbf{B}^1_{D_i}\mathbf{U}(m) + \mathbf{B}^2_{D_i}\mathbf{U}(m+1), \tag{3.11}$$

$$\mathbf{U}(m) = -\mathbf{F}_D\mathbf{X}(m), \tag{3.12}$$

where

$$\mathbf{A}_D = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & \mathbf{A} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{A}^2 \\ \vdots & & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{A}^N \end{bmatrix}, \qquad \mathbf{F}_D = \begin{bmatrix} \mathbf{F} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{F} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{F} \end{bmatrix},$$

$$\mathbf{B}^1_{D_i} = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & \mathbf{B} \\ \mathbf{0} & \cdots & \mathbf{0} & (\mathbf{AB}+\mathbf{B}) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \sum_{j=0}^{i-1} \mathbf{A}^j\mathbf{B} \\ \mathbf{0} & \cdots & \mathbf{0} & \sum_{j=1}^{i} \mathbf{A}^j\mathbf{B} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \sum_{j=N-i}^{N-1} \mathbf{A}^j\mathbf{B} \end{bmatrix}, \quad \mathbf{B}^2_{D_i} = \left.\begin{bmatrix} \mathbf{0} & & \cdots & & \mathbf{0} \\ \vdots & \ddots & \cdots & & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{B} & \cdots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{AB} & \cdots & \mathbf{0} \\ \vdots & \cdots & \vdots & & \vdots \\ \mathbf{0} & \cdots & \mathbf{A}^{N-i-1-\delta(i)}\mathbf{B} & \cdots\ \mathbf{B} & \mathbf{0} \end{bmatrix}\right\} N \tag{3.13}$$

Suppose the occurrence of computer failure(s) is binomially distributed with parameter $P$ at each sampling time. Let $q_0, q_1 \cdots, q_N$ be the probabilities of delays $0, T_s, \cdots, NT_s$, respectively, such that $\sum_{i=1}^{N} q_i = P$, where the maximum delay is assumed to be $NT_s$. This delay distribution can be derived in practice from the knowledge of the failure-handling policy and environmental characteristics. Since the delay equal to the time interval from failure occurrence/detection to its recovery — called the *Fault-Tolerance Latency* (FTL) — it can be measured via fault-injection experiments or can be evaluated analytically by a certain *pdf* (probability density function) considering all fault-tolerance features [28]. From the experimental samples or the fitted *pdf* of the FTL, one can estimate the parameters of the multinomial distribution ($q_i$'s).

By combining the state equations (3.11) with the delay distribution, the state equation including the effects of delay no greater than $NT_s$ becomes:

$$\mathbf{X}(m+1) = \mathbf{A}_D\mathbf{X}(m) + \sum_{i=0}^{N} \xi_i \left(\mathbf{B}^1_{D_i}\mathbf{U}(m) + \mathbf{B}^2_{D_i}\mathbf{U}(m+1)\right), \tag{3.14}$$

where $\xi_i \in \{0,1\}$ is a binomially-distributed random variable with parameter $q_i$, i.e., $\Pr[\xi_i = 1] = q_i$. Then, the first moment of Eq. (3.14) is:

$$\bar{\mathbf{X}}(m+1) = \mathbf{A}_D\bar{\mathbf{X}}(m) + \sum_{i=0}^{N} q_i \left\{\mathbf{B}^1_{D_i}\mathbf{U}(m) + \mathbf{B}^2_{D_i}\mathbf{U}(m+1)\right\}. \tag{3.15}$$

Since the period of the index $m$ is $N$, the complex variable $z$ in the $Z$-transform of Eq. (3.15) corresponds to $z_k^N$, where $z_k$ is the complex variable in the $Z$-transforms of equations with index $k$, that is, the period of Eq. (3.15) is $\frac{\omega}{N}$ in the frequency domain, where $\omega$ indicates the period of equations with index $k$ in the frequency domain (sub-sampling). Using Eqs. (3.12) and (3.15), the characteristic equation of the control system in the presence of stationary occurrences of feedback delay is represented by:

$$\det\left[(\mathbf{I} + \sum_{i=0}^{N} q_i\mathbf{B}^2_{D_i}\mathbf{F}_D)z^N - \mathbf{A}_D + \sum_{i=0}^{N} q_i\mathbf{B}^1_{D_i}\mathbf{F}_D\right] = 0. \tag{3.16}$$

The asymptotic stability can be tested with the pole positions of Eq. (3.16) whose characteristic equation reduces to a simple form due to the simple structure of $\mathbf{A}_D$ despite its augmented dimension. The characteristic equation of the zero-delay case (i.e., $q_0 = 1$ and thus $P = \sum_{i=1}^{N} q_i = 0$) is:

$$\det\left[z^N\mathbf{I} - (\mathbf{A} - \mathbf{BF})^N\right] = 0. \tag{3.17}$$

Further, one can get the following characteristic equation for the worst case in which $q_N = 1$, or the control input is updated only once every $N$ sampling intervals due to the periodic delay of an active duration $NT_s$:

$$\det\left[z^N\mathbf{I} - \mathbf{A}^N + \sum_{i=0}^{N-1}\mathbf{A}^i\mathbf{BF}\right] = \det\left[z^N\mathbf{I} - \sum_{i=0}^{N-1}\mathbf{A}^i(\mathbf{A} - \mathbf{BF}) + \sum_{i=1}^{N-1}\mathbf{A}^i\right] = 0, \tag{3.18}$$

where $\mathbf{A} = \mathbf{QDQ}^{-1}$ by the similarity transformation if $\mathbf{D} = diag[d_1, \cdots, d_n]$ and $d_i$'s are the eigenvalues of $\mathbf{A}$.

The conditions for asymptotic stability are used to derive the CSD. Let $NT_s$ and $DT_s$ be the assumed maximum and the actual maximum delays, respectively. Then, the CSD can be obtained by iteratively testing the necessary conditions for system stability in the modified charcteristic equation Eq. (3.16) while changing $N$ from 1 to $D$.

## 3.4 CSD in the Presence of Delays/Disturbances

The hard-deadline information is also derived for a linear time-invariant controlled process in the presence of disturbances as well as delays in the control input. We use the same method in Section 3.4, i.e., iteratively testing for the stability of the modified (stochastic) dynamic equation while changing $N$ from 1 to $D$, where $NT_s$ and $DT_s$ are redefined as the assumed maximum and actual maximum duration of delays and/or disturbances, respectively.

To modify the given state equation of Eq. (3.1) by incorporating all stochastic behaviors of controller-computer failures, we need the following random sequences and assumptions:

- $d$ is the conditional probability of successful detection given that a computer failure had occurred.
- $q_i^w$ is the conditional probability of an input disturbance for $i$ sampling intervals ($\sum_{i=1}^{N} q_i^w = 1$) if a computer failure occurs and is not detected until its disappearance. It can be estimated by using the experimental data or the analytic model for the error latency.

- $q_{\Delta u}$ is the probability density function (*pdf*) of $\Delta u$ which is the magnitude of a control input disturbance at time $kT_s$, i.e., $\mathbf{u}_{actual}(k) = \mathbf{u}_{desired}(k)\mathbf{I}_{\Delta}$. The mean and variance of $q_{\Delta u}$ are given as $\mu_{\Delta u}$ and $\sigma^2_{\Delta u}$, respectively.

- The probability that two transient failures occur sequentially within a small number of sampling intervals, $(N-i)T_s$, where the delay (recovery duration) or duration of the erroneous control input (active duration of a transient failure) is $i$ sampling intervals and $NT_s$ is the assumed maximum value of such intervals — is so small as to be ignored. Thus, we consider only one computer failure possible during a group of $N$ intervals.

- Any random sequence will be independent identically distributed (*i.i.d*) for the time index $k$.

The control input at $(mN+i)T_s$ is updated to be $\mathbf{u}(mN+i)\mathbf{I}_{\Delta}$ for the disturbance case or $\mathbf{u}(mN)$ for the delay case. We, then, obtain the augmented state equations by using the augmented state and control vectors for a group of $N$ sampling intervals:

$$\mathbf{X}(m+1) = \mathbf{A}_D\mathbf{X}(m) + \mathbf{B}^1_{D_i}\mathbf{U}(m) + \mathbf{B}^2_{D_i}\mathbf{U}(m+1), \tag{3.19}$$

$$\mathbf{U}(m) = -\mathbf{F}_D\mathbf{X}(m), \tag{3.20}$$

where $[\mathbf{B}^1_{D_i}, \mathbf{B}^2_{D_i}]$ becomes $[\mathbf{B}^{n1}_{D_0}, \mathbf{B}^{n2}_{D_0}]$ for the normal behavior, $[\mathbf{B}^{d1}_{D}, \mathbf{B}^{d2}_{D_i}]$ for the delay case, and $[\mathbf{B}^{w1}_{D_i}, \mathbf{B}^{w2}_{D_i}]$ for the disturbance case, respectively. (In [30], these augmented matrices are described in full.) Eqs. (3.19) and (3.20) are, in turn, modified by combining the random squences representing the behavior of computer failures:

$$\begin{aligned}\mathbf{X}(m+1) = \mathbf{A}_D\mathbf{X}(m) &+ \left((1-\psi)\mathbf{B}^{n1}_{D_0} + \psi(1-\varphi)\mathbf{B}^{w1}_{D} + \psi\varphi\sum_{i=1}^{N}\xi_i\mathbf{B}^{d1}_{D_i}\right)\mathbf{U}(m) + \\ &\left((1-\psi)\mathbf{B}^{n2}_{D_0} + \psi(1-\varphi)\sum_{i=1}^{N}\zeta_i\mathbf{B}^{w2}_{D_i} + \psi\varphi\sum_{i=1}^{N}\xi_i\mathbf{B}^{d2}_{D_i}\right)\mathbf{U}(m+1)\end{aligned} \tag{3.21}$$

where $\psi, \varphi \in \{0,1\}$ are binomially-distributed random sequences with probabilities $P, d$, and $\xi_i, \zeta_i \in \{0,1\}$ are multinomially-distributed random sequences with probabilities $q^d_i, q^w_i$, i.e., $\Pr[\xi_i = 1] = q^w_i$.

The asymptotic stability of Eq. (3.21) can be examined deterministically or stochastically.

**A. Deterministic Approach**

Similar to the method used in the Section 3.4, the deterministic value of the CSD is obtained by examining the pole positions of the first moment (ensemble average) of

Eq. (3.21). Although the resulting CSD has little practical meaning, it can show the trend of the ensemble system behavior with an uncertainty (in the state and output) which can be characterized by the second moment of Eq. (3.21). The first moment of Eq. (3.21) is:

$$\bar{\mathbf{X}}(m+1) = \mathbf{A}_D\bar{\mathbf{X}}(m) + \left((1-P)\mathbf{B}_{D_0}^{n1} + P(1-d)\bar{\mathbf{B}}_D^{w1} + pd\sum_{i=1}^{N} q_i^d \mathbf{B}_{D_i}^{d1}\right)\mathbf{U}(m) + \left((1-P)\mathbf{B}_{D_0}^{n2} + P(1-d)\sum_{i=1}^{N} q_i^w \bar{\mathbf{B}}_{D_i}^{w2} + pd\sum_{i=1}^{N} q_i^d \mathbf{B}_{D_i}^{d2}\right)\mathbf{U}(m+1). \quad (3.22)$$

Using Eqs. (3.20) and (3.22), one can get the characteristic equation of Eq. (3.22):

$$\det \left[\left(\mathbf{I} + [(1-P)\mathbf{B}_{D_0}^{n2} + P(1-d)\sum_{i=1}^{N} q_i^w \bar{\mathbf{B}}_{D_i}^{w2} + pd\sum_{i=1}^{N} q_i^d \mathbf{B}_{D_i}^{d2}]\mathbf{F}_D\right) z^N - \left(\mathbf{A}_D - [(1-P)\mathbf{B}_{D_0}^{n1} + P(1-d)\bar{\mathbf{B}}_D^{w1} + Pd\sum_{i=1}^{N} q_i^d \mathbf{B}_{D_i}^{d1}]\mathbf{F}_D\right)\right] = 0. \quad (3.23)$$

The characteristic equation of the zero-delay case (i.e., no computer failure, or $p = 0$) is:

$$\det\left[z^N\mathbf{I} - (\mathbf{A} - \mathbf{BF})^N\right] = 0, \quad (3.24)$$

where the magnitutes of eigenvalues are equal to those obtained from:

$$\det\left[z\mathbf{I} - (\mathbf{A} - \mathbf{BF})\right] = 0 \quad (3.25)$$

which is the characteristic equation of the original state equation (Eq. (3.1)) in the absence of computer failures.

**B. Stochastic Approach**

The effectiveness of the deterministic approach decreases as the variance of $q_{\Delta u}$ gets large. In such a case, we can derive the probability mass function ($pmf$) of the CSD with respect to $q_{\Delta u}$ rather than the deterministic value of the CSD based on the mean of $q_{\Delta u}$. Now, the mapping between the CSD and the magnitudes of disturbance ($\Delta u$'s) is not one-to-one, and the CSD can be derived numerically for each sample value of $\Delta u$'s. In all but simplest cases it is impossible to derive a closed-form expression for the $pmf$ of the CSD or the exact relation between the CSD and $\Delta u$. The method we use is therefore to quantize uniformly the $q_{\Delta u}$ continuum in the interval $[a, b]$, where $\int_a^b q_{\Delta u} d\Delta u = \gamma$. Let this quantization result in $M$ equal-length subintervals (cells). There is a tradeoff between the accuracy and the amount of computation in determining $\gamma$ and $M$, and $a$ and $b$ are determined appropriately according to $\gamma$. Then, points are allocated to the quantized cells, and let the point of the $i$-th cell $([a+(i-1)\frac{M}{b-a}, a+i\frac{M}{b-a}])$ be $\Delta u_i$ which corresponds to the midpoint of the cell, i.e., $\Delta u_i = a + \frac{(2i-1)M}{2(b-a)}$, then the probability of the point is calculated

as $q^i_{\Delta u} = \int_{a+(i-1)\frac{M}{b-a}}^{a+i\frac{M}{b-a}} q_{\Delta u}(s)ds$. A CSD is derived for each $\Delta u_i$, and let it be $D_i$ whose probability is equal to that of $q^i_{\Delta u}$. Finally, the *pmf* of the CSD is derived numerically by multiplying $D_i$ and $q^i_{\Delta u}$, $1 \leq i \leq M$. The accuracy of the resulting *pmf* depends on $a, b$, and $M$, which must be determined by considering the controlled-process state equations, the environment, and the amount of computation.

Although the above method uses a stochastic approach in deriving the *pmf* of the CSD, it is still based on the mean values of binomially- and multinomially- distributed random sequences since the control system deadlines of all possible samples cannot be derived due to the excessive number of possible samples. However, the stability of each individual system (i.e., samples) has more practical meaning than the stability of the average system or the ensemble of all possible systems which might be built. Thus, in addition to the deterministic analysis (or combined with the stochastic analysis) of the averaged system stability, we will attempt to stochastically analyze system stability by using the *almost-sure* sample stability concept (which is actually almost deterministic).

**Definition 2:**

- *Probabilistic Stability*: For every pair of positive numbers $a$ and $b$, there exists a positive number $d(a, b, t_0)$ such that

$$P[\sup_{t \geq t_0} \|x_t\| > a] < b \quad \text{for } x_{t_0} \text{ such that } \|x_{t_0}\| < d, \tag{3.26}$$

  where $x_t = \{x(s) : t_0 \leq s \leq t\}$ is a segment of the past history.

- *Almost-Sure Stability*: For every pair of positive numbers $a$ and $b$, there exists a positive number $d(a, b, t_0)$ such that

$$P[\sup_{\|x_{t_0}\| < d} \left( \sup_{t \geq t_0} \|x_t\| \right) > a] < b. \tag{3.27}$$

In fact, the almost-sure sample stability means that almost every possible difference equation for a given ensemble of such systems has a state which is stable in the Lyapunov sense.

## 3.5 One-Shot Event Model

The pole locations do not change in case of only one failure with a relatively long ($> T_s$) active period (Fig. 3.3). The (asymptotic or global) stability condition discussed thus far is therefore no longer applicable. Instead, the terminal state constraints can be used to test whether or not the system leaves its allowed state space. Note that every critical process

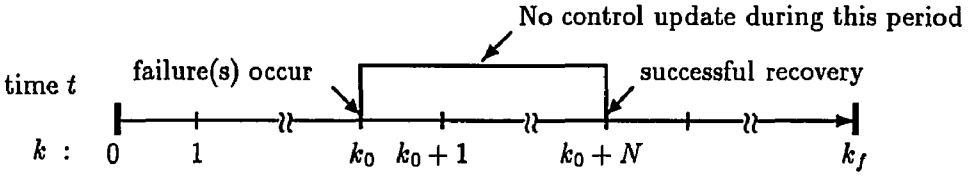



**Figure 3.3:** Time index for the one-shot event/delay model.

must operate within the state space circumscribed by given constraints, i.e., the allowed state space. When the control input is not updated correctly for a period exceeding the CSD, the system may leave the allowed state space, thus causing a dynamic failure. The allowed state space consists of two sets of states $\mathbf{X}_A^1$ and $\mathbf{X}_A^2$ defined as follows:

- $\mathbf{X}_A^1$: the set of states in which the system must stay to avoid an *immediate* dynamic failure, e.g., a civilian aircraft flying upside down is viewed as an immediate dynamic failure. This set can usually be derived *a priori* from the physical constraints.
- $\mathbf{X}_A^2$: the set of states that can lead to meeting the terminal constraints with appropriate control inputs. This set is determined by the terminal constraints, the dynamic equation, and the control algorithm used.

The system must not leave $\mathbf{X}_A^1$ nor $\mathbf{X}_A^2$ in order to prevent catastrophic failure.

Assuming that some computer failure may not be detected upon its occurrence but every detected failure can always be recovered successfully, we can consider three cases for the analysis of the effects of computer failures during a finite time: (i) delay: when a computer failure is detected upon its occurrence, (ii) disturbance: when a computer failure is not detected until its disappearance, and (iii) disturbance and delay: when a computer failure is detected at some time after its occurrence but before its disappearance.

Let $k_0, k_f, N_1$, and $N_2$ denote the indices for the failure occurrence time, the mission completion time, and the period of disturbance, the period of delay measured in sampling periods, respectively, where $N = N_1 + N_2$, $0 \leq N_1, N_2 \leq N$. The dynamic equation for a one-shot event model is:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\left[\mathbf{u}(k) + (\mathbf{u}(k_0) - \mathbf{u}(k))\Pi_{k_0}(N_1) + \mathbf{u}(k)(\mathbf{I}_\Delta - \mathbf{I})\Pi_{k_0+N_1}(N_2)\right], \quad (3.28)$$

where $\Pi_{k_0}(N)$ is the rectangular function as defined in Section 3.2, and $N_1$ and $N_2$ are random variables and determined by the conditional probability of successful detection ($d$) if $N$ is given:

$$\Pr[N_1 = i] \;=\; d(1-d)^i \qquad\qquad 0 \leq i \leq N-1$$

$$\begin{aligned} \Pr[N_2 = i] &= d(1-d)^{N-i} \qquad 1 \le i \le N \\ \Pr[N_1 = N] &= \Pr[N_1 = 0] = 1 - d(1-d)^{N-1}. \end{aligned} \tag{3.29}$$

Thus, the first moment of Eq. (3.28) is:

$$\bar{\mathbf{x}}(k+1) = \mathbf{A}\bar{\mathbf{x}}(k) + \mathbf{B}\left(\mathbf{u}(k) + \sum_{i=0}^{N}[q_i(\mathbf{u}(k_0) - \mathbf{u}(k))\Pi_{k_0}(N_1) + q_i^w \mathbf{u}(k)(\mathbf{I}_\Delta - \mathbf{I})\Pi_{k_0+N_1}(N_2)]\right). \tag{3.30}$$

Using Eq. (3.30), one can derive the states at time $k_0 + N$ and $k_f$, and examine whether or not the state trajectory satisfies the immediate and terminal constraints, iteratively for each $N$ $(1 \le N \le D)$.

$$\bar{\mathbf{x}}(k_0 + N) = \mathbf{A}^N \bar{\mathbf{x}}(k_0) + \sum_{i=k_0}^{k_0+N-1} \mathbf{A}^{k_0+N-i-1}\mathbf{B}[q_i \mathbf{u}(k_0) + q_i^w \mathbf{u}(i) I_\Delta], \tag{3.31}$$

$$\begin{aligned} \bar{\mathbf{x}}(k_f) &= \mathbf{A}^{k_f-k_0}\bar{\mathbf{x}}(k_0) + \sum_{i=k_0}^{k_0+N-1} \mathbf{A}^{k_f-i-1}\mathbf{B}[q_i \mathbf{u}(k_0) + q_i^w \mathbf{u}(i)\mathbf{I}_\Delta] + \sum_{i=k_0+N}^{k_f-1} \mathbf{A}^{k_f-i-1}\mathbf{B}\mathbf{u}(i) \\ &= \mathbf{A}^{k_f-k_0-N}\bar{\mathbf{x}}(k_0+N) + \sum_{i=k_0+N}^{k_f-1} \mathbf{A}^{k_f-i-1}\mathbf{B}\mathbf{u}(i). \end{aligned} \tag{3.32}$$

In addition to this deterministic approach, the $pmf$ of the CSD that depends on $q_{\Delta u}$ is also derived by using the same stochastic approach employed in the stationary model. Without using the first moment of the state equation (3.30) the probability of the CSD being $N$ (i.e., $\Pr[N = D]$) is thus obtained as the sum of the probabilities[2] of the sample equations (3.28) in which a dynamic failure occurs at time $N$. One must continue this process until a dynamic failure occurs for all samples (i.e., $\Pr[N = D]$=1).

The following pseudo-code is used to iteratively derive the CSD for the system with the initial state $\mathbf{x}_0$ at time $k_0$, where $T_d$ is the duration of abnormality due to a computer failure.

```
\* Recursive testing the allowed state space while increasing T_d *\
m = 1      \* sub-sampling period, mΔ *\
D = integer_time(N_tim)      \* integer times of T_s *\
if (D ≤ N_tim) then
   while (m ≠ T_s/Δ or test_constraints() ≠TRUE) do
      T_d := (D − 1)T_d + mΔ
      if test_constaints(T_d) then return T_d      \* CSD is T_d *\
      else m := m + 1      \* test larger T_d by increasing m *\
   end_while
else return no CSD

integer_time(N_tim)
N = 1
```

[2] $N^2$ testings are required for each $N$.

```
while (N ≠ N_lim or test_constaints() ≠TRUE) do
   T_d := NT_s
   if test_constaints(T_d) then return N      \* D = N_max is obtained *\
   else N := N + 1      \* test larger N *\
end_while
test_constraints(T_d)
compute x((k_0 + N)T_s) from x_0
if (x((k_0 + N)T_s) ∈ X_A^1) then
   begin
      compute x(k_f T_s) from either x((k_0 + N)T_s) or x_0
      if (x(k_f T_s) ∈ X_A^f) then return FAILURE
      else return TRUE
   end
else return TRUE
```

$\mathbf{x}_f \in \mathbf{X}_A^f$ can be tested indirectly by the following relation:

$$\mathbf{x}(k_f) \in \mathbf{X}_A^f \Longleftrightarrow \mathbf{x}(k_0+N) \in \mathbf{X}_A^2, \tag{3.33}$$

$$\text{where } \mathbf{X}_A^2 = \{\, \mathbf{x} \mid [\mathbf{A}^{k_f-k_0-N}\mathbf{x} + \sum_{i=k_0+N}^{k_f-1} \mathbf{A}^{k_f-i-1}\mathbf{B}\mathbf{u}(i)] \in \mathbf{X}_A^f \,\}. \tag{3.34}$$

In practice, it is difficult to obtain $\mathbf{X}_A^2$. Although there may be a one-to-one mapping between $\mathbf{x}(k_0+N)$ and $\mathbf{x}(k_f)$, $\mathbf{X}_A^f$ is usually a continuum, which requires an excessive amount of computation due to the curse of dimensionality. The size of $\mathbf{X}_A^2$ will decrease as either $N$ increases or $k$ approaches $k_f$, but the size of $\mathbf{X}_A^2$ is usually larger than that of $\mathbf{X}_A^f$ due to the (asymptotic) stability of a controlled process.

## 3.6 CSD for Time-Invariant Nonlinear Systems

Nonlinear control systems generally differ from linear systems in two important aspects:

1. It is not always possible to obtain closed-form solutions for nonlinear systems, where the sequences of approximating functions converging to (or estimates for) the true solution are mostly satisfying forms of the solution.

2. The analysis requires more complex and difficult mathematics.

In spite of these difficulties, control system deadlines are derived for nonlinear control systems since the dynamic equations of most control systems consist of nonlinear properties such as nonlinear gain, saturation, deadband, backlash, hysteresis, and nonlinear characteristic curves. The nonlinear differential equation of the continuous-time domain is generally given by:

$$\dot{\mathbf{x}}(t) = \mathbf{h}[t, \mathbf{x}(t), \mathbf{u}(t)], \qquad \mathbf{x}(0) = \mathbf{x}_0. \tag{3.35}$$

Assuming that the function $h(t,\cdot)$ is globally Lipschitz-continuous, the state at the sampling time $(k+1)T_s$ is represented by using a Taylor series:

$$\mathbf{x}((k+1)T_s) = \mathbf{x}(kT_s) + T_s\dot{\mathbf{x}}(kT_s) + \frac{T_s^2}{2}\ddot{\mathbf{x}}(kT_s) + \cdots, \tag{3.36}$$

where the first-order derivative term can be calculated using Eq. (3.35), i.e., $\dot{\mathbf{x}}(kT_s) = \mathbf{h}[kT_s, \mathbf{x}(kT_s), \mathbf{u}(kT_s)]$. Since higher-order methods require the calculation of many partial derivatives, the first-order method is applied as a useful starting point in understanding more sophsticated methods. Then, the nonlinear discrete-time state equation is approximated by:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \mathbf{f}[k, \mathbf{x}(k), \mathbf{u}(k)], \quad \forall k \geq k_0, \tag{3.37}$$

where $\mathbf{f}(\cdot) = T_s\mathbf{h}(\cdot)$ and $\mathbf{x}(k)$ corresponds to $\mathbf{x}(kT_s)$.

### 3.6.1 CSD of the Stationary Model

The effect of the stationary occurrences of computation-time delay due to the failure(s) of the controller computer can also be analyzed by examining the stability of nonlinear systems like linear systems. In this analysis, every failure is assumed to be detected upon its occurrence and call for a recovery mechanism, i.e., $d = 1$. This assumption ignores the effects of erroneous control inputs. However, one cannot simply modify the dynamic equations, nor can he calculate pole positions efficiently. The stability of nonlinear systems is generally analyzed by the Lyapunov's second method. The drawback of this method, which seriously limits its use in practice, is that it is not easy to find the required Lyapunov function and it gives only sufficient conditions for stability or instability. Thus, the Lyapunov's first method, which linearizes the nonlinear system around an equilibrium point and examines the stability of the resulting linearized system, is used for general nonlinear systems.

The state difference equation of a nonlinear control system is assumed to be given as in Eq. (3.37) and a suitable feedback control input is calculated to stabilize the system. Let the control input have been updated at $t = mNT_s$. If the control inputs were not updated for $i$ sampling periods from that time due to a long computation-time delay, where $0 \leq i \leq N$, the corresponding state equations for the group of intervals during which the system failed to update the control inputs become:

$$\begin{aligned}
\mathbf{x}(mN+1) &= \mathbf{x}(mN) + \mathbf{f}[mN, \mathbf{x}(mN), \mathbf{u}(mN)] \\
\mathbf{x}(mN+2) &= \mathbf{x}(mN+1) + \mathbf{f}[mN+1, \mathbf{x}(mN+1), \mathbf{u}(mN)] \\
&= \mathbf{x}(mN) + \mathbf{f}[mN, \mathbf{x}(mN), \mathbf{u}(mN)] + \mathbf{f}[mN+1, \mathbf{x}(mN+1), \mathbf{u}(mN)]
\end{aligned}$$

$$
\begin{aligned}
&\vdots \\
\mathbf{x}(mN+i) &= \mathbf{x}(mN)+\sum_{j=0}^{i-1}\mathbf{f}[mN+j,\mathbf{x}(mN+j),\mathbf{u}(mN)] \\
\mathbf{x}(mN+i+1) &= \mathbf{x}(mN)+\sum_{j=0}^{i-1}\mathbf{f}[mN+j,\mathbf{x}(mN+j),\mathbf{u}(mN)] \\
&\quad +\mathbf{f}[mN+i,\mathbf{x}(mN+i),\mathbf{u}(mN+i)] \\
&\vdots \\
\mathbf{x}((m+1)N) &= \mathbf{x}(mN)+\sum_{j=0}^{i-1}\mathbf{f}[mN+j,\mathbf{x}(mN+j),\mathbf{u}(mN)] \\
&\quad +\sum_{j=i}^{N-1}\mathbf{f}[mN+j,\mathbf{x}(mN+j),\mathbf{u}(mN+j)]
\end{aligned}
$$

where $m$ is the time index for the groups of $N$ sampling intervals each. Then, we get the following augmented state difference equation:

$$\mathbf{X}(m+1)=\mathbf{X}(m)+\mathbf{F}_i[mN,\cdots,mN+N-1,\mathbf{X}(m),\mathbf{X}(m+1),\mathbf{U}(m),\mathbf{U}(m+1)], \quad (3.38)$$

where

$$
\mathbf{F}_i[\cdot]=\begin{bmatrix}
\mathbf{f}[mN,\mathbf{E}_N\mathbf{X}(m),\mathbf{E}_N\mathbf{U}(m)] \\
\mathbf{f}[mN,\mathbf{E}_N\mathbf{X}(m),\mathbf{E}_N\mathbf{U}(m)]+\mathbf{f}[mN+1,\mathbf{E}_1\mathbf{X}(m+1),\mathbf{E}_N\mathbf{U}(m)] \\
\vdots \\
\sum_{j=0}^{i-1}\mathbf{f}[mN+j,\mathbf{E}_j\mathbf{X}(m+1),\mathbf{E}_N\mathbf{U}(m)] \\
\sum_{j=0}^{i-1}\mathbf{f}[mN+j,\mathbf{E}_j\mathbf{X}(m+1),\mathbf{E}_N\mathbf{U}(m)]+\mathbf{f}[mN+i,\mathbf{E}_i\mathbf{X}(m+1),\mathbf{E}_i\mathbf{U}(m+1)] \\
\vdots \\
\sum_{j=0}^{i-1}\mathbf{f}[mN+j,\mathbf{E}_j\mathbf{X}(m+i),\mathbf{E}_N\mathbf{U}(m)]+\sum_{j=i}^{N-1}\mathbf{f}[mN+j,\mathbf{E}_j\mathbf{X}(m+i),\mathbf{E}_j\mathbf{U}(m+1)]
\end{bmatrix},
$$

where $\mathbf{E}_i=[\overbrace{0,\cdots,0,\underbrace{1}_{i},0,\cdots,0}^{N}]$.

Using the probabilities of delays $(q_0,q_1\cdots,q_N)$, we get a final form of the state difference equation:

$$\mathbf{X}(m+1)=\mathbf{X}(m)+\sum_{i=0}^{N}\xi_i\mathbf{F}_i[mN,\cdots,mN+N-1,\mathbf{X}(m),\mathbf{X}(m+1),\mathbf{U}(m),\mathbf{U}(m+1)] \quad (3.39)$$

where $\xi_i\in\{0,1\}$ is a binomially-distributed random variable with parameter $q_i$, i.e., $\Pr[\xi_i=1]=q_i$. Then, the first moment of the above equation is:

$$\bar{\mathbf{X}}(m+1)=\bar{\mathbf{X}}(m)+\sum_{i=0}^{N}q_i\mathbf{F}_i[mN,\cdots,mN+N-1,\bar{\mathbf{X}}(m),\bar{\mathbf{X}}(m+1),\mathbf{U}(m),\mathbf{U}(m+1)]. \quad (3.40)$$

To derive the maximum value of $N$ which maintains local stability, the stability of Eq. (3.40) is examined by linearizing Eq. (3.40) around an equilibrilum point. Let the equilibrilum point be 0 without loss of generality. Then, the linearized form of Eq. (3.40) is:

$$\mathbf{A}_D^2\bar{\mathbf{X}}(m+1) = \mathbf{A}_D^1\bar{\mathbf{X}}(m) + \mathbf{B}_D^1\mathbf{U}(m) + \mathbf{B}_D^2\mathbf{U}(m+1), \tag{3.41}$$

where $\mathbf{F}_i[0,\cdots,0] = 0, \mathbf{A}_D^1 = \mathbf{I} - \left[\frac{\partial\sum_{i=0}^N q_i\mathbf{F}_i}{\partial\mathbf{X}(m)}\right]_{\mathbf{X}=0,\mathbf{U}=0}, \mathbf{A}_D^2 = \mathbf{I} + \left[\frac{\partial\sum_{i=0}^N q_i\mathbf{F}_i}{\partial\mathbf{X}(m+1)}\right]_{\mathbf{X}=0,\mathbf{U}=0}$,
$\mathbf{B}_D^1 = \mathbf{I} - \left[\frac{\partial\sum_{i=0}^N q_i\mathbf{F}_i}{\partial\mathbf{U}(m)}\right]_{\mathbf{X}=0,\mathbf{U}=0}$, and $\mathbf{B}_D^2 = \mathbf{I} - \left[\frac{\partial\sum_{i=0}^N q_i\mathbf{F}_i}{\partial\mathbf{U}(m+1)}\right]_{\mathbf{X}=0,\mathbf{U}=0}$.
When a state feedback controller — which is also obtained from the linearization method — is used (i.e., $\mathbf{U}(m) = -\mathbf{P}_D\mathbf{X}(m)$ where $\mathbf{P}_D = \frac{\partial\mathbf{G}}{\partial\mathbf{X}(m)}$ and $\mathbf{U}(m) = \mathbf{G}[\mathbf{X}(m)]$), Eq. (3.41) becomes:

$$(\mathbf{A}_D^2 + \mathbf{B}_D^2\mathbf{P}_D)\bar{\mathbf{X}}(m+1) = (\mathbf{A}_D^1 - \mathbf{B}_D^1\mathbf{P}_D)\bar{\mathbf{X}}(m). \tag{3.42}$$

Using Eq. (3.42), we examine local stability by calculating the pole positions of the following characteristic equation:

$$\det\left[(\mathbf{A}_D^2 + \mathbf{B}_D^2\mathbf{P}_D)z^N - (\mathbf{A}_D^1 - \mathbf{B}_D^1\mathbf{P}_D)\right] = 0. \tag{3.43}$$

This method has two limitations: (i) the conclusions based on linearization are purely local, i.e., it is effective only in the vincinity of the equilibrium point, and (ii) if some poles are located on the unit circle and the others are located within the unit circle, the result is inconclusive, which is called a *critical problem.*

### 3.6.2 CSD of the One-Shot Delay Model

The trajectory of Eq. (3.37) is determined by the following equations if the conditions for the existence and uniqueness of solutions for the nonlinear difference equation are met:

$$\mathbf{x}(k) = \mathbf{x}_0 + \sum_{i=0}^{k-1}\mathbf{f}(i,\mathbf{x}(i),\mathbf{u}(i)). \tag{3.44}$$

For the existence of a unique trajectory there must exist finite constants $T, r, h, k$ such that

$$\|\mathbf{f}(k,\mathbf{x}(k),\mathbf{u}(k)) - \mathbf{f}(k,\mathbf{y}(k),\mathbf{u}(k))\| \le k\|\mathbf{x}-\mathbf{y}\|, \quad \forall\mathbf{x},\mathbf{y}\in B,\ \forall k\in[0,K]$$

$$\|\mathbf{f}(k,\mathbf{x}_0(k),\mathbf{u}(k))\| \le h, \quad \forall k\in[0,K]$$

where $B = \{\mathbf{x}\in R^n : \|\mathbf{x}-\mathbf{x}_0\| \le r\}$.
Then, Eq. (3.37) has exactly one solution over $[0,\delta]$ whenever

$$h\delta e^{k\delta} \le r \ \text{ and } \ \delta \le \min[T, \frac{\rho}{h}, \frac{r}{h+kr}] \ \text{ for some constant } \rho < 1.$$

Let $k_0, k_f$, and $N$ denote the indices of delay/failure occurrence time, the mission completion time, and the period of delay measured in sampling periods, respectively. Then, the dynamic equation of a one-shot delay model for nonlinear control systems described by Eq. (3.35) is:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \mathbf{f}[\, k, \mathbf{x}(k), \{(\mathbf{u}(k) + (\mathbf{u}(k_0) - \mathbf{u}(k))\Pi_{k_0}(N)\}\,].$$

To test if the constraints at time $k_0 + N$ and $k_f$ are met, one must derive $\mathbf{x}(k_0 + N)$ and $\mathbf{x}(k_f)$ as:

$$\begin{aligned} \mathbf{x}(k_0 + N) &= \mathbf{x}(k_0) + \sum_{i=k_0}^{k_0+N-1} \mathbf{f}(i, \mathbf{x}(i), \mathbf{u}(k_0)) \\ \mathbf{x}(k_f) &= \mathbf{x}(k_0) + \sum_{i=k_0}^{k_0+N-1} \mathbf{f}(i, \mathbf{x}(i), \mathbf{u}(k_0)) + \sum_{i=k_0+N}^{k_f-1} \mathbf{f}(i, \mathbf{x}(i), \mathbf{u}(i)). \end{aligned}$$

## 3.7 Examples

To demonstrate the concept of CSD, we derive the deadlines for eight simple, yet practical, example systems. The former four examples (7.1–4), among which the first two ones calculate control system deadlines via stability analysis, consider only the effects of the computation-time delay with a perfect error detection coverage. The latter four examples (7.5–8) involve the effects of an imperfect error detection coverage, i.e., the derivation of the CSD in the presence of stationary occurrences of input delays/disturbances due to computer failures.

**Example 3.7.1:** Consider a simple controlled process:

$$x(k+1) = 1.05x(k) + 1.8u(k), \quad \text{where } \mathbf{Q} = 2,\ \mathbf{R} = 7.$$

This system is unstable without any feedback control but is controllable. The optimal feedback control is given by:

$$u(k) = \left[R^{-1}B^T P^{-1}(k+1) + BP^{-1}(k+1)B^T\right]^{-1} Ax(k) = Fx(k), \tag{3.45}$$

where $P(k)$ is the solution of the discrete Riccati equation:

$$P(k) = Q + A^T P(k+1)\left[I + BR^{-1}B^T P(k+1)\right]^{-1} A, \quad P(k_f) = 0.$$

Hence, we obtain a steady-state feedback gain $F = -0.3594$ by plugging $P(k) = P(k+1)$ into the above equation (making the *algebraic* Riccati equation), that is, $u(k) = -0.3594x(k)$

| N | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $\|\lambda\|$ | 0.4729 | 0.9589 | 1.1198 | 1.1811 | 1.2049 | 1.2128 |

**Table 3.1:** Relation between the pole position and $N$ when $P = q_N = 1$.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $P = 0.3$ | $\beta$ | 0 | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 | 1 |
| | $\|\lambda\|$ | 0.5596 | 0.5642 | 0.5731 | 0.5818 | 0.5903 | 0.5985 | 0.6024 |
| $P = 0.7$ | $\beta$ | 0 | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 | 1 |
| | $\|\lambda\|$ | 0.7917 | 0.7970 | 0.8076 | 0.8178 | 0.8255 | 0.8375 | 0.8423 |

**Table 3.2:** Relation between the pole position and the probability distribution of delays when $N = 3$.

as $k \to \infty$. The system is stabilized by the feedback control that results from minimizing the performance index $J$ (Eq. (3.9)), since the pole position $\lambda$ is changed from 1.05 to 0.4031. The relation between the pole position and $N$ for the worst case [3] is given in Table 3.1, yielding the CSD $D = 4$ since the pole moves outside the unit circle beginning at $D = 4$.

The pole position is affected by the probability distribution of delays as well as the magnitude of the maximum delay ($NT_s$), which is shown in Table 3.2 where $q_1 = 1 - P$, $q_2 = P(1 - \beta)$, and $q_3 = P\beta$. Since the pole in the worst case or when $q_N = 1$ is located at 0.9589 for $N = 3$, the pole in all other cases (i.e., $q_3 < 1$), for example $\lambda = 0.9033$ when $P = q_2 = 1$, $q_3 = 0$, must reside inside of the unit circle. However, the pole approaches the unit circle quickly as the probability of a large delay increases.

**Example 3.7.2:** For a more practical example of stationary occurrence of delays/failures, the dynamic behavior of the altitude of a spinning satellite is described in terms of the long-term control of the roll ($\varphi$) and yaw ($\psi$) angles, which is based on the dynamic coupling resulting from the rotation of the satellite around the earth:

$$\dot{\varphi}_x = 2.1\varphi_x + 1.5\psi_z + 0.1u_x + 0.2u_z, \tag{3.46}$$

$$\dot{\psi}_z = -2.3\psi_z + 0.2u_x + 0.1u_z, \tag{3.47}$$

where the coefficients depend upon the orbital frequency, i.e., the angular velocity of the satellite with respect to the inertial frame, and $u_x$ and $u_z$ are control signals. The goal of

[3] The 'worst case' means the periodic occurrence of the largest delay possible, that is, deterministically $q_N = 1$, while updating the control input only once every $N$ sampling intervals.

| N | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|
| $\|\lambda_{max}\|$ | 0.3017 | 0.5219 | 0.7732 | 1.0381 | 1.3055 | 1.5682 |

**Table 3.3:** Maximum magnitude of eigenvalues.

the control is to maintain a desired orientation of the satellite in the orbit around the earth (the stabilization problem) with the minimum-control effort, which can be represented by a quadratic performance index (Eq. (3.9)) of the state and the control input. Discrete state equations are derived from Eqs. (3.46) and (3.47) with $T_s = 1$ second. The corresponding coefficient matrices are then:

$$\mathbf{A} = \begin{bmatrix} 8.1660 & 2.7490 \\ 0 & 0.1003 \end{bmatrix}; \ \mathbf{B} = \begin{bmatrix} 0.5470 & 0.7855 \\ 0.0782 & 0.0391 \end{bmatrix}; \ \mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}; \ \mathbf{R} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}.$$

One can derive the optimal (feedback) control gain matrix, $\mathbf{F}$:

$$\mathbf{F} = \begin{bmatrix} 4.7623 & 1.6249 \\ 6.6508 & 2.2661 \end{bmatrix}.$$

This feedback control changes the eigenvalues from $\{8.166, 0.1003\}$ to $\{0.1102 \pm 0.0045j\}$, thus stablizing the satellite. Let the rate of failure occurrence be $\frac{1}{500}$ per hour. Then, the change of poles as a result of incrementing $N$ is derived for the occurrence of the largest delay possible ($P = q_N = 5.556 \times 10^{-7}$) and is given in Table 3.3.

Since the CSD $D = 8T_s$, the controller computer must have some mechanism of fault-tolerance, which can recover from any controller-computer failure within $8T_s$ in order not to lose the (asymptotic) stability in rotating the satellite. Whereas the previous examples calculated control system deadlines via stability, the following examples will calculate control system deadlines using state constraints.

**Example 3.7.3:** Consider the system of a double integrator [21], whose sampled model with $T_s = 0.01s$ is:

$$\mathbf{x}(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} u(k),$$

where the constraints on the control and state for $k_0 \leq k \leq k_f$ are given by:

$$\Omega = \{u(k) \ : \ -1 \leq u(k) \leq 1\}, \ \mathbf{X}_A^1 = \{(x_1(k), x_2(k)) \ : \ -25 \leq x_1(k) \leq 25, \ -5 \leq x_2(k) \leq 5\}.$$

The terminal state must belong to the set $\mathbf{X}_A^f = \{(x_1, x_2) \ : \ |x_i| \leq 0.2, \ i = 1, 2\}$. From these constraints, one can find a simple (non-maximal) $\Omega$-*invariant* set $\mathbf{X}$ which is a polyhedron

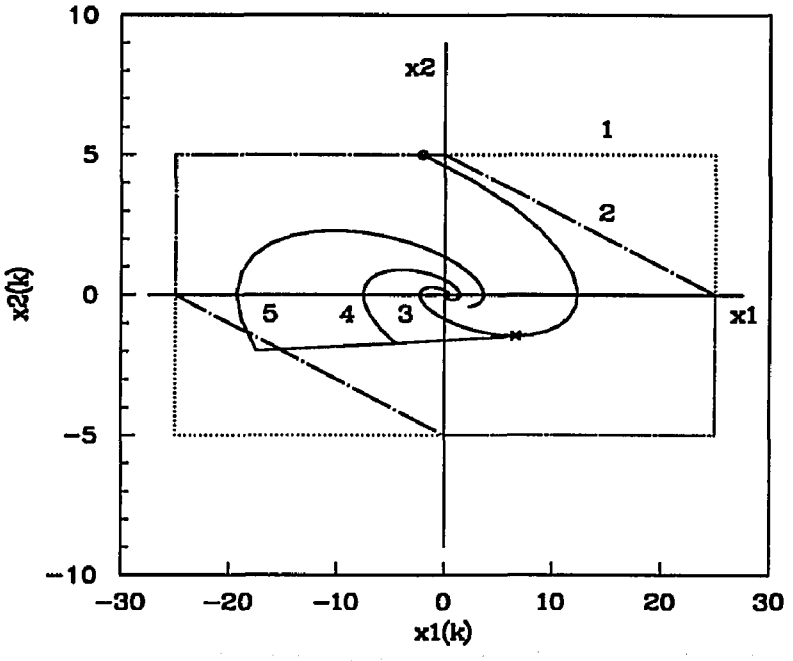



**Figure 3.4:** State trajectory in the absence of delay, and the state trajectories in the presence of one-shot delay equal to the CSD with and without terminal constraints.

defined by the vertices: $v_1 = (0,5)$, $v_2 = (25,0)$, $v_3 = (25,-5)$, $v_i = -v_{i-3}$, $i = 4,5,6$. A set $\mathbf{X}$ is said to be $\Omega$-*invariant* with respect to $\mathbf{G}$ if $\mathbf{X} \subseteq \mathbf{G}$ such that (i) $\forall \mathbf{x}(k_0) = x_0 \in \mathbf{X}$, $\exists u(k) \in \Omega\ \forall k$, such that if $\mathbf{x}(k_0) = \mathbf{x}_0$ then $\mathbf{x}(k) \in \mathbf{G}\ \forall k$ and $\lim_{k\to\infty} \mathbf{x}(k) = 0$, and (ii) $\mathbf{x}(k) \in \mathbf{X}\ \forall k$. An $\Omega$-invariant set clearly belongs to the allowed state-space $(\mathbf{X}_A^1 \cap \mathbf{X}_A^2)$, since the state $\mathbf{x}(k)\ \forall k \in [k_0, k_f]$ must stay in the given state constraint set $\mathbf{G}$ and satisfy the terminal condition $\mathbf{x}(k_f) \in \mathbf{X}_A^f$ by the convergence property of $\mathbf{x}(k)$. A constant feedback control input was simply derived by using Theorem 3.1 in [21]:

$$u(k) = -0.04x_1(k) - 0.2x_2(k). \tag{3.48}$$

The state constraint $\mathbf{X}_A^1$, $\Omega$-invariant set $\mathbf{X}$ with respect to $\mathbf{X}_A^1$, and the state trajectory in the absence of delay, and the state trajectories in the presence of one-shot delay equal to the CSD with and without terminal constraints are plotted in Fig. 3.4, where the curves 1–5 indicate $\mathbf{X}_A^1$, $\mathbf{X}$ (or an $\Omega$-invariant set), state trajectory in the absence of delay, state trajectories in the presence of delay equal to the CSD with terminal constraints, and without terminal constraints, respectively.

The control system deadlines associated with the states on the trajectory in the absence

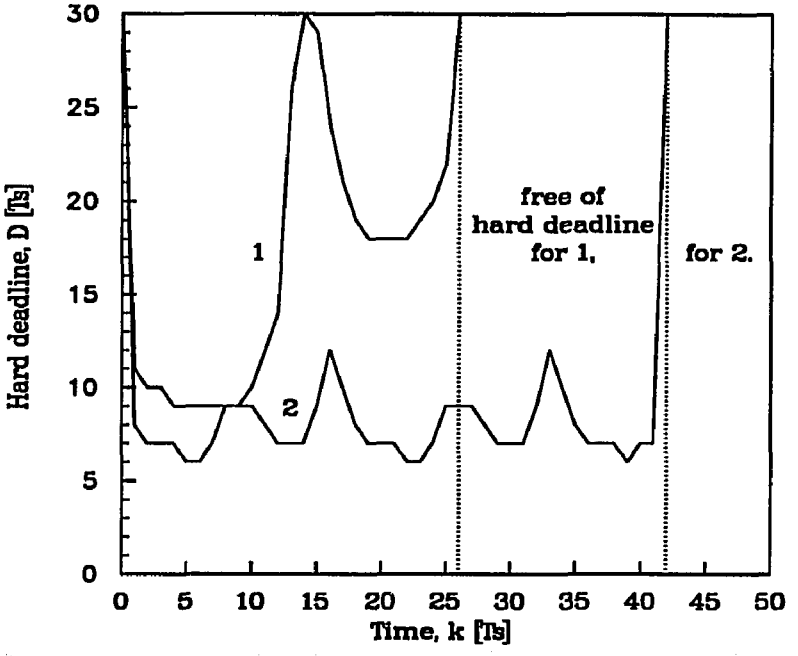



**Figure 3.5:** Control system deadlines of the state trajectory 3 of Fig. 3.4 in the absence of delay, (1) without terminal constraints and (2) with terminal constraints.

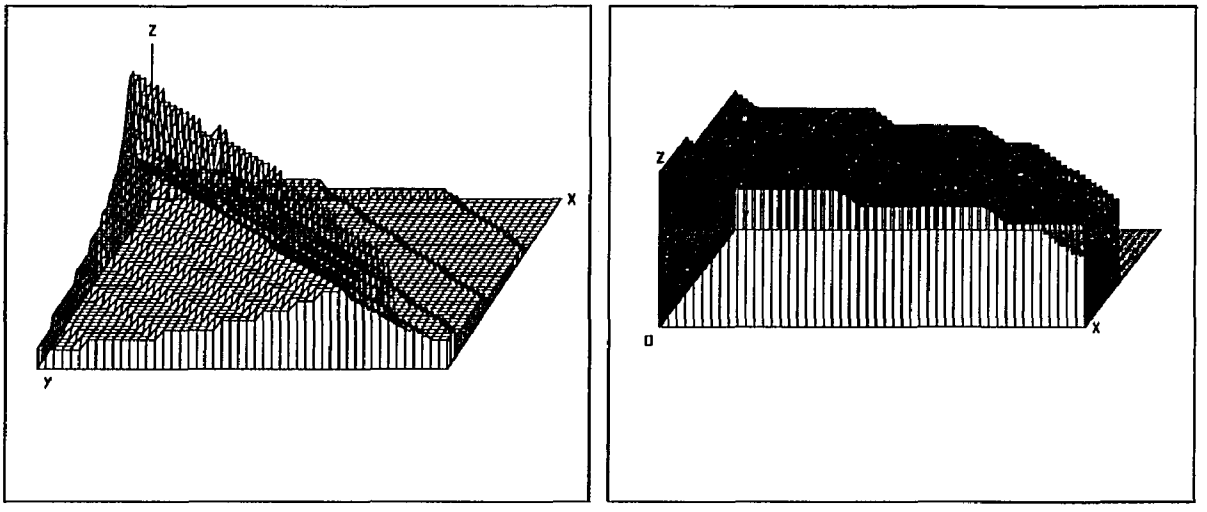
of delay are derived by the algorithm in Section 3.3 for both cases with and without terminal constraints, and are plotted in Fig. 3.5.

In Fig. 3.6, the control system deadlines on the subset, $\{5 \leq x_1 \leq 15;$ (a) $-5 \leq x_2 \leq -1$ and (b) $0 \leq x_2 \leq 2\} \subset \mathbf{X}$, are derived under the assumption that the remaining mission time is $38T_s$ for all states in the subset.

**Example 3.7.4:** The physical meaning of a CSD based on the one-shot delay model can be explained with the real-time controller of a robotic manipulator, where the obstacles in the robot's workplace are translated into state constraints. That is, the system states (robot's positions) must be constrained to avoid collision with the obstacles. In addition to these state constraints, there are usually control input constraints due to the bounds on joint motor torques and energy. In [26], a point robot of Cartesian-coordinate class was modeled by a set of decoupled double integrators:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{u} \end{bmatrix},$$

where $\mathbf{x}$, $\mathbf{v}$, and $\mathbf{u}$ are 2-dimensional position, velocity, and control vectors, respectively,




(a) (b)

**Figure 3.6:** (a) Control system deadlines ($DT_s$) of the region $\{ 5 \leq x_1 \leq 15, -5 \leq x_2 \leq -1 \}$ ($X = x_1$, $Y = x_2$, $Z = N$), where the top and the bottom values of Z–axis are $24T_s$ and $8T_s$, respectively; (b) $\{ 5 \leq x_1 \leq 15, 0 \leq x_2 \leq 2 \}$, where the top and bottom values of Z–axis are $10T_s$ and $T_s$.

and $\mathbf{I} \in \mathcal{R}^{2\times 2}$. The robot's end–effector is guided by the control input determined by:

$$\min_{\mathbf{u}} ||\dot{\mathbf{v}}_d - \dot{\mathbf{v}}||_2 \tag{3.49}$$

subject to the control constraints $|u_i| \leq u_i^{max}$ for $i = 1, 2$, and the state constraints specified by the obstacles. The *Obstacle Avoidance Strategy* (OVS) used in [26] considers the dynamic environment and real-time control needs, where the obstacle-related state constraints are transformed into state-dependent control constraints by mapping each end-effector's position relative to an obstacle into the *P-functionals*:

$$P = (\mathbf{x}^T\mathbf{x})^k - (r^2)^k,$$

where the obstacle is assumed to be centered at the origin and covered by a circle of radius $r$. The $k$ in the P-functional must be chosen such that the set of permissible controls remains nonempty, and the system state and each obstacle's position determine $\mathbf{x} = [x_1 - a, x_2 - b]^T$, where $(a, b)$ is the coordinate of a 2D obstacle. According to Theorem 1 in [26], $k$ is set to 0.5 so that the control input constraint set may remain nonempty over the duration from the detection of a potential collision to the disappearance of this collision danger. A collision-free trajectory is guaranteed if $P(t) > 0\ \forall t$. A potential collision was detected by checking

$$P(\hat{t}) = P(0) + \hat{t}\dot{P}(0) + \frac{\hat{t}^2}{2}\ddot{P}_{max} \geq 0\ \ \forall t\ \Rightarrow\ \ddot{P}_{max} \geq \frac{(\dot{P}(0))^2}{2P(0)}, \tag{3.50}$$

where $\hat{t} = -\dot{P}(0)/\ddot{P}_{max}$ is the time at which the minimum of $P$ occurs. If the condition (3.50) is violated, the instantaneous value of $\ddot{P}_{max}$ is calculated, and an additional constraint (Eq. (3.51)) is included until $\dot{P}$ becomes positive, i.e., the danger of collision disappears.

$$\Phi(\mathbf{x}, \mathbf{v}) = \{\mathbf{u} : \mathbf{P_x}^T\mathbf{u} + \mathbf{v}^T\mathbf{P_{xx}}^T\mathbf{v} \geq \ddot{P}_{max}\}, \tag{3.51}$$

where $\mathbf{P_x} = (\mathbf{x}^T\mathbf{x})^{-1/2}\mathbf{x}$, $\mathbf{P_{xx}} = \frac{m(\mathbf{x})\mathbf{I}}{\mathbf{x}^T\mathbf{x}} - (\mathbf{x}^T\mathbf{x})^{-1/2}$, and $m(\mathbf{x}) = 2k(\mathbf{x}^T\mathbf{x})^{k-1}$. The intersection of these constraints with the admissible control set $\Omega$ results in a polygonal control space:

$$\Lambda = \Phi(\mathbf{x}, \mathbf{v}) \cap \Omega.$$

The *Optimal Decision Strategies* (ODS) in [43] can be used to solve such a constrained minimization problem, similarly to a class of pointwise-optimal control laws constrained by hard control bounds. The CSD is derived using a pointwise-optimal control law with the OVS. Since the computation-time delay causes the failure to update the control input, a collision (or a dynamic failure) occurs if the computation-time delay is longer than a certain

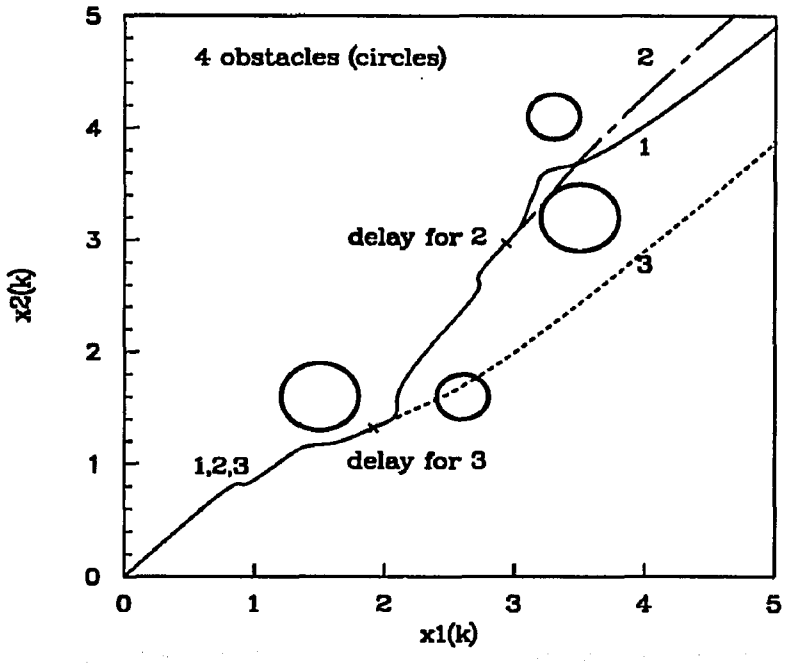



**Figure 3.7:** State trajectories for a point robot with four obstacles, 1) in the absence of delay and 2,3) in the presence of delay equal to $DT_s$.

threshold, which is a function of the system state and time. In this example, the state constraints change with system state (time), i.e., the state-dependent control constraints. Thus, the control input must be updated on the basis of new information to avoid any collision.

The trajectories derived in both the absence and presence of delay $DT_s$, and the control system deadlines on these trajectories in the absence of delay are plotted in Figs. 3.7 and 3.8.

**Example 3.7.5:** To show the CSD of a linear time-invariant system in the presence of stationary occurrences of input delays/disturbances due to computer failures, we consider a simple controlled process:

$$\begin{aligned} x_1(k+1) &= 11.02x_1(k) + 1.08x_2(k) + 3.5u_1(k) \\ x_2(k+1) &= 0.95x_2(k) + 0.5u_1(k) + 1.07u_2(k), \end{aligned} \tag{3.52}$$

where the coefficient matrices of a quadratic performance index are given as:

$$\mathbf{R}_{xx} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}; \quad \mathbf{R}_{uu} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}.$$

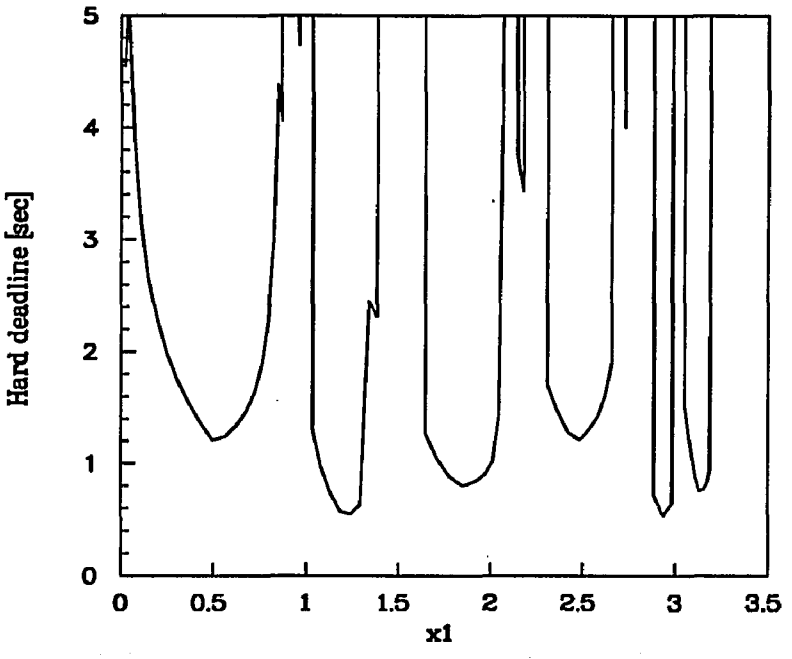



**Figure 3.8:** Control system deadlines of state trajectory 1 of Fig. 3.7 in the absence of delay without terminal constraints.

| $N$ | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| $d = 1$ | 0.7537 | 0.8945 | 0.9659 | 1.0021 | 1.0204 |
| $d = 0.9$ | 0.7579 | 0.9985 | 1.1698 | 1.2922 | 1.3798 |

**Table 3.4:** Maximum magnitude of eigenvalues, $|\lambda_{max}|$.

One can derive the optimal feedback control gain matrix $\mathbf{F}$ that stabilizes the controlled process by solving a discrete Riccati equation as:

$$\mathbf{F} = \begin{bmatrix} 3.1251 & 0.3090 \\ -1.0791 & 0.5512 \end{bmatrix}.$$

This feedback control changes the system eigenvalues from $\{0.95, 11.02\}$ to $\{0.0777, 0.2101\}$. We then derived deterministically the change of poles as a result of iteratively incrementing $N$ for the occurrence of the largest delay possible ($p = q_N = 0.045$). The results are given in Table 3.4, where the first case represents the perfect coverage ($d = 1$) and the second case represents the presence of an input disturbance ($d = 0.9$ and $\mu_{\Delta u} = -5$).

The deterministic value of the CSD is $D = 6T_s$ in the absence of input disturbances under instant failure detection, whereas it decreases to $D = 5T_s$ with some (infrequent) input dis-

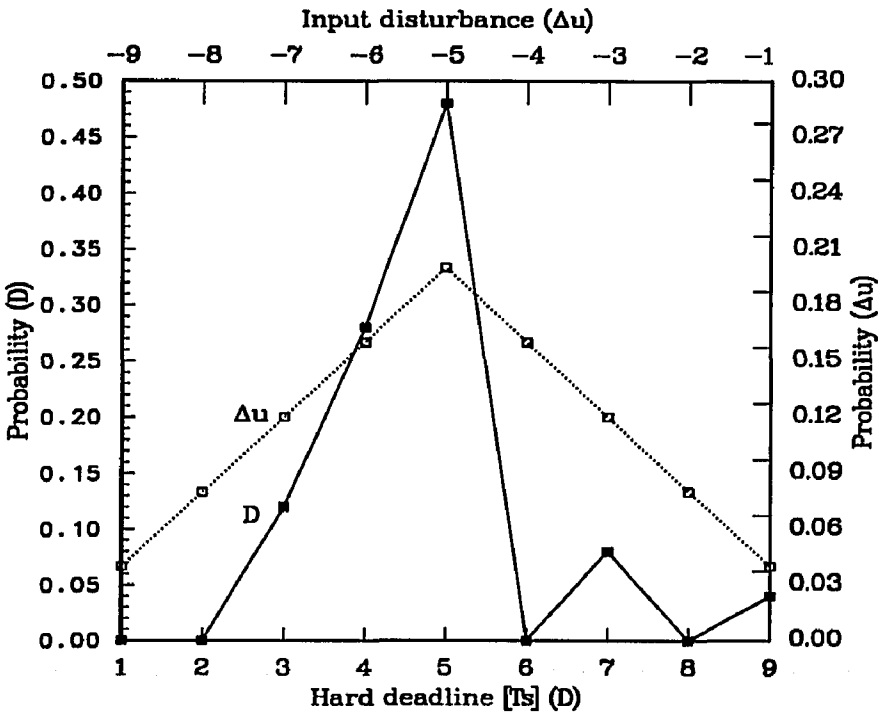



**Figure 3.9:** Probability mass functions of $\Delta u$, which is given *a priori*, and $D$, which is derived.

turbances. The *pmf* of the CSD is plotted in Fig. 3.9 along with the *pmf* of the magnitude of input disturbances.

**Example 3.7.6:** The CSD of a one-shot event model is derived for a double-integrator system which was also used for a one-shot delay model in [56]. The state difference equation of the discretized process with sampling rate, $T_s = 0.01s$, is:

$$\mathbf{x}(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} u(k). \tag{3.53}$$

With the same (state/terminal) constaints and the same feedback control input as those in [56], control system deadlines are deterministically derived in the absence (curve 1)/presence (curve 2: $\mu_{\Delta u} = 10$, curve 3: $\mu_{\Delta u} = -10$) of input disturbances, which are shown in Fig. 3.10.

The *pmf* of control system deadlines at time $T = 15T_s$ is derived for a Gaussian probability density function of $\Delta u$, $q_{\Delta u} = \frac{1}{\sqrt{2\pi}10} e^{-\frac{(\Delta u - 10)^2}{200}}$, and is given in Fig. 3.11. Since the disturbance to the control input is significant, the CSD is likely to be small as shown in the calculation except for $D = 21T_s$, which is the CSD in the absence of input disturbances.

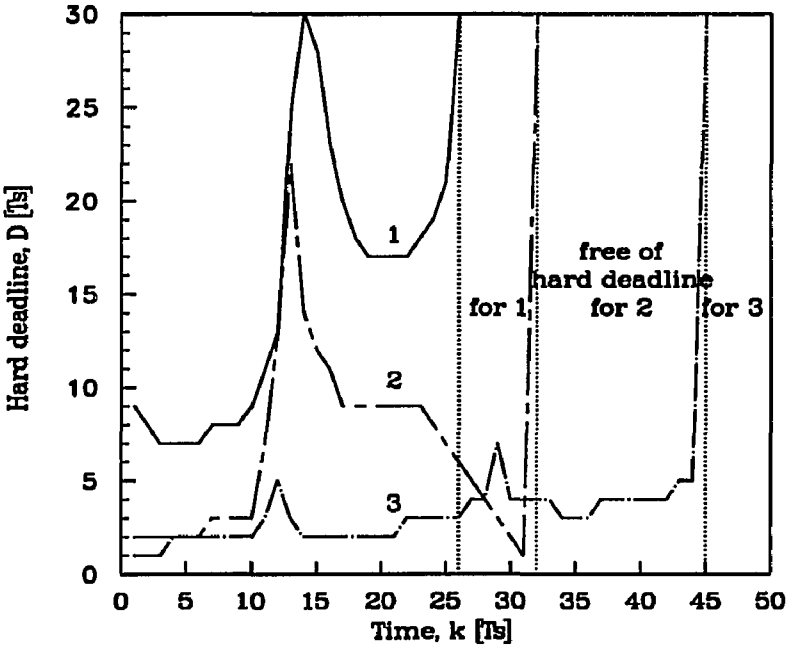



**Figure 3.10:** Control system deadlines of one-shot event model in the absence/presence of input disturbances.

**Example 3.7.7:** The CSD of a nonlinear system in the presence of stationary occurrence of delays is derived for a second-order system described by:

$$\begin{aligned} x_1(k+1) &= 3x_1(k) + x_2(k)^2 - \text{sat}(2x_2 + u(k)) \\ x_2(k+1) &= \sin x_1(k) - x_2(k) + u(k), \end{aligned} \tag{3.54}$$

where the *sat* function is defined as

$$\text{sat}(\rho) = \begin{cases} \rho & |\rho| \leq 1 \\ \text{sign}(\rho) & |\rho| > 1. \end{cases}$$

Then, $\mathbf{f}[k, \mathbf{x}(k), \mathbf{u}(k)]$ in Eq. (3.37) is equal to $[2x_1(k) + x_2(k)^2 - \text{sat}(2x_2 + u(k)), \sin x_1(k) - 2x_2(k) + u(k)]^T$. Control system deadlines around some operating points are given in Table 3.5, for which the optimal feedback control inputs are calculated by a linearization method, and local asymptotical stability around such operating points is examined by using the eigenvalues of linearized equations in the presence of random occurrence of feedback delays (Lyapunov's first method).

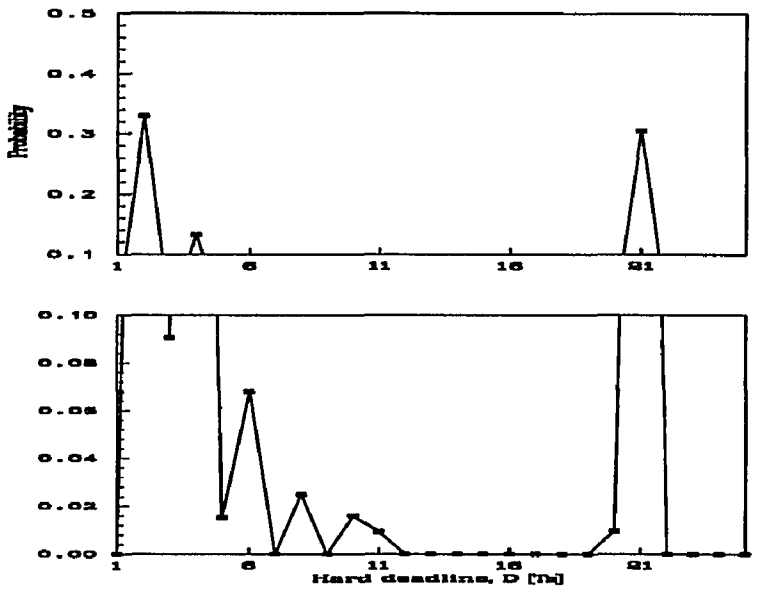



**Figure 3.11:** $Pmf$ of the CSD of one-shot event model when $\mu_{\Delta u} = 10$ and $\sigma_{\Delta u} = 10$.

| $(x_{1o}, x_{2o})$ | $(0,0)$ | $(\pi,0)$ | $(\frac{\pi}{2},0)$ | $(0,1)$ | $(0,-0.24)$ |
|---|---|---|---|---|---|
| $D\ [T_s]$ | 5 | 2 | 3 | 3 | 7 |

**Table 3.5:** Control system deadlines around some operating points ($u_0 = 0$).

**Example 3.7.8:** As an example of the one-shot delay models for nonlinear control systems, we consider the brachistochrone problem with an inequality constraint on the admissible state space. Specifically, a particle is falling in a constant gravitational field $g$ for a fixed time $t_f$ with a given initial speed $x_3(t_0) = x_{30}$. Then, we wish to find a path maximizing the final value of the horizontal coordinate $x_1(t_f)$ with unspecified final values of vertical coordinate $x_2(t_f)$ and the velocity $x_3(t_f)$. The continuous-time system dynamic equations, which were treated in [13], are modified by using a certain sampling period $T_s$ to obtain the following difference equations:

$$\begin{aligned} x_1(k+1) &= x_1(k) + x_3(k)\cos u(k), & x_1(0) &= 0 \\ x_2(k+1) &= x_1(k) + x_3(k)\sin u(k), & x_2(0) &= 0 \\ x_3(k+1) &= x_3(k) + g\cos u(k), & x_3(0) &= 0.05, \end{aligned} \tag{3.55}$$

where $u(k)$ is the control input to drive the particle to an optimal path at $kT_s (0 \leq k \leq 100)$, and $g$ is given by 0.02. The state constraint is described by the state variable inequality, $x_2(k) - 0.4x_1(k) - 20 \leq 0$, $\forall k$, which is converted to a difference equation by introducing a

dummy variable $x_4$:

$$x_4(k+1) = x_4(k) + [x_2(k) - 0.4x_1(k) - 20]^2 W(x_2(k) - 0.4x_1(k) - 20), \qquad x_4(0) = 0$$

where $W(g) = 0$ if $g \leq 0$ and 1 if $g > 0$. The performance index is represented by a modified cost function by including the effect of $x_4(k_f)$ as:

$$J = -x_1(k_f) + \frac{1}{2} S x_4^2(k_f) \qquad k_f = 100.$$

The optimal control input minimizing $J$ is derived by using the gradient method for multistage decision processes, where the Hamiltonian $H$ and the adjoint equation are defined by the system dynamic equation $\mathbf{f}$ and a new vector $\lambda$:

$$\begin{aligned} H &= \lambda^T(K+1)\mathbf{f}[\mathbf{x}(k), u(k), k], \\ \frac{\partial H}{\partial \mathbf{x}(k)} &= -\lambda(k) = \frac{\partial \mathbf{f}^T}{\partial \mathbf{x}(k)} \lambda(k+1), \end{aligned}$$

where the terminal condition on the adjoint equation is $\lambda^T(k_f) = [-1, 0, 0, S x_4(k_f)]$. The control input is updated by an iterative equation (for more detailed derivation of $u_{opt}(k)$, see [54]):

$$u^{N+1}(k) = u^N(k) + \Delta u^N(k), \quad \text{where } \Delta u^N(k) = -K(k) \frac{\partial H}{\partial u(k)}. \tag{3.56}$$

The state trajectories during $[0, 100T_s]$ are plotted in Fig. 3.12, where curve 1 is based on an initial control input ($u(k) = \frac{\pi}{6}$) and curve 2 being close to the optimal path is derived by an optimal control input obtained with 11 iterations of Eq. (3.56) and curve 3 indicates a path of the particle when a long controller-computer failure occurring at $k = 50$ (marked by X). The control system deadlines along curve 2 in the presence of a long one-shot delay are derived as a function of time index $k$, which is shown in Fig. 3.13. As the state gets closer to the boundary of the constraint space, the CSD gets reduced significantly ($61 \leq k \leq 75$). When it leaves the boundary by changing the direction, the system (i.e., a falling particle) instantly enters the non-critical region ($76 \leq k$) — which is free of control system deadlines — since the control inputs from that time do not drive the particle close to the constraint space.

## 3.8 Application of CSD

The hard-deadline information allows us to deduce the timing constraints of the controlled process. It can be used to evaluate the fault-tolerance requirements of a real-time

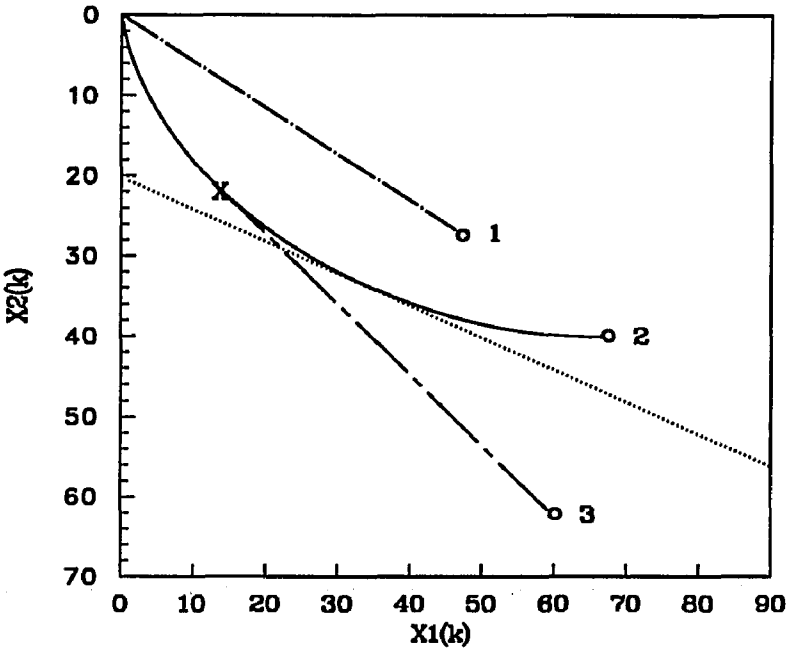



**Figure 3.12:** State trajectories of the brachistochrone problem.

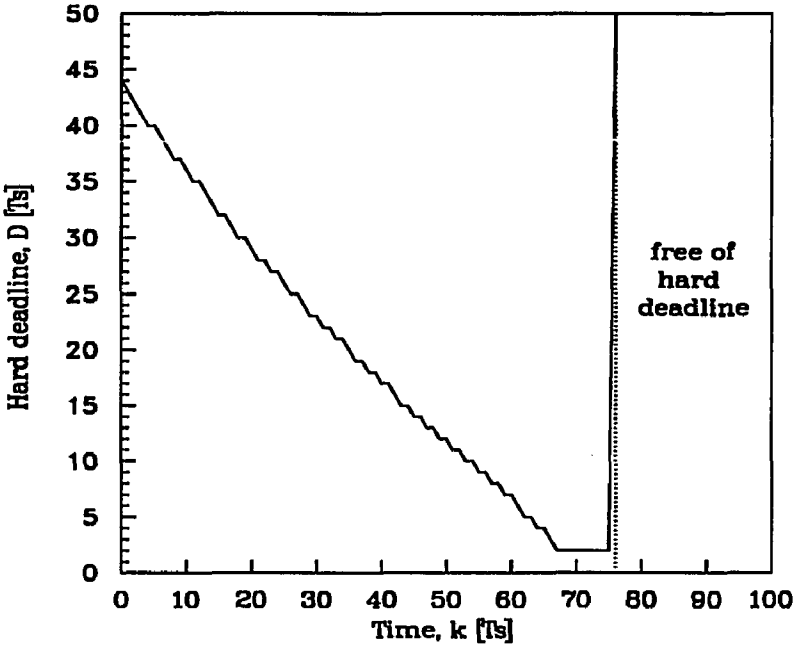



**Figure 3.13:** The CSD along the optimal path as a function of time.

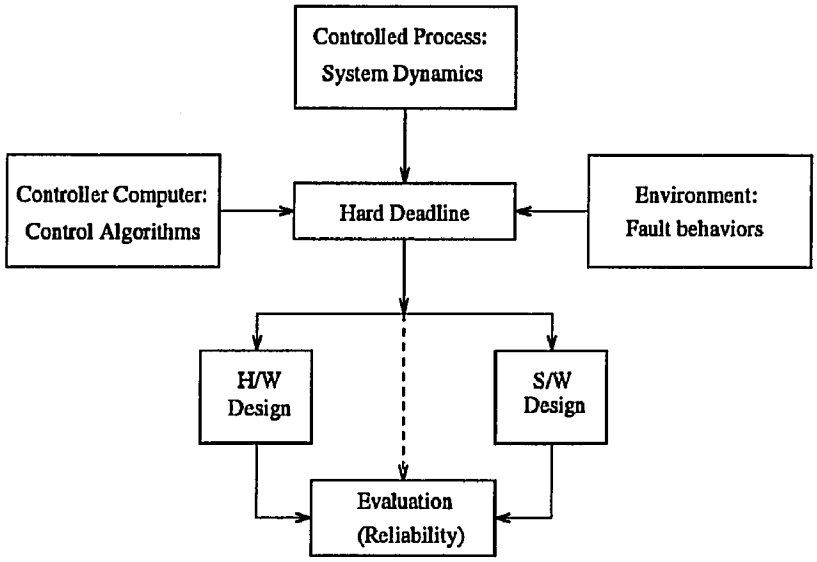



**Figure 3.14:** The source and application of hard-deadline information in a real-time control system.

control system in terms of its timing constraints. Fig. 3.14 shows the source and application of hard-deadline information. As we shall see, this deadline information about the controlled process is quite useful for the design and evaluation of a controller computer.

When designing a controller computer, one has to make many design decisions in the context of controlled processes that are characterized by their control system deadlines and cost functions [59], including:

- hardware design issues dealing with the number of processors and the type of interconnection network to be used, and how to synchronize the processors,
- software design issues related to the implementation of control algorithms, task assignment and scheduling, redundancy management, error detection and recovery.

Since the timing constraints of the controlled processes are manifested as control system deadlines, the deadline information is also essential to evaluate the system reliability, an important yardstick to measure the goodness of the controller computer.

To illustrate the general idea of applying the knowledge of the deadline information (i.e., system inertia), we consider two specific examples; (i) a design example that optimizes time-redundancy recovery methods such as retry or rollback, and (ii) an evaluation example that assesses the system reliability.

**Example 3.8.1:** When an error is detected the simplest recovery method is to re-execute the previous instruction, called simply *retry*, which is effective in case of immediate error detection [34, 38]. When retrying an instruction, one must determine a retry period, which is long enough for the present fault(s) to die away. If the retry does not succeed in recovering from the error, we have to use an alternative recovery method like rollback or restart. So, the retry period must also be short enough not to miss the deadline by considering the amount of time to be taken by the subsequent recovery method in case of an unsuccessful retry. Let $T_t$, $T_a$, and $t_r$ be the "nominal" task execution time in the absence of error, the actual task execution time, and the retry period, respectively. Then, one can obtain a set of samples of $T_a$:

$$T_a \in \{T_t, T_t + \frac{1}{\lambda_a}, (\bar{T} + T_s + t_r) + T_t, (\bar{T} + T_s + t_r) + T_t + \frac{1}{\lambda_a}, 2(\bar{T} + T_s + t_r) + T_t, \cdots\}.$$

where $T_s$, $\bar{T}$, and $\frac{1}{\lambda_a}$ are the resetting time, the mean occurrence time of an error, and the mean active duration of a fault. Since $T_a$ has discrete values, the probability mass function ($pmf$) of $T_a$ is:

$$\begin{aligned} f_{T_a}^k &= Prob[T_a = k(\bar{T} + T_s + t_r) + T_t + \delta\frac{1}{\lambda_a}], \quad 0 \le k \le \infty, \quad \delta \in \{0, 1\} \\ &= p_e^{k+\delta}(T_t)(1 - p_s(t_r))^k(1 - p_e(T_t))^{1-\delta}p_s(t_r)^\delta, \end{aligned} \tag{3.57}$$

where $p_e(T_t)$ and $p_s(t_r)$ are the probability of occurrence of an error during $T_t$ (i.e., after restart) and the probability of a successful retry with a retry period $t_r$.[4] Then, the probability of missing a CSD is:

$$p_{mh}(T_t, D) = \int_0^\infty \sum_{k > \lfloor (D - T_t)/(\bar{T} + T_s) \rfloor} f_{X_a}^k(x) f_D(y) dy, \tag{3.58}$$

where $f_D(y)$ is the probability density function of the CSD. When the CSD is deterministic, $f_D(y)$ is a delta function and the corresponding $p_{mh}(T_t, D)$ becomes simpler. Consequently, the optimal retry period can be determined by minimizing $p_{mh}(T_t, D)$ with respect to $t_r$, using the derived hard-deadline information $f_D(y)$.

Similarly, the hard-deadline information is also useful to rollback recovery, where checkpoints must be placed optimally. The checkpoints are usually placed so as to minimize the mean execution cost [69]. However, the mean cost must be minimized while keeping the *probability of dynamic failure* — the probability of missing a deadline [59] — below a prespecified level in a real-time control system [62]. The hard-deadline information is necessary to compute the probability of dynamic failure, which can, in turn, be used for

[4] See [27] for the derivation of $p_e(T_t)$ and $p_s(t_r)$ in a TMR system.

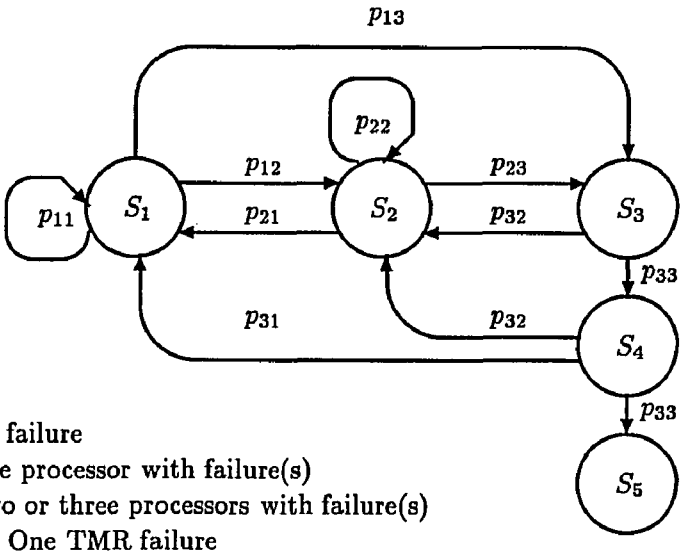
the optimal placement of checkpoints.

**Example 3.8.2:** We consider a simple example of a triple modular redundant (TMR) controller computer in which three identical processors execute the same set of cyclic tasks. The TMR controller computer updates, once every $T_s$ seconds or every sampling period, the control input to the controlled process (plant). That is, the period of each cyclic task is equal to $T_s$. The input of the cyclic task is a discretized output of the plant and the output of the cyclic task will be used to control the plant during the next sampling interval. The output of the TMR controller is correct for each task only if at least two of the three processors in the TMR controller produce correct outputs. A *TMR failure* is said to occur if more than one processor in the TMR controller fail during $T_s$. Thus, the output of the TMR controller would not be correctly updated in case of a TMR failure. The condition for a system (dynamic) failure resulting from controller-computer (TMR) failures[5] is derived from the CSD, which is the allowable maximum duration of TMR failure(s). In other words, this condition gives us the knowledge about the controlled system's inertia against controller-computer failures.

More than 90% of computer failures have been reported to be transient, especially with short active durations [66]. Thus, the controller computer may recover from most failures in a few sampling intervals, and it can correctly update the control input without causing any dynamic failure, if the active duration of controller-computer failure is smaller than the CSD.

Suppose the CSD derived from the controlled process is three sampling periods and a TMR controller computer is used. That is, no dynamic failure occurs if the faults inducing computer failures disappear (or are recovered by a fault-tolerance mechanism) within three sampling periods. Then, the reliability model for this controller computer (Fig. 3.15) is built by extending a Markov chain model whose parameters are to be estimated at a given level of confidence from empirical data. The additional states account for the system inertia, i.e., a dynamic failure results from only three consecutive erroneous (missing the update of) outputs of the controller computer or for a period of $3T_s$, not immediately from one or two erroneous outputs. Without the information of CSD, one can over-estimate the probability of a system failure under the assumption that the system has no delay-tolerance,

[5] As mentioned earlier, the other sources of system failure(s), such as failures in actuators or sensors or mechanical parts and failures of A/D and D/A converters, are not considered in this chapter, because the main intent is to analyze the coupling between a controlled process and its (fault-tolerant) controller computer.




$S_1$: No failure
$S_2$: One processor with failure(s)
$S_3$: Two or three processors with failure(s)
= One TMR failure
$S_4$: Two sequential TMR failures
$S_5$: Dynamic failure

**Figure 3.15:** A Markov reliability model with knowledge of the system inertia.

i.e., one incorrect output can lead to a dynamic failure.

## 3.9 Conclusion

In this chapter, we derived CSDs for linear time-invariant systems in the presence of the computation-time delay and/or the control input disturbances due to a controller-computer failure.

First, when the occurrence of computer failures is stationary or represented by an appropriate probability density function, the CSD is obtained as an integer multiple of the sampling period by using the (asymptotic) stability condition for a modified state equation. Second, a heuristic algorithm is proposed to iteratively compute the CSD as a function of the system state and time by testing for the given (or derived) constraints of control inputs and states.

We presented several examples to demonstrate the derivation of the CSD and its application for the design and evaluation of a fault-tolerant controller computer, which will also be treated in detail in Chapter 6.

# CHAPTER 4

# EVALUATION AND USE OF FAULT-TOLERANCE LATENCIES

## 4.1 Introduction

In this chapter, we propose to evaluate FTL analytically, covering all practical fault-tolerance mechanisms based on the tradeoff between temporal and spatial redundancy. We first investigate the times taken for all individual fault-/error- handling stages. Then, we tailor these results appropriately to represent all possible fault-/error- handling scenarios or policies. (A policy/scenario is composed of sequential fault-/error- handling stages.) Our analysis is based on the assumption that the latencies of fault-handling stages are stochastic, depending upon the random characteristics of fault/error detection (or a random error latency) and fault behaviors; the active duration of a fault affects significantly the success/failure of a spatial-redundancy method (i.e., instruction retry or program rollback). Our results — that focus on a sequence of error-/failure- handling stages — can also be used in those well-developed reliability or dependability models [8, 14, 19].

From the evaluated FTL, one can obtain timing information on the controller computer. That is, the FTL reveals an increase of the execution time of a task in the presence of faults, which in turn enables one to estimate system reliability with timing information on the controlled process (i.e., control system deadlines derived in Chapter 3). Thus, the FTL is to be used as one of design criteria for a fault-tolerant controller computer. In other words, a fault-tolerance policy should be selected and implemented to recover completely from faults/failures within the time limit (deadline) of the underlying controlled process.

In Section 4.2, general fault-tolerance features are described by classifying fault-tolerance mechanisms and considering the tradeoff between temporal and spatial redundancy. Section 4.3 examines the effects on the FTL of individual fault-handling stages from the occurrence

of an error to its recovery, and combines these results to evaluate the FTL of a general fault-handling policy covering all possible fault-handling stages. In Section 4.4 we argue for the importance of FTL information to the design and validation of fault-tolerant controller computers. We present there an example that selects an appropriate fault-handling policy based on the FTL information. The chapter concludes with some remarks in Section 4.5.

## 4.2 Generic Fault-Tolerance Features

Computer system failures occur due to errors, which are deviations from the program-specified behaviors. An error is the manifestation of a fault resulting from component defects, environmental interferences, operator or design mistakes. It is desirable to select an appropriate policy so as to continue the program-specified functions even in the presence of faults.

Fault-tolerance is achieved via spatial and/or temporal redundancy, i.e., systematic and balanced selection of protective redundancy among hardware (additional components), software (special programs), and time (repetition of operations). Thus, design methodologies for fault-tolerant computers are characterized by the tradeoff between spatial and temporal redundancy. Using these two types of redundancy, a fault-tolerant computer must go through as many as ten stages in response to the occurrence of an error, including fault location, fault confinement, fault masking, retry, rollback, diagnosis, recovery, restart, repair and reintegration. The design of fault-tolerant computers involves selection of an appropriate failure-handling policy that combines some or all of these stages.

Spatial redundancy is classified into two categories: static and dynamic. Static redundancy, also known as masking redundancy, can mask erroneous results without any delay as long as a majority of participant modules (processors or other H/W components) are nonfaulty. However, the associated spatial cost is high, e.g., three (four) modules are required to mask a non-Byzantine (Byzantine) failure in a TMR (QMR) system. The time overhead of managing redundant modules — for example, voting and synchronization — is also considerable for static redundancy. Dynamic redundancy is implemented with two sequential actions: fault/error detection and recovery of the contaminated computation. In distributed systems, upon detection of an error it is necessary to locate the faulty module before replacing it with a nonfaulty spare. Although this approach may be more flexible and less expensive than static redundancy, its cost may still be high due to the possibility

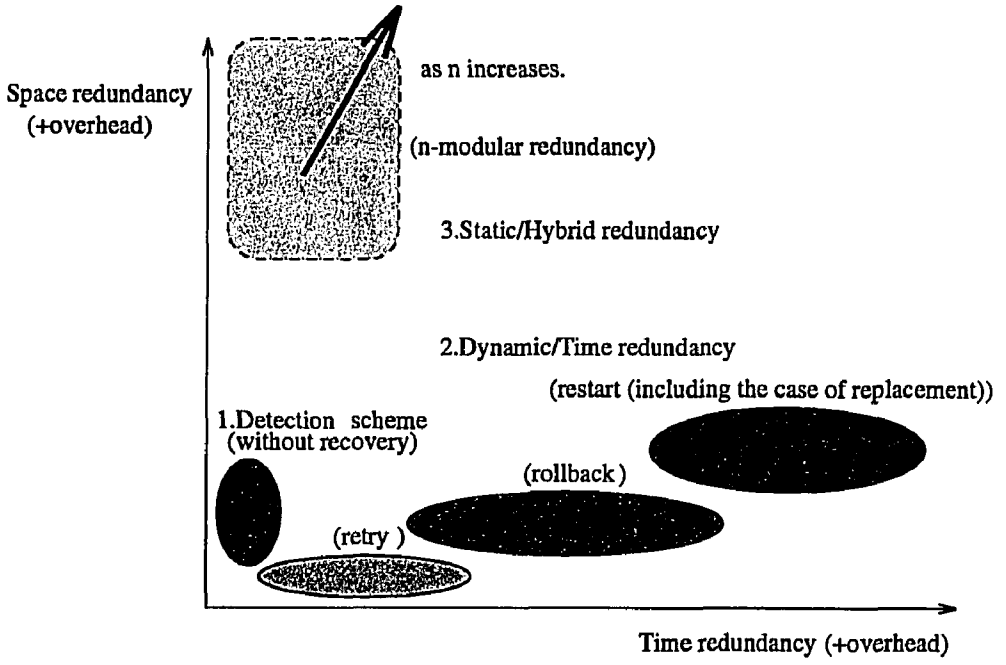



**Figure 4.1:** Tradeoff between temporal and spatial redundancy for various fault-tolerance mechanisms.

of hastily eliminating modules with transient faults[1] and it may also increase the recovery time because of its dependence on time-consuming fault-handling stages such as fault diagnosis, system reconfiguration, and resumption of execution.

To overcome the above disadvantages, temporal redundancy can be used by simply repeating or acknowledging machine operations at various levels: micro-operation/single instruction (retry), program segment (rollback), or the entire program (restart). In fact, one of these recovery schemes is also needed to resume program execution in case of dynamic redundancy. This temporal-redundancy method requires high coverage of fault/error detection so as to invoke the recovery action quickly. (The same is also required for dynamic redundancy.) The main advantages of using temporal redundancy are not only its low spatial cost but also its low recovery time for transient faults. However, the time spent for this method would have been wasted in case of permanent or long-lasting transient faults, which may increase the probability of dynamic failure.

The relations between the temporal and spatial redundancy required (and the associated redundancy-management overhead) are shown in Fig. 4.1 for several fault-tolerance mechanisms. In case of time-critical applications, an appropriate fault-tolerance mechanism can

[1]Note that more than 90% of faults are known to be non-permanent; as few as 2% of field failures are caused by permanent faults [46].

be found from the top left of Fig. 4.1, i.e., paying a small amount of temporal redundancy at the cost of spatial redundancy like $N$-modular redundancy. When the timing constraint imposed by the controlled process is not tight, we can save the cost of spatial redundancy by increasing temporal redundancy (i.e., a larger time for retry, rollback, or restart recovery), which enhances the system's ability in recovering from more transient faults before the faulty modules are replaced with spares. Increasing temporal redundancy, however, increases the possibility of missing task deadlines or dynamic failure.

## 4.3 Evaulation of FTL

The recovery process that begins from the occurrence of an error consists of several stages, some of which depend on each other, and the FTL is defined as the time spent for the entire recovery process. Thus, all the stages necessary to handle faults/failures upon occurrence of an error should be studied and their effects on the FTL must be analyzed.

In a specific application context, the recovery times were estimated in [4, 53] by decomposing the fault recovery process into stages and analyzing the effects of various fault-tolerance features on the FTL. We also use a similar approach to the problem of evaluating the FTL, but for more general fault-tolerance strategies. For completeness, these approaches are summarized below.

### 4.3.1 FTLs of a Pooled-Spares System

In [53], the FTL was estimated for a pooled-spares system implemented in the Dynamic Reconfiguration Demonstration System (DRDS) Program.[2] Phase 1 of the DRDS Program is reported to have shown potential benefits of the dynamic run-time reconfiguration implied by the pooled-spares approach, such as increased functional availability and flexibility, higher reliability, and less complexity than classical $N$-modular redundancy.

This work includes the analysis of FTL and has indicated that the pooled-spares approach can be used for many contemporary applications. The FTL was estimated for the demonstration system and the near-future (5–10 years from today) systems under the assumption that each fault-handling stage — fault detection/isolation, reconfiguration, and recovery — requires time within a deterministic range. In other words, (i) an upper bound of detection/isolation time was determined by using a fail-fast approach with health messages

[2]The DRDS Program is being developed to prove the feasibility of pooled spares for next generation weapon systems by Texas Instruments (TI) Incorporated under a contract from the Naval Air Warfare Center, Indianapolis, IN.

and the system is assumed to use a combination of continuous Built-In-Test (BIT), periodic BIT, and application-level detection/isolation methods, (ii) the reconfiguration times were actually measured on the demonstration system with cold backups for a specific load of size 8K 16-bit words, and (iii) recovery of the application code was performed via application checkpointing and rollback, and the required time was approximated to be a single checkpoint period. The expected reduction of FTL in the next 5–10 years was also estimated by considering such improvement factors as throughput and memory capacities.

In another paper dealing with the pooled-spares system [4], various fault-tolerance techniques covering both software and hardware issues were addressed by focusing on their latencies. This work also analyzed the FTL in the pooled-spares system based on the results of the DRDS Program in [53], and included part of the fault-masking method of $N$-modular redundancy. Possible fault detection/isolation, reconfiguration, recovery, and fault-masking features were considered to examine their effects on the FTL while considering the CPU speed and the relative rates of fault occurrences in individual components such as memory, I/O, buses, and processors.

Memory or data-path parity checking, error-correcting memory, checksum, reasonableness checks, health messages, data-type checks, watchdog timer, and periodic BIT were given as candidate fault detection/isolation mechanisms, while cold, warm, and hot spares were considered for classifying the system reconfiguration with the estimated data of the download and initialization delays depending upon the program size, bus speed, and CPU speed. Several characteristics of checkpointing such as consistency, independence, programmer-transparency, and conversation were introduced with the methods of software re-execution and rollback, where the time required for this recovery procedure was assumed to be a single checkpoint period as in [53].

Finally, the examples to select appropriate fault-handling policies to meet the given FTL requirement were presented for the cases of using cold, warm, and hot spares, illustrating how the analysis of FTL is applied.

### 4.3.2 The FTL of General Fault-Tolerance Mechanisms

Fig. 4.2 depicts all possible scenarios from an error occurrence to its recovery, covering static/dynamic redundancy, temporal-redundancy methods, and combinations thereof. Each *path* represents one fault-handling scenario which may occur as a result of selecting a fault-handling policy corresponding to the path and the success/fail result of the selected method, depending upon the fault behavior when a temporal-redundancy method is applied.

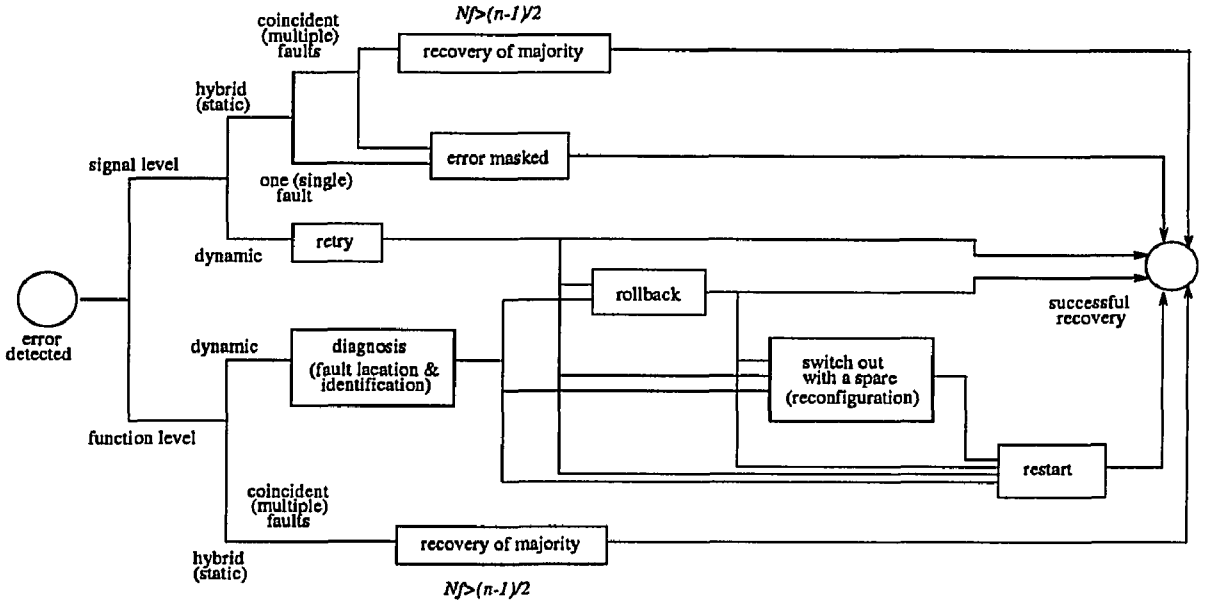



**Figure 4.2:** All possible failure-handling scenarios.

For example, an unsuccessful retry implies another error detection, which may trigger a second retry or rollback or restart. As shown in Fig. 4.2, fault-handing processes are classified by fault/error detection and recovery mechanisms.

First, we divide the fault-handing process into several stages, and evaluate the time spent on each individual stage of Fig. 4.2. When a temporal-redundancy method such as retry or rollback is used upon detection of an error, we need such stages as fault diagnosis, system reconfiguration, and resumption of execution only after the temporal-redundancy method became unsuccessful in recovering from the fault. Whether the temporal-redundancy method is successful or not depends upon the policy used and the underlying fault behavior. Thus, we have to represent the effects of certain stages on the FTL in probabilistic terms.

**Fault/Error Detection Process:** The time interval between the occurrence of a fault and the detection of an error caused by the fault is divided into two parts by the time of error occurrence: *fault latency* for the time interval from the fault occurrence to the error generation, and *error latency* for the time interval from the error generation to the error detection. The distribution of fault latency was estimated in [61] by using the Gamma and Weibull distributions. Since the FTL begins with the occurrence of an error, we are mainly interested in the error latency which depends upon the active duration of a fault and the underlying detection mechanism. Error detection mechanisms are classified

into (i) signal-level detection mechanisms, (ii) function-level detection mechanisms, and (iii) periodic diagnostics.

Let $t_{el}$ and $F_{el}(t)$ be the error latency and its cumulative probability distribution, respectively. Several well-known *pdf*'s, such as Weibull, Gamma, and lognormal distributions, were examined in [16] to model the error latency. If $\Delta t_i$ is the mean execution time of an instruction, $F_{el}(\Delta t_i) \approx 1$ for a high-coverage signal-level detection mechanism. For a function-level detection mechanism, $t_{el}$ depends on the detection mechanism used and the executing task. The coverage of the function-level detection is generally lower than that of the signal-level detection (and significantly less than one), and its error latency is thus larger than the signal-level detection's. We will therefore use the mean error latency of the function-level detection which is much larger than the signal-level detection's for the examples presented in Section 4.4. Although the periodic diagnostic whose coverage depends on both its period and duration is also a popular method to locate faults, only the first two types of detection mechanisms are considered in our analysis, because the results thus obtained can be extended to the case of periodic diagnostic.

**Fault Masking (Static/Hybrid Redundancy):** This method filters out the effects of faulty modules as long as the number of faulty modules is not larger than $\frac{n-1}{2}$ for $n$-modular redundancy. The method induces the time overhead of redundancy management such as synchronization and voting/interactive consistency techniques even in the absence of faults, which increases with the degree of redundancy [35]. Although the time required for this type of recovery is almost zero it induces high spatial costs; when the number of modules available is limited, this method is not as reliable as the dynamic-redundancy method [53] and must be equipped with separate detection and recovery mechanisms for ultra-reliable systems [9].

A hazardous environment, like the one resulting from EMI, will affect the entire system and induce coincident, or common-source, faults in the multiple modules of an $n$-modular redundant system. If the number of faulty modules is larger than $\frac{n-1}{2}$ in such a harsh environment, then the recovery time, which also depends upon the adopted temporal- or spatial- redundancy method, is no longer negligible. We do not treat such a case because of the similarity of its recovery process to the case of dynamic redundancy.

**Fault Diagnosis:** When a function-level detection mechanism is used, upon detection

of an error it is necessary to locate the faulty module[3] and/or to determine certain fault behaviors. Let $t_d$ and $p_d$ be the time spent for fault diagnosis and the probability of locating the faulty module (i.e., diagnostic coverage). Then, there is a tradeoff between $t_d$ and $p_d$, which is usually difficult to quantify. The accuracy of diagnosis, which increases with the diagnosis time, affects greatly the results of the subsequent recovery and hence the FTL. Note that the time, $t_d$, taken for diagnosis is likely to be deterministic, because it is usually programmed *a priori*. We assume that $t_d$ is sufficiently large to locate faulty modules, that is, $p_d \approx 1$.

**System Reconfiguration:** When a fault is located and identified as a permanent fault, the faulty module must be isolated from the rest of the system by replacing it with a spare module or switching it off without replacement (thus allowing for graceful degradation). This process is necessary for both dynamic and hybrid redundancy. Specific hardware like the Configuration Control Unit (CCU) in FTMP [23], may be dedicated to handling system reconfiguration. This process (of using cold spares) generally consists of (i) switching power and bus connections, (ii) running built-in-test (BIT) on the selected spare module, (iii) loading programs and data, (iv) initializing the software. When warm spares are used, steps (i) and (ii) are not needed. The time taken for this process is also likely to be deterministic, which depends upon program size, system throughput, processor speed, and bus bandwidth. Let $t_r$ be the time spent for system reconfiguration. We assume that $t_r$ lies in a deterministic interval, $t_{r1} \leq t_r \leq t_{r2}$, where $t_{r1}$ and $t_{r2}$ are determined by the type of reconfiguration and several other factors described above. In fact, these values can be determined experimentally as was done in [4, 53].

**Retry:** This is the simplest recovery method using temporal redundancy, which repeats the execution of a micro-operation or instruction. To be effective, this method requires immediate error detection, i.e., almost perfect coverage of a signal-level detection mechanism yielding an error latency smaller than the execution time of a micro-operation or an instruction ($F_{el}(\Delta t_i) \approx 1$). The *retry period*, which is defined as a continuous-time interval or the number of re-executions, is the maximum allowable time for retry. In other words, a retry must be terminated when the retry period expires, regardless whether it is successful or not. Let $t_{rp}$, $t_a$, and $F_a(t)$ be the retry period, the active duration of a fault, and the probability distribution of the active duration, respectively. The result of a retry depends

[3] in distributed systems

upon $t_{rp}$ and $t_a$. When the retry is successful, the time it took is certainly smaller than the retry period and is equal to the fault duration, $t_a$. However, it is equal to the retry period, $t_{rp}$, when the retry became unsuccessful, and an alternative recovery method will be followed, thus increasing the FTL.

**Rollback with Checkpoints:** The inquality $t_{el} > \Delta t_i$ is allowed in this method, i.e., $F_{el}(\Delta t_i) < 1$. When an error is detected by a signal- or function- level detection mechanism, this method rolls back past the contaminated part of a program following a system reconfiguration in case of dynamic redundancy. It is invoked as the first step of recovery after an unsuccessful retry. The time taken for the rollback process is dependent upon the error latency, the inter-checkpoint interval, the number of checkpoints maintained, and the way checkpoints are selected for rollback. Let $\Delta t_c$ and $N_c$ be the inter-checkpoint interval and the maximum number of checkpoints necessary for rollback recovery, respectively. For simplicity we assume that the inter-checkpoint intervals are simply equidistant. (It is not difficult to extend our method to the case of non equidistant checkpoints, though the notation will become more complex.) When the rollback is successful, the time taken to restore the contaminated segment of a program is larger than the error latency but smaller than the error latency plus one inter-checkpoint interval; that is, equal to $\lceil \frac{t_{el}}{\Delta t_c} \rceil \Delta t_c$, where $\lceil x \rceil$ is the smallest integer larger than $x$. If the fault is active during the entire period of rollback or the contaminated part is larger than the re-executed part of the program, the rollback recovery will fail and the corresponding "wasted" time is equal to $N_c \Delta t_c$.

**Restart:** If too much of the program is contaminated by an error due to a long error latency, its execution is repeated from the beginning. The time (computation loss) taken for the restart process depends upon (i) the time to detect an error and (ii) the types of restart (i.e., hot, warm, and cold restarts) following a system reconfiguration. We use $t_e$ and $F_e(t)$ to denote the error-detection time measured from the beginning of the program execution and the probability distribution of error occurrences in a program (determined by the *pdf* of fault occurrence), respectively.

**Combination of Failure-Handling Stages:** As mentioned earlier, all failure-handling scenarios are described by the paths from error detection to the corresponding recovery in Fig. 4.2. It is clear that a fault-handling policy depends on several mutually exclusive events, where one event represents a scenario and its occurrence depends upon

fault behaviors and the policy parameters. The probability of the occurrence of each event can thus be calculated by using the *pdf* of fault active duration ($F_a$) and the policy parameters such as $\Delta t_c$, $N_c$, or $t_{rp}$. The FTL of a certain fault-handling policy is thus obtained by using the probabilities of all possible events/scenarios and the times spent for those events/scenarios. Note that the time spent for each scenario is obtained by adding the times spent for all fault-handling stages on the path representing the scenario. Likewise we can obtain the probability distribution of a fault-handling policy ($F_l$) as:

$$F_l(t) = \sum_{i=1}^{n} F_l(t|S_i)P(S_i), \tag{4.1}$$

where $S_i$ indicates the $i$-th scenario of a fault-handling policy, and $P(S_i)$ and $n$ are the probability of the occurrence of $S_i$ and the number of all possible scenarios in the selected fault-handling policy, respectively. Eq. (4.1) describes $F_l(t)$ as a weighted sum of conditional distribution functions. Each $S_i$'s conditional distribution function is computed by convolving the probability distribution functions of the times spent for all the stages on the corresponding path or fault-handling policy. The times spent for all possible fault-handling stages are described as deterministic values or random variables with certain probability distribution functions. We will investigate each fault-handling stage individually.

Now, we characterize the fault-handling process into four policies according to the types of error-detection mechanisms and recovery methods combined with temporal and spatial redundancy; (i) restart after reconfiguration, (ii) rollback, (iii) retry, and (iv) retry then rollback. These cover all possible dynamic- and/or temporal- redundancy methods. Specifically, we can describe the fault-handling policies as follows. (Note that the number of all possible scenarios in each fault-handling policy is equal to $n$.)

- Policy 1 ($n = 2$): $S_1$ = successful restart after diagnosis and reconfiguration, $S_2$ = unsuccessful restart due to incorrect diagnosis then repeat.
- Policy 2 ($n = 2$): $S_1$ = successful rollback after diagnosis, $S_2$ = unsuccessful rollback then restart after diagnosis and reconfiguration.
- Policy 3 ($n = 2$): $S_1$ = successful retry, $S_2$ = unsuccessful retry then restart after reconfiguration.
- Policy 4 ($n = 3$): $S_1$ = successful retry, $S_2$ = unsuccessful retry and successful rollback, $S_3$ = unsuccessful retry and unsuccessful rollback then restart after reconfiguration.

While signal-level detection mechanisms can capture the faulty module immediately upon occurrence of an error and can thus invoke retry in Policies 3 and 4, function-level detection

mechanisms — that cause a nonzero error latency and thus require the diagnosis process to locate the faulty module — may be used for Policies 1 and 2. The probabilities of scenario occurrences and the (conditional) distribution functions of the above scenarios are derived by using the variables defined earlier for individual fault-handling stages.

For simplicity, we do not consider the occurrence of an error/failure due to a second fault during the recovery from the first fault. If we need to consider the effects of such an error/failure, we cannot derive a closed-form distribution function of FTL, but can instead derive the moments of FTL by using recursive equations, which can then be used to derive the distribution function of FTL numerically.

Policy 1 is a simple form of dynamic redundancy, i.e., to restart the task from the beginning after identifying and replacing the faulty module with a nonfaulty spare. The first scenario is a successful restart with correct diagnosis. Thus, the probability of its occurrence is equal to that of successful diagnosis ($p_d$), and the time ($t_l$) spent on this scenario becomes:

$$t_l = t_{el} + t_d + t_r + t_e,$$

where $t_d$ and $t_r$ are deterministic variables, and $t_{el}$ is a random variable with the distribution function, $F_{el}$. $t_e$ is also a random variable with a certain conditional distribution function given that an error had occurred during the execution of a task, i.e., $F_e(t|an\ error\ occurred)$. Let $t = t_l - t_d - t_r$, then:

$$\begin{aligned} P(S_1) &= p_d, \\ F_l(t|S_1) &= F_{el}(t) * F_e(t|an\ error\ occurred). \end{aligned} \tag{4.2}$$

Similarly, $P(S_2)$ and $F_l(t|S_2)$ are derived for the second scenario. Since an unsuccessful restart (of the second scenario) wastes more time than the first scenario by the amount of the incorrect–diagnosis time plus the (error) latency for a second error detection due to this incorrect diagnosis, $t_l$ is changed to:

$$t_l = t_{el} + t_d + t_{el} + t_d + t_r + t_e = 2t_{el} + 2t_d + t_r + t_e.$$

Let $t = t_l - 2t_d - t_r$, then:

$$\begin{aligned} P(S_2) &= 1 - p_d, \\ F_l(t|S_2) &= F_{el}(\frac{t}{2}) * F_e(t|an\ error\ occurred). \end{aligned} \tag{4.3}$$

Policies 2 and 3 use rollback with diagnosis and retry upon error detection, respectively. Reconfiguration is also called for if the temporal-redundancy approach became unsuccessful.

Since the first scenario of Policy 2 is a successful rollback, the probability of its occurrence depends on the probability of successful diagnosis ($p_d$), the parameters of rollback ($\Delta t_c$ and $N_c$), the error latency ($t_{el}$), and the fault active duration ($t_a$). For a successful rollback, (i) a faulty module must be identified with correct diagnosis, (ii) $t_{el}$ must be smaller than $N_c\Delta t_c$, which is the maximum allowable time for rollback, and (iii) the fault must disappear within $N_c\Delta t_c$. Let $p_t$ be the percentage of transient faults, then:

$$P(S_1) = p_t p_d F_a(N_c\Delta t_c)F_{el}(N_c\Delta t_c). \tag{4.4}$$

The time spent for this case is simply obtained as:

$$t_l = t_{el} + t_d + \lfloor\frac{t_{el}}{\Delta t_c}\rfloor\Delta t_c.$$

Let $t = t_l - t_d$, then:

$$F_l(t|S_1) = F_{el}(t) * F_m(\frac{t}{\Delta t_c}), \tag{4.5}$$

where $F_m$ is a cumulative probability mass function for $m = \lfloor\frac{t_{el}}{\Delta t_c}\rfloor$. The probability of the second scenario being exclusive of the first one is equal to $1 - P(S_1)$:

$$P(S_2) = 1 - p_t p_d F_a(N_c\Delta t_c)F_{el}(N_c\Delta t_c). \tag{4.6}$$

The time spent for this scenario is increased to:

$$t_l = t_{el} + t_d + N_c\Delta t_c + t_d + t_r + t_e = t_{el} + 2t_d + N_c\Delta t_c + t_r + t_e.$$

Let $t = t_l - 2t_d - N_c\Delta t_c - t_r$, then:

$$\bar{F}_l(t|S_2) = \bar{F}_{el}(t) * \bar{F}_e(t|an\ error\ occurred). \tag{4.7}$$

For Policy 3 which does not require fault diagnosis due to the assumed immediate and correct detection of errors with signal-level detection mechanisms, the probabilities of scenario occurrences and distribution functions are derived similarly to Policy 2. For a successful retry, the error must be detected before contaminating the result of executing the instruction that will be retried ($\Delta t_i$) and the fault must become inactive within $t_{rp}$, if the time spent is $t_{el} + t_a$. Thus,

$$\begin{aligned} P(S_1) &= p_t F_a(t_{rp})F_{el}(\Delta t_i), \\ F_l(t|S_1) &= F_{el}(t) * F_a(t). \end{aligned} \tag{4.8}$$

When a retry is unsuccessful, the time spent for this becomes:

$$t_l = t_{el} + t_{rp} + t_r + t_e.$$

Let $t = t_l - t_{rp} - t_r$, then:

$$\begin{aligned} P(S_2) &= 1 - p_t F_a(t_{rp}) F_{el}(\Delta t_i), \\ F_l(t|S_2) &= F_{el}(t) * F_e(t|\textit{an error occurred}). \end{aligned} \tag{4.9}$$

Policy 4 has three scenarios whose probabilities and distribution functions are obtained by combining those of Policies 2 and 3. The first scenario is a successful retry, for which $P(S_1)$ and $F_l(t|S_1)$ are equal to those of the first scenario in Policy 3 (i.e., Eq. (4.8)). The second scenario is a successful rollback following an unsuccessful retry. Thus, $P(S_2)$ and $F_l(t|S_2)$ can be obtained by modifying Eqs. (4.4) and (4.5) to include the effects of an unsuccessful retry. Let $t = t_l - t_{rp}$, then:

$$\begin{aligned} P(S_2) &= p_t[F_a(N_c\Delta t_c) F_{el}(N_c\Delta t_c) - F_a(t_{rp}) F_{el}(\Delta t_i)], \\ F_l(t|S_2) &= F_{el}(t) * F_m(\frac{t}{\Delta t_c}). \end{aligned} \tag{4.10}$$

The third scenario is to restart with reconfiguration following an unsuccessful retry then rollback when the time spent is $t_l = t_{el} + t_{rp} + N_c\Delta t_c + t_r + t_e$. Thus, if $t = t_l - t_{rp} - N_c\Delta t_c - t_r$ then:

$$\begin{aligned} P(S_3) &= 1 - p_t F_a(N_c\Delta t_c) F_{el}(N_c\Delta t_c), \\ F_l(t|S_3) &= F_{el}(t) * F_e(t|\textit{an error occurred}). \end{aligned} \tag{4.11}$$

With these derived probabilities and conditional distribution functions, we can compute the probability distribution of FTL for each policy from Eq. (4.1).

## 4.4 Application of FTL

A real-time control system is composed of a controlled process/plant, a controller computers, and an environment, all of which work synergistically. A control system does not generally fail instantaneously upon occurrence of a controller failure. Instead, for a certain duration the system stays in a safe/stable region or in the admissible state space even without updating the control input from the controller computer. However, a serious degradation of system performance or catastrophe called a *dynamic failure* (or system failure), occurs if the duration of missing the update (or incorrect update) of the control input due to malfunctioning of the controller computer exceeds the CSD [59]. The CSD represents system inertia/resilience against a dynamic failure, which can be derived experimentally

or analytically using the state dynamic equations of the controlled process, the information on fault behaviors involving environmental characteristics (such as electro-magnetic interferences), and the control algorithms programmed in the controller computers [56].

When an error/failure occurs in a controller computer, the error must be recovered within a certain period, called the *Application Required Latency* (ARL) [53], in order to avoid a dynamic failure. Roark *et al.* [53] presented several empirical examples of ARL for flight control, missile guidance, air data system, automatic tracking and recognition applications. It is important to note that one can derive the ARL analytically using the hard-deadline information [56], because the sum of ARL and the minimum time to execute the remaining control task to generate a correct control input is equal to the system's deadline. Using this information about the ARL/deadline of the controlled process and the FTL of the controller computer, one can select (or design) an appropriate fault-handling policy by making a tradeoff between temporal and spatial redundancy while satisfying the strict timing constraint, FTL$\leq$ARL. One can also estimate the system's ability of meeting the timing constraint in the presence of controller-computer failures, which is characterized by the probability of no dynamic failure using the evaluated ARL and FTL.

We now present an example to demonstrate the usefulness of the evaluated FTL. Consider a pooled-spares system which consists of multiple modules connected to a backplane. The system consists of power supplies, input/output modules, and a set of identical data processing modules, a subset of which are assigned to processing tasks. The remaining modules can be used as spares in case of an error/failure. Let the basic time unit be one millisecond, and let the task execution time in the absence of error/failure and the mean execution time of one instruction be given as $T = 50$ $(= 0.05sec)$ and $\Delta t_i = 0.002$ $(= 2\mu sec)$, respectively. The error latency is assumed to follow an exponential distribution with mean 12 and 0.002 for function- and signal- level detection mechanisms, respectively. The fault occurrence and duration are also governed by exponential distributions, where the mean value of active duration is 0.5, and the percentage of transient faults ($p_t$) is about 0.9. Then, given that an error occurred during $T$, the occurrence time, $t_e$, is uniformly distributed over $T$. The diagnosis time, $t_d$, is 50, which is assumed to yield coverage $p_d = 0.95$. When cold spares are used, it is assumed to take 500 units of time for system reconfiguration. We also assume that this value can be reduced to 100 by using warm spares. When applying rollback recovery, we set $\Delta t_c = 5$ and $N_c = 4$, whereas the retry period, $t_{rp}$, is set to 1. Under these conditions, the probability distribution functions of FTL are evaluated for the four representative policies by using the method developed in Section 4.3. These functions

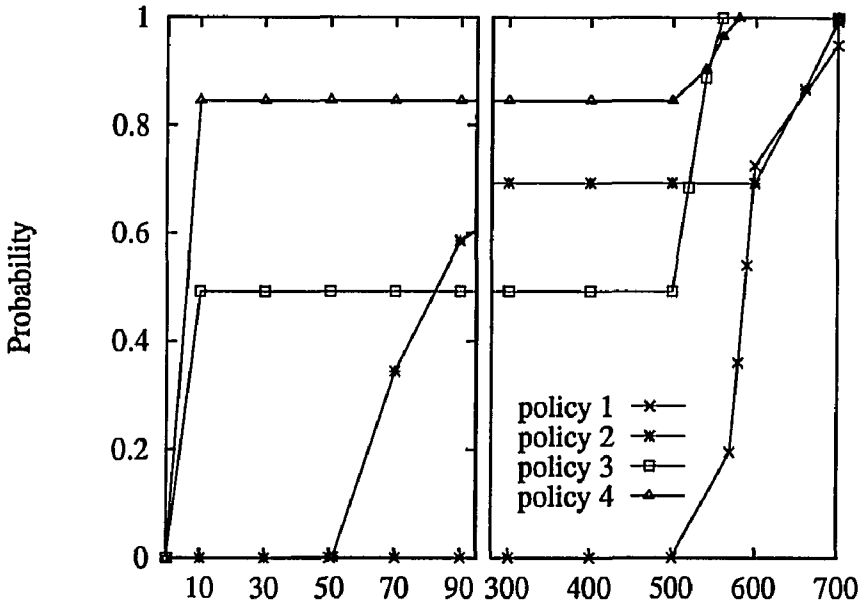



**Figure 4.3:** Probability distribution functions (PDF) of FTL with cold spares.

are plotted in Fig. 4.3.

From the above evaluations of FTL, one can conclude that Policies 1 and 2 are acceptable only when $t_{hd} > 14T(= 700 = 0.7sec)$. While the FTL of Policy 1 is distributed around $12T(= 600)$ with a small variance, Policy 2 has a wide range bounded by $14T$, indicating that Policy 2 is less likely than Policy 1 to violate the timing constraint, FTL$\leq$ARL, under the above chosen conditions. Policies 3 and 4 that use retry have better distributions as compared to Policies 1 and 2, which satisfy the constraint $t_{hd} > 12T$. However, to be effective, retry usually requires dedicated hardware and immediate error detection. (Note that the mean error latency of Policies 3 and 4 in Fig. 4.3 is 0.002.)

Fig. 4.4 plots the FTL while varying the policy parameters. We adopt a more accurate diagnosis process with $p_d = 0.97$, and change rollback and retry policies to $N_c = 5$ and $t_{rp} = 2$, respectively. The error-detection mechanisms are also improved to decrease the error latencies to 10 and 0.001 for both function- and signal- level mechanisms. In this case, the mean values of FTL become smaller, but the upper bounds of FTL are not changed. Thus, we can draw the same conclusion as Fig. 4.3 in selecting an appropriate fault-handling policy.

We also consider different fault parameters: $p_t$ and the mean value of active duration are changed to 0.95 and 0.25, respectively. Since the temporal-redundancy approaches get

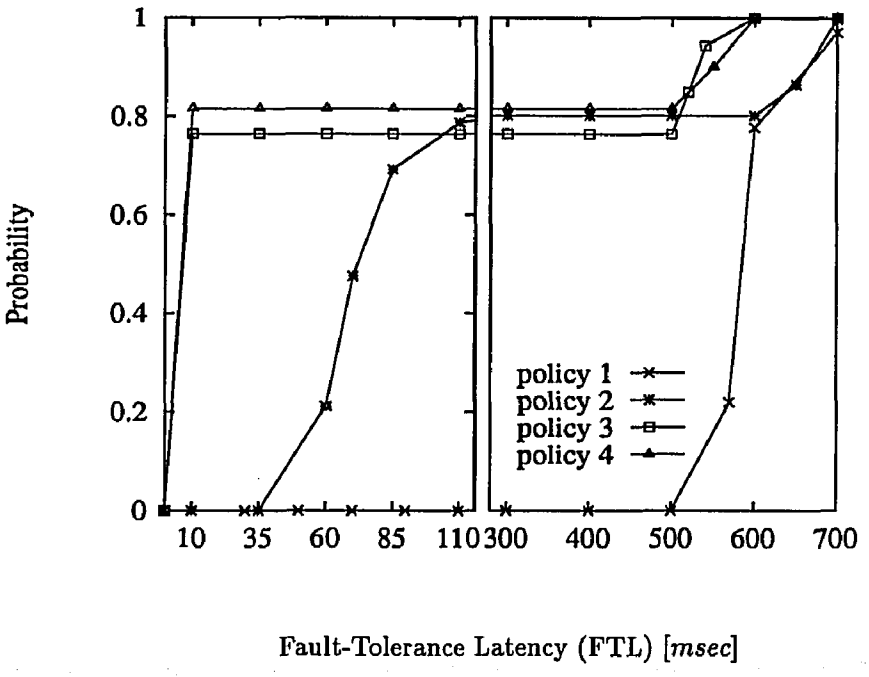



**Figure 4.4:** PDFs of the FTLs with policy parameters different from those of Fig. 4.3: $p_d = 0.97$, $N_c = 5$, and $t_{rp} = 2$.

better under such conditions, the FTLs of Policies 2, 3, and 4 cluster small values, as depicted in Fig. 4.5. However, one cannot still neglect some possible FTLs larger than $10T$, albeit with small probabilities.

When the CSD $t_{hd}$ is tight like $t_{hd} \leq 10T(= 500)$, no policy can meet the timing constraint, FTL$\leq$ARL. It is shown in Figs. 4.4 and 4.5 that the FTL does not change significantly even if the policy and/or fault parameters are changed. Considering the fact that reconfiguration is the most time-consuming among all the fault-handling stages, we use warm spares to reduce $t_r$, which skews significantly the probability distribution functions of the FTLs to the right, as shown in Fig. 4.6 where all parameters but $t_r$ are the same as those in Fig. 4.3. In that case, Policies 3 and 4 are suitable for systems with $t_{hd} > 4T(= 200)$.

If the timing constraint is tighter, e.g., $t_{hd} \leq 4T$, we can conclude that static redundancy (or hot spares) must be used at the expense of spatial redundancy, since no policy using dynamic or temporal redundancy can satisfy the stringent constraint.

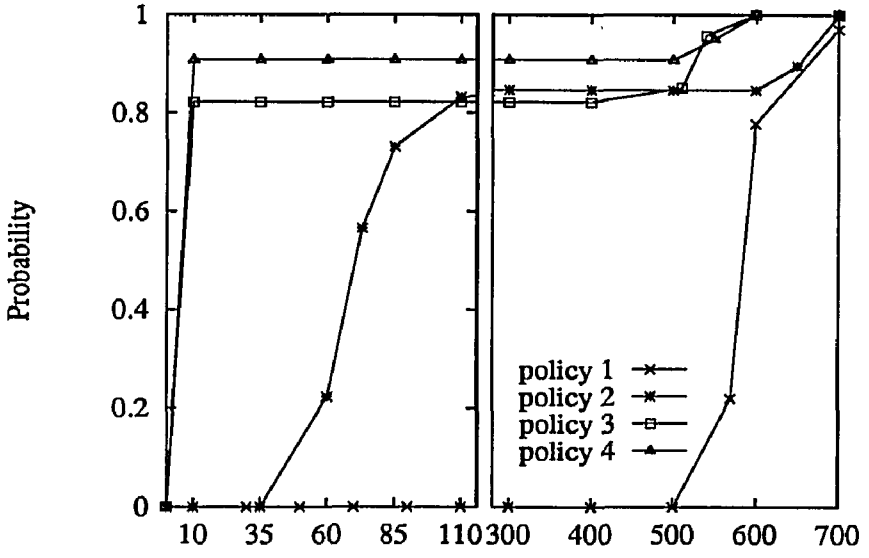



**Figure 4.5:** PDFs of the FTL under fault environments different from those of Fig. 4.3: $p_t = 0.95$ and $E(t_a)$ (the mean active duration)= 0.25.

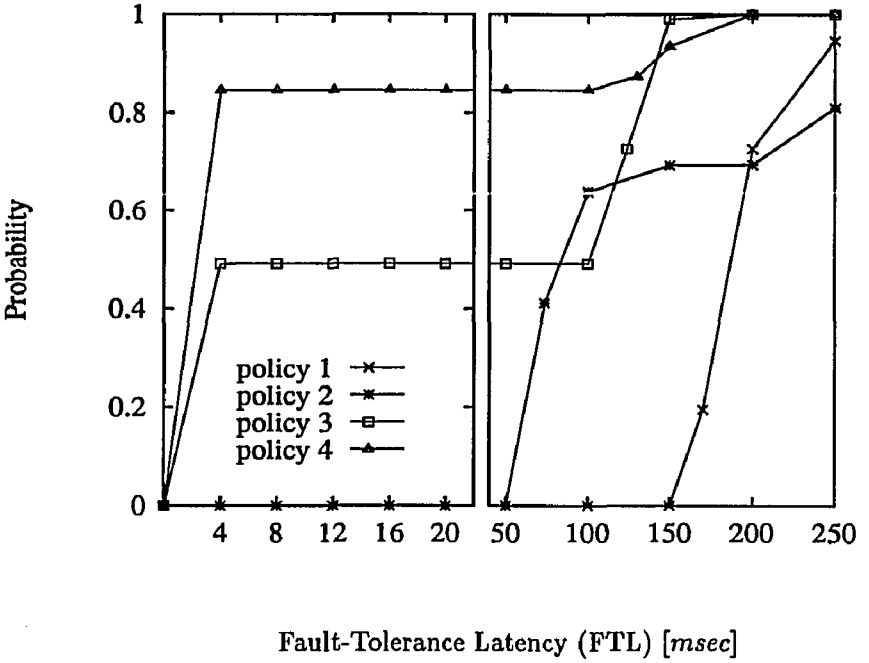



**Figure 4.6:** PDFs of the FTLs with a reconfiguration strategy different from that of Fig. 4.3: warm spares.

## 4.5 Conclusion

In this chapter, we evaluated the FTL for general failure-handling policies that combine temporal and spatial redundancy. We investigated all the individual fault-handling stages from error detection to its complete recovery, and used some deterministic and random variables to model the times taken for these stages. This is in sharp contrast to the previous work that evaluated fault-recovery times with simple models or deterministic data collected from experiments.

As shown in a simple example — although the parameters used in the example are chosen arbitrarily, their choice would not change the conclusion we draw — the evaluated FTL is a key to the selection of an appropriate fault-handling policy, especially for real-time controller computers.

# CHAPTER 5

# EVALUATION OF RECONFIGURATION LATENCIES

## 5.1 Introduction

In Chapter 4, several aspects that affect the reconfiguration latency were mentioned in evaluating the FTL, because reconfiguration is an important error-/failure- handling stage. However, the reconfiguration latency was modeled as a variable lying in a deterministic interval as was done in [53].

In this chapter, we focus on the reconfiguration latency by analyzing the effects of all parameters associated with the reconfiguration process. We classify reconfiguration techniques into four types; reconfigurable duplication, reconfigurable N-Modular Redundancy (NMR), backup sparing, and graceful degradation. For each type of reconfiguration, we describe all the features of the reconfiguration process, which is generally composed of switching in power and bus connections, running power-up BIT on the spare, loading software programs and data from a permanent storage medium to a spare CPU and memory, and initializing software. First, we define several parameters accounting for task sizes, CPU speed, transfer rate of bus/interconnection, the number of interconnections (links) between a faulty module and its replacement module, and the types of spares. We evaluate qaulitative and quantitative effects of these parameters on the reconfiguration latency for the four types of reconfiguration.

In Section 5.2 we specify task sizes, and processor & system capabilities by defining several parameters. Some assumptions required to formalize our analysis are also presented there. In Section 5.3 we investigate the reconfiguration latency of each of the four types of reconfiguration as a function of the parameters defined in Section 5.2. Section 5.4 presents an example of evaluating and using the reconfiguration latency, especifically for backup sparing. The chapter concludes with Section 5.5.

## 5.2 Preliminary

When a failure/error is detected and its source is identified, the system (hardware and/or software) should be reorganized to remove the failed component from active use. Although this process of changing the system organization or the interconnection among its components may be invoked due to a mode change and can occur while the system is non-operational, we focus on dynamic reconfiguration that enables the system to tolerate faults occurring dynamically and randomly during a mission. Obviously, the time spent on dynamic reconfiguration is critical to the reliability of a real-time system, because reconfiguration, which is the most time-consuming stage of fault-/error- handling, is the only way to remove the effects of any permanent fault and because it should be completed within a certain time bound, i.e., the control system deadline. For the purpose of our analysis, we assume prompt and perfect fault detection and isolation (diagnosis).

In this section, we define several parameters affecting the reconfiguration latency. First, the task (application-program) size determines the size of application code and data to be reloaded. (We assume that necessary core operating system components are preloaded.) It is well-known that software download, if required, has the greatest impact on the reconfiguration latency. Let $s_T$ be the task size measured in $K$ bytes. The reconfiguration latency is also dependent on the speed of each individual processor, because the CPU speed greatly affects the program download time as well as the initialization time. Note that the time required for setting up the transfer, the operating system overhead, and the processing time for the transfer are intrinsically sensitive to the CPU speed. Let $s_C$ be the CUP speed measured in MIPS. Bus/Interconnection speed also influences (slightly) the program download time. This speed is determined by bus/interconnection bandwidth, network OS, and memory access time. Let $s_B$ be the bus speed measured in $M$ bytes/second.

In case of graceful degradation, task redistribution is required to transfer the tasks of a faulty module to the remaining active modules, where the transfer time depends upon the system architecture and the adopted reconfiguration algorithm. The reconfiguration algorithm decides which module(s) to take over the tasks of a faulty module. Let $n_R$ be the number of interconnections between the faulty module and the module receiving the remaining tasks.

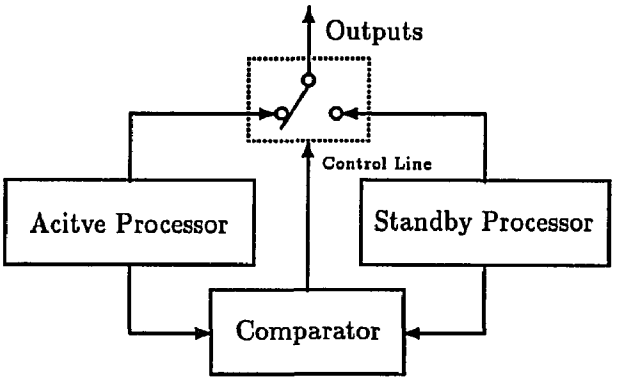



**Figure 5.1:** Structure of reconfigurable duplication.

## 5.3 Evaluation of Reconfiguration Latency

A majority of reconfiguration techniques are included into four classes of dynamic redundancy: (i) reconfigurable duplication, (ii) reconfigurable NMR, (iii) backup sparing, and (iv) graceful degradation. In this section, we derive the reconfiguration latency, defined as $t_{rl}$, by investigating the effects of several parameters such as task sizes, processor capabilities, the system architecture, and the reconfiguration strategy for the four classes of dynamic reconfiguration.

### 5.3.1 Reconfigurable Duplication

A duplicated system is generally used to provide the capability of fault detection by comparing two modules' outputs.[1] When a fault is detected by a mismatch between two outputs, the duplicated system can be reconfigured by disconnecting the faulty module, which can be identified by a certain diagnosis method such as a diagnostic or test program, a watchdog timer, self-checking circuits, and connecting a 'standby spare' running in parallel with the active module. As shown in Fig 5.1, a signal generated by a detected mismatch during comparison triggers reconfiguration through the control line.

Let $t_s$ be the switching time for disconnecting the active module and connecting the standby spare from the output line, and let $t_I$ be the time (overhead) required for initialization. Then, the reconfiguration latency is equal to the time required for switching two modules and initializing the standby module:

$$t_{rl} = t_s + t_I, \tag{5.1}$$

where $t_{rl} = t_s$ if the standby module is always executing the same task for comparison-based

[1] A module is defined as a logical block mapping a binary input vector to a binary output vector.

fault detection. Since the standby spare is generally ready for performing the same tasks as the active module, the process of loading software (program and data) is not necessary. Thus, the reconfiguration latency depends upon switching and/or initialization delay. Note that duplicated modules are generally located on the same bus and the bus interface unit in each module performs the switching function.

### 5.3.2 Reconfigurable NMR

The combination of $N$-modular redundancy and standby sparing, known as hybrid redundancy, is a promising approach to meeting the requirement of high reliability and availability. An NMR core is formed by connecting $N$ identical modules to a majority voter, and several extra modules are used as standby spares. The output of a faulty module differs from the majority-voted result, which is indicated by a disagreement detector. The reconfiguration process of this scheme corresponds to switching the faulty module with one of standby modules, which is also signaled by a disagreement detector.

Let $t_s^{off}$ and $t_s^{on}$ be respectively the time required for cutting off the faulty module and the time required for switching the standby module in power and letting power-up transients settle, where $t_s = t_s^{off} + t_s^{on}$. Let $t_b$ and $t_d$ be the time required for power-up Built-In-Test (BIT) on the standby module and the time required for download and initialization on the spare module, respectively.

A switching strategy decides which modules to be switched in to replace the faulty module in the NMR core. In [65], two switching designs were proposed: (i) a *sequential* switch where all spares are ordered and the $i$-th spare is switched in to replace the $i$-th faulty module and (ii) a *rotary* switch where the spares arrange themselves in numerically increasing order of the voter positions they occupy with the lowest-numbered spare rotating to the highest voter position. The time spent for switching ($t_s^{off}$ and $t_s^{on}$), thus, depends on the adopted switching strategy and the switch complexity that intrinsically depends on the number of spares and the core size. The states of spares are also the key factor in determining the reconfiguration latency. Since a module in an unpowered state probably has a lower failure rate, the standby modules may be unpowered until they are switched in. In that case, the reconfiguration latency significantly increases due to the time ($t_b + t_d$) required to make the selected spare powered up and to load and initialize the software (the application program and/or data of intermediate results) on the spare.

Consequently, we obtain the reconfiguration latency, which depends on the switching

strategy and the states of standby modules:

$$t_{rl} = t_s^{off} + t_s^{on} + t_b + t_d, \tag{5.2}$$

where $t_b$ depends upon the coverage/thoroughness of the BIT, the complexity of the spare, and BIT software (requiring a certain degree of hardware assistance). In Section 5.3.3, we will examine the factors affecting $t_d$ and derive $t_d$ using the parameters introduced in Section 2.

### 5.3.3 Backup Sparing

In addition to hybrid redundancy using the NMR core, the concept of backup sparing can be used in general multiprocessor structures like meshes and hypercubes. For effectiveness of this method, we assume appropriate schemes of fault detection and isolation that will be followed by the replacement of faulty module(s) with spare(s). If any sparing module can be used to replace any other working module, the set of spares are called "pooled spares". The module periodically checkpoints its state on its backups so that a selected backup among the set of pooled spares may have state information to maintain consistency. In other words, the new active module restores the last checkpoint and reexecutes all the operations that were executed by the previously active module since the checkpoint. The new active module can then start executing the remaining tasks and service new requests from the consistent state.

Let $t_w$ be the time spent for selecting a spare to take over the remaining tasks of the faulty module. In case of dedicated spares which associate some spares locally with specific groups of active modules to minimize the interconnection complexity, $t_w$ can be made negligible by using well-controlled procedures. However, it could be considerable depending on the applied reconfiguration strategy for nondedicated spares.

As shown in Fig. 5.2, the reconfiguration process for this class is generally composed of (i) switching power and bus connections, (ii) running built-in-test (BIT) on the selected spare module, (iii) loading programs and data, (iv) initializing the software, among which some steps are not always needed depending on the state of on-line spares. When a spare to be switched in is determined upon fault detection and isolation by an appropriate method, the selected spare is powered up and ready to become an active module. Since unpowered modules are likely to have lower failure rates and the standby power requirements are lower than the active ones, unpowered spares are often kept in the form of *cold spares* despite their large time overhead to become active. Extensive testing can be done during power-up

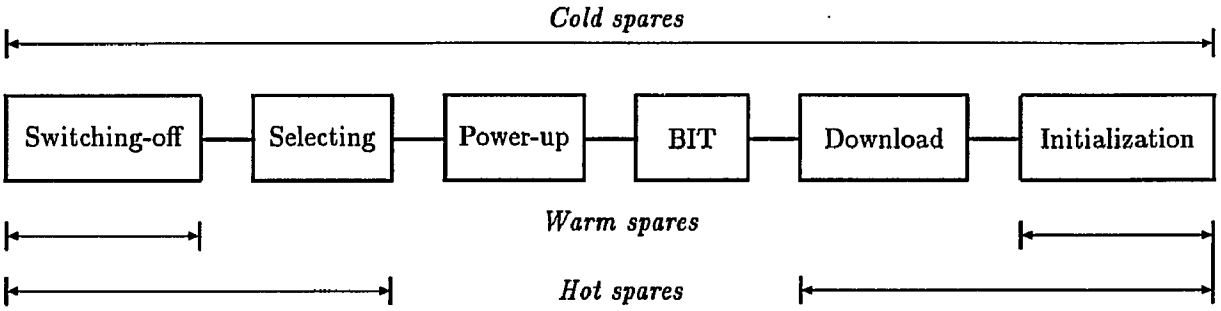



**Figure 5.2:** Reconfiguration steps of backup sparing

before starting any normal operation, such as a comprehensive memory test. As mentioned in Section 5.3.2, the time required for this power-up BIT ($t_b$) depends upon the complexity of the module, the accuracy of BIT, and the degree of hardware support for BIT software. In cold spares, the reconfiguration latency is computed as:

$$t_{rl} = t_s^{off} + t_w + t_s^{on} + t_b + t_d. \tag{5.3}$$

Cold spares require power-up transients to settle and to run BIT to ensure that a healthy (nonfaulty) module is switched in the system, whereas warm spares are kept powered up on-line ready to load and run the application software. Using Fig. 5.2, we get the reconfiguration latency of warm spares:

$$t_{rl} = t_s^{off} + t_w + t_d. \tag{5.4}$$

The software program that is downloaded from a certain permanent storage medium to a spare CPU generally causes the longest delay. Let $t_O$, $t_B$, $t_P$, and $t_I$ be the time required for setting up data transfer plus the operating system overhead, the processing time for the transfer, the transfer time on the bus/interconnection, and the time for initialization, respectively. Then, for a task of size $s_T$, we have:

$$t_d = t_O + t_I + s_T(t_B + t_P), \tag{5.5}$$

where $t_O$, $t_P$, and $t_I$ are sensitive to the change of CPU speed, and $t_B$ is sensitive to the change of bus/interconnection speed. If some specific values are measured for a certain protocol having $s_C^0$ and $s_B^0$ as $\{t_O^0, t_B^0, t_P^0, t_I^0\}$, the estimated values of $\{t_O, t_B, t_P, t_I\}$ for different $s_C$ and $s_B$ are calculated by:

$$\begin{aligned} t_O &= c_C t_O^0, \quad t_P = c_C t_P^0, \\ t_B &= c_B t_B^0, \quad t_I = c_C t_I^0, \end{aligned} \tag{5.6}$$

where $c_C$ and $c_B$ are the coefficients indicating the amounts of change in $\{t_O, t_I, t_P\}$ and $t_B$ by $s_C$ and $s_B$, respectively. Although other factors affect $c_C$ and $c_B$ (for example, $c_C$ and $c_B$ also depend on the inherent synchronization factors and the type of buffering, respectively) we consider only the effects of $s_C$ and $s_B$ because they are most influential. Then, $c_C$ and $c_B$ are inversely proportional to the module throughput (CUP speed $s_C$) and transfer rate of bus/interconnection (bus/interconnection speed $s_B$), respectively. (Note that $c_C$ and $c_B$ are decreased by using better $s_C$ and $s_B$.) Thus, we obtain

$$c_C = \frac{s_C^0}{s_C} \quad \text{and} \quad c_B = \frac{s_B^0}{s_B}. \tag{5.7}$$

For example, from the actual measurements of a pooled-spares system implemented in Phase 1 of the DRDS Program [2] in [48], i.e., 2.2 1750A Digital Avionics Information System (DAIS) $MIPS$ and $s_B = 2.105$ $[Mbytes/second]$ (bus bandwidth 200 $[Mbytes/second]$ for 16 bit bus-width and memory access time 250 $[nanoseconds]$), we obtain:

$$t_O^0 + t_I^0 = 29.8 \ [milliseconds],$$
$$t_B^0 = 0.2375 \ [milliseconds/1Kbyte],$$
$$t_P^0 = 0.3375 \ [milliseconds/1Kbyte],$$

where $c_C = 2.2/s_C$ and $c_B = 2.105/s_B$.

By using Eqs. (5.5), (5.6), and (5.7), we evaluate the estimated value of $t_d$ under the various conditions of $s_T$, $s_C$ and $s_B$. When only the task size $s_T$ varies with $s_C$ and $s_B$ fixed, we obtain a linear equation for $t_d$:

$$t_d = As_T + B, \tag{5.8}$$

where

$$A = t_B + t_P = c_B t_B^0 + c_C t_P^0 = \frac{s_B^0}{s_B} t_B^0 + \frac{s_C^0}{s_C} t_P^0,$$
$$B = t_O + t_I = c_C(t_O^0 + t_I^0) = \frac{s_C^0}{s_C}(t_O^0 + t_I^0).$$

When only the CPU speed $s_C$ varies with $s_T$ and $s_B$ fixed, we obtain a equation containing two constants $A$ and $B$ for $t_d$:

$$t_d = \frac{A}{s_C} + B, \tag{5.9}$$

where

$$A = s_C^0(t_O^0 + t_I^0 + s_T t_P^0),$$
$$B = s_T t_B = s_T c_B t_B^0 = \frac{s_B^0}{s_B} s_T t_B^0.$$

[2] This was mentioned in Introduction.

Likewise, for various $s_B$ values we obtain $t_d$ from:

$$t_d = \frac{A}{s_B} + B, \tag{5.10}$$

where

$$\begin{aligned} A &= s_T s_B^0 t_B^0, \\ B &= t_O + t_I + s_T t_P = c_C(t_O^0 + t_I^0 + s_T t_P^0) = \frac{s_C^0}{s_C}(t_O^0 + t_I^0 + s_T t_P^0). \end{aligned}$$

In case of hot spares that are always on-line executing the target software in parallel with the active hardware (dedicated hot spares) as in Section 5.3.1, the reconfiguration latency reduces to:

$$t_{rl} = t_s^{off} + t_I, \tag{5.11}$$

which is suitable for applications requiring short latencies because all the steps but switching out and initialization are not necessary for hot spares.

### 5.3.4 Graceful Degradation

When an error is detected and the contaminated module is located in a multiprocessing system, the system is reconfigured to isolate the faulty module from the rest of the system. The faulty module may be replaced by a backup spare as discussed in Section 5.3.3, and alternatively, it may simply be switched off, thus degrading the system capability, i.e., *graceful degradation.* This technique uses redundant hardware as part of the normal operating-resources at all times and allows the system performance to degrade gracefully while compensating for failures.

In a complex multiprocessor like a mesh or a hypercube, faulty modules are disconnected upon fault detection and identification, because a faulty module cannot be immediately repaired in many cases (nor replaced in the mode of graceful degradation). The remaining modules should be reconfigured into a functioning connected network of smaller size and/or tasks are also redistributed by assigning the tasks of the failed modules to the remaining modules. For our analysis, we assume that each module can test its neighbor modules to determine their state (fault or fault-free), and the neighboring modules exchange predetermined test information and the intermediate task results at regular intervals. The intermediate results of each module are stored in some of its neighbors and will be used by those neighbors for reconfiguration in case the module fails. We also assume that this procedure of testing and updating the intermediate results is synchronized throughout the system.

| $s_T$ | 1 | 2 | 4 | 8 | **16** | 32 | 64 | 128 |
|---|---|---|---|---|---|---|---|---|
| $t_d$ | 30.38 | 30.95 | 32.1 | 34.4 | **39** | 48.2 | 66.6 | 103.4 |

**Table 5.1:** $t_d$ for various task sizes ($s_T$) [$K$Bytes].

The reconfiguration latency is, thus, equal to the time spent for transferring and initializing the remaining tasks of faulty modules. We assume that it takes $t_w$ for a certain reconfiguration strategy to decide which modules take over the tasks of the faulty modules, as has been done in various system architectures [3, 10, 37, 50, 71]. We define $n_R$ as the number of processor-interconnections between the faulty module and a module taking over the tasks of the faulty module. Let $t_n$ be the time to transfer a unit of task (say 1 [$Kbyte$]) between the faulty module and its replacement module, which is called the *network latency* and depends upon the speed/bandwidth of an interconnection network and the distance (number of interconnections/links to go through). The network latency includes both the overhead to prepare for transferring a task in the source module (address generation, packaging, etc.) and the overhead in the destination module induced due to acknowledging, error check, unpackaging, etc.). If the size of the remaining tasks is $s_T$ and the tasks are transferred in block-data transfer mode, in which one unit of latency is required for a block of data/task elements, the reconfiguration latency is:

$$t_{rl} = t_s^{off} + t_w + t_n + s_T(t_B + t_P), \tag{5.12}$$

where

$$t_n = t_O + t_I + (n_R - 1)t_B,$$

and the effects of $s_C$ and $s_B$ upon $\{t_O, t_I, t_P, t_B\}$ were described in the previous subsection. In the mode of single-data transfer where each data/task element requires one unit of latency and a transfer time, the reconfiguration latency becomes:

$$t_{rl} = t_s^{off} + t_w + s_T(t_n + t_B + t_P). \tag{5.13}$$

## 5.4 Example

In this section, we present an example of evaluating the reconfiguration latency for the demonstration system of [48] using *milliseconds* as the basic time unit. In the measurements of a pooled-spares system implemented in Phase 1 of the DRDS Program using 2.2 DAIS MIPS experimental system, the data of $\{t_O^0, t_B^0, t_T^0\}$ is given as $\{28.8, 0.2375, 0.3375\}$ with the condition of $s_T = 1$ [$Kbyte$], $s_C^0 = 2.2$ [$MIPS$], and $s_B^0 = 2.105$ [$Kbytes/millisecond$].

| $s_C$ | 1 | 2 | **2.2** | 5 | 10 | 50 | 100 | 1000 |
|---|---|---|---|---|---|---|---|---|
| $c_C$ | 2.2 | 1.1 | **1** | 0.44 | 0.22 | 0.044 | 0.022 | 0.0022 |
| $t_d$ | 81.24 | 42.52 | **39** | 19.29 | 11.54 | 5.35 | 4.57 | 3.88 |

**Table 5.2:** $t_d$ for various CPU speeds ($s_C$) [$MIPS$].

| $s_B$ | 1.0525 | **2.105** | 4.21 | 8.42 | 16.84 | 21.05 | 33.68 | 42.1 |
|---|---|---|---|---|---|---|---|---|
| $c_B$ | 2 | **1** | 0.5 | 0.25 | 0.125 | 0.1 | 0.0625 | 0.05 |
| $t_d$ | 42.8 | **39** | 37.1 | 36.15 | 35.67 | 35.58 | 35.44 | 35.39 |

**Table 5.3:** $t_d$ for various Bus speeds ($s_B$) [$Kbytes/millisecond$].

We assume that $t_s^{off} = 1, t_w = 5, t_s^{on} = 80, t_b = 20$, and $t_I^0 = 4$ are given, and $\{t_s^{off}, t_w, t_b, t_I^0\}$ are inversely proportional to CPU speed $s_C$.

First, we begin with evaluating the time required for download and initialization $t_d$, which is most sensitive to task sizes and processor-capability parameters. Under the given condition, $t_d$ is computed as $28.8 + (0.2375 + 0.3375) + 1 = 39$. If we change $s_T$, $s_C$, or $s_B$, then the estimated values of $t_d$ are computed using Eqs. (5.8), (5.9) and (5.10), as given in Tables 5.1, 5.2, and 5.3.

Similarly, we derive $t_{rl}$ for various conditions, i.e., type of spares, CPU speed, and bus speed, with a fixed $s_T = 16$ [$Kbytes$]. Fig. 5.3 plots the value of $t_{rl}$ while varying $s_C$ over three types of spares. Since most steps of system reconfiguration are sensitive to CPU speed, $t_{rl}$ decreases as $s_C$ increases. However, $t_{rl}$ of cold spares is not decreased below a certain value due to insensitiveness of the time required for power-up transients to settle. The $t_{rl}$ values of warm spares (and cold spares) are not scaled directly with the CPU speed (but hot spares are directly scaled down) because the bus/interconnection transfer time $t_B$ is independent of $s_C$ (and also $t_s^{on}$). In Fig. 5.4, we also plot $t_{rl}$ while varying $s_B$. In this case, $t_{rl}$ does not change significantly because only one step in system reconfiguration ($t_B$) is dependent on $s_B$.

Cold spares are generally useful for applications that require low fault rates of spares (faults occur more frequently in powered states) and do not have tight control system deadlines. In this type of spares, it takes more time to become operational due to large $t_s^{on}$ and $t_b$, relative to the time required for other steps of reconfiguration. In Section 5.3, we observed that the time required for most steps with cold (and warm) spares depends on CPU speed. Fast CPU speed significantly decreases $t_{rl}$ of cold or warm spares as shown

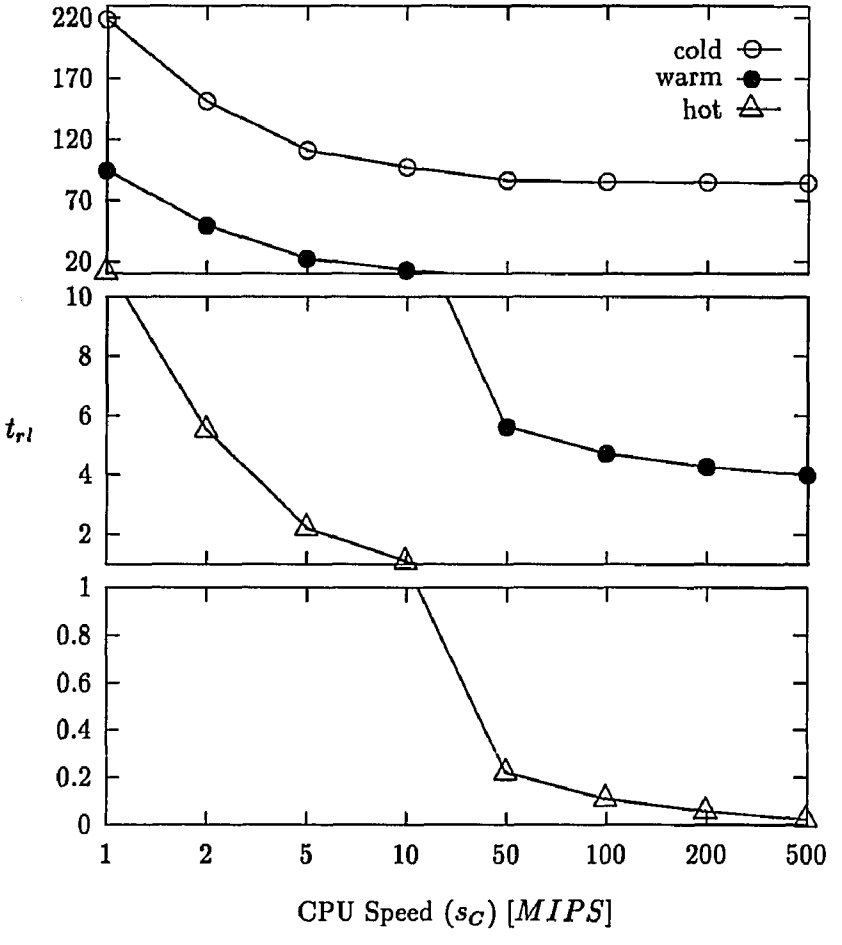



**Figure 5.3:** Reconfiguration latency vs. CPU speed for three types of spares: $s_B$ = 2.105 [*Kbytes/milliseconds*].

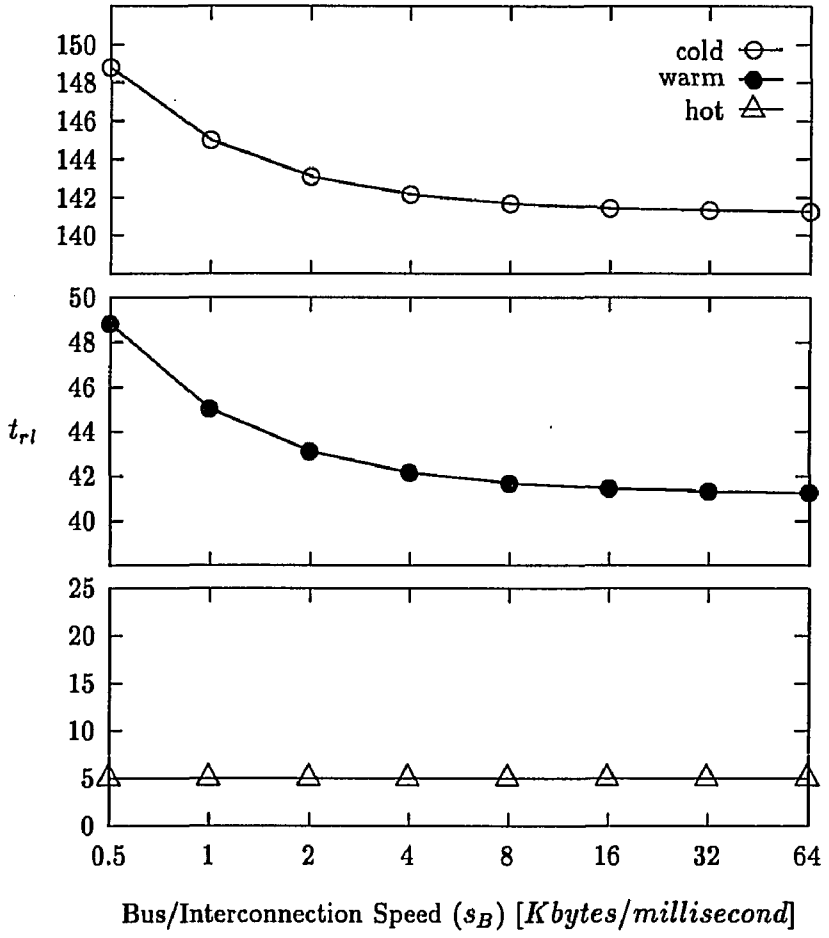



**Figure 5.4:** Reconfiguration latency vs. bus/interconnection speed for three types of spares: $s_C = 2.2$ [$MIPS$].

in Fig. 5.3, which allows cold or warm spares to be effectively used for meeting a certain system deadline. A higher transfer rate of bus/interconnection raises cold or warm spares' usefulness, as shown in Fig. 5.4. However, only hot spares can satisfy a stringent CSD, for example, requiring the reconfiguration latency less than 1 [*millisecond*] in the example, for which $s_C$ should be improved as fast as tens of $MIPS$ for even hot spares.

## 5.5 Conclusion

We evaluated the reconfiguration latency of four classified reconfiguration techniques by using the times required for such steps that make up those techniques. Specifically, we analyzed the effects of the task size, CPU speed, and bus/interconnection speed (and the number of links between a faulty module and its replacement module) upon the download and initialization time, which is the most contributing factor of the reconfiguration latency in backup sparing (as well as graceful degradation), for the three types of spares. Via an example, we observed that the reconfiguration latency can be decreased to meet a given CSD by using, for example, fast CPU or high transfer rate of bus/interconnection, in case of cold or warm spares. However, hot spares should be used to satisfy a tight CSD, because the time required for some steps with cold (or warm) spares is insensitive to these improved capabilities.

# CHAPTER 6

# AN OPTIMAL INSTRUCTION-RETRY POLICY FOR TMR CONTROLLER COMPUTERS

## 6.1 Introduction

Using the deadline information obtained from the controlled process in Chapter 3, we propose in this chapter an optimal recovery policy to enhance system reliability by adding time redundancy to controller computers equipped with a minimum degree of spatial redundancy. Specifically, instruction retry is used "optimally" (in a sense of minimizing a certain cost) for triple modular redundant (TMR) controller computers.

A TMR system is a typical example of static redundancy which can tolerate one faulty module without any delay [9, 23, 25, 64, 73]. The TMR system can tolerate even multiple faults, if they occur sequentially with a relatively long inter-occurrence interval, by using appropriate detection, identification, and replacement of a faulty module (whose error was 'masked') before a new fault occurs to another module within the TMR. Detect–diagnose–reconfigure is a conventional recovery policy for handling multiple faults in TMR systems [23, 64]. Alternatively, the system can also recover from the masked error induced by a transient fault by retrying the failed operation a fixed number of times on the same hardware [9]. Note that these two policies can tolerate only a subset of multiple faults. That is, *TMR failures* — failure to establish a majority of module outputs due to multiple faulty modules or a faulty voter — caused by coincident/common-cause faults require a different recovery method. Note that a harsh environment with electromagnetic interferences (EMI), such as lightning, high-intensity radiated fields (HIRF), or nuclear electromagnetic pulses (NEMP), may cause coincident faults in all modules.

A TMR system uses a minimum degree of spatial redundancy to mask one faulty module (in general, $2n + 1$ modules needed to mask up to $n$ faulty modules), and more than

90% of field failures are reported to be caused by transient faults [46]. Most TMR failures can thus be recovered by (a) using the capability of a TMR system that can mask one (permanent/transient) faulty module, and (b) retrying instructions on the same redundant hardware in case of a TMR failure resulting from additional transient fault(s). This may reduce the hardware cost by avoiding the premature retirement of modules with transient faults, and reduce the time overhead of recovery and the probability of dynamic failure resulting from spares exhaustion as well as deadline misses. Note that system reconfiguration is more time-consuming than a simple retry. Reconfiguration and (cold) restart generally consist of (i) switching power and bus connections, (ii) running built-in-test (BIT) on the spare module, (iii) loading programs and data, (iv) initializing the software (even when warm spares are used, thus unneeding (i) and (ii), this is still time-consuming), and (v) there are only a limited number of spares available during each mission.

As the simplest form of time redundancy, instruction retry has been proposed and analyzed by several researchers. The authors of [7] specified the retry period *a priori* in an ad hoc manner. The retry period was also derived by minimizing an average task-oriented measure (mean execution time per instruction) [34], or mean task-completion time by using a Bayesian decision approach [38] or the maximum likelihood principle [40], under the assumption that an infinite number of spares are available. The retry periods derived in these papers were intended for use in simplex systems.

In contrast to the above cited approaches, we derive in this chapter the optimal retry period, $r_{opt}$, of a TMR controller computer by minimizing the probability of missing deadlines or dynamic failure upon detection of a masked error or a TMR failure. A retry will terminate if it becomes successful or the retry period expires, whichever occurs first. (Since the time required for repeated execution of an instruction cannot be cascaded into a single continuous duration, a retry period should be discrete, i.e., we define the "retry period" as a number of retry attempts throughout the discussion to follow.) If retry during a given period cannot recover the system from a TMR failure, the system will be reconfigured. Clearly, whether or not retry is successful depends upon the retry period as well as the *error latency* defined as the time interval from the occurrence of an error to its detection. The retry period should be large enough for the transient fault(s) inducing the detected error/failure to die away. This may in turn increase the recovery time when retry is used for permanent fault(s), and/or when the error latency is larger than the execution time of the retried instruction. We assume the use of a detection scheme with high (but not necessarily

perfect) coverage for both masked errors and TMR failures as required by a usual retry policy. For example, any error in a module can be caught by using a simple disagreement detector [51, 74]. One can ultimately adopt a detection scheme like a Totally Self Checking Circuit (TSCC) in [17], which has the capability of detecting a masked error as well as a TMR failure and is both self-testing and fault-secure.

The probability of dynamic failure, $P_{dyn}$, depends on the mission lifetime, the number of spares, the CSD, and the retry period $r$.[1] We calculate $P_{dyn}$ as a function of these parameters and derive $r_{opt}$ by minimizing $P_{dyn}$ in each case of masked error or TMR failure. Using the optimal retry period, we also determine the minimum number of spares needed to attain the largest acceptable $P_{dyn}$ for a given mission.

This chapter is organized as follows. Section 6.2 describes the characteristics of a real-time control system, the basic assumptions used, and the required property and structure of a detection scheme adopted. In Section 6.3, we numerically derive the optimal retry period by minimizing $P_{dyn}$ in both cases of TMR failure and masked error. The effect of the number of available spares on $P_{dyn}$ is also analyzed there. Section 6.4 presents representative numerical examples. The chapter concludes with Section 6.5.

## 6.2 Notation, Assumptions, and Models

A real-time control system executes missions between maintenances, and usually no repair is assumed during a single mission. System diagnosis and the subsequent repair, if needed, are performed during a maintenance period between missions. As shown in Fig. 6.1, a mission lifetime generally consists of many *task periods* during each of which a sequence of instructions are executed to generate a control command or display output. During a mission, a *dynamic failure* is said to occur due to several consecutive TMR failures whose total period exceeds the CSD, the maximum delay in the feedback loop the controlled process can tolerate without losing system stability or leaving its allowed state space [56, 57]. This delay could be as long as the time of executing several consecutive tasks or task invocations. Using the CSD, one can derive the deadline of each task.

The main computational load of a controller computer consists of a set of periodic tasks that are executed repetitively, each time with a different input. A dynamic failure may occur due to either missing the CSD[2] or exhausting spares as a result of frequent system

[1] $r$=0 means system reconfiguration without retry.

[2] That is, missing the update of the control command for a period longer than the CSD due to consecutive

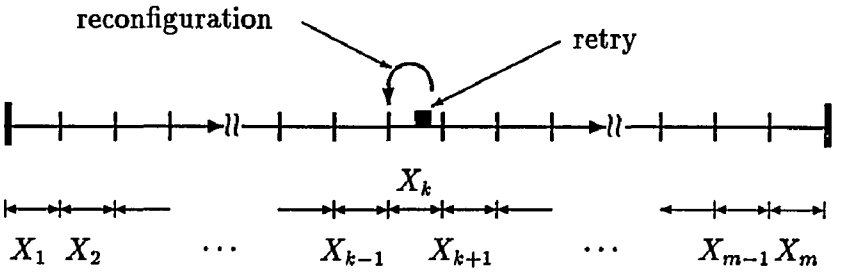



**Figure 6.1:** Time index of a mission lifetime: $X_M = \sum_{i=1}^{m} X_i$.

reconfigurations. Note that the latter may occur because there are only a limited number of spares aboard, depending on the weigh, volume, cost, MTTF of each spare and the given mission. To study the effects of retry/reconfiguration on the probability of dynamic failure, we need to introduce the following variables:

- $X_M$: the lifetime of a mission consisting of $m$ computational tasks.
- $X_i$: the execution time of the $i$-th task of the mission, i.e., $X_M = \sum_{i=1}^{m} X_i$. $X = \frac{X_M}{m}$ is the average execution time of a task in the mission, or it is the execution time of a single task invocation if the mission consists of $m$ invocations of a periodic task.
- $\Delta x$: the inter-voting interval such that $X = K\Delta x$, where $K$ is the number of times (intermediate) computation results are voted on during the average execution time, $X$, of a task or a task invocation.
- $N$: the number of spares available during a single mission.
- $D$: the CSD characterized by a probability density function $f_D(t)$.
- $r = \{r_e, r_t\}$: the maximum retry period allowed for a masked error ($r_e$) or a TMR failure ($r_t$).

Throughout the paper, we will classify faults to be *external* or *internal*, depending on whether their causes are inside or outside the system. Internal (system component) faults reside inside the system inducing errors, while external faults are caused by environmental interferences. One can observe that external faults are likely to be transient because adverse environmental conditions are generally temporary/transient and environmental disruptions result in functional error modes without actually damaging system components [5]. A

TMR failures or generating incorrect execution results for several tasks.

harsh environment resulting from lightning, HIRF, or NEMP is likely to cause coincident or 'common mode' faults/errors. In other words, external faults are likely to result in multiple non-permanent fault modules, which will in turn cause TMR failures.

We assume that all faults arrive according to time-invariant Poisson processes with rate $\lambda_{ip}$ ($\lambda_{in}$) for permanent (non-permanent) internal faults such that the total internal fault arrival rate $\lambda_i = \lambda_{ip} + \lambda_{in}$, and rate $\lambda_{ep}$ ($\lambda_{en}$) for permanent (nonpermanent) external faults such that the total external fault arrival rate $\lambda_e = \lambda_{ep} + \lambda_{en}$. The active duration of a non-permanent internal (external) fault is assumed to be exponentially distributed with mean $\frac{1}{\lambda_{ia}}$ ($\frac{1}{\lambda_{ea}}$). We also assume occurrences of *internal* faults in one module to be independent of those in other modules.

As mentioned earlier, retry could be effective only if the corresponding masked error/TMR failure is detected upon its occurrence. To achieve immediate and accurate detection of such errors/failures, we employ a special detection scheme like a Totally Self Checking Circuit (TSCC) [17], which detects the existence of a masked error and/or a TMR failure and is both self-testing and fault-secure. The TSCC provides two output indications distinguishing a masked error (and/or a fault in the error checking circuit) from a TMR failure (an output-information error generating a stop signal). However, it may not detect the incorrect output(s) before the completion of one instruction, thus making retry inapplicable. If a more complicated recovery operation such as rollback with checkpoints is applied to complement retry, an error/failure detected late can also be recovered by using only time redundancy, i.e., a non-zero error latency is allowed. However, in our approach to using retry or reconfiguration, any error/failure detected late is assumed to be recovered only through reconfiguration.

Let $c_e$ and $c_t$ be the detection coverages of a masked error and a TMR failure, respectively. Then, the probability that the detection scheme detects late (or misses) a masked error (or TMR failure) is $(1 - c_e)$ (or $(1 - c_t)$). There are two cases in which the detection scheme may fail to find a masked error/TMR failure: (i) a short-lived fault inducing the masked error/TMR failure disappears after contaminating one or more tasks, (ii) a permanent or long-lived transient fault keeps generating incorrect outputs to lead to a dynamic failure. Although an undetected permanent or long-lived transient fault can cause serious damages, the probability of not capturing such a long-lived fault is so small as to be ignored because consecutive instructions with incorrect results are likely to be detected before the CSD which is usually larger than the execution time of one or more tasks. Furthermore, the probability of missing a short-lived fault, which may not induce any error, is not negligible,

but its effect is not significant to independent periodic tasks. Thus, we will only deal with late detection of errors/failures due to an imperfect detection scheme while assuming that long-active faults are detected eventually.

## 6.3 Optimal Retry Policy

In our model of a TMR system, the key variables in determining the optimal retry period $r_{opt}$ are the mission lifetime $X_M = mX$, the number of spares $N$, and the CSD $D$.

When a masked error/TMR failure is detected, there are several state-transition scenarios as shown in Fig. 6.2. Suppose a masked error/TMR failure occurs during $X_i$. If retry is chosen to recover from this error/failure ($I_{r_e} = I_{r_t} = 1$), it may terminate when the fault that had caused the error/failure disappears (a successful retry), or the retry period expires (an unsuccessful retry), whichever occurs first. In the case of a TMR failure, a successful retry results when the system is in one of two possible states: *fault-free* state due to disappearance of all existing (non-permanent) faults and *one masked-error* state due to the existence of one still-active faulty module. A successful retry for a masked error moves the controller computer to fault-free state. Unsuccessful retries may lead to a dynamic failure in case of a TMR failure,[3] or may trigger a system reconfiguration in both cases of masked error and TMR failure. Note that retry for a masked error is performed only on the faulty module while retaining the execution results from the other two healthy modules. The occurrence of a new fault in another module before the disappearance of the current fault or initiation of a system reconfiguration will result in a TMR failure. The system may be reconfigured immediately without retry ($I_{r_e} = I_{r_t} = 0$) or after an unsuccessful retry. If no spares are available, or $N = 0$ for a masked error and $N < 3$ for a TMR failure, a dynamic failure occurs. System reconfiguration increases the probability of exhausting spares during the remaining mission even if it could prevent an immediate dynamic failure. When the CSD is tight, system reconfiguration may also lead to a dynamic failure due to its setup and restart delays, during which another TMR failure may occur.

All of the above phenomena are captured in the Markov-chain of Fig. 6.3, each state of which is distinguished by the number of spares available. In this model, the probabilities $p_{E_r}$ and $p_{T_r}$ account for replacing the faulty module(s) that had caused a masked error and a TMR failure, respectively, and $p_{mh}$ represents the probability of dynamic failure due to missing the CSD during the execution of a task. These probabilities determine the

[3] due mainly to missing a CSD

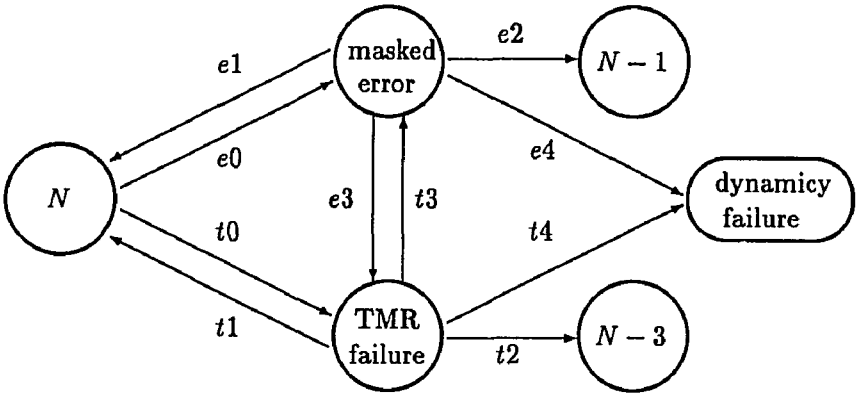



**Figure 6.2:** A system state diagram. In case of a masked error ($e0$): $e1$ = recovery by retry, $e2$ = recovery by immediate reconfiguration ($I_{r_e} = 0$) or after unsuccessful retry ($I_{r_e} = 1$), $e3$ = a TMR failure due to occurrence of another faulty module during retry, $e4$ = spares exhaustion during reconfiguration or missing the CSD. In case of a TMR failure ($t0$): $t1$ = restoration to a fault-free state by retry, $t2$ = recovery by reconfiguration, $t3$ = restoration to one masked-error state by retry, $t4$ = the same as $e4$.

probability of dynamic failure over the mission lifetime. They all depend upon whether retry is used or not (represented by a pair of indicator functions $(I_{r_e}, I_{r_t})$) and how long retry lasts (i.e., values of $(r_e, r_t)$), or how many times the failed instruction is retried.

$P_{dyn}(r, m, X, N, D)$ is obtained simply by adding the probability of exhausting spares $P_{es}(r, m, X, N)$ and the probability of missing the CSD $P_{mh}(r, m, X, D)$ during the mission lifetime $X_M = mX$, i.e.,

$$P_{dyn}(r, m, N, D, X) = P_{es}(r, m, N, X) + P_{mh}(r, m, D, X). \tag{6.1}$$

$N$ spares can withstand $j$ and $k$ reconfigurations resulting from TMR failures and masked errors during $m$ invocations of tasks, respectively, such that $3j + k \leq N$. Thus,

$$P_{es}(r, m, X, N) = \sum_{3j+k>N} \frac{m!}{j!k!(m-j-k)!} p_{T_r}^j(r, X) p_{E_r}^k(r, X) \left[1 - p_{T_r}(r, X) - p_{E_r}(r, X)\right]^{m-j-k}, \tag{6.2}$$

where $p_{E_r}(r, X)$ $(p_{T_r}(r, X))$ is the probability of reconfiguration due to a masked error (TMR failure) during the average task-execution time $X$ with a retry period $r = \{r_e, r_t\}$. Suppose that in Eq. (6.2) of a multinomial function, $m \gg 1$, $p_{T_r} + p_{E_r} \ll 1$, but $mp_{T_r}$ and $mp_{E_r}$

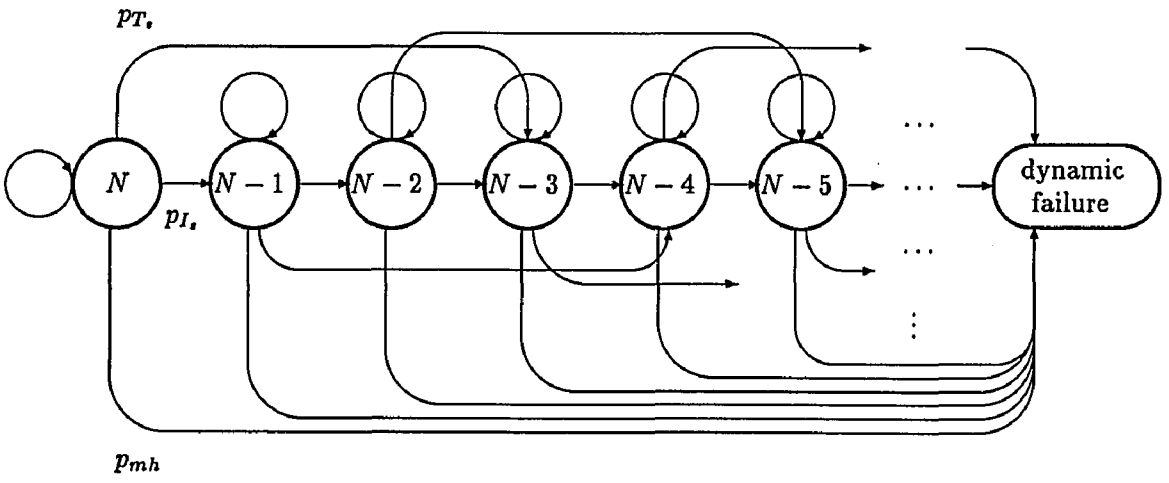



**Figure 6.3:** A modified Markov-chain model based on the number of spares upon occurrence of masked errors or TMR failures: $p_0 = 1 - p_{E_s} - p_{T_s} - p_{mh}$, $p_{E_s}$ ($p_{T_s}$) = probability of reconfiguration due to a masked error (TMR failure), $p_{mh}$ = probability of dynamic failure due to missing the CSD.

remain constant, say $mp_{T_s} = a_1$ and $mp_{E_s} = a_2$. Then, Eq. (6.2) can be approximated as $\frac{1}{j!k!}a_1^j a_2^k(1 - a_1 - a_2)^{m-j-k}$, where $m(m-1)\cdots(m-j-k+1) \simeq m^{k+j}$, if $m$ is allowed to become large enough and if $j$ and $k$ are fixed. Hence as $m \to \infty$, $p_{T_s} + p_{E_s} \to 0$, and $j + k \ll m$, we obtain

$$\frac{1}{j!k!}a_1^j a_2^k(1 - a_1 - a_2)^{m-j-k} \longrightarrow \frac{1}{j!k!}a_1^j a_2^k e^{-a_1-a_2}.$$

Thus in situations where the multinomial law applies with $m \gg 1$, $p_{T_s} + p_{E_s} \ll 1$, but $mp_{T_s} = a_1$ and $mp_{E_s} = a_2$ are finite constants, we can use the following approximation for Eq. (6.2):

$$\frac{1}{j!k!}\,(mp_{T_s}(r_t, X))^j (mp_{E_s}(r_i, X))^k e^{-m[p_{T_s}(r_t,X)+p_{E_s}(r_i,X)]}.$$

Eq. (6.2) is based on the assumption of at most one masked error or TMR failure during $X$, which is reasonable because masked errors and TMR failures occur very rarely. $P_{mh}(r, m, D, X)$ is also derived from the probability, $p_{mh}(r, X, D)$, of missing the CSD $D$ during $X$ when retry is applied for a period $r$. For the successful completion of a mission, every task/invocation must not miss its deadline, which is represented by $\prod_{i=1}^{m}[1 - p_{mh}(r, X, D)]$. Thus, $P_{mh}(r, m, X, D)$ is computed as:

$$\begin{aligned} P_{mh}(r, m, X, D) &= 1 - \prod_{i=1}^{m}[1 - p_{mh}(r, X, D)] = 1 - [1 - p_{mh}(r, X, D)]^m, \\ &\simeq mp_{mh}(r, X, D), \quad \text{if } p_{mh} \ll 1. \end{aligned} \tag{6.3}$$

We now analyze quantitatively the effects of retries for masked errors and TMR failures on $P_{dyn}$ separately, and then combine the two results to derive $r_{opt}$.

### 6.3.1 Reconfiguration without Retry ($I_{r_e} = I_{r_t} = 0$)

This is the usual recovery method for masked errors and TMR failures without using time redundancy, i.e., $r_e = r_t = 0$. The diagnosed faulty module is replaced with a healthy spare. In case of a TMR failure, all three modules are switched out due to the considerable time overhead of identifying (diagnosing) faulty modules, and the current task is re-executed with the reloaded data as shown in Fig. 6.1.

$p_{E_s}(X)$ and $p_{T_s}(X)$ in this case are equal to the probabilities of a masked error $p_E(X)$ and a TMR failure $p_T(X)$, respectively. Since masked errors are also recovered through reconfiguration (or retry in the following methods) with high detection coverage $c_e$, a TMR failure is assumed to occur due mainly to (near) coincident faults occurring within an inter-voting interval, rather than sequentially occurring faults. Let $p_{T_e}(X)$ and $p_{T_i}(X)$ be the probabilities of TMR failures caused by external and internal faults, respectively. Then, $p_{T_e}(X)$ is equal to $1 - e^{-\lambda_e X}$, which is the probability of occurrence of an external fault during $X$, because external faults are assumed to cause coincident internal faults/errors. $p_{T_i}(X)$ is also obtained by using the probability of coincident internal faults in two or three modules in $K$ inter-voting intervals during $X$, that is:

$$p_{T_i}(X) = 1-[1 - 3(1 - e^{-\lambda_i \Delta x})^2 + 2(1 - e^{-\lambda_i \Delta x})^3]^K \simeq K\,[3(1 - e^{-\lambda_i \Delta x})^2 - 2(1 - e^{-\lambda_i \Delta x})^3]\,.$$

Since TMR failures caused by external and internal faults are not exclusive to each other, $p_T(X)$ is not the direct sum of $p_{T_e}(X)$ and $p_{T_i}(X)$, but equals $p_{T_e}(X)+p_{T_i}(X)-p_{T_e}(X)p_{T_i}(X)$. $p_E(X)$ is the probability of occurrence of an internal fault in only one module during $X$, which is calculated as:

$$p_E(X) = 3(1 - e^{-\lambda_i X}) - 3(1 - e^{-\lambda_i X})^2 + (1 - e^{-\lambda_i X})^3 - p_{T_i}(X) \simeq 3K(1 - e^{-\lambda_i \Delta x})e^{-2\lambda_i \Delta x}.$$

Let $X_a$ be the actual time required to complete the computation corresponding to $X$,[4] and $f_{X_a}$ and $f_D$ are the *pdf*'s of $X_a$ and the control system deadline $D$, respectively. Using $f_{X_a}$ and $f_D$, we can obtain $p_{mh}(X, D)$ as the probability that $X_a > D$:

$$p_{mh}(X, D) = \int_0^\infty \int_D^\infty f_{X_a}(x) f_D(y) dx\, dy.$$

[4] Because of the time overhead of retry and/or reconfiguration for errors during the execution of a mission segment, $X_a$ is usually larger than $X$.

The set of samples of $X_a$ is obtained as:

$$X_a \in \{X, (\bar{X} + t_r) + X, 2(\bar{X} + t_r) + X, 3(\bar{X} + t_r) + X, \cdots\},$$

where $t_r$ is the resetting time and $\bar{X}$ is the mean occurrence time of a TMR failure, which was derived in [58]. Since $X_a$ has discrete values, the probability mass function ($pmf$) of $X_a$ is:

$$f^k_{X_a} = Prob[X_a = k(\bar{X} + t_r) + X] = p^k_T(X)(1 - p_T(X)).$$

Note that $f_D$ is given *a priori* from experimental data or the analysis of controlled processes [56, 57]. Consequently,

$$p_{mh}(X, D) = \int_0^\infty \sum_{k > \lfloor (D-X)/(\bar{X}+t_r) \rfloor} f^k_{X_a}(x) f_D(y) dy.$$

### 6.3.2 Retry for Masked Errors ($I_{r_e} = 1,\ I_{r_t} = 0$)

In this case, retry of a period $r_e$ is initiated upon detection of a masked error, and reconfiguration is the only recovery mechanism for TMR failures. Thus, $p_{E_r}(X)$ and $p_{T_r}(X)$ are no longer equal to $p_E(X)$ and $p_T(X)$. $p_{E_r}$ decreases with $r_e$, because most masked errors are induced by non-permanent faults and because a simple retry is likely to recover from them (only if they are detected before the completion of an instruction that is to be retried). However, $p_{T_r}$ increases with $r_e$ due to the increased probability of a TMR failure during the retry period, i.e., a TMR failure may be induced by faults occurring sequentially during the retry for a masked error as well as by coincident faults.

Let $R^{dec}_e(r_e)$ and $R^{inc}_t(r_e)$ be the coefficients indicating respectively the decrease of masked errors and the increase of TMR failures after retrying for masked errors. Then,

$$R^{dec}_e(r_e) = (\frac{\lambda_{ip}}{\lambda_i} + \frac{\lambda_{in}}{\lambda_i} e^{-\lambda_{ia} r_e}) e^{-(2\lambda_i + \lambda_e) r_e} + \frac{2\lambda_{in}}{\lambda_i}(1 - e^{-\lambda_{ia} r_e})(1 - e^{-\lambda_i r_e}) e^{-(\lambda_i + \lambda_e) r_e},$$

where the first term represents the effect of an unsuccessful retry and the second term represents a second fault occurrence after successful retry on the first fault occurrence.

$$\begin{aligned} R^{inc}_t(r_e) \;=\; & (\frac{\lambda_{ip}}{\lambda_i} + \frac{\lambda_{in}}{\lambda_i} e^{-\lambda_{ia} r_e})(1 - e^{-(2\lambda_i + \lambda_e) r_e}) + \frac{\lambda_{in}}{\lambda_i}(1 - e^{-\lambda_{ia} r_e}) \Big( 1 - e^{-\lambda_e r_e} \\ & + (1 - e^{-\lambda_i r_e})^2 - (1 - e^{-\lambda_e r_e})(1 - e^{-\lambda_i r_e})^2 \Big), \end{aligned}$$

where the first term represents an unsuccessful retry (occurrences of fault in any healthy module) and the second term represents a successful retry (occurrences of fault in two or three modules). Then, using $R^{dec}_e(r_e)$, $R^{inc}_t(r_e)$, $c_e$, $p_E(X)$, and $p_T(X)$, we can obtain

$p_{E_r}(r_e, X)$ and $p_{T_r}(r_e, X)$ as:

$$\begin{aligned} p_{E_r}(r_e, X) &= (1 - c_e)p_E(X) + c_e R_e^{dec}(r_e)p_E(X), \\ p_{T_r}(r_e, X) &= p_T(X) + c_e R_t^{inc}(r_e)p_E(X). \end{aligned} \tag{6.4}$$

$p_{mh}(r_e, X, D)$ is derived in the same way as before except for the change of $f_{X_a}^k$, i.e., substituting $p_{T_r}(r_e, X)$ for $p_T(X)$ since the probability of a TMR failure is changed:

$$f_{X_a}^k = Prob[X_a = k(\bar{X} + t_r) + X] = p_{T_r}^k(r_e, X)\,(1 - p_{T_r}(r_e, X)), \quad 0 \le k \le \infty.$$

### 6.3.3 Retry for TMR Failures ($I_{r_e} = 0,\ I_{r_t} = 1$)

This is the opposite to the previous policy, that is, retry of a period $r_t$ is initiated upon detection of a TMR failure, and a masked error calls for an immediate reconfiguration. Obviously, $p_{T_r}$ decreases with $r_t$, because most TMR failures are also recovered by a simple retry during which non-permanent faults are likely to disappear. Even if retry is successful, there may still exist a faulty module. Those masked-error states transited from TMR failures during a retry increase $p_{E_r}(X)$.

Let $p_{ij}$ denote the conditional probability of $i$ faulty modules and $j$ permanent-fault modules given a TMR failure (i.e., $i \le 2$), which is computed as:

$$\begin{aligned} p_{20} &= (\frac{\lambda_{in}}{\lambda_i})^2 \frac{B}{A+B+C}, \quad p_{21} = \frac{2\lambda_{in}\lambda_{ip}}{\lambda_i^2}\frac{B}{A+B+C}, \quad p_{22} = (\frac{\lambda_{ip}}{\lambda_i})^2 \frac{B}{A+B+C}, \\ p_{30} &= \frac{\lambda_{en}}{\lambda_e}\frac{A}{A+B+C} + (\frac{\lambda_{in}}{\lambda_i})^3 \frac{C}{A+B+C}, \quad p_{31} = \frac{3\lambda_{in}^2\lambda_{ip}}{\lambda_i^3}\frac{C}{A+B+C}, \\ p_{32} &= \frac{3\lambda_{in}\lambda_{ip}^2}{\lambda_i^3}\frac{C}{A+B+C}, \quad p_{33} = \frac{\lambda_{ep}}{\lambda_e}\frac{A}{A+B+C} + (\frac{\lambda_{ip}}{\lambda_i})^3 \frac{C}{A+B+C}, \end{aligned}$$

where $A = 1 - e^{-\lambda_e X}$ and $B = 3K(1 - e^{-\lambda_i \Delta x})^2 e^{-\lambda_i \Delta x}$ ($C = K(1 - e^{-\lambda_i \Delta x})^3$) are the probabilities of occurrences of external faults and near-coincident internal faults inducing two (three) faulty modules during $X$. Let $R_e^{inc}(r_t)$ and $R_t^{dec}(r_t)$ be the coefficients indicating respectively the increase of masked errors and the decrease of TMR failures after retrying for TMR failures. $R_e^{inc}(r_t)$ is obtained by computing the probability that only one faulty module remains in each case of $p_{ij}$, and $R_t^{dec}(r_t)$ is derived from the cases of more than two permanent or long-lived transient faults, thus:

$$\begin{aligned} R_e^{inc}(r_t) &= p_{21}(1 - e^{-\lambda_{ia} r_t}) + p_{31}(1 - e^{-\lambda_{ia} r_t})^2 + 2p_{20}(1 - e^{-\lambda_{ia} r_t})e^{-\lambda_{ia} r_t} \\ &\quad + 3p_{30}C_i(1 - e^{-\lambda_{ia} r_t})^2 e^{-\lambda_{ia} r_t}, \\ R_t^{dec}(r_t) &= p_{33} + p_{32} + p_{22} + p_{21}e^{-\lambda_{ia} r_t} + p_{31}(2e^{-\lambda_{ia} r_t} - e^{-2\lambda_{ia} r_t}) \end{aligned}$$

$$+p_{20}e^{-2\lambda_{ia}r_t} + p_{30}\{C_e e^{-\lambda_{ea}} + C_i(3e^{-2\lambda_{ia}r_t} - 2e^{-3\lambda_{ia}r_t})\},$$

where

$$C_e = \frac{\lambda_{en}}{\lambda_e}A\left(\frac{\lambda_{en}}{\lambda_e}A + (\frac{\lambda_{in}}{\lambda_i})^3 C\right)^{-1}, C_i = (\frac{\lambda_{in}}{\lambda_i})^3 C\left(\frac{\lambda_{en}}{\lambda_e}A + (\frac{\lambda_{in}}{\lambda_i})^3 C\right)^{-1}.$$

Using these coefficients, we obtain $p_{E_s}(r_t, X)$ and $p_{T_s}(r_t, X)$ as:

$$\begin{aligned} p_{E_s}(r_t, X) &= p_E(X) + c_t R_e^{inc}(r_t) p_T(X), \\ p_{T_s}(r_t, X) &= (1 - c_t) p_T(X) + c_t R_t^{dec}(r_t) p_T(X). \end{aligned} \tag{6.5}$$

Now, a dynamic failure may occur due mainly to unsuccessful retries if the CSD, $D$, is tight. The controller computer may fail to generate a correct control command within $D$ units of time due to TMR failures after repeating/retrying the execution of an instruction. Thus, the derivation of $p_{mh}(r_t, X, D)$ is different from that of the previous two cases. By approximating the mean end-of-retry period with $\frac{1}{\lambda_{ia}}$ in case of a successful retry, the samples of $X_a$ are:

$$X_a \in \{X, X + \frac{1}{\lambda_{ia}}, (\bar{X} + t_r + r_t) + X, (\bar{X} + t_r + r_t) + X + \frac{1}{\lambda_{ia}}, 2(\bar{X} + t_r + r_t) + X, \cdots\}.$$

The $pmf$ of $X_a$ is then:

$$\begin{aligned} f_{X_a}^k &= \Pr[X_a = k(\bar{X} + t_r + r_t) + X + \delta\frac{1}{\lambda_i a}], \quad 0 \le k \le \infty, \quad \delta \in \{0, 1\} \\ &= p_T^{k+\delta}(X)(1 - p_s(r_t))^k (1 - p_T(X))^{1-\delta} p_s(r_t)^\delta, \end{aligned} \tag{6.6}$$

where $\delta \in \{0, 1\}$ indicates that the mission segment corresponding to $X$ is completed because of a successful retry upon detection of a TMR failure or because of no TMR failure, while repeating the execution of the mission segment $k$ times with $k$ reconfigurations. $p_s(r_t)$ of Eq. (6.6) represents the probability of successful retry, which is computed by considering all cases of no more than one faulty module remaining after retrying in each case of $p_{ij}$:

$$\begin{aligned} p_s(r_t) = c_t \big[&p_{21}(1 - e^{-\lambda_{ia}r_t}) + p_{31}(1 - e^{-\lambda_{ia}r_t})^2 + p_{20}(1 - e^{-2\lambda_{ia}r_t}) \\ &+ p_{30}\left\{C_e(1 - e^{-\lambda_{ea}r_t}) + C_i(1 - e^{-\lambda_{ia}r_t})^2(1 + 2e^{-\lambda_{ia}r_t})\right\}\big]. \end{aligned}$$

Consequently,

$$p_{mh}(r_t, X, D) = \int_0^\infty \sum_{k > \lfloor (D - X - \delta\frac{1}{\lambda_i a})/(\bar{X} + t_r + r_t) \rfloor} f_{X_a}^k(x) f_D(y) dy.$$

### 6.3.4 Retry for Both Cases ($I_{r_e} = 1,\ I_{r_t} = 1$)

Finally, we present a retry policy for both cases of TMR failure and masked error with retry periods $r_t$ and $r_e$, respectively. To show a significant decrease in the frequency of reconfigurations, $p_{E_s}(r, X)$ and $p_{T_s}(r, X)$ are also derived by using all the coefficients indicating the increase and/or decrease of masked errors and TMR failures, i.e., in the same way of deriving Eqs. (6.4) and (6.5):

$$\begin{aligned} p_{E_s}(r, X) &= \{p_E(X) + c_t R_e^{inc}(r_t) p_T(X)\}\{1 - c_e + c_e R_e^{dec}(r_e)\}, \\ p_{T_s}(r, X) &= \{p_T(X) + c_e(1 - c_t) R_t^{inc}(r_e) P_E(X)\}\{1 - c_t + c_t R_t^{dec}(r_t)\}, \end{aligned} \tag{6.7}$$

which are obtained by considering both the effects of retry for masked errors ($R_e^{dec}(r_e)$ and $R_t^{inc}(r_e)$) and the effects of retry for TMR failures ($R_e^{inc}(r_t)$ and $R_t^{dec}(r_t)$). The derivation of $p_{mh}(r, X, D)$ is similar to the previous case because both cases use retry for a TMR failure. The only difference from Eq. (6.6) is the change of $p_T(X)$ to $p_{T0}$ in $f_{X_a}^k$, where $p_{T0}$ is the increased probability of TMR failures after retrying for masked errors:

$$f_{X_a}^k = p_{T0}^{k+\delta}(X)(1 - p_s(r_t))^k (1 - p_{T0}(X))^{1-\delta} p_s(r_t)^\delta, \tag{6.8}$$

where

$$p_{T0} = p_T(X) + c_e R_e^{inc}(r_e) p_E(X).$$

### 6.3.5 Optimal Retry Period and Minimum Number of Spares

Using the derived $p_{E_s}(r_e, X)$, $p_{T_s}(r_t, X)$, and $p_{mh}(r, X, D)$, one can compute $P_{es}$ and $P_{mh}$ from Eqs. (6.2) and (6.3), which are in turn used to calculate $P_{dyn}$ with Eq. (6.1). Now, $r_{opt} = \{r_{eopt}, r_{topt}\}$ is determined by minimizing the derived $P_{dyn}$ with respect to $r_e$ and $r_t$. (There always exist $r_{eopt}$ and $r_{topt}$ that minimize $P_{dyn}$ over a closed interval, $0 \le r_e, r_t \le D - X$.) The derivation of $r_{opt}$ involves the following three steps:

**Step 1:** Compute $r_{eopt}$ from the case of $I_{r_e} = 1$ and $I_{r_t} = 0$ and let it be $r_e^*$.

**Step 2:** Compute $r_{topt}$ from the case of $I_{r_e} = 1$ and $I_{r_t} = 1$ by using $r_e^*$ and let it be $r_t^*$

**Step 3:** Calculate $P_{dyn}(r = 0)$ and compare it with $P_{dyn}(r_e^*, r_t^*)$. If $P_{dyn}(r = 0) \le P_{dyn}(r_e^*, r_t^*)$, then $r_{opt} = 0$ else $r_{opt} = \{r_e^*, r_t^*\}$.

Steps 1 and 2 to minimize $P_{dyn}$ with respect to $r_e$ and $r_t$ separately are reasonable, because the frequency (thus, effects on $P_{dyn}$) of masked errors is significantly larger than that of

TMR failures. Step 3 is taken to choose a better recovery policy between reconfiguration and retry (with the derived retry period).

When the maximum acceptable probability of dynamic failure is given as $p_{dyn}^{max}$, the minimum number of spares, $N_{min}$, can be computed by using the derived optimal retry period. Obviously, $P_{dyn}$ decreases with $N$ when other variables are fixed. We can derive the relation between $P_{dyn}$ and $N$ iteratively by increasing $N$ from a certain initial value $N_0$. From this relation $N_{min}$ is determined using $p_{dyn}^{max}$. The derivation of $N_{min}$ is described in pseudo-code, where $N_0$ is simply determined by using the case of $I_{r_e}=0$ and $I_{r_t}=0$:

*Recursive method to derive $N_{min}$*\
* $\Delta$ = acceptable error in the upper bound of $p_{dyn}^{max}$ *\
* $PROB\_DYN\_1()$ : program to compute $P_{dyn}$ for $I_{r_e}=0, I_{r_t}=0$.*\
* $PROB\_DYN\_2()$ : program to compute $P_{dyn}$ using $r_{opt}$ for $I_{r_e}=1, I_{r_t}=1$.*\
$N_0 := K$ * $K$ : arbitrary number *\
$P_{dyn} := PROB\_DYN\_1(N_0)$
**while** $(p_{dyn}^{max} - \Delta \leq P_{dyn} \leq p_{dyn}^{max})$
  **if** $(P_{dyn} \geq p_{dyn}^{max})$ **then**
    $N_0 := N_0 + 1$
  **else** $N_0 := N_0 - 1$
**end_while**
$P_{dyn} := PROB\_DYN\_2(N_0)$
**while** $(p_{dyn}^{max} \leq P_{dyn} \leq p_{dyn}^{max} + \Delta)$
  **if** $(P_{dyn} \leq p_{dyn}^{max})$ **then**
    $N := N - 1$
  **else** $N := N + 1$
**end_while**
**return** $N_{min} := N$

## 6.4 Numerical Examples

In this section, we present numerical examples of $r_{opt}$ and $N_{min}$ under a certain condition of fault occurrences. A brief performance analysis of several retry policies is also presented using numerical values of $P_{dyn}$. All variables have the same time unit and thus are listed without any specific unit. Specifically, the basic (time) unit is defined as a task period ($X = 1$) for convenience, i.e., the mission lifetime is simply represented by $m$. The CSD is

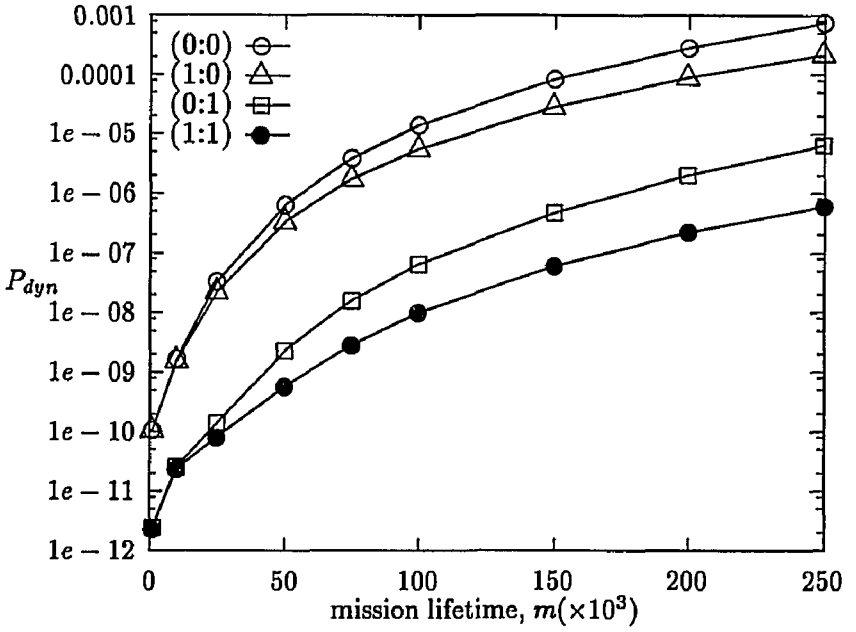



**Figure 6.4:** Probabilities of dynamic failure ($P_{dyn}$'s) of several retry policies ($I_{r_e} : I_{r_t}$): $c_e = c_t = 1$.

specified by a distribution function $F_D(d) = \frac{d-2}{2}$ (uniformly distributed) for $2 \leq d \leq 4$, the result of which can be extended to any other model of the CSD. In the numerical results below we used the resetting time $t_r = 0.1$ and the number of spares $N = 10$. (These numbers are chosen somewhat arbitrarily, but their choice would not change the conclusion we draw.)

Fault occurrences are also governed by the following set of fault parameters:

$$\lambda_{ep} = 5 \times 10^{-8}, \quad \lambda_{en} = 10^{-6}, \quad \lambda_{ip} = 2 \times 10^{-7};$$
$$\lambda_{in} = 10^{-6}, \quad \frac{1}{\lambda_{ia}} = 0.01, \quad \frac{1}{\lambda_{ea}} = 0.005. \tag{6.9}$$

The examples of $P_{dyn}$ of the four retry policies in Section 6.3 are shown in Figs. 6.4, 6.5, and 6.6, while varying the mission lifetime $m$. A simple-minded retry period ($r_i = r_t = 0.02$) is used to demonstrate the effects of retrying for TMR failures and masked errors. Retrying for TMR failures significantly reduces $P_{dyn}$ over the whole range of mission lifetime when the detection coverages are perfect ($c_e, c_t = 1$). When the mission is short, retrying for masked errors may increase $P_{dyn}$; this is demonstrated by comparing $P_{dyn}$'s of $(I_{r_i} : I_{r_t}) = (0 : 0)$ and $(1 : 0)$, or $(0 : 1)$ and $(1 : 1)$, when $m = 1000$ or $10000$. In case of short missions, a dynamic failure is likely to occur due to missing a CSD, rather than exhausting spares. Retrying for masked errors clearly increases the probability of TMR failures (thus the probability

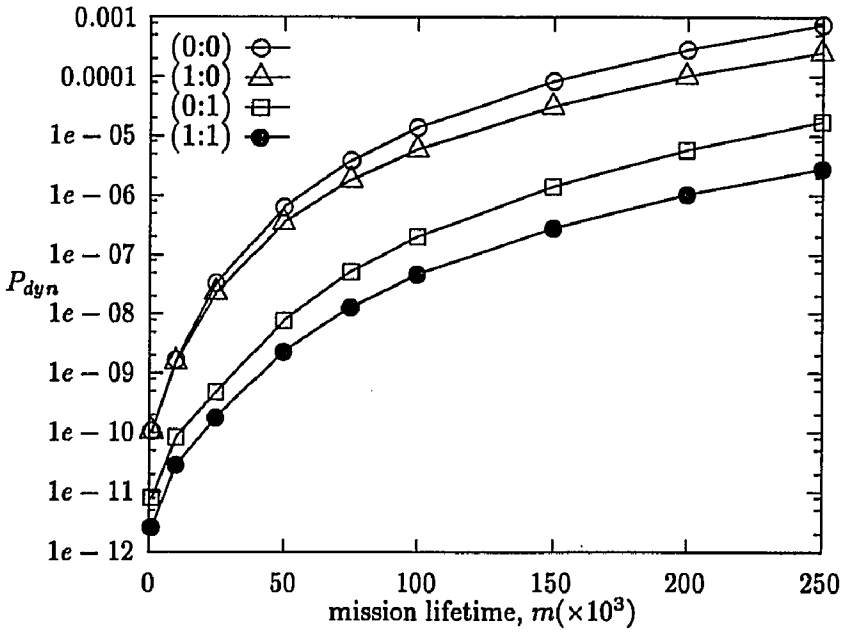



**Figure 6.5:** $P_{dyn}$'s of several retry policies $(I_{r_e} : I_{r_t})$: $c_e = c_t = 0.9$.

of missing a CSD), and decreasing the probability of exhausting spares is not effective in reducing $P_{dyn}$ for short missions. However, retrying for masked errors also decreases $P_{dyn}$ as the mission lifetime increases.

When the detection coverages are not perfect $(c_e, c_t < 1)$ the relative advantage of retry gets diminished; one can see this by comparing Fig. 6.4 with Fig. 6.5 or 6.6. Since the retry periods are generally by far smaller than a CSD, the retry policy retains its superiority to reconfiguration even if the detection coverages are not perfect, i.e., the time spent on an unsuccessful retry does not significantly affect the probability of missing control system deadlines.

Several examples of $r_{opt}$ are shown in Table 6.1 while varying the mission lifetime $m$ and the task period $K$. The task (invocation) period is measured by the number of instructions executed for the task. Those values are derived by the method proposed in Section 6.3, i.e., $r_i$ is derived first and $r_t$ computed by using the derived $r_i$. Case (i) uses the same task period ($K$ = 5000) which is equal to one time unit (i.e., X=1), while Case (ii) takes the same mission lifetime ($m$ = 10000). Both cases use the same values of fault parameters, the number of spares, the resetting time and the control system deadlines as given earlier. Although the absolute values of $r_{opt}$ ($r_{opt} \times K$) change a little with $m$ and $K$, the relative values of $r_{opt}$ with respect to a task period ($X$) changes significantly with $K$. One can observe that the mission lifetime does not affect the optimal retry period as much as the

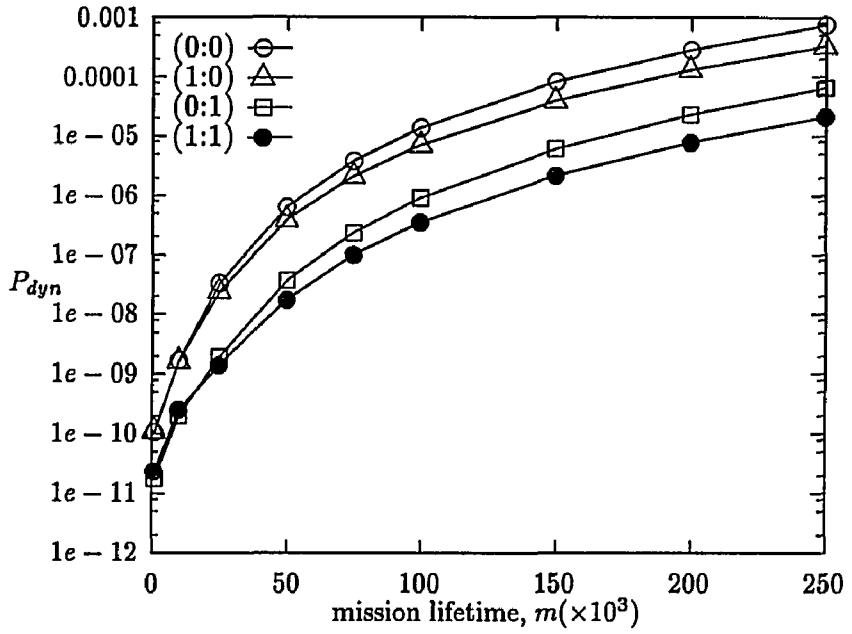



**Figure 6.6:** $P_{dyn}$'s of several retry policies $(I_{r_e} : I_{r_t})$: $c_e = c_t = 0.7$.

task period under the same condition.

$N_{min}$ is given in Table 6.2 for various mission lifetimes with a required level of $P_{dyn} = 10^{-9}$/mission. Although other parameters affect the values of $N_{min}$, $N_{min}$ heavily depends upon $m$. $P_{dyn}$ decreases with an increase of $N$, but there is a lower bound of $P_{dyn}$ because the minimum value of the probability of missing the CSD cannot be decreased beyond a certain value only by increasing $N$.

## 6.5 Conclusion

In this chapter, we have analyzed the effects of retry policies for TMR failures and masked errors on the probability of dynamic failure. We have also proposed a method to derive the optimal retry periods for both TMR failures and masked errors by minimizing the probability of dynamic failure. For this purpose, we adopted a detection scheme with high coverages of TMR failures as well as masked errors. The instruction-retry policy outperforms reconfiguration even with low detection coverage, as shown in Fig. 6.6.

Although the TMR structure can mask only the effects of one manifested faulty module, the occurrence of fault(s) in another module or coincident faults in multiple modules can lead to a TMR failure. Our retry policy reduces effectively the frequency of TMR failures by recovering from masked erroneous module(s), and it also recovers from a TMR failure as a result of coincident faults by considering the control system deadlines and the number

| (i) $m(\times 10^3)$ | 1 | 10 | 25 | 50 | 100 | 250 |
|---|---|---|---|---|---|---|
| $r_i$ | 0.1002 | 0.1248 | 0.1324 | 0.1402 | 0.1448 | 0.1524 |
| $r_t$ | 0.0562 | 0.0564 | 0.0560 | 0.0546 | 0.0542 | 0.0762 |
| (ii) $K(\times 10^3)$ | 2.5 | 5 | 7.5 | 10 | 15 | 25 |
| $r_i$ | 0.2392 | 0.1248 | 0.0844 | 0.0767 | 0.0462 | 0.0279 |
| $r_t$ | 0.0456 | 0.0564 | 0.0443 | 0.0440 | 0.0164 | 0.0177 |

**Table 6.1:** Optimal retry periods $r_{opt} = \{r_i, r_t\}$ when $X = 1$: (i) $K = 5000$ instructions/task, (ii) $m = 10000$ tasks/mission.

| $m(\times 10^3)$ | 1 | 10 | 50 | 100 | 150 | 250 | 300 | 400 |
|---|---|---|---|---|---|---|---|---|
| $N_{min}$ | 6 | 7 | 9 | 10 | 12 | 13 | 14 | 15 |

**Table 6.2:** Numerical examples of $N_{min}$ vs. $m$.

of spares. Our analysis also includes the relation between the number of spares and the probability of dynamic failure, which is a key factor in determining the minimum number of spares so as to satisfy the required level of the probability of dynamic failure for a given mission.

# CHAPTER 7

# A TIME REDUNDANCY APPROACH TO TMR FAILURES USING FAULT-STATE LIKELIHOODS

## 7.1 Introduction

Instruction retry used in Chapter 6 intrinsically assumes almost-perfect fault detection, for which TMR systems require frequent voting, thereby inducing high time overhead. However, the probability of system crash due to multiple-channel faults is shown in [63] to be usually insignificant for general TMR systems, even when the outputs of computing modules are infrequently voted on as long as the system is free of latent faults. This chapter thus applies a simpler time-redundancy to TMR systems, re-execution of the whole task (program), than the approach of instruction retry taken in Chapter 6,

If the TMR failure had been caused by transient faults, system reconfiguration or Replacement of HardWare and Restart (RHWR), upon detection of a TMR failure, may not be desirable due to its high cost in both time and hardware. To counter this problem, we propose to, upon detection of a TMR failure, Re-execute the corresponding task on the Same HardWare (RSHW) without module replacement. Unlike simplex systems, program rollback is not adequate for TMR systems due to the associated difficulty of checkpointing and synchronization. So, we consider re-execution of tasks on a TMR system with infrequent voting. For example, since more than 90% of faults are known to be non-permanent — as few as 2% of field failures are caused by permanent faults [46] — simple re-execution may be an effective means to recover from most TMR failures. This may reduce (i) the hardware cost resulting from the hasty elimination of modules with transient faults and (ii) the recovery time that would otherwise increase, i.e., as a result of system reconfiguration. Note that system reconfiguration is time-consuming because it requires the location and replacement of faulty modules, program and data reloading, and resuming execution.

We shall propose two RSHW methods for determining when to reconfigure the system instead of re-executing a task without module replacement. The first (non-adaptive) method is to determine the maximum number of RSHWs allowable (MNR) before reconfiguring the system for a given task according to its nominal execution time without estimating the system (fault) state — somewhat similar to the multiple-retry policy applied to a general rollback recovery scheme in [70]. By contrast, the second (adaptive) method (i) estimates the system state with the likelihoods of all possible states and (ii) chooses the better of RSHW or RHWR based on their expected costs when the system is in one of the estimated states. RHWR is invoked if either the number of unsuccessful RSHWs exceeds the MNR in the first method or the expected cost of RSHW gets larger than that of RHWR in the second method. For the second method, we shall develop an algorithm for choosing between RSHW and RHWR upon detection of a TMR failure. We shall also show how to calculate the likelihoods of all possible states, and how to update them using the RSHW results and the Bayes theorem.

The chapter is organized as follows. In the following section, we present a generic methodology of handling TMR failures, and introduce the assumptions used. Section 7.3 derives the optimal voting interval ($X_v$) for a given nominal task-execution time $X$. The MNR of the first method and the optimal recovery strategy of the second method are computed for given $X$. We derive the probability density function (pdf) of time to the first occurrence of a TMR failure, the probabilities of all possible types of faults at that time, transition probabilities up to the voting time, the costs of RSHW and RHWR, and the problem of updating likelihoods of the system state and the recovery policy after an unsuccessful RSHW. Section 7.4 presents numerical results and compares two recovery methods of RSHW and RHWR. The chapter concludes with Section 7.5.

## 7.2 Detection and Recovery of a TMR Failure

Detection and location of, and the subsequent recovery from, faults are crucial to the correct operation of a TMR system, because the TMR system fails if either a voter fails at the time of voting or faults manifest themselves in multiple modules during the execution of a task. The fault occurrence rate is usually small enough to ignore coincident faults which are not caused by a common cause, but non-coincident fault arrivals at different modules are not negligible and may lead to a TMR failure.

Disagreement detectors which compare the values from the different voters of a TMR system can detect single faults, but may themselves become faulty. FTMP [23], JPL-STAR [2], and C.vmp [64] are example systems that use disagreement detectors. In FTMP, any detected disagreement is stored in error latches which compress fault-state information into error words for later identification of the faulty module(s). System reconfiguration to resolve the ambiguity in locating the source of a detected error is repeated depending on the source of the error and the number of units connected to a faulty bus. Two fault detection strategies — hard failure analysis (HFA) and transient failure analysis (TFA) — are provided according to the number and persistency of probable faulty units. These strategies may remove the unit(s) with hard failures or update the fault index (demerit) of a suspected unit. Frequent voting is required to make this scheme effective, because any faulty module must be detected and recovered before the occurrence of a next fault on another module within the same TMR system.

Voting in a TMR system masks the output of one faulty module, but does not locate the faulty module. One can, however, use a simple scheme to detect faulty modules and/or voter. Assuming that the probability of two faulty modules producing an identical erroneous output is negligibly small, the output of a module-level voter becomes immaterial when multiple modules are faulty [33]. A TMR failure can then be detected by using two identical voters and a self-checking comparator as shown in Fig. 7.1. These voters can be implemented with conventional combinational logic design [74]. The comparator can be easily made self-checking for its usually simple function: for example, a simple structure made of two-rail comparators in [41] for each bit can be utilized for its high reliability and functionality. This TMR structure can also detect a voter fault. When a TMR failure or a voter fault occurs, the comparator can detect the mismatch between the two voters that results from either the failure to form a majority among three processing modules, or a voter fault. (Note that using three voters, instead of two, would not make much difference in our discussion, so we will focus on a two-voter TMR structure.)

If the comparator indicates a mismatch between two voters at the time of voting, an appropriate recovery action must follow. Though RHWR has been widely used, RSHW may prove more cost-effective than RHWR in recovering from most TMR failures. To explore this in-depth, we will characterize RSHW with the way the MNR is determined. The simplest is to use a constant number of RSHWs irrespective of the nominal task-execution time and the system state which is defined by the number of faulty modules and the fault

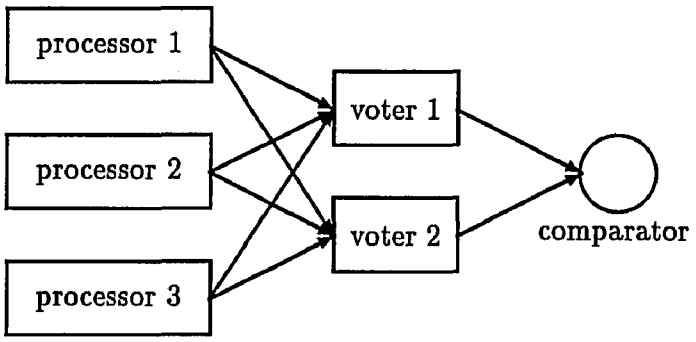



**Figure 7.1:** The structure of a TMR system with two voters and a comparator.

type(s). Taking into account the fact that the time overhead of an unsuccessful RSHW increases with the nominal task-execution time $X$, one can determine the MNR simply based on $X$, without estimating the system state. A more complex, but more effective, method is to decide between RSHW and RHWR based on the estimated system state. Since the system state changes dynamically, this decision is made by optimizing a certain criterion which is dynamically modified with the additional information obtained from each unsuccessful RSHW. In this adaptive method, the probabilities of all possible states will be used instead of one accurately-estimated state. Upon detection of a TMR failure, the expected cost of RSHW is updated and compared with that of RHWR. The failed task will then be re-executed, without replacing any module, either until RSHW recovers from the corresponding TMR failure or until the expected cost of RSHW becomes larger than that of task execution.[1] As the number of unsuccessful RSHWs increases, the possibility of permanent faults having caused the TMR failure increases, which, in turn, increases the cost of RSHW significantly.

Throughout this paper, we assume that the arrival of permanent faults and the arrival and disappearance of non-permanent faults are Poisson processes with rates $\lambda_p, \lambda_n$, and $\mu$, respectively.

[1]This procedure is described in the algorithm of Fig. 7.4.

## 7.3 Optimal Recovery from a TMR Failure Using RSHW

### 7.3.1 The Optimal Voting Interval

Let $X_i$ $(2 \leq i \leq n)$ be the nominal task-execution time measured in CPU cycles between the $(i-1)$-th and $i$-th voting, and let $X_1$ be that between the beginning of the task and the first voting, in the absence of any TMR/voter failure. As shown in Fig. 7.2, for $1 \leq i \leq n$ let $w_i$ represent the task-execution time from the beginning of the task to the first completion of the $i$-th voting possibly in the presence of some module failures, and let $W_i = E(w_i)$. Then $E(w_n) = W_n$ is the expected execution time of the task. Upon detection of a TMR failure, let $p$ and $q$ be the probabilities of recoverying a task with RSHW and RHWR, respectively, where $p + q = 1$. Assuming that the time overhead of reconfiguration is constant $T_c$, $W_n$ is expressed as a recursive equation in terms of $W_i$, $1 \leq i \leq n$. Let $F_i(t)$ $(2 \leq i \leq n)$ be the probability of a TMR failure in $t$ units of time from the system state at the time of the $(j-1)$-th voting, and let $F_1(t)$ be that from the beginning of the task. The probability of a recovery attempt (i.e., RSHW or RHWR) being successful depends upon $F_i(t)$. When a TMR failure is detected at the time of first voting (i.e., it occurred during the execution of the task portion corresponding to $X_1$), the system will try RSHW (or RHWR) with probability $p$ (or $q$) to recover from the failure. This process is renewed probabilistically for the variable $w_1$ which is the actual task-execution time corresponding to the nominal task-execution time $X_1$. Thus,

$$w_1 = \begin{cases} X_1 & \text{with probability } 1 - F_1(X_1) \\ X_1 + w_1 & \text{with probability } F_1(X_1)p \\ X_1 + T_c + w_1 & \text{with probability } F_1(X_1)q. \end{cases}$$

where $T_c$ is the setup time for system reconfiguration.

Let $T_v$ be the time overhead of voting which is in practice negligible. The above equation is also renewed for all $w_i$'s $(2 \leq i \leq n)$ after each successful recovery. Hence,

$$w_i = w_{i-1} + V_i + T_v \quad \text{for } 2 \leq i \leq n,$$

where $V_i$ is defined as the actual task-execution time between the $(i-1)$-th and $i$-th votings, i.e., $V_1 = w_1$ and

$$V_i = \begin{cases} X_i & \text{with probability } 1 - F_i(X_i) \\ X_i + w_i & \text{with probability } F_i(X_i)p \\ X_i + T_c + w_i & \text{with probability } F_i(X_i)q. \end{cases}$$

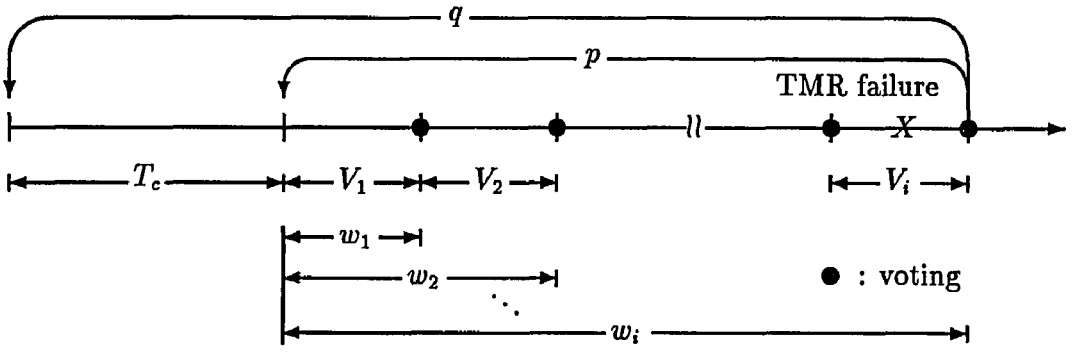



**Figure 7.2:** Graphical explanation for $V_i$ and $w_i$ for $1 \leq i \leq n$.

From the above equations, the following recursive expressions are derived for $2 \leq i \leq n$:

$$W_i = \frac{1}{1 - F_i(X_i)} (W_{i-1} + T_v + X_i + F_i(X_i) q T_c) .$$

Applying this recursively $n - 1$ times, we can get:

$$W_n = \sum_{j=1}^{n} \prod_{i=j}^{n} \left( \frac{F_j(X_j) q T_c + X_j + T_v}{1 - F_i(X_i)} \right) . \tag{7.1}$$

The optimal voting frequency is derived by minimizing $W_n$ with respect to $n$ and $X_i$, $1 \leq i \leq n$, subject to

$$\sum_{i=1}^{n} X_i = X.$$

If all inter-voting intervals are assumed to be identical then the constant voting interval is $X_i = X_v = \frac{X}{n}$ for $1 \leq i \leq n$, where an optimal value of $n$ must be determined by minimizing Eq. (7.1). Examples of $n$ for a given $X$ with typical values of $p$, $q$, $T_v$, and $T_c$ are shown in Table 7.1. The voting points can be inserted by a programmer or a compiler.

| $X$ $(hr.)$ | 3 | 5 | 10 | 20 | 30 | 40 |
|---|---|---|---|---|---|---|
| $n$ | 2 | 3 | 9 | 24 | 42 | 61 |

**Table 7.1:** $n$ vs. $X$ for $(p, T_c, T_v) = (0.9, 0.1X, 0.001)$

### 7.3.2 Pre-determination of Non-adaptive RSHWs

In the first method, we determine *a priori* the maximum number of RSHWs (MNR), $k_m$, based on $X$ without estimating the system state. The associated task will be re-executed up to $k_m$ times. As $X$ increases, the effect of an unsuccessful RSHW becomes more pronounced; that is, the possibility of successful recovery with RSHW (instead of RHWR) will decrease

with $X$ due to the increased rate of TMR failures, and the time overhead of an unsuccessful RSHW also increases with $X$ while the time overhead of RHWR remains constant. So, $k_m$ decreases as $X$ increases.

Let $C_1(k, X)$ be the actual time/cost of task execution in the presence of up to $k$ RSHWs for a task with the nominal execution time $X$, which can be expressed as:

$$\begin{aligned} C_1(k,X) &= Xp_s^1 + 2Xp_u^1p_s^2 + \cdots + kX\prod_{n=1}^{k-1} p_u^n p_s^k + \left(kX + \frac{T_c + X}{1 - F_1(X)}\right)\prod_{n=1}^{k} p_u^n, \\ &= \sum_{m=1}^{k-1} mX \prod_{n=1}^{m-1} p_u^n p_s^m + kX\prod_{n=1}^{k-1} p_u^n + \left(\frac{T_c + X}{1 - F_1(X)}\right)\prod_{n=1}^{k} p_u^n, \end{aligned} \tag{7.2}$$

where $p_s^n$ ($p_u^n$) and $F_1(X)$ denote the probability of the $n$-th RSHW becoming successful (unsuccessful) and the probability of a TMR failure during $X$ after system reconfiguration, respectively, where $p_s^n + p_u^n = 1, \quad 1 \le n \le k$. In fact, $p_s^n$ and $p_u^n$ cannot be determined without knowledge of the system state after the $(n-1)$-th unsuccessful RSHW, which is too complicated to derive *a priori*. We will approximate these probabilities using the following useful properties of a TMR system. Since the probability of permanent faults having caused the TMR failure increases with the number of unsuccessful RSHWs, $p_s^n$ is monotonically decreasing in $n$:

$$p_s^1 > p_s^2 > \cdots > p_s^k \Longleftrightarrow p_u^1 < p_u^2 < \cdots < p_u^k.$$

Though $p_s^1$ and $R(n) \equiv p_s^{n+1}/p_s^n$ depend upon $X$ and fault parameters, it is assumed for simplicity that $p_s^1$ is given *a priori* as a constant $P$ and $R(n)$ is a constant $R$ for all $n$. $C_1(k, X)$ of Eq. (7.2) is then modified in terms of $P$ and $R$:

$$C_1(k,X) = \sum_{m=1}^{k-1} mX \prod_{n=1}^{m-1}(1-PR^{n-1})PR^{m-1} + kX\prod_{n=1}^{k-1}(1-PR^{n-1}) + \left(\frac{T_c + X}{1 - F_1(X)}\right)\prod_{n=1}^{k}(1-PR^{n-1}). \tag{7.3}$$

The cost of RHWR, denoted by $C_2(X)$, is derived by using recursive equations:

$$C_2(X) = \frac{T_c + X}{1 - F_1(X)}. \tag{7.4}$$

Now, $k_m$ can be determined as the integer that minimizes $C_1(k, X)$ subject to $C_1(k, X) < C_2(X)$. Example values of $k_m$ for typical values of $P$ and $R$ are shown in Table 7.2.

### 7.3.3 Adaptive RSHW

In this method, the system chooses, upon detection of a TMR failure, between RSHW and RHWR based on their expected costs. RSHW will continue either until it becomes successful or until the expected cost of the next RSHW becomes larger than that of RHWR.

| $X/T_c$ $(hr.)$ | 1/1 | 2/1 | 3/1 | 4/1 | 5/1 |
|---|---|---|---|---|---|
| $k_m$ | 4 | 3 | 2 | 2 | 1 |

**Table 7.2:** $k_m$ vs. $X$ for $(P, F, R) = (0.8, 0.1, 0.8)$

The system state is characterized by the likelihoods of all possible states because one can observe only the time of each TMR failure detection, which is insufficient to accurately estimate the system state. The outcome of one RSHW, regardless whether it is successful or not, is used to update the likelihoods of states in one of which (called a *prior state*) the RSHW started. The possible states upon detection of a TMR failure can be inferred from the *posterior states* which are the updated prior states using the RSHW result and the Bayes theorem.

Unlike a simplex model, there are too many possible states and events to analyze a TMR system accurately. We will thus use the simplified Markov-chain model in Fig. 7.3 to derive the state probabilities and transition probabilities in a TMR system. The model consists of six states which are distinguished by the number of permanent faults and that of non-permanent faults, where two- and three- fault states are merged into one state due to their identical effects in our analysis. In Fig. 7.3 the transitions over the bidirectional horizontal lines result from the behavior of non-permanent faults and the transitions over the unidirectional vertical lines are caused by the occurrence of permanent faults. Note that even occurrences of near-coincident faults can be represented by sequential occurrences with slightly different interarrival times. The model, thus, includes only transitions between neighboring states — any transition from a state due to multiple faults occurs in two steps through one of its neighboring states.

Some faults may disappear without affecting the execution of a task. This happens when the latency of a fault is greater than its active duration, i.e., it will not manifest itself. Note that the occurrence of an error in a module during the task execution may produce an erroneous output for the task, even if the fault which had induced the error disappeared before producing the final output of the task. In other words, a transient fault may have permanent effects on task execution.[2]

The optimal recovery algorithm based on the adaptive method in Fig. 7.4 can be illus-

[2]In fact, this problem can be eliminated by resynchronizing the processors after a transient fault is detected [72]. This, however, requires frequent voting and additional mechanisms for detecting errors in each processor and resynchronizing the processors.

trated as follows. Upon detection of a TMR failure, the first step is to derive the probabilities of all possible states at time $X_f$ evolved from each prior state. Let $T_f^i$ be the time when the TMR system moved to the failure state from prior state $i$ during $[0, X_f]$, where $X_f$ is the time of detecting a TMR failure (i.e., a voting time). Occurrence of a TMR failure is then represented by an event $(T_f^i \leq X_f)$ for prior state $i$. We want to calculate the probabilities of all possible states $\pi_n^i(X_f)$ at voting time $X_f$ evolved from prior state $i$, which are actually conditional probabilities given the observed event $(T_f^i \leq X_f)$. They can be calculated from the probabilities of all types of TMR failures $\pi_m^i(T_f^i)$ at time $T_f^i$ and the transition probabilities $P_{mn}(X_f - T_f^i)$ during the remaining task-execution time, $X_f - T_f^i$. The probabilities of all possible states are thus

$$\pi_n(X_f) = \sum_i \sum_m \frac{\pi_i(0)}{F_{T_f^i}(X_f)} \int_0^{X_f} P_{mn}(X_f - t)\pi_m^i(t) dF_{T_f^i}(t), \tag{7.5}$$

where subscripts $i, m$ and $n$ indicate the prior state, the state at time $T_f^i$, and the state at the time, $X_f$, of detecting a TMR failure, respectively. As mentioned earlier, a voting failure may result from a voter fault or multiple-module faults. Multiple-module faults can be classified based on the number of modules with permanent faults: Type-I, Type-II, and Type-III failures represent zero, one, and more than one permanent-fault module, respectively, where all possible states of each type are listed in Fig. 7.3. Let $S(x, y)$ be the state with $x$ permanent-fault modules, $y$ non-permanent-fault modules, and $3 - x - y$ nonfaulty modules.

Although there are ten different states, we only need to consider six of them by merging (i) $S(0,3)$ into $S(0,2)$, (ii) $S(1,2)$ into $S(1,1)$, and (iii) both $S(2,1)$ and $S(3,0)$ into $S(2,0)$. This merger of states simplifies the model of a TMR system without losing model accuracy, because:

- By modifying the transition rates, one can make the simplified Markov-chain model in Fig. 7.3 represent a TMR system very accurately, and
- The merger is based on a realistic assumption that simultaneous occurrence of faults in different processor modules is highly unlikely.

Moreover, the merger does not change the analysis of a TMR failure because merged states have similar effects on the TMR failure as compared to the original states. For example, the merged states induce the same type of TMR failure, where the 'type' is determined by the number of permanent-fault modules. There are four possible states, $\{S(0,0), S(0,1), S(0,2), S(0,3)\}$ at time $X_f$, which led to Type-I failures (i.e., it was $S(0,1)$,

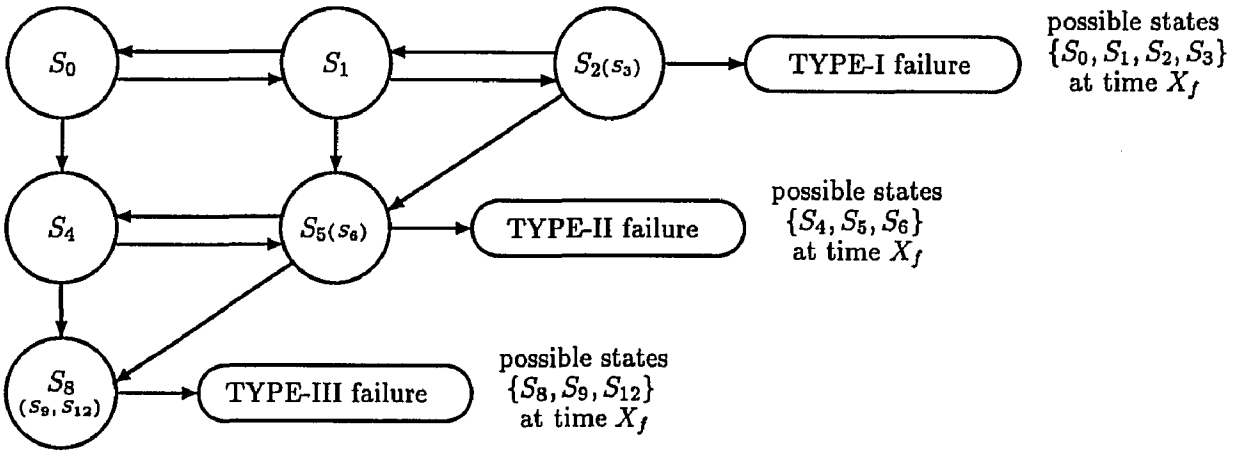



**Figure 7.3:** A simplified Markov-chain model for a TMR system.

$S(0,2)$, or $S(0,3)$ at time $T_f^i$, because a non-permanent fault might disappear after inducing error(s).). Type-II and Type-III failures have three possible states, $\{S(1,0), S(1,1), S(1,2)\}$ and $\{S(2,0), S(2,1), S(3,0)\}$, respectively, at time $T_f^i$ and $X_f$.

For notational simplicity, let state $S_i \equiv S(x,y)$ where $i = 4x + y$. Then, the set of all possible states after the merging is $\{S_i : i = 0,1,2,4,5,8\}$; out of these, $\{S_1, S_2, S_5, S_8\}$ are the set of possible fault states transited from $S_0, S_1$, and $S_2$ at time $T_f^0, T_f^1$, and $T_f^2$, respectively. $S_4$ and $S_5$ may change to $S_5$ (or $S_8$) at $T_f^4$ (or $T_f^5$), and $S_8$ remains unchanged due to the persistence of a permanent fault.

Let a *path* denote the transition trajectory between a pair of states. Since there are usually more than one path between a given pair of nodes, each of these paths is assigned an ID number. From the simplified model in Fig. 7.3, $T_f^i$ is the minimum-time path from $S_i$ to any type of TMR failure. Let $t_j^i$ be the time taken from $S_i$ to a TMR failure via path $j$. Then, $T_f^i = \min_j [t_j^i]$, where the *pdf* of $t_j^i$ is calculated by convolving the *pdf*'s of all sub-paths that make up path $j$. The *pdf* of a sub-path between two states $S_{j_k}$ and $S_{j_{k+1}}$ is obtained by using the distribution of sojourn time $t_{j_k}$ of $S_{j_k}$ with several exits in the Markov chain model (Fig. 7.3):

$$f_{j_k j_{(k+1)}}(t) = \frac{\lambda_{j_k j_{(k+1)}}}{\sum_{y \in \{E_{j_k}\}} \lambda_{j_k y}} \sum_{y \in \{E_{j_k}\}} \lambda_{j_k y} e^{-(\sum_{y \in \{E_{j_k}\}} \lambda_{j_k y})t}, \tag{7.6}$$

where $\{E_{j_k}\}$ represents the set of all outgoing arcs of $S_{j_k}$. Then, the *pdf* of $t_j^i$ is

$$f_{t_j^i}(t) = f_{ij_1}(t) * f_{j_1 j_2}(t) * \cdots * f_{j_f m}(t),$$

where path $j$ is composed of sub-paths $\{ij_1, j_1j_2, \cdots, j_fm\}$ and $S_m$ must be one of possible fault states: $S_m \in \{S_1, S_2, S_4, S_5, S_8\}$. (When the inter-arrival time of events such as fault occurrence, fault disappearance, and fault latency, is not exponentially distributed, we need a semi-Markov chain model in place of a Markov chain model.) Let $J_m^i$ represent the set of all paths to a fault state $S_m$ from $S_i$. The likelihood of a fault state $S_m$ at time $T_f^i$ is, then, equal to $\sum_{j \in J_m^i} \text{Prob}(t_j^i = T_f^i)$, which is obtained by:

$$\pi_m^i(T_f^i) = \sum_{j \in \{J_m^i\}} \frac{f_{t_j^i}(T_f^i)}{\sum_{y \in \{E^i\}} f_{t_y^i}(T_f^i)}, \tag{7.7}$$

where $E^i$ is the set of all paths to all possible fault states evolved from $S_i$, i.e., $E^i = \bigcup_m J_m^i$ and $m \in \{1, 2, 4, 5, 8\}$. The probabilities of $S_1$ and $S_2$ leading to Type-I failure are computed based on the behavior of non-permanent faults, i.e., depending on whether or not a non-permanent fault, after having induced some error(s), is still active when a second non-permanent fault occurs. Likewise, the probabilities of $S_4$ and $S_5$ leading to Type-II failure are computed by the behavior of a non-permanent fault, if it had occurred earlier than permanent fault(s). When an intermittent fault is considered, the fault state must be divided by fault active and fault benign states as in [60], which makes the problem too complicated to be tractable. The numerical examples of $F_{T_f^i}(X)$ and the mean of $T_f^i$ $(i = 0, 4)$ for several $X$ are given in Figs. 7.5 and 7.6, in which analytic results are compared against the results obtained from Monte-Carlo simulations.

In addition to $f_{T_f^i}$ and $\pi_m^i$, the transition probabilities $P_{mn}$ from $S_m$ to $S_n$ during $X_f - T_f^i$ must be derived in order to obtain the likelihood of every possible state at the time of voting (failure detection), $X_f$. Although the matrix algebra using the transition matrix or Chapman-Kolmogrov theorem can be applied to give accurate expressions, we will use a simplified method for computational efficiency at an acceptably small loss of accuracy. For the transition probabilities from $T_f^i$, we need not consider subsequent errors but can focus on only those states useful in choosing between RSHW and RHWR.

Observe that the occurrence rate, $\lambda_p$, of permanent faults is much smaller than both the appearance and disappearance rates of non-permanent faults. Using this observation, one can analyze the behavior of permanent faults separately from that of non-permanent faults. The transition probabilities due to the occurrence of permanent faults are represented by $P_{mn}(X_f - T_f^i)$ for $S_m \in \{S(x_1, y)\}$, $S_n \in \{S(x_2, y) : x_2 > x_1\}$, that is, $P_{mn}(X_f - T_f^i) = 0$ for $S_m \in \{S(x_1, y)\}$, $S_n \in \{S(x_2, y) : x_2 < x_1\}$, because of the persistence of permanent faults. Although these probabilities depend upon $\pi_m^i(t)$ $\forall t$, $T_f^i \le t \le X_f$, they are approximated by

using only the prior probabilities of source states, $\pi_m^i(T_f^i)$. This approximation causes only a very small deviation from the exact values because the occurrence rate of permanent faults is usually very small as compared to the other rates. For example, consider $P_{1n}$ for $n \geq 4$, i.e., transitions from $S_1$ due to the occurrence of permanent fault(s). The corresponding transition probabilities are derived from the model in Fig. 7.3 in terms of the *pdf*'s of sub-paths between two states. Let $T = X_f - T_f^i$, then

$$
\begin{aligned}
P_{18}(T) &= \int_0^T F_{58}(T-t) f_{15}(t) dt \\
P_{15}(T) &= \left( \frac{2\lambda_p}{2\lambda_p + \mu} + \frac{\mu}{2\lambda_p + \mu} e^{-(\lambda_p + \mu)(X_f - T_f)} \right) \int_0^T (1 - F_{58}(T-t)) f_{15}(t) dt \\
P_{14}(T) &= \left( \frac{\mu}{2\lambda_p + \mu} - \frac{\mu}{2\lambda_p + \mu} e^{-(\lambda_p + \mu)(X_f - T_f)} \right) \int_0^T (1 - F_{58}(T-t)) f_{15}(t) dt.
\end{aligned}
$$

The probability $\pi_1^i(T_f^i)$ for $S_1$ is thus reduced to $(1 - F_{15}(T))\pi_1^i(T_f^i)$. Likewise, transitions from other source states due to the occurrence of permanent faults can be derived. Consequently, the prior probabilities are transformed into $(1 - F_{25}(T))\pi_2^i(T_f^i)$, $(1 - F_{48}(T))\pi_4^i(T_f^i)$, and $(1 - F_{58}(T))\pi_5^i(T_f^i)$, respectively. Using these transformed prior probabilities, we will derive the transition probabilities based only on the behavior of non-permanent faults.

Considering only the behavior of non-permanent faults divides the above model into a two-state model $\{S_4, S_5\}$ and a three-state model $\{S_0, S_1, S_2\}$, as shown in Fig. 7.3. The transition matrix of the three-state model $\{S_0, S_1, S_2\}$ is derived by (i) using the Laplace transform which reduces the linear differential equations of three states to algebraic equations in $s$, (ii) solving the algebraic equations, and (iii) transforming the solution back into the time domain.

The linear differential equation of $\{S_0, S_1, S_2\}$ with only the effects of non-transient faults is $\dot{\Pi}(X_f) = T(X_f - T_f^i)\Pi(T_f^i)$, where

$$
T = \begin{bmatrix} -3\lambda_n & \mu & 0 \\ 3\lambda_n & -2\lambda_n\mu & 2\mu \\ 0 & 2\lambda_n & -2\mu \end{bmatrix}.
$$

The Laplace transform of T is:

$$
A = \begin{bmatrix} s + 3\lambda_n & -\mu & 0 \\ -3\lambda_n & s + 2\lambda_n\mu & -2\mu \\ 0 & -2\lambda_n & s + 2\mu \end{bmatrix}.
$$

The solution requires the inverse of $A$:

$$A^{-1} = \frac{\begin{bmatrix} s^2+(2\lambda_n+3mu)s+2\mu^2 & \mu(s+2\mu) & 2\mu^2 \\ 3\lambda_n(s+2\mu) & s^2+(2\lambda_n+3\mu)s+6\lambda_n\mu & 2\mu(s+3\lambda_n) \\ 6\lambda_n^2 & 3\lambda_n(s+2\lambda_n) & s^2+(5\lambda_n+\mu)s+6\lambda_n^2 \end{bmatrix}}{s^3+(5\lambda_n+3\mu)s^2+(6\lambda_n^2+6\lambda_n\mu+2\mu^2)s.}$$

Let the roots of $s^2+(5\lambda_n+3\mu)s+6\lambda_n^2+6\lambda_n\mu+2\mu^2$ be $\alpha$ and $\beta$, then $a_{ij}$, the $ij$-th element of $A$, can be obtained by partial fraction expansion:

$$a_{ij} = \frac{c_{(ij)1}}{s} + \frac{c_{(ij)2}}{s+\alpha} + \frac{c_{(ij)3}}{s+\beta}.$$

Since $c_{(ij)2}$ and $c_{(ij)3}$ are conjugates, $c_{(ij)2} = k_{ij}(\alpha,\beta)$ if $c_{(ij)3} = k_{ij}(\beta,\alpha)$. The effect of permanent faults changes the initial probabilities of $\{S_0, S_1, S_2\}$ to:

$$\Pi'(T_f^i) = [A_0\pi_0(T_f^i), A_1\pi_1(T_f^i), A_2\pi_2(T_f^i)]^T,$$

where $A_0 = (1-F_{04}(T))$, $A_1 = (1-F_{15}(T))$, $A_2 = (1-F_{25}(T))$.
Thus, the $i$-th column of the $3\times 3$ transition matrix $P(T)$ reduces to:

$$\begin{bmatrix} (\frac{2\mu^2}{\alpha\beta} + k_{1i}(\alpha,\beta)e^{-\alpha T} + k_{1i}(\beta,\alpha)e^{-\beta T})A_{i-1} \\ (\frac{6\lambda_n\mu}{\alpha\beta} + k_{2i}(\alpha,\beta)e^{-\alpha T} + k_{2i}(\beta,\alpha)e^{-\beta T})A_{i-1} \\ (\frac{6\lambda_n^2}{\alpha\beta} + k_{3i}(\alpha,\beta)e^{-\alpha T} + k_{3i}(\beta,\alpha)e^{-\beta T})A_{i-1} \end{bmatrix},$$

where

$$\begin{aligned} k_{11}(x,y) &= \frac{x^2+(2\lambda_n+3\mu)x+2\mu^2}{x(y-x)}, \\ k_{22}(x,y) &= \frac{x^2+(3\lambda_n+2\mu)x+6\lambda_n\mu}{x(y-x)}, \\ k_{33}(x,y) &= \frac{x^2+(5\lambda_n+\mu)x+6\lambda_n^2}{x(y-x)}, \\ k_{12}(x,y) &= \frac{\mu x+2\mu^2}{x(y-x)}, \qquad k_{21}(x,y) = \frac{3\lambda_n x+6\lambda_n\mu}{x(y-x)}, \\ k_{13}(x,y) &= \frac{2\mu^2}{x(y-x)}, \qquad k_{31}(x,y) = \frac{6\lambda_n^2}{x(y-x)} \\ k_{23}(x,y) &= \frac{2\mu x+6\lambda_n\mu}{x(y-x)}, \qquad k_{32}(x,y) = \frac{2\lambda_n x+6\lambda_n^2}{x(y-x)} \end{aligned}$$

The above equations indicate that the coefficients of exponentials in $A_0$, $A_1$, and $A_2$ include the effects of the occurrence of permanent fault(s) on the prior probabilities. Likewise, the transition matrix of a two-state model for $\{S_4, S_5\}$ can be derived as:

$$\begin{bmatrix} (\frac{\mu}{2\lambda_n+\mu} + \frac{2\lambda_n}{2\lambda_n+\mu}e^{-(2\lambda_n+\mu)T})A_4 & (\frac{\mu}{2\lambda_n+\mu} - \frac{\mu}{2\lambda_n+\mu}e^{-(2\lambda_n+\mu)T})A_5 \\ (\frac{2\lambda_n}{2\lambda_n+\mu} - \frac{2\lambda_n}{2\lambda_n+\mu}e^{-(2\lambda_n+\mu)T})A_4 & (\frac{2\lambda_n}{2\lambda_n+\mu} + \frac{\mu}{2\lambda_n+\mu}e^{-(2\lambda_n+\mu)T})A_5 \end{bmatrix},$$

where $A_4 = 1 - F_{48}(T)$ and $A_5 = 1 - F_{58}(T)$ also represent the effects of permanent-fault occurrences on the transitions to $S_8$. These transition matrices and probabilities (resulting from the occurrence of permanent faults) can describe all possible transitions in the simplified model of Fig. 7.3.

When the TMR system is in $S_2$, $S_5$ or $S_8$ at time $X_f$, RSHW will be unsuccessful again due to multiple active faults (in more than one module). If it is not in those states at time $X_f$ due to disappearance of active fault(s) after inducing some error(s), the system moves to a recoverable state by RSHW. Let $F_{T_f^i}(X)$ be the probability of a TMR failure evolved from $S_i$ during the execution time $X$, where $F_{T_f^i}$ is the probability distribution function of $T_f^i$. Since exact knowledge of the system state is not available, we estimate the state probabilities, which are then used to calculate the expected cost of a single RSHW as follows:

$$C_1(X) = X + \frac{T_c + X}{1 - F_{T_f^0}(X)} \left( \sum_{i \in \{2,5,8\}} \pi_i(0) + \sum_{i \in \{0,1,4\}} F_{T_f^i}(X) \pi_i(0) \right), \tag{7.8}$$

where $\pi_i(0)$ is the probability that the state before starting one RSHW (upon detecting a TMR failure) is $S_i$, i.e., the probabilities of the present states become those of the prior states for the next RSHW. The expected cost of RHWR is obtained similarly to Eq. (7.4):

$$C_2(X) = \frac{T_c + X}{1 - F_{T_f^0}(X)}. \tag{7.9}$$

When RSHW is unsuccessful or a voting failure occurs again, the (prior) state probabilities are updated with the additional information obtained from the RSHW using the Bayes theorem. The observed information tells us that a TMR failure has occurred again during the current execution. (Note that the TMR failure detection time during the current execution is $X_f$.) As a result, the prior probabilities of all possible fault states for the $(k+1)$-th RSHW ($\pi_i^{k+1}$) are renewed from those of the $k$-th RSHW ($\pi_i^k$):

$$\pi_i^{k+1} = \frac{\pi_i^k \text{ Prob(a TMR failure during } X_f \text{ from } S_i)}{\text{Prob(a TMR failure during } X_f)}, \tag{7.10}$$

where Prob(a TMR failure during $X_f$) $= \sum_i \pi_i^k$ Prob(a TMR failure during $X_f$ from $S_i$) $= \sum_{i \in S_T} \pi_i^k F_{T_f^i}(X_f)$. From Eq. (7.10) one can see that the probability of the TMR system being in a permanent-fault state increases with each unsuccessful RSHW, which, in turn, increases the chance of adopting RHWR over RSHW upon detection of next TMR failure. Using the above updated state probabilities, we can get the conditional probabilities of all states upon detection of a TMR/voting failure.

When RSHW is successful, one can likewise update the probabilities of possible states, which will then be used to guess the prior state of the next voting interval.

When the hardware cost is high and the time constraint is not stringent, one may do the following. Since the fault occurrence rate is much smaller than the disappearance rate of (existing) non-permanent faults, we may wait for a certain period of time (called a *back-off time*) in order for the current non-permanent fault(s) to disappear before task re-execution. An optimal back-off time is determined by minimizing the expected time overhead. When a task is re-executed without any back-off, the cost of one RSHW is equal to Eq. (7.8). When re-execution starts after backing off for $r$ units of time, the cost changes (due to the change of prior states):

$$C_1(r) = X + r + \frac{T_c + X}{1 - F_{T_j^0}(X)} \left( \sum_{i \in \{2,5,8\}} \pi_i(r) + \sum_{i \in \{0,1,4\}} F_{T_j^i}(X) \pi_i(r) \right),$$

$$\text{where } \pi_i(r) = \sum_{j \in S_T} F_{ji}(r) \pi_i(0).$$

The optimal back-off time is obtained by minimizing $C_1(r)$ with respect to $r$.

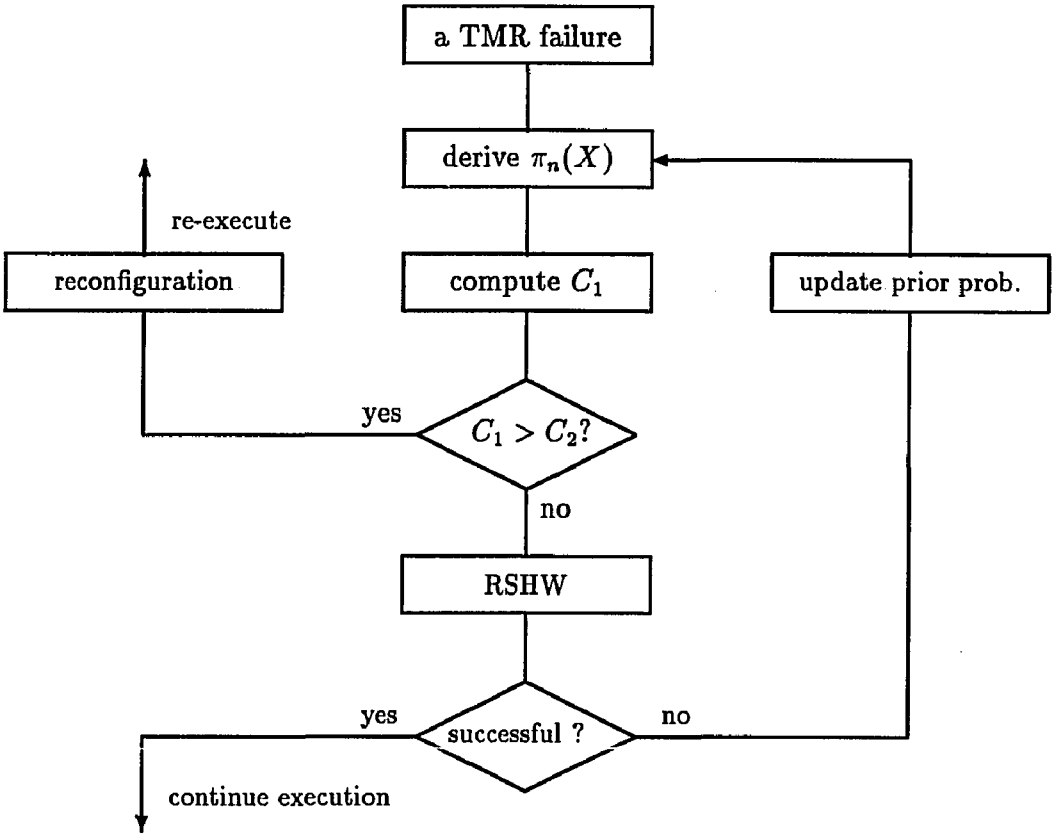



**Figure 7.4:** Algorithm to recover from a TMR failure by estimating the system state and comparing the costs of RSHW and RHWR.

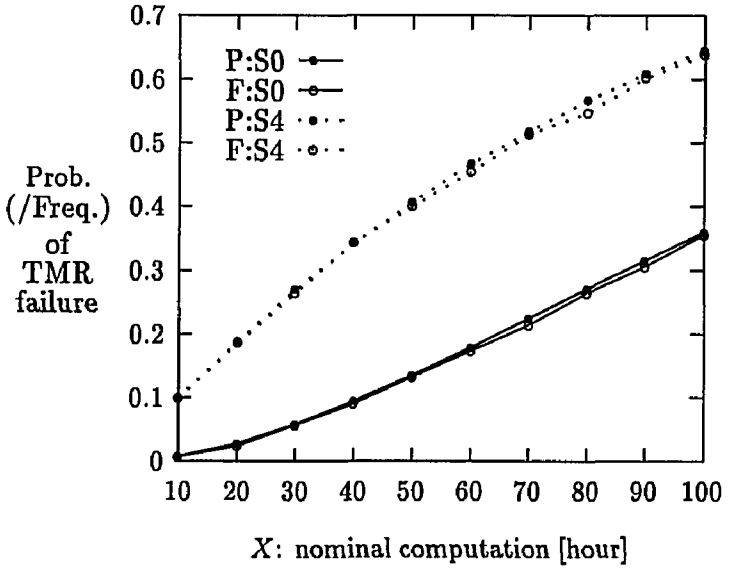



**Figure 7.5:** Probability/Frequency of a TMR failure obtained from the Markov-chain model (P:S0=from $S_0$ and P:S4=from $S_4$)/from simulations (F:S0=from $S_0$ and F:S4=from $S_4$).

## 7.4 Numerical Results and Discussion

A system with three replicated processing modules, two voters, and a comparator is simulated to compare the proposed method (called Method 1) with an alternative which is based on RHWR (called Method 2). Upon detection of a TMR failure, Method 1 will decide between RSHW and RHWR according to their respective costs. Method 2, however, will reconfigure the TMR entirely with a new healthy TMR or partially with healthy spare modules following an appropriate diagnosis. If a non-permanent fault does not disappear during the diagnosis, it will be treated as a permanent fault and replaced by a new, non-faulty spare. We assume that (A1) an unlimited number of tasks with the same nominal task-execution time are available to keep the running module busy, which simplifies the description of system workload, and (A2) there are an unlimited number of spares available. The performances of these two methods are characterized by the *overhead ratio*:

$$\mathrm{OVR}(X) \equiv \frac{E - X}{X},$$

where $E$ is the real execution time (including the RSHW and/or RHWR overheads) of a task whose nominal execution time is $X$.

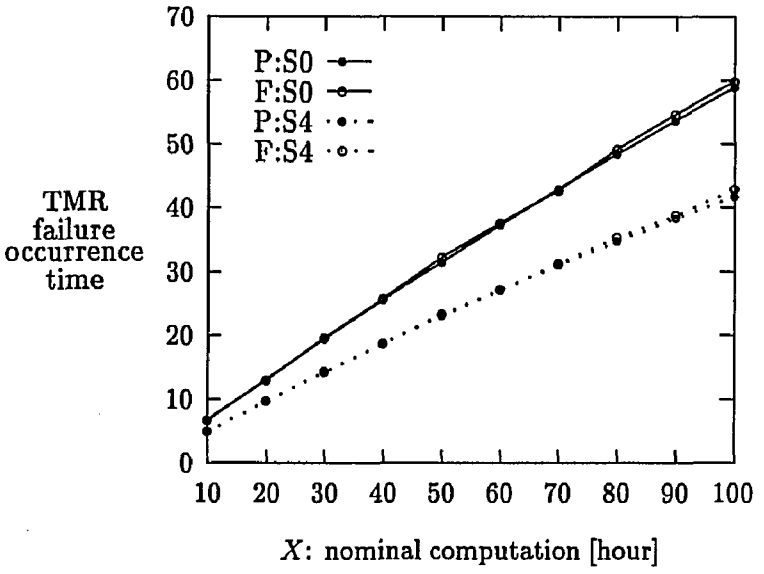



**Figure 7.6:** Mean TMR failure time ($E[T_f^0]$) obtained from analysis (P:S0=from $S_0$ and P:S4=from $S_4$), and from simulations (F:S0=from $S_0$ and F:S4=from $S_4$).

| $\frac{1}{\lambda_n}$ | $\frac{1}{\lambda_p}^*$ | $\frac{1}{\mu}$ | $\frac{1}{\mu}$ | $X^*$ |
|---|---|---|---|---|
| 200 | 3000 | 0.0001 | 0.002 | 50 |

**Table 7.3:** Parameter values used in simulations, all measured in hours.

We ran simulations under the fault generation process with the parameters as given Table 7.3, where the symbol * indicates a parameter varied while the others are fixed, in order to observe the effects of the parameter on OVR in both methods. Since fault occurrence/disappearance rates are difficult to estimate on-line, some experimental data or numerical data based on a model reflecting the maturity of design/fabrication process, the environmental effects, operating conditions, and the number and ages of components, can be used [66].

In Figs. 7.5 and 7.6, the probabilities of a TMR failure and the failure times from $S_0$ and $S_4$ are computed from the Markov-chain model and simulations, and are then compared. The simulation and modeling results are very close to each other. The modeling analyses proved to be very effective in determining when and how to choose between RSHW and

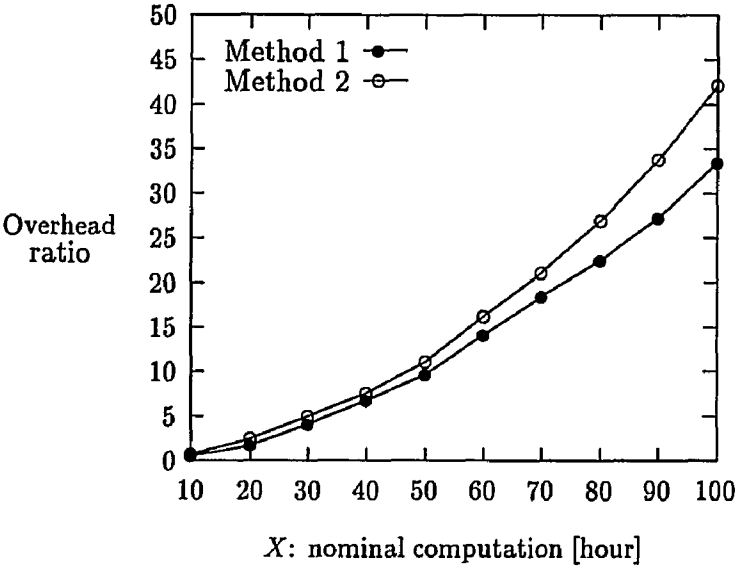



**Figure 7.7:** OVR($X$) [%] vs. $X$ for RSHW and RHWR, with the optimal number of votings for $T_v = 0.0005$ hour: (13,34,61,87,110,133,164,181,198,216).

RHWR under various conditions, as shown in Figs. 7.7–7.11.

The results obtained while varying $X$ from 10 to 100 hours with $T_c = 0.15X$, are plotted in Figs. 7.7–7.9. The OVRs of Methods 1 and 2 with the optimal number of votings are compared in Fig. 7.7. The difference between the OVRs of Method 1 and Method 2 increases significantly with $X$. When $X$ is small, the OVRs of the two methods are too small to distinguish, which is due mainly to the small probability of a TMR failure. Fig. 7.8 compares the multi-voting policy (with the optimal number of votings) and one voting policy. Generally, the overhead of a TMR system with infrequent voting increases significantly as $X$ increases, because the probability of a TMR failure increases with $X$; e.g., if there is no voting during the task execution, a TMR failure means the waste of the entire nominal execution time, $X$. As $X$ increases, the OVR of a one-voting policy increases more rapidly than that of multi-voting policy. The number of RHWRs — which is represented by the percentage of RHWR from the total number of simulations in Fig. 7.9 — will determine the hardware cost of spares used. The increase in this percentage is much larger in Method 2 than Method 1, since the number of TMR failures increases with $X$, and Method 1 can recover from most TMR failures with RSHW.

The second comparison is made while varying $T_c$ — the resetting time for system recon-

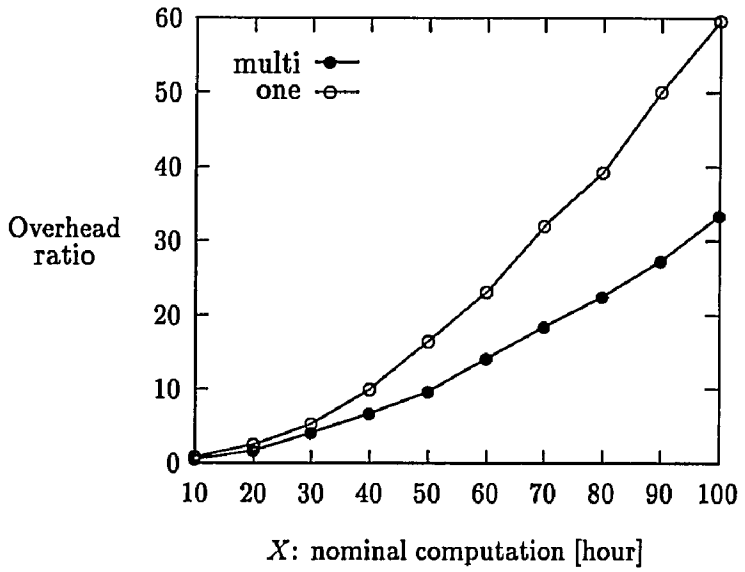



**Figure 7.8:** OVR($X$) [%] vs. $X$ for one voting and multi-votings with the optimal number of votings.

figuration — from 2.5 to 12.5 hours for $X$=50 hours, and the results are plotted in Fig. 7.10. A larger resetting time generally results in a larger OVR. Increasing $T_c$ greatly affects the performance of Method 2. But, it has little influence on the OVR of Method 1, since the system recovers from most TMR failures with RSHW which has nothing to do with $T_c$.

The third comparison in Fig. 7.11 is made while varying $\frac{\lambda_n}{\lambda_p}$ from 5 to 25, where $\lambda_n$ is fixed at 0.005 /hr, and $X$ = 50 hours and $T_c$ = 7.5 hours. The OVRs of both methods decrease with $\frac{\lambda_n}{\lambda_p}$, but the magnitude of decrease in Method 1 is larger than that in Method 2. This is because the probability of a TMR failure decreases as $\lambda_p$ decreases with $\lambda_n$ fixed, and because the probability of successful RSHW increases with $\frac{\lambda_n}{\lambda_p}$.

We simulated the proposed and other schemes for $10^5$ units of time with the fault parameters of Table 7.3 for each comparison (of the mean overhead ratios of different schemes). The fault parameters are assumed not to change during the simulation. Since the estimation of system states depends upon the fault parameters, they must be estimated first. This problem can be solved by assuming the parameters to be time-varying and estimating them on-line with certain adaptive methods which, in turn, require more samples.

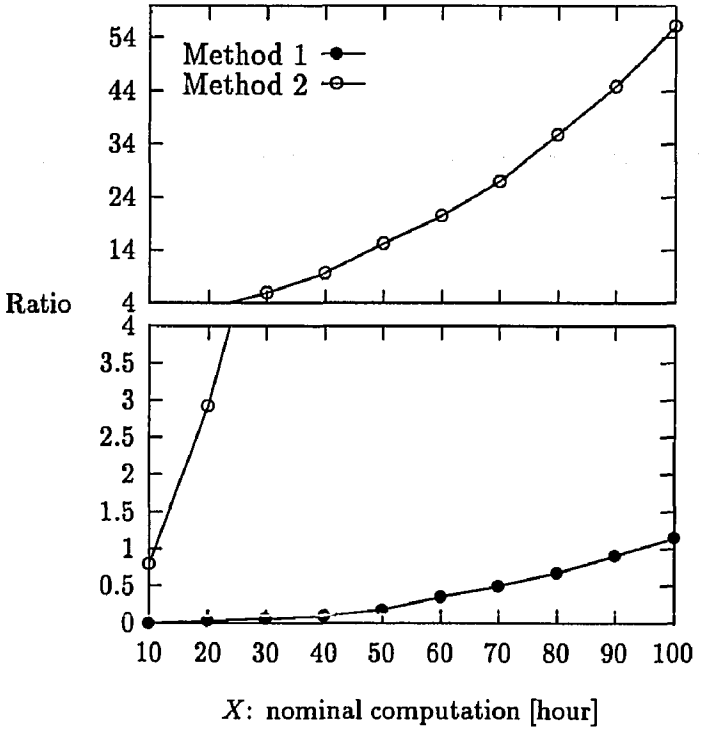



**Figure 7.9:** Ratio [%] of the number of reconfigurations to the total number of simulation runs.

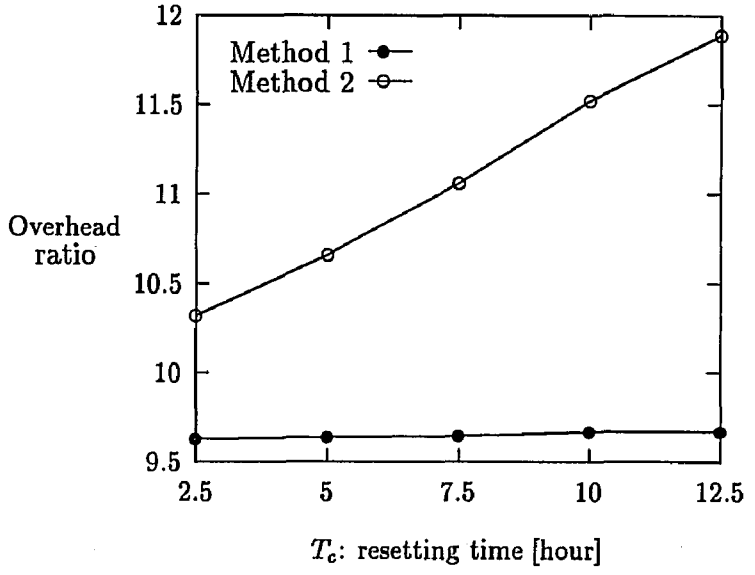



**Figure 7.10:** OVR($X$) [%] vs. $T_c$ for RSHW and RHWR.

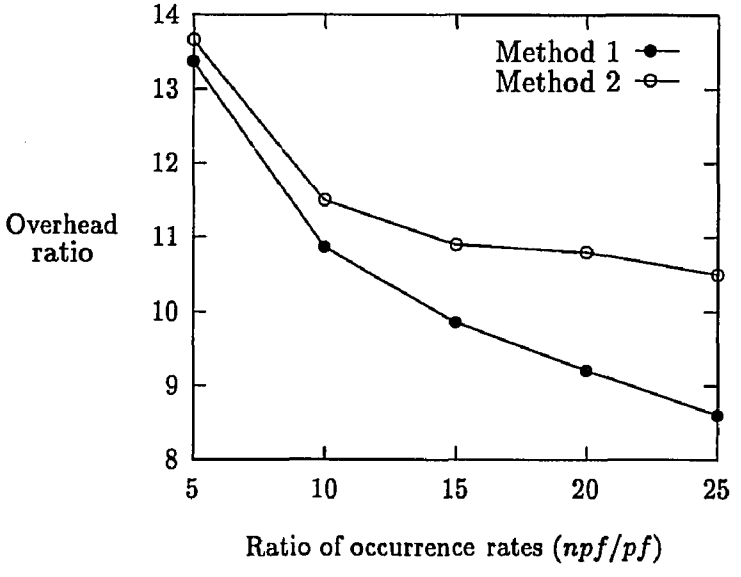



**Figure 7.11:** OVR($X$) [%] for different occurrence rates of non-permanent and permanent faults.

## 7.5 Conclusion

In this chapter, we have proposed a strategy for recovering TMR failures using two different methods that determine when and how to apply RHWR. Both methods are shown to outperform the conventional method based solely on reconfiguration. This finding is consistent with the fact that most faults are non-permanent, so simple re-execution can recover from non-permanent faults and the TMR structure can mask the effects of one faulty module.

The distinct characteristic of the proposed strategy is that it uses the estimated state of a TMR system even with incomplete observation of system states. Detection of a TMR failure and/or an unsuccessful RSHW does not always call for reconfiguration (RHWR) but requires us to derive and compare the expected costs of reconfiguration and one additional RSHW. Most TMR failures are represented by using a simplified Markov-chain model, and the TMR failure time and the probability of another unsuccessful RSHW are also analyzed with the model. One can therefore conclude that combining time and spatial redundancy appropriately can be effective in handling component failures.

# CHAPTER 8

# TASK SEQUENCING TO MINIMIZE THE EFFECTS OF NEAR-COINCIDENT FAULTS IN TMR CONTROLLER COMPUTERS

## 8.1 Introduction

TMR used in Chapters 6 and 7 is effective only if (A1) the voter is fault-tolerant, (A2) module faults are statistically independent, and (A3) additional detection and recovery schemes are available to retain its fault masking capability or prevent a *TMR failure* that results from sequentially-occurring faults in different modules. (A *TMR failure* is said to occur if the TMR system fails to form a majority of its module outputs.)

The triplicated voters in [1, 49] can overcome any critical single-point fault in the voter, thus satisfying A1. Most TMR systems in the field are based on A2 and adopt appropriate detection and recovery schemes to preserve the capability of masking faults occurring sequentially with sufficiently large inter-occurrence time intervals so as to recover from the effects of a faulty module before a next module fault occurs (thus satisfying A3). For example, several researchers [2, 9, 23, 58, 64, 72] have proposed the policies tolerating multiple faults by treating them only as sequential fault occurrences, one at a time, *with repair/recovery* between occurrences, thus requiring effective detection and identification of each faulty module in the TMR system.

However, under certain conditions we cannot ignore (near) coincident faults occurring in different modules, especially when the system is exposed to a harsh environment and/or the system is required to have very high reliability. For example, faults caused by EMI are likely to induce coincident faults in a TMR system [29]. Thus, a TMR failure caused by common-cause faults in multiple modules must also be detected and recovered. In [25], the effect of dependent faults inducing a TMR failure was eliminated by periodic resynchronization at

an optimal time interval. In that paper, the reliability of a TMR microcomputer system with dependent faults was considered, and a dependent-fault-tolerant operating chart was produced by executing different programs on different CPUs. In [27] we proposed an optimal instruction-retry policy to minimize the probability of *dynamic failure* (i.e., missing a task's deadline) for a TMR controller computer while considering the possibility of coincident faults that cause TMR failures. However, all of these approaches still focus on the behaviors of independent faults, and do not present any adequate means of tolerating, or minimizing the effects of, coincident faults.

The key idea of this chapter is based on the observation that two coincident module faults in a TMR system will *not* result in a TMR failure if (i) at any given time all three processor modules execute *different* tasks, (ii) the source of the two coincident module faults does not last long, and (iii) tasks are independent of one another. (As we shall see in Section 8.3.3, condition (iii) can be relaxed.) This fact implies that some TMR failures can be avoided by properly sequencing tasks on the three modules of the TMR system. Such task sequencing will be effective in dealing with coincident faults induced by transient environmental disruptions like EMI. This is different from the approach proposed in [25] in which processors 1, 2, and 3 execute tasks $(n+2), (n+1)$, and $n$ during the $n$-th cycle with a *fixed* pattern. Our approach is much more general and flexible than this. We first compute the probability mass function (*pmf*) of the number of TMR failures (i.e., two or more modules are faulty when each corresponding task is executed) for:

- *random task sequencing* in which tasks are randomly selected for execution, and
- *conventional task sequencing* in which all three modules execute the same task at the same time.

Then, we develop an *optimal task sequencing* by maximizing the mean number of TMR failures. We will not consider TMR failures due to fault occurrences during the voting process, which usually do not affect significantly the comparative numbers of TMR failures for both conventional and random task sequencing. Moreover, since the voters are generally implemented with simple combinational logic components [74] and the time required for voting is relatively small compared to task execution times, the probability of fault occurrences during the voting process is very small.

The rest of the chapter is organized as follows. In Section 8.2, we discuss fault and task models along with the assumptions used. In Section 8.3, we analyze the effects of

independent and coincident faults on both the random and conventional sequencing of tasks. Specifically, we compute the mean number of independent and/or dependent tasks producing correct executions results. An optimal task sequencing is also developed there. Section 8.4 presents demonstrative examples. The chapter concludes with Section 8.5.

## 8.2 Basic Model and Assumptions

### 8.2.1 Fault Model

Hardware modules are subjected to the same environment (temperature, humidity, and EMI) and often share the same clock and power supply, for which the commonly-used independence assumption for fault occurrences in different modules no longer holds. In particular, the harsh environment resulting from EMI (lightning, high-intensity radio frequency field, or nuclear electro-magnetic pulses) will affect the entire system and induce coincident or common-cause faults in the multiple modules of a TMR system. These faults may cause a TMR failure, or failure to form a majority of module outputs. In this paper, we consider a fault/failure model that accounts for *both* independent and common-cause faults. Physical defects during manufacture or component-aging effects are the main cause of independent module faults. Multiple (near-) coincident faults occur due mainly to environmental disruptions.

Occurrences of independent and common-cause module faults are assumed to follow Poisson processes with rates $\lambda_i$ and $\lambda_c$, respectively. [1] The durations of external disruptions inducing these faults are also assumed to follow Poisson processes with rates $\mu_i$ and $\mu_c$, respectively. In fact, since a fault may disappear without causing any error/failure or the effects of a fault may persist even after its disappearance, the duration of an error/failure is not always equal to the duration of a fault. However, we approximate error/failure durations using the knowledge of fault-duration information.

### 8.2.2 Task Model

A *mission* is broken down into several sequential *phases* and is accomplished by executing a set of computational tasks, three copies of each task on the three modules of a TMR system, during a *task interval* (TI), a basic time unit, defined as the time required to execute a task. The module outputs are voted on when these tasks are completed or

[1] Failure occurrences are modeled using information on the occurrences of faults, and the fault and error latencies.

produce outputs. We will begin with a basic task model in which all tasks have an identical execution time (= $\Delta t$ = one TI) and are independent of one another. This basic task model is covered in Sections 8.3.1 and 8.3.2. In Section 8.3.3, we extend the results of the basic task model to a more general (thus realistic) task model including tasks of different execution times and/or dependent tasks.

When the three copies of a task are executed by all three modules during the same TI, the execution results can be voted immediately upon completion of the task execution. However, if the three modules execute different tasks during each TI according to a certain sequencing policy, it is no longer simple to determine when to vote on the execution results. To circumvent this difficulty, we assume that all task execution results are saved and voted on later.[2] If there are tasks with tight timing constraints, they will be given priority to be executed early and their execution results will immediately be voted on. The tasks requiring synchronous interaction of all three modules with their environments such as actuators or peripherals, should also be executed simultaneously in all three processors and voted on immediately.

## 8.3 Task Sequencing Policies

If more than one copy of a task produce incorrect results due to multiple coincident module faults, the TMR system cannot generate a correct output for the task, thus resulting in a TMR failure. Let a mission or a mission phase consist of $N$ tasks and let $N_d$ ($N_f$) be the number of tasks producing correct (incorrect) results in the absence (presence) of TMR failures. We want to develop a method for sequencing tasks in a TMR system that maximizes the mean value of $N_d$ or $E(N_d)$. We first focus on the basic task model and then extend the basic model results to a more general task model. This problem is equivalent to minimizing $E(N_f) \equiv N - E(N_d)$, which is the mean number of TMR failures or voting failures. Occurrences of TMR failures depend on the fault model governing the fault occurrences and durations in individual modules as well as on the method of sequencing tasks in different modules.

[2] In typical current-generation systems, memory components have higher failure rates than any other components. Thus, the saved data may also be vulnerable to some faults. However, the memory can be equipped with another well-developed fault-tolerance scheme (like error detection and correction codes). Our idea and analysis will be based on the assumption that the saved data waiting for voting is safe by using appropriate fault-tolerance methods for memory components.

### 8.3.1 Comparison of Conventional and Random Task Sequencing

We derive and compare $E(N_f)$ for the conventional task sequencing that executes the copies of the same task in the three modules during the same TI and the random task sequencing that selects a task randomly for each module during a TI.

We consider only one occurrence of common-cause or independent fault in each module during the time interval of interest. Since fault inter-arrival times are significantly larger than the task execution time, the results derived under such consideration differ little from those under more realistic conditions assuming multiple-fault occurrences. We will first consider the simplest task model, where all tasks are assumed to have an identical execution time ($= \Delta t =$ one TI) and to be independent of one another; that is, tasks can be ordered arbitrarily as $\{t_1, t_2, \cdots, t_N\}$

Let $P^c_{jk}$ be the probability that a TMR failure occurs during the $j$-th TI and the durations of coincident faults inducing this failure are larger than $(k-1)\Delta t$ and smaller than $k\Delta t$ $(1 \le k \le N-j+1)$. We will consider such a duration equal to $k$ TIs. Given that a TMR failure occurs during the $j$-th TI, the time of this occurrence will be uniformly distributed over that interval, because a Poisson process has stationary and independent increments. Then, by using the probability density functions (*pdf*) of fault occurrences and durations, $P^c_{jk}$ can be derived for $1 \le j \le N-1$ as:

$$
\begin{aligned}
P^c_{j1} &= (e^{-\lambda_c(j-1)\Delta t} - e^{-\lambda_c j\Delta t}) \int_0^{\Delta t} \frac{1}{\Delta t} \int_0^{\Delta t - x} \mu_c e^{-\mu_c t} dt\, dx \\
&= (e^{-\lambda_c(j-1)\Delta t} - e^{-\lambda_c j\Delta t})(1 + \frac{1}{\mu_c \Delta t}(e^{-\mu_c \Delta t} - 1)), \\
P^c_{jk} &= (e^{-\lambda_c(j-1)\Delta t} - e^{-\lambda_c j\Delta t}) \int_0^{\Delta t} \frac{1}{\Delta t} \int_{(k-1)\Delta t - x}^{k\Delta t - x} \mu_c e^{-\mu_c t} dt\, dx \\
&= (e^{-\lambda_c(j-1)\Delta t} - e^{-\lambda_c j\Delta t}) \frac{1}{\mu_c \Delta t}(1 - e^{-\mu_c \Delta t})^2 e^{-\mu_c(k-2)\Delta t}, \\
&\qquad\qquad \text{for} \quad 2 \le k \le N-j \\
P^c_{j(N-j+1)} &= (e^{-\lambda_c(j-1)\Delta t} - e^{-\lambda_c j\Delta t}) \int_0^{\Delta t} \frac{1}{\Delta t} \int_{(N-j)\Delta t - x}^{\infty} \mu_c e^{-\mu_c t} dt\, dx \\
&= (e^{-\lambda_c(j-1)\Delta t} - e^{-\lambda_c j\Delta t}) \frac{1}{\mu_c \Delta t}(1 - e^{-\mu_c \Delta t}) e^{-\mu_c(N-j-1)\Delta t}, \qquad (8.1)
\end{aligned}
$$

where $P^c_{j(N-j+1)}$ includes both relatively long active transient faults (whose durations are larger than $(N-j)\Delta t$) and permanent faults during the $j$-th TI. All of the above is also applicable to independent faults by using the probability, $P^i_{jk}$, of coincident independent faults during the $j$-th TI with a duration of $k$ TIs.

Now, we derive the probability mass functions (*pmf*) of $N_f$ for both the conventional and random task sequencing. Let the dummy variable for the number of TMR failures be $\ell$,

where $0 \leq \ell \leq N$. In the conventional sequencing, the $N_f$ due to coincident faults is simply equal to $k$ (i.e., $\ell = k$ and $1 \leq j \leq N-\ell+1$) because only one possible fault occurrence is considered during the entire mission or mission phase. In the case of independent faults, we ignore the rare case when all three modules become faulty at the same time. A TI of a module $M_i$ is said to be *faulty* if $M_i$ is faulty during that TI. The $N_f$ due to independent faults is then the number of TIs during which two of the three modules are faulty — we call it two modules' *overlapping faulty TIs*. Let $P^i_{jk}$ and $P^i_{uv}$ be the probabilities of independent faults occurring in two modules. For any pair of $j$ and $k$ ($1 \leq j \leq N$ and $1 \leq k \leq N-j+1$), when $u \leq j$, $\ell = v-(j-u)$ for $1 \leq \ell \leq k-1$, and all module faults lasting longer than $j-u+k$ TIs cause $\ell = k$ TMR failures. When $u > j$, $\ell = v$ for $1 \leq \ell \leq k-(u-j)-1$, and all module faults lasting longer than $k-(u-j)$ TIs cause $\ell = k-(u-j)$ TMR failures. Considering the above and the fact that the three modules in a TMR system can be paired in three different ways, the *pmf* of $N_f$ due to coincident faults and independent faults is obtained by using $P^c_{jk}$ and $P^i_{jk}$ as follows:

$$P[N_f = \ell] = \sum_{j=1}^{N-\ell+1} P^c_{j\ell} + 3\sum_{j=1}^{N}\sum_{k=1}^{N-j+1} P^i_{jk}\left[\sum_{u=1}^{j}\left(P^i_{u(j-u+\ell)}\Pi_\ell(1,k-1) + \sum_{v=j-u+k}^{N-u+1} P_{uv}\delta_\ell(k)\right)\right.$$
$$\left. + \sum_{u=j+1}^{j+k-1}\left(P^i_{u\ell}\Pi_\ell(1,k-u+j-1) + \sum_{v=k-u+j}^{N-u+1} P_{uv}\delta_\ell(k-u+j)\right)\right], \tag{8.2}$$

where $\Pi_\ell(1,k)$ is a rectangular function of $\ell$ over $[1,k]$ (i.e., $\Pi_\ell(1,k) = 1$ if $\ell \in \{1,2,\cdots,k\}$ and 0 otherwise), and $\delta_\ell(k)$ is a delta function such that $\delta_\ell(k) = 1$ if $\ell = k$ and 0 otherwise.

In the random task sequencing, the $N_f$ due to common-cause faults is not $k$ but equal to an integer smaller than or equal to $k$. That is, the *pmf* of $N_f$ due to common-cause faults is derived as:

$$P(N_f = \ell) = \sum_{j=1}^{N}\sum_{k=\ell}^{N-j+1} P^c_{jk} P(k,\ell), \tag{8.3}$$

where $P(k,\ell)$ is the probability that any pair of modules execute same $\ell$ tasks during $k$ consecutive faulty TIs. $P(k,\ell)$ is obtained by using the combinatorics of three generic urns containing $N$ balls, because the method of sequencing tasks is random just like drawing balls from urns. Consider one pair of modules, say $M_1$ and $M_2$, which contribute $\ell_1$ TMR failures because of their overlapping faulty TIs. Suppose $k$ tasks (black balls) assigned to the faulty TIs of $M_1$ are distinguishable from the remaining $N-k$ tasks (white balls). Then one can cast the random task sequencing on the other module, $M_2$, into the problem of *sampling of balls without replacement* in the urn containing $k$ black balls and $N-k$ white balls. Noting that there are also $k$ faulty TIs of $M_2$, $k$ balls are drawn at random (simultaneously or

| Sequencing method \ $N$ | 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| Conventional | 1.142e−4 | 2.979e−4 | 4.948e−4 | 6.942e−4 | 8.941e−5 |
| Random | 7.995e−5 | 1.671e−4 | 2.612e−4 | 3.484e−4 | 4.202e−4 |

**Table 8.1:** Examples $E(N_f)$'s when $\lambda_i = 1e{-}4$, $\lambda_i = 1e{-}5$, $\mu_c = 1/3$, and $\mu_i = 1/3$, i.e., independent faults are likely to occur.

one-by-one without replacement) from $M_2$. The probability that $\ell_1$ black balls are selected among those $k$ balls drawn is simply derived as:

$$P(k,\ell_1) = \frac{{}_kC_{\ell_1}\ {}_{N-k}C_{k-\ell_1}}{{}_NC_k}, \quad \ell_1 \le k,\ k-\ell_1 \le N-k \Rightarrow 2k-N \le \ell_1 \le k. \tag{8.4}$$

Let $\ell_2$ be the number of TMR failures contributed by two pairs of modules, ($M_2$, $M_3$) and ($M_1$, $M_3$), then $\ell = \ell_1 + \ell_2$. Again, suppose $2(N-\ell_1)$ tasks (black balls) assigned to the faulty TIs of $M_1$ and $M_2$ are distinguishable from the remaining $N-2(N-\ell_1)$ tasks (white balls) in $M_3$. Since there are also $k$ faulty TIs of $M_3$, the conditional probability that $\ell_2$ black balls are selected among $k$ balls drawn from $M_3$ given $\ell_1$ is derived as:

$$P(k,\ell_2|\ell_1) = \frac{{}_{2(N-\ell_1)}C_{\ell_2}\ {}_{N-2(N-\ell_1)}C_{k-\ell_2}}{{}_NC_k},$$
$$\ell_2 \le 2(N-\ell_1),\ k-\ell_2 \le N-2(N-\ell_1) \Rightarrow k+N-2\ell_1 \le \ell_2 \le 2(N-\ell_1). \tag{8.5}$$

From Eqs. (8.4) and (8.5), $P(k,\ell_1,\ell_2) = P(k,\ell_1)P(k,\ell_2|\ell_1)$ is computed, and thus $P(k,\ell) = \sum_{\ell_1=1}^{\ell} P(k,\ell_1,\ell-\ell_1)$.

The number, $N_f$, of TMR failures caused by coincident independent faults in the case of random task sequencing is an integer not greater than the number of two modules' overlapping faulty TIs. The *pmf* of $N_f$ is thus derived from Eqs. (8.2) and (8.3) as:

$$P(N_f = \ell) = 3\sum_{j=1}^{N}\sum_{k=1}^{N-j+1} P_{jk}^{i}\left(\sum_{u=1}^{N}\sum_{v=\ell}^{N-u+1} P_{uv}P(k,v,\ell)\right), \tag{8.6}$$

where $P(k,v,\ell)$ is the probability that any pair of modules execute same $\ell$ tasks during their $k$ and $v$ faulty TIs, respectively, thus resulting in $\ell$ TMR failures. This probability can be obtained similarly to $P(k,\ell)$ by sampling balls in the urn containing $k$ (or $v$) black balls out of $N$ balls. (Recall that $k$ or $v$ represents the number of a module's faulty TIs.) $P(k,v,\ell)$ is equal to the probability that $\ell$ black balls are selected among $v$ (or $k$) balls drawn from the urn (the other module), because there are $v$ (or $k$) faulty TIs of the other module. That is, $P(k,v,\ell) = {}_kC_\ell\ {}_{N-k}C_{v-\ell}/\ {}_NC_v$ for $\ell \le k$ and $k-\ell \le N-v$ (i.e.,

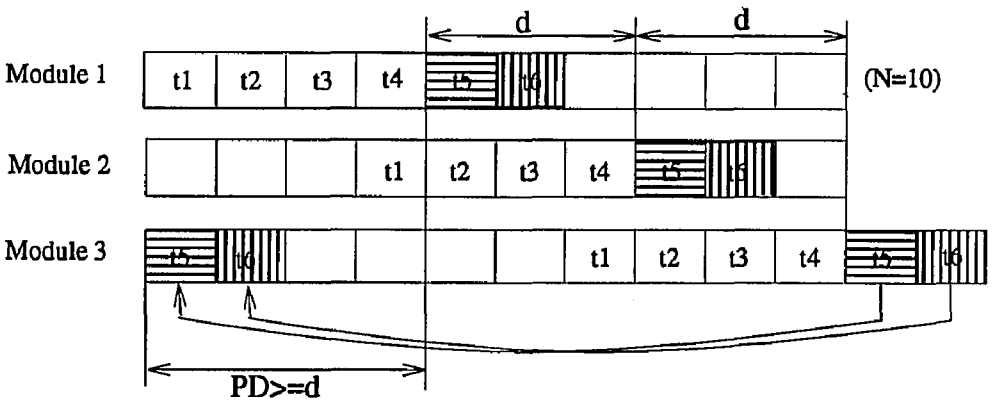



**Figure 8.1:** Wrapping tasks around in $M_3$ with $N = 20$ and $d = 3$, i.e., $t_{5+2\times3\ mod(10)} = t_1$.

$k+v-N \leq \ell \leq k$). Using the derived *pmf* of $N_f$, we finally compute:

$$E(N_f) = \sum_{\ell=1}^{N} \ell P(N_f = \ell). \tag{8.7}$$

Using a particular set of fault parameters, we have derived several example $E(N_f)$'s while varying $N$ and presented them in Table 8.1 for both the conventional and random task sequencing. These examples do not cover the entire set of missions or mission phases (represented by $N$ TIs) and fault environments. But the results indicate that the methods of sequencing tasks in which three modules execute different tasks during each TI may be effective in dealing with coincident faults and suggest the existence of an optimal task sequence that results in a smaller $E(N_f)$ than the random sequencing.

### 8.3.2 The Optimal Task Sequencing

We search for an optimal sequence of tasks that minimizes $E(N_f)$. This may be more complex to use but yield more reliable task execution results than the random and conventional sequencing. Let $M_1$, $M_2$, and $M_3$ be the first, the second, and the third modules labeled arbitrarily in a TMR system.

We consider a simple sequencing strategy such that the three copies of a task are executed on three different modules at different times separated by $d\Delta t$, which is called the *task distance* (TD). This is formally stated as follows.

1. In $M_1$, $N$ tasks are assigned to $N$ different TIs in an arbitrary order and let $t_j$ denote

the task assigned to the $j$-th TI.

2. In $M_2$ and $M_3$, $t_j$ is assigned to the $(j+d)_{mod\,N}$-th TI and the $(j+2d)_{mod\,N}$-th TI, respectively.

$A_{mod\,N}$ represents that the $A$-th TI is wrapped around the $N$-th TI if $A > N$, like $\{t_5, t_6, \cdots, t_{10}\}$ in Fig. 8.1, because we consider only $N$ consecutive TIs for executing all copies of $N$ tasks on three modules. That is,

$$A_{mod\,N} = \begin{cases} A & \text{if } A \leq N \\ N - A & \text{otherwise.} \end{cases}$$

TD should not be smaller than $d$ even after wrapping, like $t_5$ in Fig. 8.1. Since wrapping happens earlier in $M_3$ than $M_2$, the following condition for $M_3$ guarantees TD not to be smaller than $d$ after wrapping in both $M_2$ and $M_3$:

$$j - (j + 2d - N) \geq d \Longleftrightarrow 3d \leq N. \tag{8.8}$$

We now want to derive $E(N_f(d))$ in this sequencing strategy and find an integer $d_{opt}$ that minimizes $E(N_f(d))$ for $0 \leq d \leq \lceil \frac{N}{3} \rceil$.

Let $S_1$ be the subset of tasks which are not wrapped around, $S_2$ be the subset of tasks wrapped around only in $M_3$, and $S_3$ be the subset of tasks wrapped around in both $M_2$ and $M_3$. Then

$$\{t_1, t_2, \cdots, t_{N-2d}\} \in S_1,$$
$$\{t_{N-2d+1}, t_{N-2d+2} \cdots, t_{N-d}\} \in S_2,$$
$$\{t_{N-d+1}, t_{N-d+2} \cdots, t_N\} \in S_3.$$

Let $N_f^1$, $N_f^2$, and $N_f^3$ be the numbers of TMR failures occurred during the execution of tasks in $S_1$, $S_2$, and $S_3$, respectively. Using $\sum_{i=1}^{3} N_f^i = N_f$, the *pmf* of $N_f$ can be obtained by convolving the *pmf*'s of $N_f^1$, $N_f^2$, and $N_f^3$:

$$P(N_f = \ell) = P(N_f^1 = \ell) * P(N_f^2 = \ell) * P(N_f^3 = \ell). \tag{8.9}$$

First, consider the tasks affected by common-cause faults. In $S_1$, the TD of a task is $d$ for all pairs of task copies on ($M_1$, $M_2$) and ($M_2$, $M_3$). When faults occur between the first TI and the $d$-th TI, no more than $N-2d$ tasks may be contaminated. If the faults have an active duration of $k$ TIs, then $N_f^1 = \ell = k-d$ for $1 \leq \ell \leq N-2d-1$, and all faults lasting longer than $N-d$ TIs cause $\ell = N-2d$ TMR failures, because TD= $d$. If faults occur

between the $(d+1)$-th TI and the $(N-d)$-th TI, then $\ell = k-d$ for $1 \le \ell \le N-d-j+1$, similarly to the above. No TMR failure will occur when $j > N-d$. Consequently, when $1 \le j \le d$, the probability of $\ell$ TMR failures is increased by $P^c_{j(d+\ell)}$ for $1 \le \ell \le N-2d-1$ and by $\sum_{k=N-d}^{N-j+1} P^c_{jk}$ for $\ell = N-2d$. When $d+1 \le j \le N-d$, the probability of $\ell$ TMR failures is increased by $P^c_{j(d+\ell)}$ for $1 \le \ell \le N-2d-1$. Thus,

$$P(N_f^1 = \ell) = \sum_{j=1}^{d} \left( P^c_{j(d+\ell)} \Pi_\ell(1, N-2d-1) + \sum_{k=N-d}^{N-j+1} P^c_{jk} \delta_\ell(N-2d) \right) + \sum_{j=d+1}^{N-d} P^c_{j(d+\ell)} \Pi_\ell(1, N-d-j+1). \quad (8.10)$$

For any task in $S_2$, the TDs of task copies in module pairs $(M_1, M_2)$, $(M_2, M_3)$, and $(M_3, M_1)$ are $d$, $N-d$, and $N-2d$, respectively. When faults occur between the first and the $d$-th TI, the task copies on $M_3$ and $M_1$ are likely to be contaminated. If these faults are active for $k$ TIs, $N_f^2 = k-(N-2d)$ $(\le d)$, because the TD of task copies on $M_3$ and $M_1$ is equal to $N-2d$. When faults occur between the $(d+1)$-th TI and the $(N-d)$-th TI $(d+1 \le j \le N-d)$, only the task copies on $M_1$ and $M_2$ can be contaminated. For $d+1 \le j \le N-2d$, the only faults lasting longer than $(N-2d-j)+d$ TIs can induce $N_f^2 = k-(N-2d-j)-d-1$ $(\le d)$ TMR failures, because no task in $S_2$ is executed on any module during $N-2d-j$ TIs and the TD of task copies on $M_1$ and $M_2$ is $d$. For $N-2d+1 \le j \le N-d$, we get $N_f^2 = k-d$ $(\le N-j-d+1)$, simply because the TD of task copies on $M_1$ and $M_2$ is $d$, and the task copies on $M_1$ before the fault occurrence (i.e., those tasks executed between the $(N-2d+1)$-th TI and the $(j-1)$-th TI) are unaffected by the fault. Thus, simliarly to Eq. (8.10), we get

$$P(N_f^2 = \ell) = \sum_{j=1}^{d} \left( P^c_{j(N-2d+\ell)} \Pi_\ell(1, d-1) + \sum_{k=N-d}^{N-j+1} P^c_{jk} \delta_\ell(d) \right) + \sum_{j=d+1}^{N-2d} P^c_{j(N-d-j+\ell+1)} \Pi_\ell(1, d) + \sum_{j=N-2d+1}^{N-d} P^c_{j(d+\ell)} \Pi_\ell(1, N-d-j+1). \quad (8.11)$$

For any task in $S_3$, the TDs of the task copies on module pairs $(M_1, M_2)$, $(M_2, M_3)$, and $(M_3, M_1)$ are $N-d$, $d$, and $N-2d$, respectively. When faults occur between the first and the $d$-th TI $(1 \le j \le d)$, the task copies on $M_2$ and $M_3$ are likely to be contaminated. If the faults stay active for $k\Delta t$, where $d < k \le 2d-j$, $2d-j < k \le N-d-j+1$, $N-d-j+1 < k < N-d$, and $N-d \le k \le N-j+1$, $N_f^3$ becomes $k-d$, $2d-j+1$, $k-N+2d$, and $d$, respectively. When faults occur between the $(d+1)$-th TI and the $2d$-th TI $(d+1 \le j \le 2d)$, only the task copies on $M_3$ and $M_1$ can be contaminated. $N_f^3 (\le d-(j-d)+1)$ is equal to $k-N+2d$, because TD between the task copies on $M_3$ and $M_1$ is $N-2d$, and the task copies of $M_3$ before the

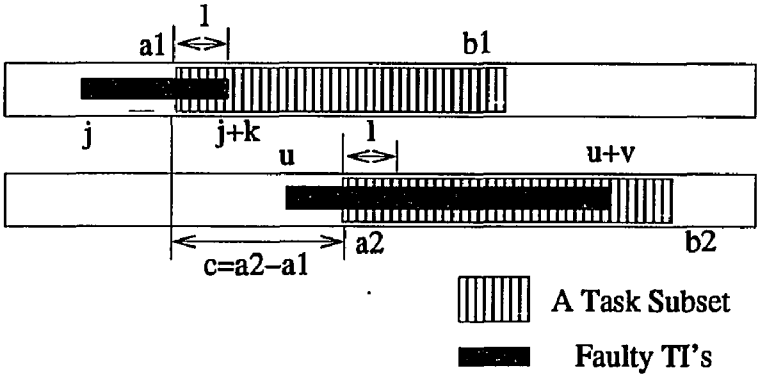



**Figure 8.2:** An example number of TMR failures ($\ell$) when $u \leq c + j$.

fault occurrence (between the $(d+1)$-th TI and $(j-1)$-th TI) are unaffected by the fault. Thus, if we treat the probability of $\ell$ TMR failures for two cases of $j$ (i.e., $1 \leq j \leq d$ and $d+1 \leq j \leq 2d$) by considering the above facts, we have

$$P(N_j^3 = \ell) = \sum_{j=1}^{d} \left( P^c_{j(d+\ell)} \Pi_\ell(1, d-j) + \sum_{k=2d-j+1}^{N-d-j+1} P^c_{jk} \delta_\ell(d-j+1) + P^c_{j(N-2d+\ell)} \Pi_\ell(d-j+2, d-1) + \sum_{k=N-d}^{N-j+1} P^c_{jk} \delta_\ell(d) \right) + \sum_{j=d+1}^{2d} P^c_{j(N-2d+\ell)} \Pi_\ell(1, 2d-j+1). \quad (8.12)$$

Similarly, we can obtain the *pmf* of the number, $N_j^i$, of TMR failures caused by independent faults. We derive the number of two modules' overlapping faulty TIs and the number of TMR failures occurred in executing each subset of tasks during these TIs, separately for both all three subsets ($S_1$, $S_2$, and $S_3$) and all three possible module pairs, $(M_1, M_2)$, $(M_2, M_3)$, $(M_3, M_1)$.

As shown in Fig. 8.2, suppose tasks in a subset, say $S_1$, be sequenced between the $a_1$-th TI and the $b_1$-th TI on a module, say $M_1$, and sequenced between the $a_2$-th TI and the $b_2$-th TI on the other module, say $M_2$. Then, TD= $c = a_2 - a_1$. Let $P^i_{jk}$ ($P^i_{uv}$) be the probabilities that an independent fault occurs in $M_1$ ($M_2$) during the $i$-th ($u$-th) TI with an active duration of $k\Delta t$ ($v\Delta t$). The values of $\{a_1, a_2, b_1, b_2\}$ of all three possible pairs of modules for all task subsets are given in Table 8.2.

Let $N_j^k(ij) = \ell$ be the number of TMR failures occurred during the execution of tasks in $S_k$ due to the coincident independent faults in $M_i$ and $M_j$. Then, the *pmf* of $N_j^k(ij) = \ell$ can be derived similarly to that of the conventional sequencing in Section 8.3.1 (Eq. (8.2)). Suppose a fault occurs at the $j$-th TI and lasts for $k\Delta t$ in a module, which is represented by $P_{jk}$. Using $P_{jk}$, we derive the probability of $N_j^k(ij) = \ell$ while varying $u$ and $v$ in the

| | $M_1$ and $M_2$ | $M_2$ and $M_3$ | $M_3$ and $M_1$ |
|---|---|---|---|
| $S_1$ | $\{1, d+1, N-2d, N-d\}$ | $\{d+1, 2d+1, N-d, N\}$ | $\{1, 2d+1, N-2d, N\}$ |
| $S_2$ | $\{N-2d+1, N-d+1, N-d, N\}$ | $\{1, N-d+1, d, N\}$ | $\{1, N-2d+1, d, N-d\}$ |
| $S_3$ | $\{1, N-d+1, d, N\}$ | $\{1, d+1, d, 2d\}$ | $\{d+1, N-d+1, 2d, N\}$ |

**Table 8.2:** $\{a_1, a_2, b_1, b_2\}$

other module.

First, let's consider the case of $a_1 = 1$. In this case, if a fault occurs after the $(a_2-a_1+j+k-1)$-th TI in a module or $b_2$-th TI in the other module, then there is no overlapping faulty TI between the two modules. Thus, we need to deal only with two cases: (i) $1 \le u \le c+j$ and (ii) $c+j+1 \le u \le r_1$, where $r_1 = \min\{c+j+k-1,\ b_2\}$. In Case (i), the number of TMR failures is $\ell = v-c-j+u$ for $1 \le \ell \le r_2-1$, where $r_2 = \min\{k,\ b_1-j+1\}$. However, all faults lasting longer than $c+j-u+r_2$ TIs cause $r_2$ TMR failures. In Case (ii), $\ell$ is simply equal to $v$ for $1 \le \ell \le r_3-1$, where $r_3 = \min\{k-u+c+j,\ b_1-j+1\}$, and all faults lasting longer than $r_3$ TIs cause $r_3$ TMR failures. Consequently, we have the *pmf* of $N_f^k(ij) \in \{N_f^1(12), N_f^1(31), N_f^2(23), N_f^2(31), N_f^3(12), N_f^3(23)\}$:

$$P(N_f^k(ij)=\ell) = \sum_{j=1}^{b_1}\sum_{k=1}^{N-j+1} P_{jk}^i \left[\sum_{u=1}^{c+j}\left(P_{u(c+j-u+\ell)}^i \Pi_\ell(1,r_2-1) + \sum_{v=c+j-u+r_2}^{N-u+1} P_{uv}^i \delta_\ell(r_2)\right)\right.$$
$$\left. + \sum_{u=c+j+1}^{r_1}\left(P_{ul}^i \Pi_\ell(1,r_3-1) + \sum_{v=r_3}^{N-u+1} P_{uv}^i \delta_\ell(r_3)\right)\right]. \quad (8.13)$$

When $a_1 > 1$, we should also deal with two cases of $j$: (I) $1 \le j \le a_1-1$ and (II) $a_1 \le j \le b_1$. We can compute the probabilities of Case II, just like Case (ii) of $a_1 = 1$ or Eq. (8.13). In Case I, a fault should last longer than $a_1-j$ TIs in the first module to have any overlapping faulty TI (i.e., $k \ge a_1-j+1$). We should also deal with two cases of $u$ different from those of $a_1 = 1$: (a) $1 \le u \le a_2-1$ and (b) $a_2 \le u \le r_4$, where $r_4(=r_1) = \min\{a_2+(k-a_1+j)-1,\ b_2\}$. In Case (a), $\ell$ is equal to $v-a_2+u$ for $1 \le \ell \le r_5-1$, where $r_5 = \min\{k-(a_1-j),\ b_1-a_1\}$, and all faults lasting longer than $a_2-u+r_5$ TIs cause $r_5$ TMR failures. In Case (b), $\ell$ is equal to $v$ for $1 \le \ell \le r_3-1$, where $r_6 = \min\{k-(a_1-j)-(u-a_2),\ b_1-a_1\}$, and all faults lasting longer than $r_6$ TIs cause $r_6$ TMR failures. Thus, we have the *pmf* of $N_f^k(ij) \in \{N_f^1(23), N_f^2(12), N_f^3(31)\}$:

$$P(N_f^k(ij)=\ell) = \sum_{j=1}^{a_1-1}\sum_{k=a_1-j+1}^{N-j+1} P_{jk}^i \left[\sum_{u=1}^{a_2}\left(P_{u(a_2-u+\ell)}^i \Pi_\ell(1,r_5-1) + \sum_{v=a_2-u+r_5}^{N-u+1} P_{uv}^i \delta_\ell(r_5)\right)\right.$$
$$\left. + \sum_{u=a_2+1}^{r_4}\left(P_{u\ell}^i \Pi_\ell(1,r_6-1) + \sum_{v=r_6}^{N-u+1} P_{uv}^i \delta_\ell(r_6)\right)\right]$$

$$+ \sum_{j=a_1}^{b_1} \sum_{k=1}^{N-j+1} P_{jk}^{i} \left[ \sum_{u=1}^{o+j} \left( P_{u(o+j-u+\ell)}^{i} \Pi_l(1,r_2-1) + \sum_{v=o+j-u+r_2}^{N-u+1} P_{uv}^{i} \delta_\ell(r_2) \right) \right.$$
$$\left. + \sum_{u=o+j+1}^{r_1} \left( P_{u\ell}^{i} \Pi_\ell(1,r_3-1) + \sum_{v=r_3}^{N-u+1} P_{uv}^{i} \delta_\ell(r_3) \right) \right]. \quad (8.14)$$

We ignored the effects of three modules' overlapping faulty TIs caused by independently-occurring faults. A TMR failure occurring due to a pair of faulty modules is treated as an exclusive event of TMR failure occurring due to the other pairs of faulty modules. In other words, $N_f^k(12)$, $N_f^k(23)$, and $N_f^k(31)$, become exclusive random variables. Hence, we can get $P(N_f^k = \ell)$ from Eqs. (8.13) and (8.14):

$$P(N_f^k = \ell) = P(N_f^k(12) = \ell) + P(N_f^k(23) = \ell) + P(N_f^k(31) = \ell) \quad k \in \{1,2,3\}. \quad (8.15)$$

just like Eq. (8.9).

The *pmf* of independent faults is obtained by convolving three equations: $P(N_f^1 = \ell)$, $P(N_f^2 = \ell)$, and $P(N_f^3 = \ell)$. The *pmf* of common-cause faults is also obtained from convolving Eqs. (8.10), (8.11), and (8.12). Consequently, $P(N_f = \ell)$ is obtained by adding the *pmf*'s of common-cause and independent faults. (Since we considered only one possible fault occurrence in each module, occurrences of common-cause and independent faults are also exclusive to each other.)

By using the constraint of $d$ in Eq. (8.8) and the *pmf* of $N_f$ derived as a function of $d$, we can determine $d_{opt}$ minimizing $E(N_f)$.[3]

### 8.3.3 The General Model for Different-Size and Dependent Tasks

We now treat a general model for tasks with different execution times. (Precedence constraints will be considered later in this section.) Let $\{t_1, t_2, \cdots, t_m\}$ be an ordered set of $m$ tasks, and let $n_i$ be the execution time of $t_i$ such that $\sum_{i=1}^{m} n_i = N$.

Suppose we use the same sequencing strategy as the basic task model, according to which three copies of each task are placed in the schedules of three modules with a separation interval $d\Delta t$, as shown in Fig. 8.3. Then, unlike the case of basic task model, when placing the copies of $t_k$ in the module schedules, one is likely to encounter a case where $2d+\sum_{i=1}^{k-1} n_i < N$ but $2d+\sum_{i=1}^{k} n_i > N$ on $M_3$, or $d+\sum_{i=1}^{k-1} n_i < N$ but $d+\sum_{i=1}^{k} n_i > N$ on $M_2$. That is, none of the remaining tasks, $t_k, t_{k+1}, \cdots, t_m$, can be executed before or at the end of the $N$-th TI of $M_2$ or $M_3$. Since we cannot generally divide a task, we have to extend the execution

[3] Although it is impossible to derive $d_{opt}$ in a closed-form, we can determine $d_{opt}$ numerically, i.e, computing $E(N_f)$ for every $d$ $(0 \le d \le \lceil \frac{N}{3} \rceil)$.

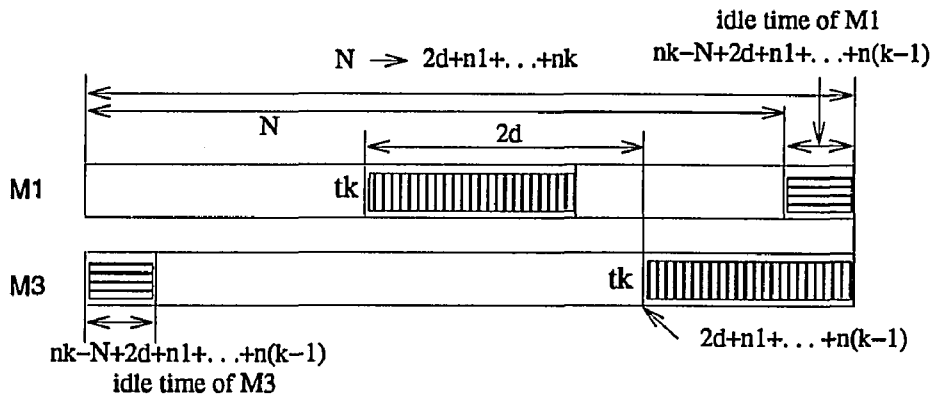



**Figure 8.3:** Sequencing $t_k$ such that $2d+\sum_{i=1}^{k-1} n_i < N$ but $2d+\sum_{i=1}^{k} n_i > N$ in $M_3$ by shifting the whole execution time in $M_3$ to the right (i.e., the end of a task) by $(2d+\sum_{i=1}^{k} n_i - N)$ TIs.

of each of these tasks beyond the $N$-th TI. For example, on $M_3$, $t_k$ is executed from the $(2d+\sum_{i=1}^{k-1} n_i)$-th TI to the $(2d+\sum_{i=1}^{k} n_i)$-th TI, and $t_{k+1}$ is not wrapped around to the first TI but extended to the $(2d+\sum_{i=1}^{k} n_i - N)$-th TI, as shown in Fig. 8.3. This extends[4] the cumulative execution time of tasks $t_1, \cdots, t_k$ from $N$ TIs to $(2d+\sum_{i=1}^{k} n_i)$ TIs.

Unlike the arbitrary ordering of tasks in the basic task model, we must develop an ordering algorithm by minimizing these extra TIs as follows.

An algorithm for ordering $m$ tasks by minimizing the extra TIs:

*Step 1*: Select any task and call it $t_1 (i = 1)$.

*Step 2*: $i := i+1$ (sequence tasks until tasks on $M_3$ need to be wrapped around).

If there is a task whose execution time is equal to $N-2d-\sum_{j=0}^{i-1} n_j$, then call it $t_i$ and go to *Step 3*.

Else if there is any task whose execution time is smaller than $N-2d-\sum_{j=0}^{i-1} n_j$, then (select one arbitrarily if more than one and) call it $t_i$ and go to *Step 2*.

Else select a task (of execution time $n_r$ TIs) among the remaining tasks by minimizing $2d+\sum_{j=0}^{i-1} n_j + n_r - N$ and call it $t_i$ and go to *Step 3*.

*Step 3*: $i := i+1$ (sequence tasks until tasks on $M_2$ need to wrapped around)

If there is a task whose execution time is equal to $N-d-\sum_{j=0}^{i-1} n_j$, then call it $t_i$ and go to *Step 4*.

Else if there is any task whose execution time is smaller than $N-d-\sum_{j=0}^{i-1} n_j$, then (select one arbitrarily if more than one and) call it $t_i$ and go to *Step 3*.

[4] adding idle TIs

Else select a task (of execution time $n_r$ TIs) among the remaining ones by minimizing $d+\sum_{j=0}^{i-1} n_j+n_r-N$ and call it $t_i$ and go to *Step 4*.

*Step 4*: Sequence the remaining tasks in arbitrary order.

Let $n_d$ be the number of TMR failures in this task model. Considering that a mission or mission phase is composed of $N$ TIs,[5] we can derive the *pmf* of $N_f$, the number of faulty TIs, similarly to the case of basic task model. From $P(N_f = \ell)$, we derive the *pmf* of $n_d$ under the assumption that if tasks are sequenced according to the proposed ordering algorithm then a TI is assigned to a task of execution time $n_i$ TIs with a probability $\frac{n_i}{N}$), i.e., a uniform distribution. Note that the size of a task for the general model is measured in number of TIs "composing" the task and the sum of the TIs composing all tasks in the mission or mission phase is $N$. A task will produce a correct output if no TI composing the task is faulty; in other words, a task would have an erroneous output if some of the TIs composing the task are faulty.

Given that $N_f = \ell$, the conditional probability of $n_d = r$ is obtained similarly to "sampling balls without replacement" in the urn containing $N$ balls of $m$ different colors (tasks) such that $\sum_{i=1}^{m} n_i = N$. Since there are $\ell$ faulty TIs, we randomly select $\ell$ balls from the urn and derive the *pmf* for the number, $r$, of different color balls among $\ell$ drawn balls. Let $S_r^\ell$ be the set of all subsets of task sizes, each subset consisting of $r$ task sizes $n_{s1}, n_{s2}\cdots, n_{sr}$ and $\sum_{i=1}^{r} n_{si} \geq \ell$ from the entire set of $m$ tasks. The number of subsets each composed of $r$ tasks is ${}_mC_r$, from which we obtain the set $S_r^\ell$. The probability that at least one ball is drawn from every color group corresponding to every task in the subset $\{n_{s1}, n_{s2}\cdots, n_{sr}\} \in S_r^\ell$ is:

$$P(n_d = r|N_f = \ell) = \sum_{S_r^\ell} \frac{1}{{}_NC_\ell} \left[ \sum_{i_1=1}^{n_{s1}} \sum_{i_2=1}^{n_{s2}} \cdots \sum_{i_r=1}^{n_{sr}} \left(\sum_{j=1}^{r} (n_{sj}-i_j)\right) C_{\ell-r} \right], \qquad (8.16)$$

which includes all the cases of $r$ different color balls drawn by using $r$ summations as well as all combinations of $\ell-r$ balls selected from the remaining balls in each case (i.e., from $\sum_{j=1}^{r}(n_{sj}-i_j)$ balls), which can be obtained by avoiding repeated cases. The *pmf* of $n_d$ is thus derived as:

$$\begin{aligned} P(n_d = r) &= \sum_{l=1}^{N} P(n_d = r|N_f = \ell)P(N_f = \ell) \\ &= \sum_{l=1}^{N} \sum_{S_r^\ell} \frac{1}{{}_NC_\ell} \left[ \sum_{i_1=1}^{n_{s1}} \cdots \sum_{i_r=1}^{n_{sr}} \left(\sum_{j=1}^{r} (n_{sj}-i_j)\right) C_{\ell-r} \right] P(N_f = \ell). \qquad (8.17) \end{aligned}$$

[5]The effects of the extra TIs required to handle different size tasks are not considered, because (i) $2d+n_1+\ldots+n_k-N$ is smaller than the execution time of any remaining task, (ii) the total number of TIs during which tasks are actually executed in each module is still $N$, and (iii) the number of the extra TIs depends on the given task set and thus is difficult to determine. Since $N >> 2d+n_1+\cdots+n_k-N \geq 0$, this approximation makes little difference.

As a result, we compute all $E(n_d)$'s for $0 \leq d \leq \lceil \frac{N}{3} \rceil$ and determine $d_{opt}$ that minimizes $E(n_d)$ numerically as we did for the basic task model.

To calculate Eq. (8.17), we need to check the inequality $\sum_{i=1}^{r} n_{si} \geq \ell$ ${}_mC_r$ times to obtain $S_r^\ell$ for $1 \leq r \leq m$, and thus a total of $\sum_{i=1}^{m}$ ${}_mC_i$ $(= 2^m)$ times. In addition, at least $r+1$ loops are required to compute the probability for each element (subset) in $S_r^\ell$, $1 \leq \ell \leq N$. Thus, the amount of computation required for Eq. (8.17) increases rapidly as $m$ and $N$ increase. Let $n_0$ be the Greatest Common Divisor $(GCD)$ of $\{n_1, n_2, \cdots, n_m\}$. If $n_0 > 1$, we reduce significantly the required computation by using the adjusted execution times, which are obtained by dividing the original execution times by $n_0$, e.g., $n_i' = \frac{n_i}{n_0}, N' = \frac{N}{n_0}$. The amount of computation is then proportional to $N'$, not $N$. It is thus better to change the size of a few tasks (typically one or two) to obtain a larger $n_0$ (thus a smaller $N'$), i.e., if $GCD(\{n_1, n_2, \cdots, n_i+\Delta n, \cdots, n_m\}) > GCD(\{n_1, n_2, \cdots, n_i, \cdots, n_m\})$ for a small $\Delta n$, then we change $\{n_1, n_2, \cdots, n_i, \cdots, n_m\}$ to $\{n_1, n_2, \cdots, n_i+\Delta n, \cdots, n_m\}$. After this change, the real execution time of the $i$-th task is $n_i$ and all three modules are idle for $\Delta n$ TIs. For example, if a set of tasks with execution times $\{18, 12, 8, 3\}$ is given, we would better use $\{18, 12, 9, 3\} \Longrightarrow \{6, 4, 3, 1\}(n_0 = 3)$ by assigning an idle time $(\Delta n = 1)$ to the third task.

Now, we want to consider a task model in which some tasks are dependent on others, i.e., there are precedence constraints. We define a new task for each subset by merging all dependent tasks within the subset into consecutive TIs, thus obtaining a new set of tasks whose sizes are equal to the sum of sizes of all merged tasks. The problem of optimally sequencing tasks in such a newly-defined set is then equivalent to a different-size task model. However, even when no task is fitted in the remaining TIs $(= N-2d-\sum_{i=1}^{k-1} n_i$ or $N-d-\sum_{i=1}^{k-1} n_i)$ on $M_3$ or $M_2$, no extra TIs are required, unlike the different-size task model. Since a newly-defined task is composed of dependent tasks, parts of it can be sequenced during both the TIs near the $N$-th TI and the TIs near the first TI after wrapping around.

## 8.4 Numerical Examples

In this section, we present the numerical results derived from both the basic and general task models under certain fault-behavior conditions.

We assume fault-related parameters, $\lambda_c = 10^{-6}$, $\lambda_i = 10^{-4}$, $\mu_c = 1/6$, and $\mu_i = 1/6$, all in number of TIs. Although independent faults occur more frequently than common-cause faults in each individual module, most TMR failures occur due to common-cause faults, as shown in Fig. 8.4. Note that coincident independent faults in multiple modules are

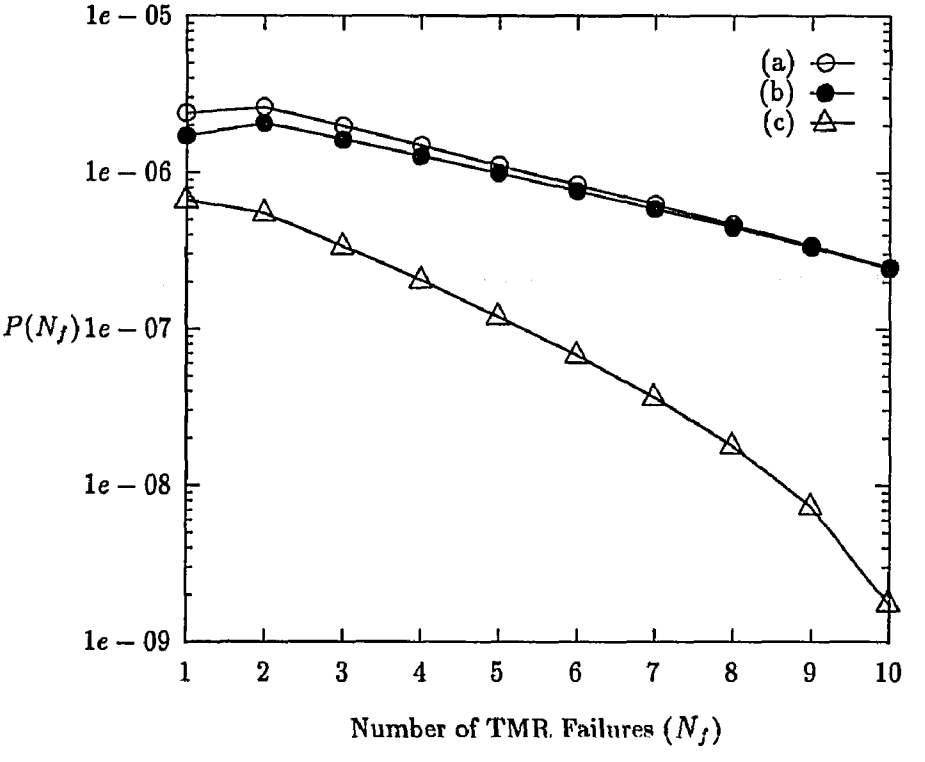



**Figure 8.4:** The *Pmf* of the number, $N_f$, of TMR failures due to (a) both common-cause and independent faults with $P(N_f = 0) = 0.99988$, (b) only common-cause faults with $P(N_f = 0) = 0.9999$, and (c) only independent faults with $P(N_f = 0) = 0.99998$.

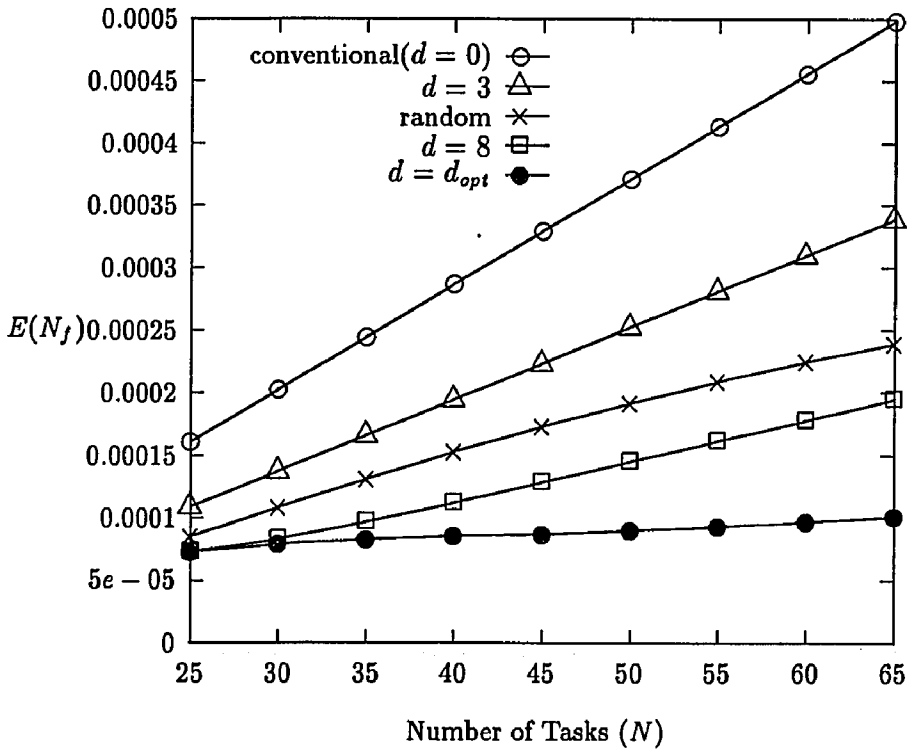



**Figure 8.5:** Mean number of TMR failures while varying the number $N$ of tasks for several sequencing methods with $d_{opt} = \{8, 10, 11, 13, 15, 16, 18, 20, 21\}$.

significantly rarer than common-cause faults during a mission or mission phase. If a large number of TMR failures occur during a mission or mission phase, from Fig. 8.4 one can see that their main causes are common-cause faults.

In Fig. 8.5, the mean numbers of TMR failures are derived for several sequencing methods while varying the number of tasks during a mission or mission phase. The conventional task sequencing turns out to be the worst, i.e., it has the largest $E(N_f)$ for any $N$ and the fastest increase of $E(N_f)$ as $N$ increases. When tasks are sequenced with a constant $TD = d > 0$, there may be a certain value of $d$ that performs similarly to the random sequencing. Clearly, the random sequencing works better than the case of $d \leq 3$ but worse than that of $8 \leq d \leq d_{opt}$. The increase of $E(N_f)$ in the optimal sequencing is negligible as compared to that in any other sequencing method.

To investigate the effects of $d$ on $E(N_f)$ under different fault-behavior conditions, we derived the mean numbers of TMR failures while varying $d$ from 0 to $d_{opt}$ for $N = 30$ in Fig. 8.6. Case (b), in which common-cause faults occur more frequently than case (a), has shown improvements by using a better $d$. In other words, the use of a better $d$ achieves

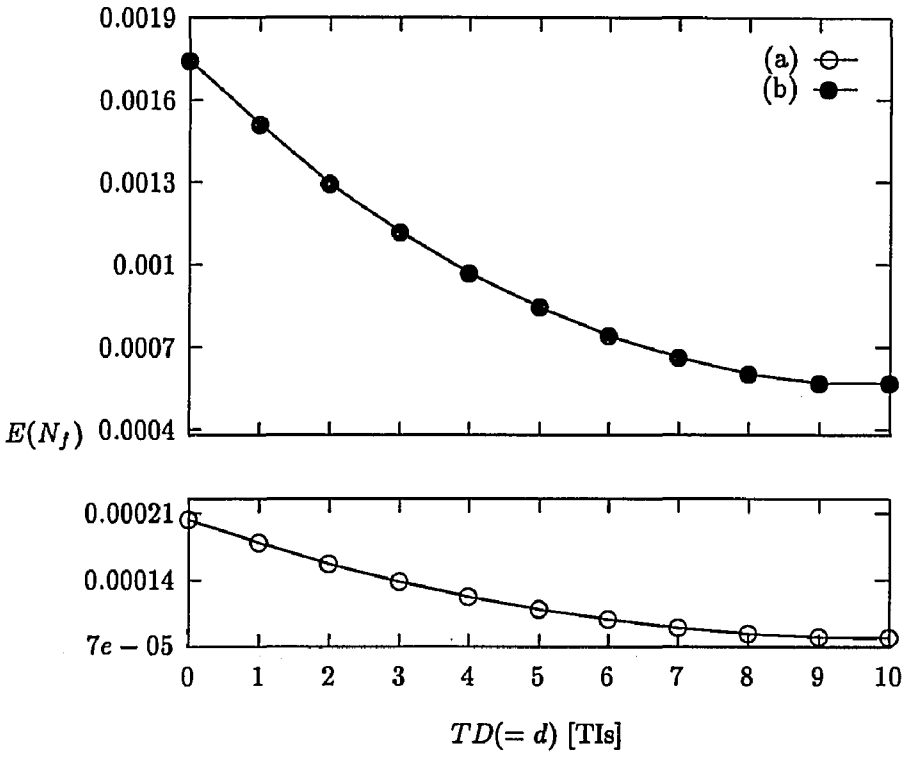



**Figure 8.6:** Mean number of TMR failures ($E(N_f)$) while varying task distance, $d$, under two different conditions of common-cause faults, when $N = 30$ (*basic model*): (a)$\lambda_c = 10^{-6}$ ($E(N_f) = 8.76e{-}4$ for the random sequencing) and (b)$\lambda_c = 10^{-5}$ ($E(N_f) = 1.089e{-}4$ the random sequencing).

more when the effect of common-cause faults is severer.

In the general task model, we obtained similar results. In Fig. 8.8, the use of a better $d$ (i.e., a larger $d \leq d_{opt}$) results in a smaller $E(n_d)$, and the relative effect of using a better $d$ is also larger when common-cause faults are more likely to occur as in cases (b) and (d) of the figure.

We also dealt with two task sets:

$$T_1 = \{1,2,4,3,2,4,1,1,3,2,4,3\} \text{ and } T_2 = \{3,5,7,5,4,6\},$$

which have the same number of TIs but different number of tasks ($m_1 = 12$ and $m_2 = 6$). Although the numbers ($N_f$) of faulty TIs of two task sets are the same as the case of $N = 30$ in Fig. 8.6, $T_1$ suffers more TMR failures (larger $E(n_d)$) because $m_1 > m_2$. However, the difference of $E(n_d)$'s in the two task sets decreases as $d$ increases. This is because the increase of $E(n_d)$ as a result of increasing $m$ becomes less pronounced as $d$ increases, which is shown similarly in Fig. 8.5.

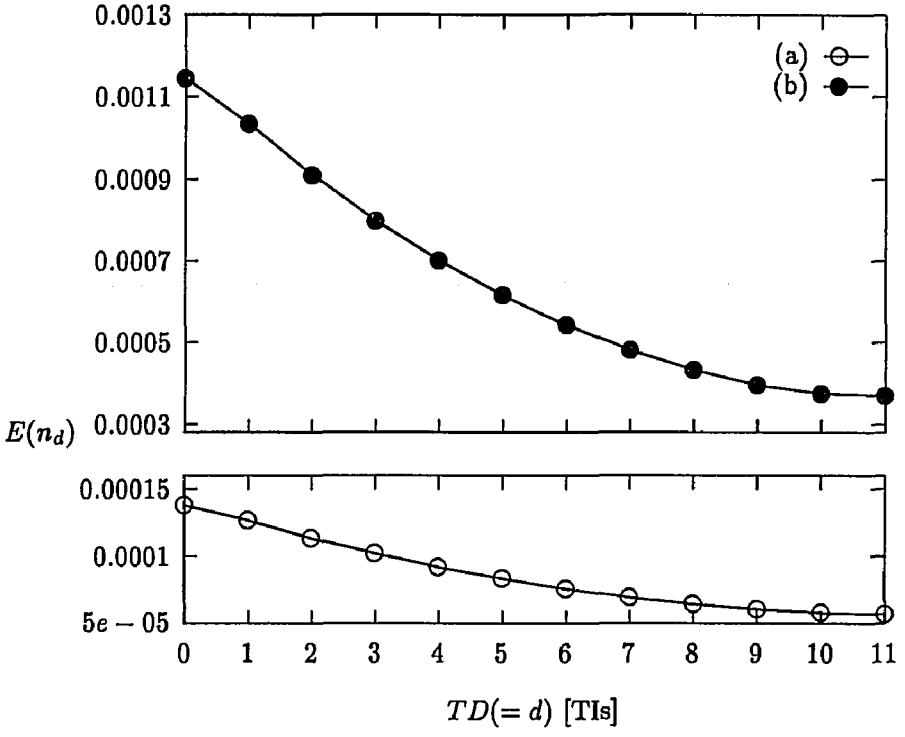



**Figure 8.7:** Mean number of TMR failures ($E(n_d)$) while varying task distance $d$ under two different conditions of common-cause faults for a *general* task set $\{(1,4,1),(3),(3,1),(4,2),(5),(3,5)\}$ ($m = 11$ and $N = 33$), where tasks within a pair of parentheses are dependent on each other: (a) $\lambda_c = 10^{-6}$ ($E(n_d) = 8.46e-5$ for the random sequencing), (b) $\lambda_c = 10^{-5}$ ($E(n_d) = 6.35e-4$ for the random sequencing).

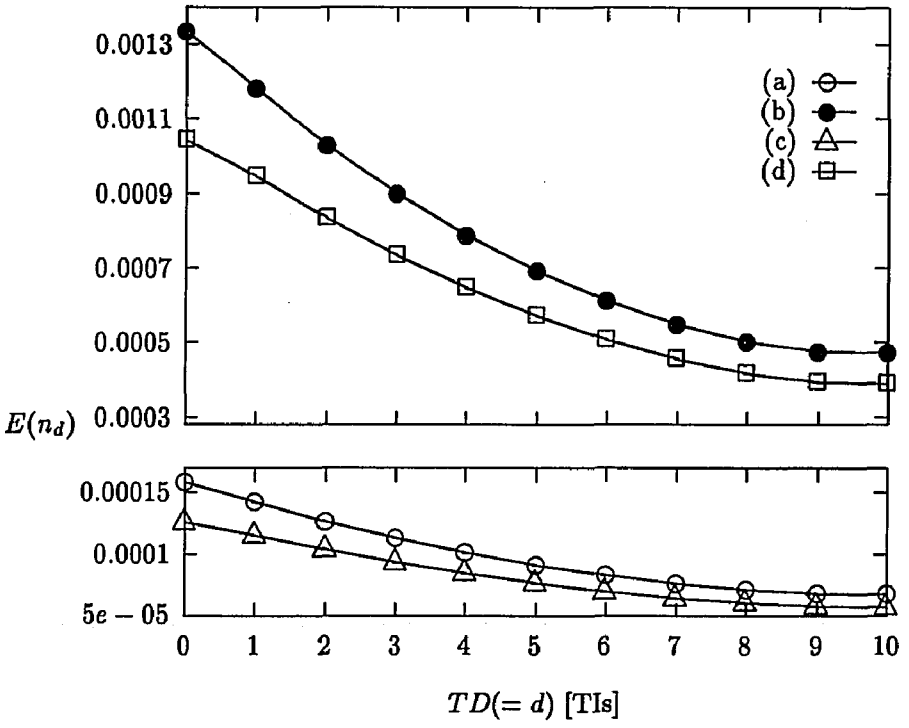



**Figure 8.8:** Mean number of TMR failures ($E(n_d)$) while varying task distance, $d$, under two different conditions of common-cause faults, and two sets of tasks (*different size model*): (a) $T_1 = \{1,2,4,3,2,4,1,1,3,2,4,3\}$ ($m = 12$ and $N = 30$) and $\lambda_c = 10^{-6}$ ($E(n_d) = 9.09e-5$ for the random sequencing), (b) task set $T_1$ and $\lambda_c = 10^{-5}$ ($E(n_d) = 7.12e-4$ for the random sequencing), (c) $T_2 = \{3,5,7,5,4,6\}$ ($m = 6$ and $N = 30$) and $\lambda_c = 10^{-6}$ ($E(n_d) = 7.65e-5$ for the random sequencing), and (d) task set $T_2$ and $\lambda_c = 10^{-5}$ ($E(n_d) = 5.83e-4$).

Fig. 8.7 deals with a task set, in which some subsets of tasks are dependent on others due to the precedence constraints:

$$\{(1,4,1),(3),(3,1),(4,2),(5),(3,5)\},$$

where tasks in a pair of parentheses are dependent on each other. The results are derived by defining a new task set $\{6,3,4,6,5,8\}$ as discussed in Section 8.3.3. In this case, the effect of using a better $d$ shows similar results to those in Figs. 8.6 and 8.8, i.e., the task sequencing with a better $d$ is more effective when common-cause faults are likely to occur.

## 8.5 Conclusion

We developed in this chapter a new method for sequencing task copies in a TMR system in order to alleviate the effects of common-cause/coincident faults, the main source of TMR failures. We first proposed a simple sequencing strategy in which three copies of each task are executed on the three modules of a TMR system with $TD = d$, and then derived the probability mass function of the number of TMR failures as a function of $d$.

For both the basic and general task models, we determined numerically the optimal TD by minimizing the mean number of TMR failures on the basis of the derived *pmf*'s. We also presented numerical examples of the mean number of TMR failures while varying $N$ and $d$, showing that the proposed sequencing strategy can significantly decrease the number of TMR failures under various conditions, especially when common-cause faults are likely to occur.

# CHAPTER 9

# CONCLUSIONS AND FUTURE DIRECTIONS

In this chapter, we summarize the contribution of this dissertation, and explore possible extensions of the work presented here.

## 9.1 Research Contributions

The timing information on the controlled process and the controller computer is a key to the design and evaluation of the controller computer in a hard real-time control system. We formally specified the needs of the controlled process understandable and meaningful to the the controller computer by using the CSD and also represented the recovery capability of the controller computer by evaluating the FTL.

In Chapter 2, we described the characteristics of real-time control systems where the controlled process and the controller computer are interacting synergically, and discussed the background of our work.

The main contribution of Chapter 3 is to provide a precise definition of the CSD and a method to derive it. This subject was not previously addressed in detail in spite of its importance for task assignment and scheduling, specification and evaluation of fault-tolerant controller computers.

In Chapter 4, we also investigated various features of fault-tolerance and several aspects of error- or failure-handling in fault-tolerant systems to evaluate the FTL, which encapsulates useful timing-information of the controller computer in the presence of faults. Especially, Chapter 5 concentrates on the reconfiguration process, a predominant contributor to the FTL.

Our attention to the design of a fault tolerant controller computer has been focused mainly on using the TMR structure, because it is the most popular and a practical fault-

tolerance scheme using the simplest form of spatial redundancy. Retry or re-execution on the same hardware without any reconfiguration can yield a successful recovery from TMR failures or masked errors caused by transient faults, and system reconfiguration is expensive both in time and hardware. Thus, a fault-tolerance strategy using time redundancy and the TMR structure proves to be cost-effective in designing a fault-tolerant controller computer by making a tradeoff between spatial and time redundancy.

Chapters 6 and 7 were based on the notion of applying time redundancy to a TMR system to develop design methods for "optimal" (in a sense of minimizing certain costs) fault-tolerant controller computers. In Chapter 6, we proposed an optimal instruction-retry policy minimizing the probability of dynamic failure based on the number of spares and the CSD derived in Chapter 3. Specifically, upon occurrence of both TMR failures and masked errors, either retry or reconfiguration whichever has a smaller $P_{dyn}$, is selected, and if retry is chosen, then the optimal retry period is derived to minimize $P_{dyn}$. In Chapter 7, we also used a simpler time redundancy for a TMR system, optimally choosing RSHW or RHWR based on the mean task-completion time.

In case of safety-critical missions under harsh environments where we cannot ignore the effects of near-coincident faults due to a common source such as an environmental disruption or malfunction of a shared component, the masking capability of a TMR system is no longer effective. In Chapter 8, we thus proposed a new method for sequencing task copies in a TMR system in order to alleviate the effects of common-cause/coincident faults, the main source of TMR failures in these harsh environments.

## 9.2 Future Directions

There are many remaining related research issues in designing and evaluating real-time fault-tolerant control systems. Discussed below are some of these issues.

### 9.2.1 Extension of Each Chapter

In Chapter 3, we considered only the failure to update the control input in case of nonlinear time-invariant control systems. It would be interesting to derive the deadlines of these systems in the presence of control input disturbances as was done for linear systems. We may extend linear control approaches after linearizing the nonlinear dynamic equation around several nominal states.

The derivation of control system deadlines for time-varying systems will also be a challenging task. This problem appears to be extremely difficult, since the elegant frequency-domain theory — that is applicable to the analysis of linear time-invariant system stability — is no longer applicable. We will begin with slowly time-varying linear systems, so that the frequency-domain theory can be applied over a certain time period. We must apply this theory many times, once per period, over the entire mission.

In Chapter 4, although we assumed the latencies of individual stages to be available by measuring or modeling relevant variables, it is in reality quite difficult to obtain their accurate values, some of which depend on one another and also on applications. We are currently investigating ways to evaluate or model the times taken by the individual stages (e.g., reconfiguration treated in Chapter 5) while considering the system architecture, task type, and the implementation of each stage.

Other related problems worth further investigation include:

- Derivation of a closed-form *pdf* expression for the time taken for each individual stage. In case an exact closed-form *pdf* is not obtainable, an approximate expression may be used to determine an appropriate fault- or error-handling policy.

- Dependence of FTL on fault coverage, which was assumed to be a constant determined by the diagnosis stage. However, fault coverage is in reality not simple to determine due mainly to the effects of many coupled testing and detecting methods. The task type and the failure occurrence rate also affect fault coverage. When periodic diagnoses are used, there is a tradeoff between accuracy (fault coverage) and time (frequency and diagnosis time). It is important to study all the factors determining fault coverage and analyze its effects on the FTL.

In Chapter 8, although we dealt only with TMR systems, we can extend our approach of this chapter to the problem of sequencing tasks in NMR systems. We can also include tight timing constraints for a certain subset of tasks and the effects of unreliable voting, which requires a more complicated sequencing strategy to minimize the mean number of NMR failures.

### 9.2.2 Reliability Model Based on CSD

It is necessary to develop a reliability model of a real-time control system with a fault-tolerant controller computer by incorporating the random CSD and the underlying fault-

tolerance strategy. System reliability is an essential attribute to any fault-tolerant system, for which its modeling is an important and popular tool. We can develop a reliability model by applying the general Markov-chain model in a systematic manner. For this purpose, a reliability model should incorporate all phenomena of general real-time control systems such as stochastic fault behaviors (transient as well as permanent faults), several possible fault-handling strategies of a fault-tolerant computer, and the characteristics of controlled process and the effects of tight timing conditions. A system crash can result from incorrect results of a controller computer as well as not meeting the stringent timing requirement, which depends on the nature of the controlled process. Specifically, all samples of a Markov-chain reliability model must be derived for all samples of control system deadlines by including additional states for each control system deadline. We then derive a desired reliability model from these samples. To reduce the necessary amount of computation, only important states and parameters will be analyzed.

### 9.2.3 Experimental Methodology for Measuring the Effects of EMI

We will outline an experimental methodology for measuring the effects of EMI on digital controller computers in flight-critical digital control systems. We view the main EMI effects as functional error modes of computing control inputs in the system/subsystem level and have already developed a theoretical basis for the assessment of EMI effects. We will measure the parameters using the testbed of NASA Airlab's HIRF Laboratory.

**Parameters to be Measured:** Executing the same tasks on multiple modules (hardware/software) like a TMR system is one of the most popular fault-tolerance methods. Since the EMI effects are likely to induce common-cause failures in these redundant modules, it is no longer effective to design a fault-tolerance strategy using massive spatial redundancy under the assumption of independent failure behaviors. In [29], we used a probability mass function *pmf* called the (compressed) *beta-binomial* distribution to model fault/failure occurrences caused by EMI. To validate such a model or develop another useful model to assess the EMI effects on the use of massive redundancy, we first propose an experiment measuring the number of faulty modules/computers in the presence of EMI.

There are two types of EMI effects on digital controller computers: (i) damage of components which should be replaced or repaired immediately and (ii) functional error modes,

called as *upset*,[1] without any component damage. In [5, 29], the main effects of EMI are functional error modes of the system/subsystem level that result in degraded system performance and/or reliability. Thus, the EMI effects are likely to be transient when the controller computer executes periodic tasks. Although programs or data remain contaminated after disappearance of EMI, a certain corrective action such as reset/reload software or internal recovery mechanisms may recover the controller computer. Secondly, we propose an experiment measuring the time taken from error occurrence to its recovery in the application context, which is equal to the FTL discussed in Chapter 4.

Finally, we propose an experiment measuring the system inertia in the presence of EMI, which characterizes the timing constraint of the controlled process. In Chapter 3, we developed a method to analytically derive the CSD.

[1] We use computer failures, errors, and upsets, in the same sense.

# APPENDIX A

## List of Symbols in Chapter 7

| | | |
|---|---|---|
| $X$ | : | Nominal task-execution time in the absence of failures, i.e., the amount of pure computation for a task measured in CPU cycles without including repetition of part of the task due to failures. |
| $X_i$ | : | Nominal execution time for the task between the $i$-th and $(i-1)$-th voting. |
| $W_n(X)$ | : | Expected execution time of a task whose nominal execution time is $X$. |
| $w_i$ | : | Actual execution time from the beginning of the task to the first completion of the $i$-th voting, where $W_i = E(w_i)$. |
| $V_i$ | : | Actual execution time during the interval $[X_{i-1}, X_i]$. |
| $p\ (q)$ | : | Probability of recovering a task with RSHW (RHWR), $p+q=1$. |
| $T_c$ | : | Resetting time in case of system reconfiguration. |
| $T_v$ | : | Time for voting on those variables changed during the previous voting interval. |
| $p_s^n(p_u^n)$ | : | Probability of the $n$-th RSHW being successful (unsuccessful). |
| $P$ | : | Probability of the first RSHW being successful. |
| $R$ | : | Ratio of the probability of success at the $(n+1)$-th RSHW to that at the $n$-th RSHW. |
| $k_m$ | : | Allowable maximum number of RSHWs. |
| $X_f$ | : | Time of detecting a TMR/voting failure. |
| $T_f^i$ | : | Time to a TMR system failure occurred first after starting the system in state $S_i$. |
| $F_{T_f^i}(X)$ | : | Probability of a TMR failure from $S_i$ during the execution time $X$ ($f_{T_f^i} \equiv pdf$ of $T_f^i$). |
| $t_i^j$ | : | Time of TMR failure occurrence via path $j$ from $S_i$ ($f_{t_i^j} \equiv pdf$ of $t_f^i$). |
| $S(x,y)$ | : | State with $x$ permanent faulty processor(s), $y$ non-permanent faulty processor(s), and $(3-x-y)$ nonfaulty processor(s) ($S_i = S(x,y)$ such that $i = 4x+y$). |
| $J_m^i$ | : | Set of all paths to a fault state $S_m$ from an initial state $S_i$ ($E^i = \bigcup^m J_m^i$). |
| $\pi_i(0)$ | : | Probability of a prior state before the first RSHW. |
| $\pi_m^i(T_f^i)$ | : | Probability of a fault state $S_m$ at time $T_f^i$ from an initial state $S_i$. |
| $P_{mn}(T)$ | : | Transition probability from $S_m$ to $S_n$ during $T$. |
| $C_1(k,X)$ | : | Expected cost of RSHW with a nominal task-execution time $X$ and MNRA $k$. |
| $C_1(X)(C_2(X))$ | : | Expected cost of RSHW (RHWR) for $X$. |
| $F_{j_k j_{(k+1)}}$ | : | Distribution of time to move to $S_{j_{(k+1)}}$ from $S_{j_k}$. |

| | | |
|---|---|---|
| $\{E_{j_k}\}$ | : | Set of all sub-paths emanating from $S_{j_k}$. |
| $\lambda_t(\lambda_p)$ | : | Occurrence rate of non-permanent (permanent) faults. |
| $\frac{1}{\mu}$ | : | Active duration of a non-permanent fault. |

# APPENDIX B

## List of Symbols in Chapter 8

| | | |
|---|---|---|
| $\Delta t$ | : | One task interval (TI), which is the execution time of a task. |
| $N$ | : | Number of tasks during a mission phase in the basic task model. |
| $N_d$ ($N_f$) | : | Number of tasks producing correct (incorrect) outputs in the absence (presence) of TMR failures. |
| $t_i$ | : | The $i$-th task ($1 \leq i \leq N$). |
| $P^c_{jk}$ ($P^i_{jk}$) | : | Probability that a common-cause (independent) failure occurs during the $j$-th TI and the durations of coincident faults inducing this failure are larger than $(k-1)\Delta t$ and smaller than $k\Delta t$ ($1 \leq k \leq N-j+1$). |
| $P(N_f = \ell)$ | : | Probability mass function (*pmf*) of $N_f$ ($0 \leq \ell \leq N$). |
| $P(k, \ell)$ | : | Probability that any pair of modules execute same $\ell$ tasks during $k$ consecutive faulty TIs, i.e., $\ell$ contaminated tasks. |
| $P(k, v, \ell)$ | : | Probability that any pair of modules execute same $\ell$ tasks during $k$ and $v$ faulty TIs of two modules, respectively. |
| $M_i$ | : | The $i$-th module ordered arbitrarily in a TMR system. |
| $d$ ($d_{opt}$) | : | Task Distance (TD) (the optimal TD minimizing $E(N_f)$). |
| $S_1$ | : | Subset of tasks which are not wrapped around. |
| $S_2$ ($S_3$) | : | Subset of tasks wrapped around only on $M_3$ (on both $M_2$ and $M_3$). |
| $N^i_f$ | : | Number of TMR failures due to common-cause faults in $S_i$. |
| $N^k_f(ij)$ | : | Number of TMR failures occurred in $S_k$ due to the independent faults in $M_i$ and $M_j$. |
| $m$ | : | Number of tasks during a mission in the general task model. |
| $n_d$ | : | Number of tasks producing incorrect results in the general task model. |
| $n_i$ | : | The $i$-th task in the general task model ($1 \leq i \leq m$). |
| $S^\ell_r$ | : | Set of all subsets of tasks that consist of $r$ tasks of sizes $n_{s1}, n_{s2} \cdots, n_{sr}$ and $\sum_{i=1}^{r} n_{si} \geq \ell$ from the entire set of $m$ tasks. |
| $n_0$ | : | Greatest Common Divisor ($GCD$) of $\{n_1, n_2, \cdots, n_m\}$. |
| $\lambda_c$ ($\lambda_i$) | : | Occurrence rate of common-cause (independent) failures. |
| $\frac{1}{\mu_c}$ ($\frac{1}{\mu_i}$) | : | Mean active duration of common-cause (independent) failures. |

# BIBLIOGRAPHY


[1] J. A. Abraham and D. P. Siewiorek, "An algorithm for the accurate reliability evaluation of triple modular redundancy networks," *IEEE Trans. on Computers*, vol. C-23, no. 7, pp. 682–692, July 1974.

[2] A. Avizienis and G. C. Gilley, "The STAR(self-testing and repairing) computer: An investigation of theory and practice of fault-tolerant computer design," *IEEE Trans. on Computers*, vol. C–20, no. 11, pp. 1312–1321, November 1971.

[3] P. Banerjee, "Strategies for reconfiguing hypercubes under faults," in *Proc. 2oth Annu. Int. Symp. on Fault–Tolerant Computing*, 1990.

[4] P. Barton, "Fault latency white paper," Technical report, Texas Instruments, Microelectronics Department, Plano, TX, January 1993.

[5] C. M. Belcastro, "Laboratory test methodology for evaluating the effects of electromagnetic disturbances on fault–tolerant control systems," *NASA TM-101665*, November 1989.

[6] A. P. Belleisle, "Stability of systems with nonlinear feedback through randomly time–varying delays," *IEEE Trans. on Automat. Contr.*, vol. AC–20, no. 1, pp. 67–75, February 1975.

[7] M. Berg and I. Koren, "On switching policies for modular redundancy fault-tolerant computing systems," *IEEE Trans. on Computers*, vol. C–36, no. 9, pp. 1052–1062, September 1987.

[8] R. W. Butler and A. L. White, "SURE reliability analysis," *NASA Technical Paper*, March 1990.

[9] P. K. Chande, A. K. Ramani, and P. C. Sharma, "Modular TMR multiprocessor system," *IEEE Trans. on Industrial Electronics*, vol. 36, no. 1, pp. 34–41, February 1989.

[10] C. Chen, A. Feng, T. Kikuno, and K. Tori, "Reconfiguration algorithm for fault-tolerant arrays with minimum number of dangerous processors," in *Proc. 21st Annu. Int. Symp. on Fault–Tolerant Computing*, 1991.

[11] B. Cuchi, "Reliability and analysis of hybrid redundancy," in *Digest of Papers, FTCS–5*, pp. 75–79, 1975.

[12] P. T. de Sousa and F. P. Mathur, "Shift–out modular redundancy," *IEEE Trans. on Computers*, vol. C–27, no. 7, pp. 624–627, July 1978.

[13] S. Dreyfus, "Variational problems with state variable inequality constraints," *RAND Coporation Paper*, vol. P-2605, pp. 72–85, 1962.

[14] J. Dugan, K. Trivedi, M. Smotherman, and R. Geist, "The hybrid automated reliability prediction," *AIAA Journal of Guidance, Control and Dynamics*, pp. 319–331, May 1986.

[15] J. B. Dugan and K. S. Trivedi, "Coverage modeling for dependibility analysis of fault-tolerant systems," *IEEE Trans. on Computers*, vol. 38, no. 6, pp. 775–787, June 1989.

[16] G. B. Finelli, "Characterization of fault recovery through fault injection on FTMP," *IEEE Trans. on Reliability*, vol. R-36, no. 2, pp. 164–170, June 1987.

[17] N. Gaitanis, "The design of tatally self–checking TMR fault–tolerant systems," *IEEE Trans. on Computer.*, vol. C-37, no. 11, pp. 1450–1454, November 1988.

[18] R. M. Geist, M. Smotherman, and R. Talley, "Modeling recovery time distributions in ultrareliable fault–tolerant systems," in *Digest of Papers, FTCS-20*, pp. 499–504, June 1990.

[19] R. M. Geist and K. S. Trivedi, "Ultrahigh reliability prediction for fault–tolerant computer systems," *IEEE Trans. on Computer.*, vol. C-32, no. 12, pp. 1118–1127, December 1983.

[20] A. Gosiewski and A. W. Olbrot, "The effect of feedback delays on the performance of multivariable linear control systems," *IEEE Trans. on Automat. Contr.*, vol. AC-25, no. 4, pp. 729–734, August 1980.

[21] P.-O. Gutman and M. Cwikel, "Admissible sets and feedback control for discrete–time linear dynamic systems with bounded controls and states," *IEEE Trans. on Automat. Contr.*, vol. AC-31, no. 4, pp. 373–376, April 1986.

[22] K. Hirai and Y. Satoh, "Stability of a system with variable time delay," *IEEE Trans. on Automat. Contr.*, vol. AC-25, no. 3, pp. 552–554, June 1980.

[23] A. L. Hopkins Jr., T. B. Smith III, and J. H. Lala, "FTMP–a highly reliable fault-tolerant multiprocessor for aircraft," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1221–1239, October 1978.

[24] G. Hostetter and J. S. Meditch, "Observing systems with unmeasurable inputs," *IEEE Trans. on Automat. Contr.*, vol. AC–18, pp. 306–307, June 1973.

[25] M. Kameyama and T. Higuchi, "Design of dependent–failure–tolerant microcomputer system using triple–modular redundancy," *IEEE Trans. on Computers*, vol. C-29, no. 2, pp. 202–205, February 1980.

[26] S. Kheradpir and J. S. Thorp, "Real-time control of robot manipulators in the presence of obstacles," *IEEE journal of Robotics and Automation*, vol. 4, no. 6, pp. 687–698, December 1988.

[27] H. Kim and K. G. Shin, "Design and analysis of an optimal instruction-retry policy for TMR controller computer," submitted for publication, 1993.

[28] H. Kim and K. G. Shin, "Evaluation of fault-tolerance latency from real-time application's perspectives," Technical Report CSE-TR-201-94, CSE Division, EECS Department, The University of Michigan, 1994.

[29] H. Kim and K. G. Shin, "Modeling externally-induced faults in controller computers," *Proc. 13rd IEEE/AIAA Digital Avionics Systems Conf.* (in press), 1994.

[30] H. Kim and K. G. Shin, "On the maximum feedback delay in a linear/nonlinear control system with input disturbances caused by controller-computer failures," *IEEE Trans. on Control Systems Technology*, vol. 2, no. 2, pp. 110–122, June 1994.

[31] H. Kim and K. G. Shin, "Reconfiguration latencies of dynamic redundancy techniques in fault-tolerant systems," in preparation, 1994.

[32] H. Kim and K. G. Shin, "Task sequencing to minimize the effects of near-coincident faults in TMR controller computers," submitted for publication, 1994.

[33] D. L. Kiskis and K. G. Shin, "Embedding triple-modular redundancy into a hypercube architecture," in *Proc. of 3rd Conf. on HCCA*, pp. 337–345, Los Angeles, January 1988.

[34] I. Koren and Z. Koren, "Analysis of a class of recovery procedures," *IEEE Trans. on Computers*, vol. C-35, no. 8, pp. 703–712, August 1986.

[35] C. M. Krishna, K. G. Shin, and R. W. Butler, "Synchronization and fault-masking in redundant real-time systems," in *Digest of Papers, FTCS-14*, pp. 152–157, June 1984.

[36] J. H. Lala, "Fault detection, isolation and configuration in FTMP: Methods and experimental results," in *Proc. 5th IEEE/AIAA Digital Avionics Systems Conf.*, pp. 21.3.1–21.3.9, 1983.

[37] Y. H. Lee and K. G. Shin, "Optimal reconfiguration strategy for a degradable multimodule computing system," *Journal of the ACM.*, vol. 34, pp. 326–348, April 1987.

[38] Y. H. Lee and K. G. Shin, "Optimal design and use of retry in fault-tolerant computing systems," *Journal of the ACM*, vol. 35, pp. 45–69, January 1988.

[39] G. Leitmann, *An Introduction to Optimal Control*, New York, NY: McGray-Hill, 1969.

[40] T.-H. Lin and K. G. Shin, "An optimal retry policy based on fault classification," *IEEE Trans. on Computers*, vol. C-43, no. 9, , September 1990.

[41] J.-C. Liu and K. G. Shin, "A RAM architecture for concurrent access and on–chip testing," *IEEE Trans. on Computers*, vol. C-40, no. 10, pp. 1153–1158, October 1991.

[42] J. Losq, "A highly efficient redundancy scheme: Self-purging redundancy," *IEEE Trans. on Computers*, vol. C-25, no. 6, pp. 569–578, June 1976.

[43] D. G. Luenberger, *Optimization by vector space methods*, New York, Wiley, 1969.

[44] R. E. Lyons and W. Vanderkulk, "The use of triple–modular redundancy to improve computer reliability," *IBM J. Res. Develop.*, vol. 6, pp. 200–209, April 1962.

[45] M. Mariton, "Detection delays, false alarm rates and the reconfiguration of control systems," *Int. J. Control*, vol. 49, no. 3, pp. 981–992, 1989.

[46] S. R. McConnel, D. P. Siewiorek, and M. M. Tsao, "The measurement and analysis of transient errors in digital computer systems," in *Digest of Papers, FTCS-9*, pp. 67–70, June 1979.

[47] J. McGough, M. Smotherman, and K. S. Trivedi, "The conservativeness of reliability estimates based on instantaneous coverage," *IEEE Trans. on Computers*, vol. C–34, no. 7, pp. 602–608, July 1985.

[48] D. Paul, C. Roark, and D. Struble, "Technical report on phase one of the dynamic reconfiguration demonstration system program," Technical report, Texas Instruments, Inc. NAWC-DRDS-P1-TR-0003, April 1992.

[49] V. B. Pradsad, "Fault-toterant digital systems," *IEEE*, pp. 17–21, February 1989.

[50] C. V. Ramamoorthy and Y. W. Eva Ma, "Optimal reconfiguration strategies for reconfigurable systems with no repair," *IEEE Trans. on Computers*, vol. C-35, no. 3, pp. 278–280, March 1986.

[51] C. V. Ramamoorthy and Y.-W. Han, "Reliability analysis of systems with concurrent error detection," *IEEE Trans. on Computers*, vol. C-24, no. 9, pp. 868–878, September 1975.

[52] Z. V. Rekasius, "Stability of digital control with computer interruption," *IEEE Trans. on Automat. Contr.*, vol. AC–31, no. 4, pp. 356–359, April 1986.

[53] C. Roark, D. Paul, D. Struble, D. Kohalmi, and J. Newport, "Pooled spares and dynamic reconfiguration," in *Proceedings of NAECON'93*, pp. 173–179, May 1993.

[54] A. P. Sage and I. C. C. White, *Optimum systems control*, Prentice–Hall Inc., Englewood Cliffs, New Jersey, 1977.

[55] K. G. Shin and X. Cui, "Effects of computing time delay on real–time control systems," in *Proc. of 1988 American Control Conf.*, pp. 1071–1076, 1988.

[56] K. G. Shin and H. Kim, "Derivation and application of hard deadlines for real–time control systems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1403–1413, Nov./Dec. 1992.

[57] K. G. Shin and H. Kim, "Hard deadlines in real-time control systems," *Control Eng. Practice*, vol. 1, no. 4, pp. 623–628, 1993.

[58] K. G. Shin and H. Kim, "A time redundancy approach to TMR failures using fault-state likelihoods," *IEEE Trans. on Computers*, vol. C-43, no. 9, , October 1994.

[59] K. G. Shin, C. M. Krishna, and Y.-H. Lee, "A unified method for evaliating real–time computer controller and its application," *IEEE Trans. on Automat. Contr.*, vol. AC–30, no. 4, pp. 357–366, April 1985.

[60] K. G. Shin and Y.-H. Lee, "Error detection process — model, design, and its impact on computer performance," *IEEE Trans. on Computers*, vol. C–33, no. 6, pp. 529–539, June 1984.

[61] K. G. Shin and Y.-H. Lee, "Measurement and application of fault latency," *IEEE Trans. on Computers*, vol. C–35, no. 4, pp. 370–375, April 1986.

[62] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal checkpointing of real-time tasks," *IEEE Trans. on Computers*, vol. C–36, no. 11, pp. 1328–1341, November 1987.

[63] K. G. Shin and J.-C. Liu., "Study on fault-tolerant processor for advanced launch system," *NASA Contractor Report*, June 1990.

[64] D. P. Siewiorek, V. Kini, and H. Mashburn, "A case study of C.mmp, Cm*, and C.vmp: Part I – experiences with fault tolerance in multiprocessor systems," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1178–1199, October 1978.

[65] D. P. Siewiorek and E. J. McCluskey, "Switch complexity in systems with hybrid redundancy," *IEEE Trans. on Computers*, vol. C–22, no. 3, pp. 276–283, March 1973.

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

[67] D. D. Siljak, "Reliable control using multiple control systems," *Int. J. Control*, vol. 31, no. 2, pp. 303–329, 1980.

[68] M. Smotherman, R. M. Geist, and K. S. Trivedi, "Provably conservative applications to complex reliability models," *IEEE Trans. on Computers*, vol. C–35, no. 4, pp. 333–338, April 1986.

[69] A. Tantawi and M. Ruschitzka, "Performance analysis of checkpinting strategies," *ACM Trans. Computer Systems*, vol. 2, pp. 123–1441, 1984.

[70] J. S. Upadhyaya and K. K. Saluja, "A watchdog processor based general rollback technique with multiple retries," *IEEE Trans. Software Eng.*, vol. SE-12, no. 1, pp. 87–95, January 1986.

[71] M. Uyar and A. Reeves, "Dynamic fault reconfiguration in a mesh-connected MIMD environment," *IEEE Trans. on Computers*, vol. 37, no. 10, pp. 1191–1205, October 1988.

[72] J. F. Wakerly, "Transient failures in triple modular redundancy systems with sequential modules," *IEEE Trans. on Computers*, vol. 63, no. 5, pp. 570–573, May 1975.

[73] J. F. Wakerly, "Microcomputer reliability improvement using triple-modular redundancy," *IEEE Trans. on Computers*, vol. 64, no. 6, pp. 889–895, June 1976.

[74] X.-Y.Zhuo and S.-L. Li, "A new design method of voter in fault-tolerant redundancy multiple-module multi-microcomputer system," in *Digest of Papers, FTCS-13*, pp. 472–475, June 1983.

[75] K. Zahr and C. Slivinsky, "Delay in multivariable computer controlled linear systems," *IEEE Trans. on Automat. Contr.*, vol. AC–19, no. 8, pp. 442–443, August 1974.