IAF0530/IAF9530

Dependability and fault tolerance

Lecture 6 and 7
Redundancy (Hardware and software)

Gert Jervan
gert.jervan@ati.ttu.ee

Lecture Outline

- ✓ **Introduction**
- ✓ **Hardware Redundancy**
- ✓ **Software Redundancy**
- ✓ **Information Redundancy**
- ✓ **Time Redundancy**

Some materials from:
Kewal Saluja
Hongyu Sun
Zaipeng Xie
Meng-Lai Yin
Rajesh Gupta
Elena Dubrova

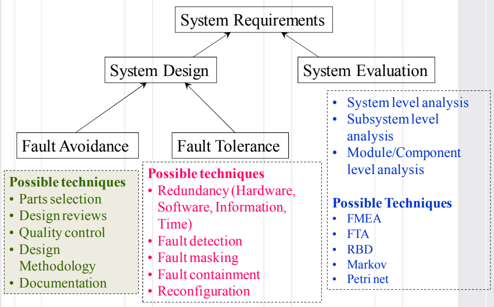
 2

Fault Tolerance

- A fault-tolerant system is one that can continue to correctly perform its specified tasks in the presence of hardware failures and/or software errors.
- Fault tolerance is the attribute that enables a system to achieve fault-tolerant operation.
- Fault tolerance is not a new field:
  - 1949, the EDVAC computer duplicated the ALU and compare the results
  - 1955, the UNIVAC computer incorporated parity check for data transfers
  - 1952, John von Neumann, lectures on the use of replicated logic modules to improve system reliability,
  - etc.

 3

System Design & Evaluation Top-Level View



Hardware Redundancy

Hardware Redundancy

- 3 basic forms: passive, active, and hybrid
  - Passive: Mask faults rather than detect faults without requiring any system or operator action
  - Active: Fault has to be detected before it can be tolerated. Actions: location, containment, recovery (for component removal)

 6

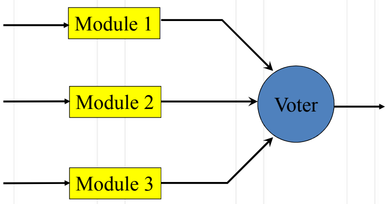
Passive Hardware Redundancy

- Use fault masking to hide the occurrence of faults and prevent the faults from resulting in errors
- Mask faults rather than detect faults
- Achieve fault tolerance without requiring any system or operator action
- Voting mechanisms, majority voting
- Do not need fault detection or reconfiguration
- Many drawbacks

Passive Hardware Redundancy

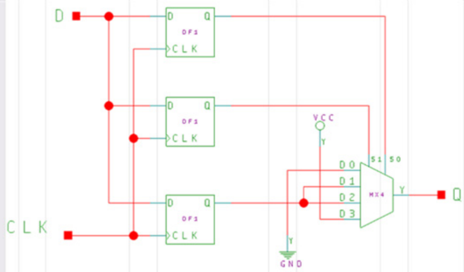
- N-Modular Redundancy (generalization of TMR or Triple Modular Redundancy)
- TMR: Triplicate the hardware and perform a majority vote to determine the output of the system
  - If one of the modules becomes faulty, the 2 remaining fault-free modules mask the results of the faulty module when the majority vote is performed

TMR Technique



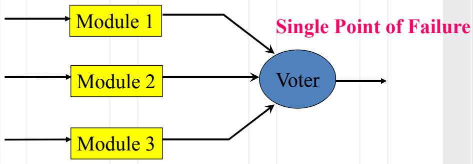
Tolerates N/2 faults

TMR/Voter Structures



Fault-Tolerance Capability

- Assuming perfect voter, how many module faults can the TMR technique tolerate?
- What if 2 modules fail the same way?
- Does TMR technique provide fault detection capability?
- How about imperfect voter?
- Performance impacts from the voter in the TMR technique

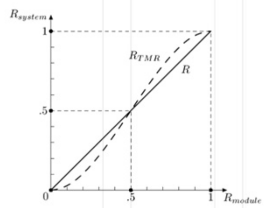


Reliability of a TMR System

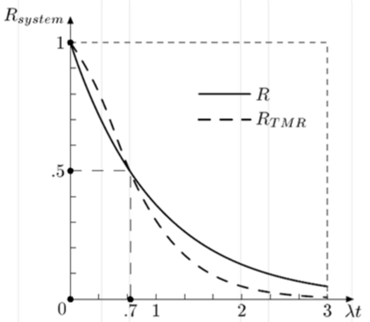
$$R_{TMR} = R_1R_2R_3 + (1-R_1)R_2R_3 + R_1(1-R_2)R_3 + R_1R_2(1-R_3)$$

$$R_1 = R_2 = R_3 = R$$

$$R_{TMR} = 3R^2 - 2R^3$$

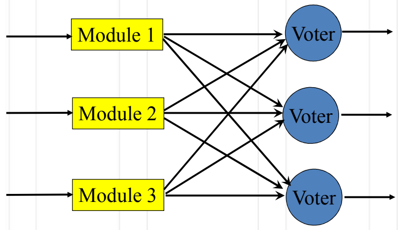


Reliability of a TMR System

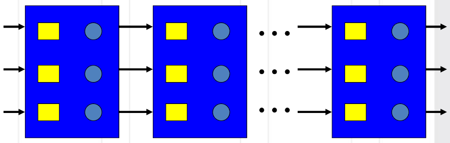


MTTF=1/λ

TMR with Triplicated Voters



Cascading TMR modules



Passive hardware redundancy

- Types of voting
  - Majority
    - in many practical situations it is meaningless
  - Average
    - can have poor performance if a sensor always provide very low value
  - Mid value
    - a good choice - can be very costly to implement in HW

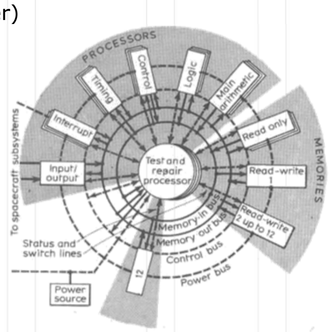
Passive Hardware Redundancy

- Comparison between hw and sw voter schemes

	HW	**SW**
cost	high	low
flexibilty	inflex	flex
synch.	tightly	loosely
perform.	high (fast)	low (slow)
types of voting	majority (others costly)	diff (no extra cost)

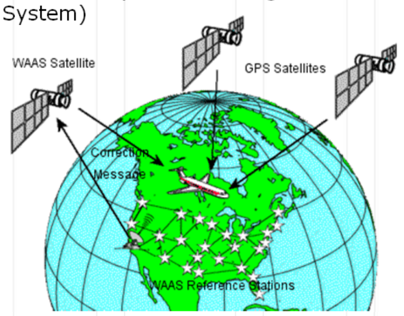
Example Systems Using TMR Technique

- JPL STAR (Self-Testing And Repairing computer)

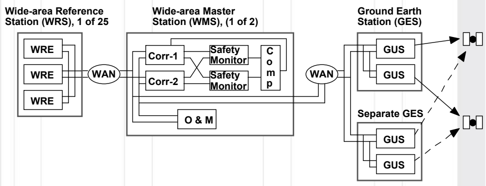


Example Systems Using TMR Technique

- FAA WAAS (Wide Area Augmentation System)



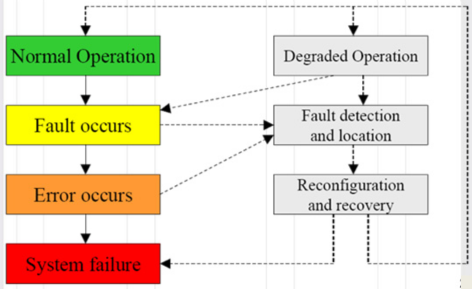
WAAS Block Diagram



Active Hardware Redundancy

- Achieve fault tolerance by detecting the existence of faults and performing some action to remove the faulty parts
- Require the system be reconfigured to tolerate faults
- 3 steps: fault detection, fault location, and fault recovery

Active Hardware Redundancy

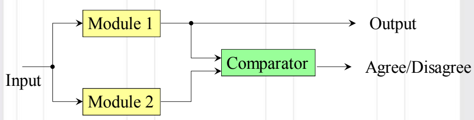


Dynamic Redundancy

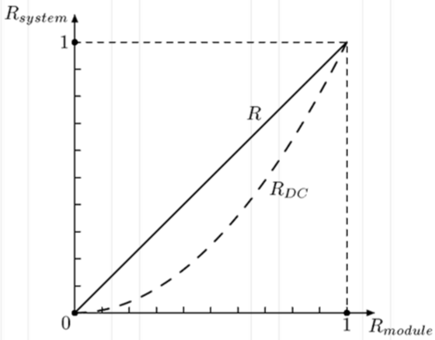
- Uses Extra Components
- Only 1 Copy Operates At A Times
  - Fault Detection
  - Fault Recovery
- Spares Are On "Standby"
  - Hot Spares
  - Cold Spares

Duplication with Comparison

- Both modules perform the same computations in parallel and compare the results
- An error message is generated if the two results disagree
- Only fault detection, no fault tolerance
- Can be used as a fundamental fault detection technique in active redundancy approach, for example, the pair-and-a-spare technique



Reliability of duplication with comparison



 25

Duplication with Comparison

- Problems:
  - if there is a fault on input line, both modules will receive the same erroneous signal and produce the erroneous result
  - comparator may not be able to perform an exact comparison
    - synchronisation
    - no exact matching
  - comparator is a single point of failure

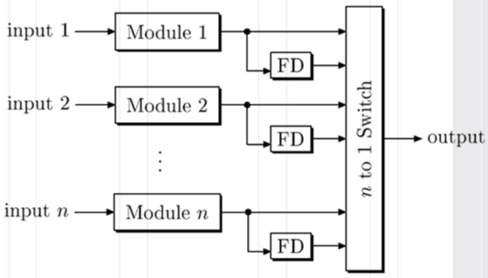
 26

Implementation of comparator

- In hardware, a bit-by-bit comparison can be done using two-input exclusive-or gates
- In software, a comparison can be implemented with a COMPARE instruction
  - commonly found in instruction sets of almost all microprocessors

 27

Standby Sparing



 28

Spares

- Hot spares
  - all modules are powered up
  - spares can be switched into use immediately after the primary module becomes failed
- Cold spares
  - the primary modules are powered up
  - the spares are powered down, which are powered up and switched into use when the primary modules fail
- Warm spares

 29

Standby Sparing (standby replacement)

- Active hardware redundancy
- One module is operational and one or more modules serve as standbys (or spares)
- Various fault detection or error detection schemes are used to determine whether a module has become faulty
- Fault location is used to determine exactly which module, if any, is faulty.

 30

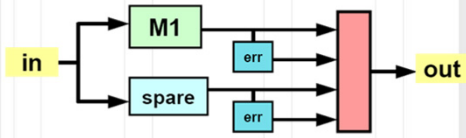
Standby Sparing (standby replacement)

- If a fault is detected and located, then the faulty module is removed from operation and replaced with a spare
- The reconfiguration can be viewed as a switch.
- Can bring a system back to full operation after the occurrence of a fault.
- Require momentary disruption in performance when reconfiguration is performed.

31

Hot Standby Sparing

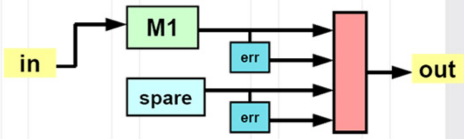
- In hot standby sparing spares operate in synchrony with on-line module and are prepared to take over any time



32

Cold Standby Sparing

- In cold standby sparing spares are unpowered until needed to replace a faulty module



33

Hot & Cold Standby Sparing

- Hot standby sparing can minimize the performance disruption. The spares operate in synchrony with the on line modules and are prepared to take over at any time.
- In cold standby sparing, the spares are unpowered until needed to replace a faulty module. Hence extra time is required to bring the module back to operation. The advantage is that spares do not consume power until needed. Satellite application is a good example for cold standby sparing.

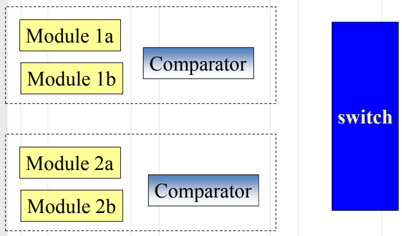
34

Pair-and-a-spare Technique

- Combine the features in standby sparing and duplication with comparison
- 2 modules are operated in parallel at all times and their results are compared to provide the error protection capability
- The error signal from the comparison is used to initiate the reconfiguration process (switch) that removes faulty modules and replaces them with spares

35

Pair-and-a-spare scheme



http://www.stratus.com/

36

Example Systems

- Apollo telescope mount pointing computer
- Saturn 5 LVDC memory section
- Compaq Himalaya architecture

 37

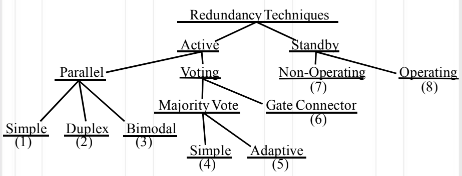
Types of Redundancy

NASA Office of Logic Design - klabs.org

- Classified on how the redundant elements are introduced into the circuit
- Choice of redundancy type is application specific
- Active or Static Redundancy
  - External components are not required to perform the function of detection, decision and switching when an element or path in the structure fails.
- Standby or Dynamic Redundancy
  - External elements are required to detect, make a decision and switch to another element or path as a replacement for a failed element or path.

 38

Redundancy Techniques



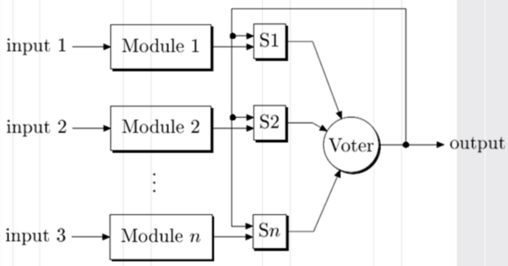
 39

Hybrid Hardware Redundancy

- Hybrid:
  - combine the attractive features of both the passive and active approaches
    - fault masking
    - fault detection
    - fault location
    - recovery

 40

Self-Purging Redundancy



Can mask n-2 module faults

 41

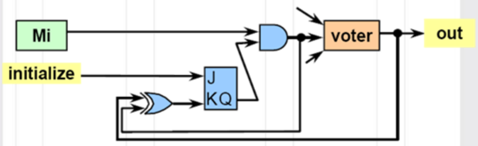
Self Purging Redundancy

- Initially start with NMR
- Purge one unit at at time till arrive at TMR
  - can tolerate more faults initially compared to NMR with spare
  - cost of the switch - higher?

 42

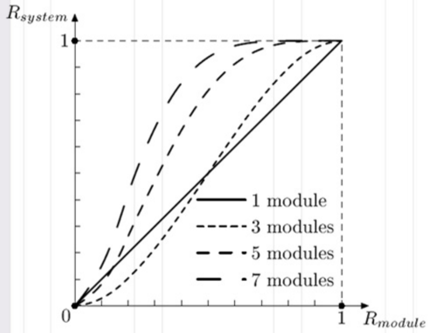
Basic Structure of a Switch

- If output of a module disagrees with the output of the system, its contribution to the voter is forced to be 0 (threshold voter)



 43

Reliability of Self-Purging System



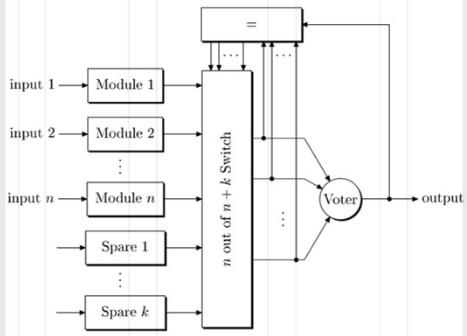
 44

N-Modular Redundancy with Spares

- Most hybrid redundancy are based on the concept of N-modular redundancy (NMR) with spares
- The idea is to provide N modules arranged in a voting configuration
- Spares are provided to replace failed modules
- The advantage of NMR with spares is that a voting configuration can be restored after a fault has occurred

 45

N-Modular Redundancy with Spares



 46

NMR with Spares

- System remains in the basic NMR configuration until the disagreement vector determines a fault
- The output of the voter is compare to the individual outputs of the modules
- Module which disagrees is labeled as faulty and removed from the NMR core
- Spare is switched to replace it

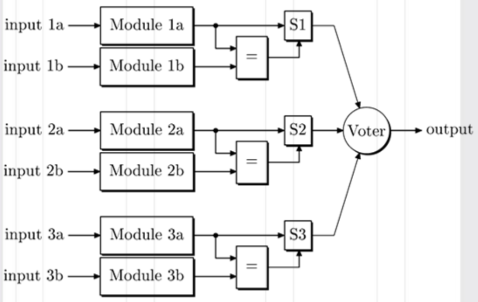
 47

NMR with Spares

- The reliability is maintained as long as the pool of spares is not exhausted
- 3-modular redundancy with 1 spare can tolerate 2 faults
- To do it in a passive approach, we would need to have 5 modules

 48

Triplex-duplex Redundancy



49

Triplex-duplex Redundancy

- TMR allows faults to be masked
  - performance without interruption
- Duplication with comparison allows faults to be detected and faulty module removed form voting
  - removal of faulty module allows to tolerate future faults
- Two module faults can be tolerated

50

Software Fault Tolerance

Introduction

- Less understood and less mature than in hardware
- Software does not degrade over time
- Design faults
- Environment

52

Introduction

- Many current techniques for software fault tolerance attempt to leverage the experience of hardware redundancy schemes
  - software N-version programming closely resembles hardware N-modular redundancy
  - recovery blocks use the concept of retrying the same operation in expectation that the problem is resolved after the second try.

53

Problems

- Traditional hardware fault tolerance techniques were developed to fight
  - permanent components faults primarily
  - transient faults caused by environmental factors secondarily.
- They do not offer sufficient protection against design and specification faults, which are dominant in software.

54

Concepts for Traditional SFT

- Software design and implementation errors cannot be detected by simple replication of identical software units, assuming the same inputs are provided to each copy.
- Some form of diversity must accompany the redundancy
  - Software redundancy → Design diversity
  - Information or data redundancy → Data diversity
  - Temporal redundancy → Temporal diversity
  - Environment diversity
  - Hardware redundancy

Single- and multi-version

- Software fault-tolerance techniques can be divided into two groups:
  - single-version
  - multi-version
- Single version techniques aim to improve fault tolerant capabilities of a single software module
  - fault detection, containment and recovery mechanisms
- Multi-version techniques employ redundant software modules, developed following design diversity rules

Redundancy Allocation

- A number of possibilities have to be examined:
  - at which level the redundancy need to be provided
- Redundancy can be applied to a procedure, or to a process, or to the whole software system
  - which modules are to be made redundant
- Usually, the components which have high probability of faults are chosen to be made redundant.
- The increase in complexity caused by redundancy can be quite severe and may diminish the dependability improvement

Single-Version (Dynamic) Techniques

- Dynamic redundancy kicks in only when an error is detected.
- Four phases
  - 1. Error detection:
    fault tolerance techniques effective only when an error is detected
  - 2. Damage assessment and containment:
    to what extent the "damage" has spread because of the delay between a fault and its manifestation/detection?
  - 3. Error recovery:
    techniques to reach from a corrupted to a safe state
  - 4. Fault treatment and continued service:
    error correction.

1 - Error Detection

- The goal is to determine that a fault has occurred within a system.
- Various types of acceptance tests are used to detect faults
  - the result of a program is subjected to a test
  - if the result passes the test, the program continues its execution
  - a failed test indicates a fault

Acceptance Test

- Acceptance test is most effective if it can be calculated in a simple way and if it is based on criteria that can be derived independently of the program application.
- The existing techniques include
  - timing checks
  - coding checks
  - reversal checks
  - reasonableness checks
  - structural checks
  - replication checks
  - dynamic reasonableness checks

Timing Checks

- Timing checks are applicable to system whose specification include timing constrains
- Based on these constrains, checks are developed to indicate a deviation from the required behavior.
  - Watchdog timer is an example of a timing check
  - Watchdog timers are used to monitor the performance of a system and detect lost or locked out modules.

Coding Checks

- Coding checks are applicable to system whose data can be encoded using information redundancy techniques
- Usually used in cases when the information is merely transported from one module to another without changing it content.
  - Arithmetic codes can be used to detect errors in arithmetic operations

Reversal Checks

- In some system, it is possible to reverse the output values and to compute the corresponding input values.
- A reversal checks compares the actual inputs of the system with the computed ones.
  - a disagreement indicates a fault.

Reasonableness Checks

- Reasonableness checks use semantic properties of data to detect fault.
  - a range of data can be examined for overflow or underflow to indicate a deviation from system's requirements
- Maximum withdrawal sum in bank's teller machine
- Address generated by a computer should lie inside the range of available memory

Structural Checks

- Structural checks are based on known properties of data structures
  - a number or elements in a list can be counted, or links and pointer can be verified
- Structural checks can be made more efficient by adding redundant data to a data structure,
  - attaching counts on the number of items in a list, or adding extra pointers

2 - Damage Assessment & Containment

- Necessary due to the delay between fault and error
- Goal of containment is to minimize damage caused by a faulty component
  - "firewalling"
- Assessment closely related to containment techniques used
- Techniques for fault containment:
  - modularization
  - partitioning
  - system closure
  - atomic actions

Modularization

- Software system is divided into modules with few or no common dependencies between them
- Modularization attempts to prevent the propagation of faults
  - by limiting the amount of communication between modules to carefully monitored messages
  - by eliminating shared resources

Partitioning

- Modular hierarchy of a software architecture is partitioned in horizontal or vertical dimensions
- Horizontal partitioning separates the major software functions into independent branches
  - The execution of the functions and the communication between them is done using control modules
- Vertical partitioning distributes the control and processing function in a top-down hierarchy.
  - High-level modules normally focus on control functions, while low-level modules perform processing

System Closure

- System closure technique is based on a principle that no action is permissible unless explicitly authorized
- In an environment with many restrictions and strict control all the interactions between the elements of the system are visible
  - prison
- It is easier to locate and disable any fault.

Atomic Action

- An atomic action among a group of components in an activity in which the components interact exclusively with each other.
  - no interaction with the rest of the system
- Two possible outcomes of an atomic action:
  - it terminates normally
  - it is aborted upon a fault detection
- Fault containment area is defined and fault recovery is limited to atomic action components

3 Fault Recovery

- Once a fault is detected and contained, a system attempts to recover from the faulty state and regain operational status
  - If fault detection and containment mechanisms are implemented properly, the effects of the faults are contained within a particular set of modules at the moment of fault detection.
- The knowledge of fault containment region is essential for the design of effective fault recovery mechanism

Exception Handling

- Exception handling is the interruption of normal operation to handle abnormal responses
- Possible events triggering the exceptions:
  - Interface exceptions
    - signaled by a module when it detects an invalid service request
  - Local exceptions
    - signaled by a module when its fault detection mechanism detects a fault
  - Failure exceptions
    - signaled by a module when it has detected that its fault recovery mechanism is enable to recover successfully

Recovery

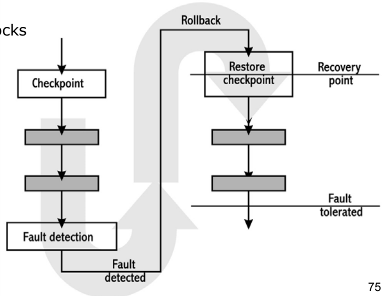
- Forward or Backward
- Forward: continues from an erroneous state by making selective corrections to the system state
  - includes making safe the controlled environment which may be hazardous or damaged because of failure
  - system specific and depends upon accurate predictions
  - e.g., redundant pointers in data structures, self-correcting codes

Recovery

- Backward: relies on restoring the system to a previous safe state and executing an alternative section of the program
  - safe functionality but different algorithm
  - the point to which a process is restored is called a recovery point and the act of establishing it is called checkpointing.
  - BER can be used to recover from unanticipated faults including design errors.
  - State restoration is not always possible in (real-time) embedded systems.

Backward Recovery

- ✓ Attempts to return the system to a correct or error-free state.
- ✓ For transient faults
- ✓ Example: recovery blocks (RcB)



Static Checkpoints

- A static checkpoint takes a single snapshot of the system state at the beginning of the program execution and stores it in the memory.
  - If a fault is detected, the system returns to this state and starts the execution from the beginning.
  - Fault detection checks are placed at the output of the module

Dynamic Checkpoints

- Dynamic checkpoints are created dynamically at various points during the execution
  - If a fault is detected, the system returns to the last checkpoint and continues the execution.
  - Fault detection checks need to be embedded in the code and executed before the checkpoints are created

Static vs. Dynamic

- In static approach, the expected time to complete the execution grows exponentially with the execution requirements.
  - static checkpointing is effective only if the processing requirement is relatively small.
- In dynamic approach, it is possible to achieve linear increase in execution time as the processing requirements grow

Strategies for dynamic checkpointing

- Equidistant
  - places checkpoints at deterministic fixed time intervals
  - the time between checkpoints is chosen depending on the expected fault rate
- Modular
  - places checkpoints at the end of the sub-modules in a module, after the fault detection checks for the submodule are completed
  - the execution time depends on the distribution of the sub-modules and expected fault rate
- Random

Advantages

- Conceptually simple
- Independent of the damage caused by a fault
- Applicable to unanticipated faults
- General enough to be used at multiple levels in a system

Problems

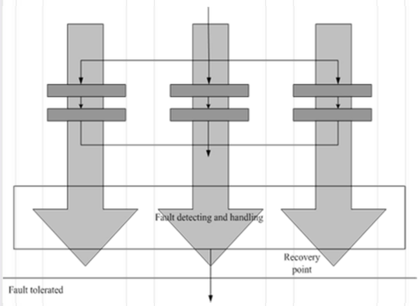
- Non-recoverable actions exist in some systems
  - these actions cannot be compensated by simply reloading the state and restarting the system
    - firing a missile
    - soldering a pair of wires
- The recovery from such actions can be done
  - by compensating for their consequences
    - undoing a solder
  - by delaying their output until after additional confirmation checks are completed
    - do a friend-or-foe confirmation before firing

Forward Recovery

- Attempts to find a new state from which the system can continue operation.
- Utilize error compensation based on redundancy to select or derive the correct answer or an acceptable answer.
- Example: N-version programming (NVP), N-copy programming (NCP), and the distributed recovery block (DRB)

Forward Recovery

- Efficient for predictable errors



4 - Fault Treatment and Continued Service

- Even with recovery, the error may recure. Need to eradicate the fault from the system
- Automatic treatment of faults is very application specific
- Make some assumptions. For instance:
  - all faults are transient
- Fault treatment in two stages
  - Fault location
  - System repair
- Fault location
  - use error detection techniques to trace a fault to a component (hardware or software)
  - System repair
    - sometimes it has to be done while the system is in operation.

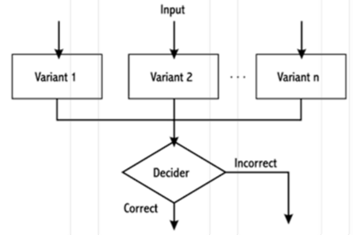
Multi-Version Techniques

- Multi-version techniques use two or more versions the same software module, which satisfy design diversity requirements.
  - different teams, different coding languages or different algorithms can be used to maximize the probability that all the versions do not have common faults

© Gert Jervan 85

Design Diversity

- Higher cost



© Gert Jervan 86

SFT Techniques Using Design Diversity

Techniques	Abbr.	Error Processing
Recovery Blocks	RcB	Error detection by AT and backward recovery
N-Version Programming	NVP	Vote
N Self-Checking Programming	NSCP	Error detection by AT and forward recovery

AT – Acceptance Test

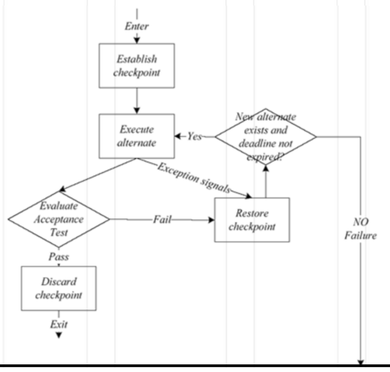
© Gert Jervan 87

Recovery Blocks

- Combines checkpoint and restart approach with standby sparing redundancy scheme
- n different implementations of the same program
  - Only one of the versions is active
  - If an error if detected by the acceptance test, a retry signal is sent to the switch
  - The system in rolled back to the state stored in the checkpoint memory and the execution is switched to another module

© Gert Jervan 88

Recovery Blocks



© Gert Jervan 89

Recovery Blocks

Method	Recovery block
Error Processing Technique	Error detection by AT and backward recovery
Criteria of Accepting Result	Absolute, with respect to specification
Execution Scheme	Sequential
Consistency of Input Data	Implicit, from backward recovery principle

© Gert Jervan 90

Recovery Blocks

- A language level support for backward error recovery
  - blocks in the normal programming language sense, but
  - at the entrance to the block is an automatic recovery point and
  - at the exit an acceptance test to test that the system is an an acceptable state
  - if the acceptance test fails, the program is restored to the recovery point at the beginning of the block and an alternative module is executed
  - repeat this process with alternative modules
  - if all fail, recovery must take place at a higher level
- In terms of four phases of software fault tolerance
  - Error detection <-> acceptance test
  - Damage assessment <-> not needed due to BER
  - Fault treatment <-> stand-by spare code

© Gert Jervan 91

Recovery Blocks

- Similarly to cold and hot standby sparing, different version can be executed either serially, or concurrently
  - Serial execution may require the use of checkpoints to reload the state before the next version is executed
  - The cost in time of trying multiple versions serially may be too expensive, especially for a real-time system.
  - A concurrent system requires n redundant hardware modules, a communications network to connect them and the use of input and state consistency algorithms.

© Gert Jervan 92

Syntax of Recovery Blocks

```
ensure <acceptance test>
by
    <primary module>
else by
    <alternative module>
else by
    <alternative module>
    ...
else by
    <alternative module>
else error
```

- Recovery blocks can be nested
- If all alternatives in a nested recovery block fail the acceptance test, the outer level recovery point will be restored
  - (and an alternative module to that block will be executed).

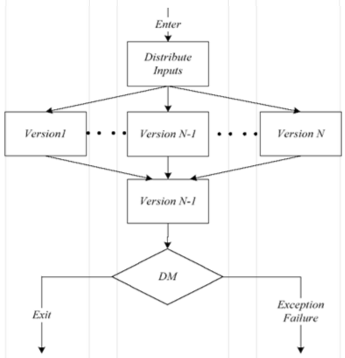
© Gert Jervan 93

N-Version Programming

- Resembles N-modular hardware redundancy
- N different software implementations of a module are executed concurrently.
- The selection algorithm (voter) decides which of the answers is correct
  - a voter is application independent
  - this is an advantage over recovery block fault detection mechanism, requiring application dependent acceptance tests

© Gert Jervan 94

NVP



© Gert Jervan 95

N-version Programming

Method	N-version programming
Error Processing Technique	Vote
Criteria of Accepting Result	Relative, on variant results
Execution Scheme	Parallel
Consistency of Input Data	Explicit by dedicated mechanisms

© Gert Jervan 96

N-Version Programming

- Consists of independent generation of N (>2) functionally equivalent programs from same initial specifications
  - Design Diversity, Different Programming Language, Methods..
- Programs execute concurrently, results are arrived at by consensus (majority voting).
- Questions
  - How are results compared? How is voting conducted?
- NVP depends upon
  - good initial specification, independence of effort, abundance of effort.
- NVP can be taken further
  - compiling, processing, ...

© Gert Jervan

97

NVP

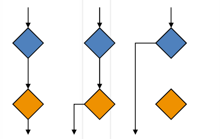
- Controlled by a driver process
  - invokes each of the versions
  - waiting for the versions to complete
  - comparing and acting on the results
- Problem: assumes programs run to completion!
  - So the versions must actually interact (with the driver program)
    - Comparison Points: points in the versions when programs must communicate their votes to the driver process
    - Defines granularity of the fault tolerance
  - How the versions communicate and synchronize depend upon the programming language used, its model of concurrency

© Gert Jervan

98

Vote Comparison in NVP

- Efficiency of vote comparison is critical
- Complicated by comparison procedure
  - Not all results are single numeric values
  - The "consistent comparison problem"
    - When using "thresholds" for comparison the errors can stack up, resulting different execution paths in *all* versions.



Two sequential thresholding lead to different execution paths in all three versions.

The problem will reappear even when using inexact comparison (just have to be near a threshold value).

And what happens when there are multiple solutions?

© Gert Jervan

99

NVP versus RB

- NVP is static where as RB is dynamic redundancy
- Both have design overheads
  - alternative algorithms
  - NVP requires a driver
  - RB requires an acceptance test
- Runtime overheads
  - NV requires more resources
  - RB requires establishing recovery points
- Both susceptible to errors in requirements
- Error detection
  - vote comparison (NVP) versus acceptance test (RB)
- Atomicity requirement
  - NV vote before it outputs to the environment, RB must output only following the passing of the acceptance test.

© Gert Jervan

100

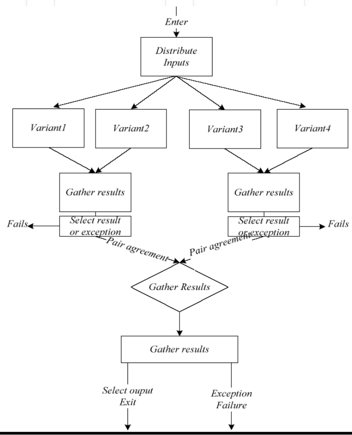
N Self-Checking Programming

- N self-checking programming combines recovery block concept with N version programming
- The checking is performed either by using acceptance tests, or by using comparison.
- Examples of applications of N self-checking programming:
  - Lucent ESS-5 phone switch
  - Airbus A-340 airplane

© Gert Jervan

101

NSCP



© Gert Jervan

102

NSCP

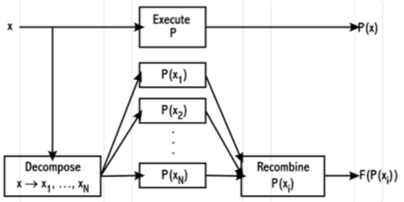
Method	N self-checking programming
Error Processing Technique	Error detection and result switching Then, Detection by comparison or by AT(s)
Criteria of Accepting Result	Relative, on variant results or Absolute with respect to specification
Execution Scheme	Parallel
Consistency of Input Data	Explicit, by dedicated mechanisms

Comparison

- N self-checking programming using acceptance tests
  - The use of separate acceptance test for each version is the main difference of this technique from recovery blocks
- N self-checking programming using comparison
  - resembles triplex-duplex hardware redundancy
  - An advantage over N self-checking programming using acceptance tests is that the application independent decision algorithm is used for fault detection

Data Diversity

- To complement design diversity
- Using data re-expression algorithms (DRA) to obtain logically equivalent variants of the input data



Data re-expression via decomposition and recombination

SFT Techniques Using Data Diversity

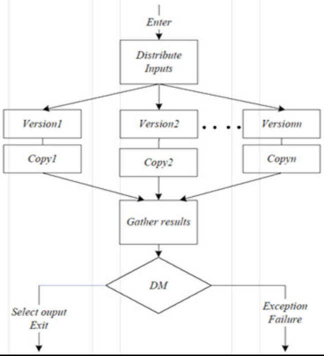
SFT Techniques	Abbr.	Error Processing
Retry Blocks	RtB	Acceptance test and Backward recovery
N-Copy Programming	NCP	Run the same process concurrently or sequentially

Retry Blocks

Retry Blocks

Method	Retry blocks
Error Processing Technique	Error detection by AT and backward recovery by DRA
Criteria of Accepting Result	Absolute, with respect to specification
Execution Scheme	Sequential
Consistency of Input Data	Implicit, from backward retry principle

NCP



N-copy Programming

Method	N-copy programming
Error Processing Technique	Decision mechanism (DM) and forward recovery
Criteria of Accepting Result	Relative, on variant results
Execution Scheme	Parallel
Consistency of Input Data	Explicit by dedicated mechanisms

Design Diversity

- The most critical issue in multi-version software fault tolerance techniques is assuring independence between the different versions of software through design diversity
- Software systems are vulnerable to common design faults if they are developed by the same design team, by applying the same design rules and using the same software tools

Design Diversity

- Decision to be made when developing a multiversion software system include
  - which modules are to be made redundant
    - usually less reliable modules are chosen
  - the level of redundancy
    - procedure, process, whole system
  - the required number of redundant versions
  - the required diversity
    - diverse specification, algorithm, code, programming language, testing technique
  - rules of isolation between the development teams

Environment Diversity

- To diversify the software operating circumstance temporarily.
- The typical examples of environment diversity technique are progressive retry, rollback rollforward recovery with checkpointing, restart, hardware reboot, etc.

An Adaptive Approach for n-Version Systems

- Model and manage different quality levels of the versions by introducing an individual weight factor to each version of the n-version system.
- This weight factor is then included in the voting procedure, i.e. the voting is based on a weighted counting.

Why Fuzzy Voting

- In traditional voting, equality relation regards two real numbers as equal if their difference is smaller than fixed tolerance ε. For different version outputs that are "closer" to each other than the fixed threshold there is no gradual comparison. As a result, certain interconnection of faults could incur incorrect selection.
- Fuzzy equivalence relation results in more reliable systems

115

Fuzzy Equality Equation

- Traditional Equality Equation

$$r_{i,j} = \begin{cases} 1, & if\ |x_i - x_j| \le \varepsilon \\ 0, & otherwise \end{cases}$$

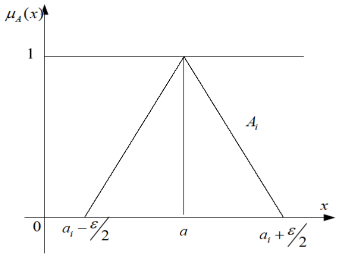
- Fuzzy Equality Equation

$$\mu_{A_i}(x_i) = \begin{cases} 1 - \frac{|a_i - x_i|}{\varepsilon/2}, & if\ |x_i - a_i| \le \varepsilon/2 \\ 0, & otherwise \end{cases}$$

116

Output of Fuzzy Sets (Triangular Shape)

- The fuzzy logic maps the input vector into an output nonlinearly



117

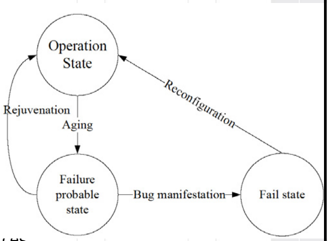
New Techniques

- Rejuvenation
- (Not classifiable in design diversity or data diversity, actually environmental diversity)

118

Reconfiguration and Rejuvenation

- Complementary ways
- Reconfiguratic
  - Reactive
  - Analogy
    - Event-drive interrupts
- Rejuvenation
  - Proactive
  - Analogy
    - Polling resources



119

Software Aging

- When software application executes continuously for long periods of time, some of the faults cause software appear to age due to the error conditions that accrue with time and/or load. This phenomenon is called software aging which is reported in
  - Telecommunication billing application over time experiences a crash or a hang failure.
  - A telecommunication switching software
  - Netscape and xrn
  - Safety critical systems Patriot missile's software, where the accumulated errors led to a failure that resulted in loss of human lives.

120

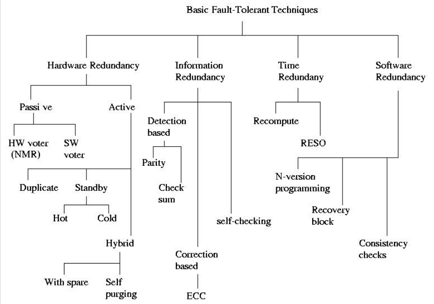
Discussion

- Each software fault tolerance technique need to be tailored to particular applications.
- This should also be based on the cost of the fault tolerance effort required by the customer. The differences between each technique provide some flexibility of application.

© Gert Jervan

121

A summary chart of all techniques



© Gert Jervan

122

Information redundancy

- Key concept - add redundancy to information/data
  - all schemes use Error detecting or Error correcting coding
- Use of parity
  - very effective single error detection
  - encoding and decoding cost is low
  - commonly used in memories, transmission over short reliable channels
  - limitations
    - unable to detect common multiple errors
    - can not be used in data transformation - for example addition does not preserve parity

© Gert Jervan

123

Information redundancy

- Error correcting codes
  - triplication
  - Hamming code
  - byte error detection/correction
  - cyclic code
- m-out-of-n codes
  - encode each word (data/control) such that the coded word is of length n and each coded word has exactly m 1's in it
    - can detect all single errors
    - can detect all unidirectional multiple errors

© Gert Jervan

124

Information redundancy

- Berger codes
  - n information bits are encoded into an n+k bit code word. The k check bits are binary encoding of the number of 1's (or 0's) in the n information bits
    - can detect all single errors
    - can detect all unidirectional multiple errors if carefully designed
- Arithmetic codes
  - AN code
    - used for arithmetic function unit designs
    - each data word is multiplied by a constant A
    - makes use of the identity A(N+M) = AN + AM
    - choice of A is important

© Gert Jervan

125

Information redundancy

- Arithmetic codes (Contd.)
  - Residue code
    - makes use of the fact (M+N) mod k = (M mod k + N mod k) mod k
  - Checksums
    - data is sent/stored with a checksum and when used the checksum is regenerated and compared to the a priory known checksum
    - functions used for checksum
      - add, exclusive-OR (bit wise), end with end around carry, LFSR, …
    - limitation
      - can only perform (normally) error detection

© Gert Jervan

126

Information redundancy

- Self-Checking
  - This is a form of hardware redundancy but often it is closely related to ECC techniques, therefore I have chosen to include it here
  - Assumptions: inputs are coded and outputs are coded
  - Objective: in the presence of a fault the circuit should either continue to provide correct output(s) or indicate by providing an error indication that there is a fault.
    - Clearly error indication can not be 1-bit output (why?)
    - With 2-bits output, 00 and 11 may indicate no failure
    - other output combinations (10, 01) may indicate a failure

© Gert Jervan 127

Information redundancy

- Self-Checking (contd.)
  - Example application
    - two devices produce identical outputs and we compare these outputs to check their equality
    - checker has two outputs encoded as follows
      - 00 equal
      - 11 unequal
      - 01 or 10 possible fault in the circuit
      - (we will discuss input encoding when we discuss an example of a 2-rail 1-bit checker)

© Gert Jervan 128

Information redundancy

- Self-Checking (contd.)
  - Definitions
    - a circuit is fault secure if in the presence of a fault, the output is either always correct, or not a code word for valid input code words
    - a circuit is self-testing if only valid inputs can be used to test it for the faults
    - a circuit is totally self-checking if it is fault secure and self-testing
  - Example: a totally self-checking 2-rail 1-bit comparator
    - assumptions
      - 2 inputs and each input x is available as x and its complement
      - x and its complement are independently generated
      - note with these assumption the input space is encoded (4 valid inputs out of 16 possible inputs)
      - single stuck-at fault model

© Gert Jervan 129

Time redundancy

- Key Concept - do a job more than once over time
  - examples
    - re-execution
    - re-transmission of information
  - different faults and capabilities of different schemes
    - transient faults
      - re-execution and re-transmission can detect such faults provided we wait for transient to subside
    - permanent faults
      - simple re-execution or re-transmission will not work. Possible solutions
        - send or process shifted version of data
        - send or process complemented data during second transmission

© Gert Jervan 130

Time redundancy

- Different faults and capabilities of different schemes (contd.)
  - faults in ALU
    - re-execution with complement or shifted version can detects permanent and transient faults
    - (RESO concept - re-computation with shifted operands)
  - multiple re-computations
    - can detect and possibly correct transient and permanent faults if properly employed/designed

© Gert Jervan 131