IAF0030
Arvutitehnika erikursus I

Advanced Issues of Fault Tolerance Redundancy

Gert Jervan

Arvutitehnika instituut (ATI)
Tallinna Tehnikaülikool

Tallinn University of Technology

Lecture Outline

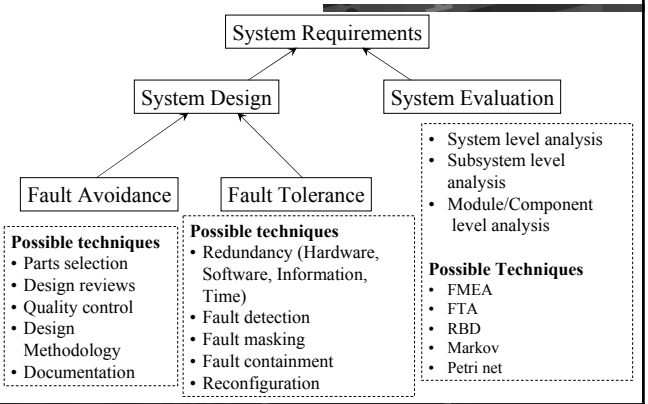
- **Introduction**
- **Hardware Redundancy**
- **Information Redundancy**
- **Time Redundancy**
- **Software Redundancy**

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

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.

System Design & Evaluation Top-Level View



Hardware Redundancy

- 3 basic forms: passive, active, and hybrid
- Passive:
  - 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

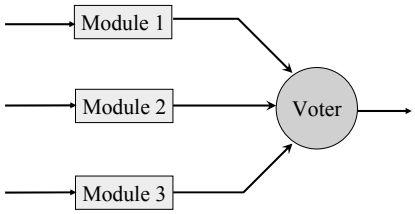
Passive Hardware Redundancy

- Tolerate the faults by fault masking
- Voting mechanisms, majority voting
- Do not need fault detection or reconfiguration

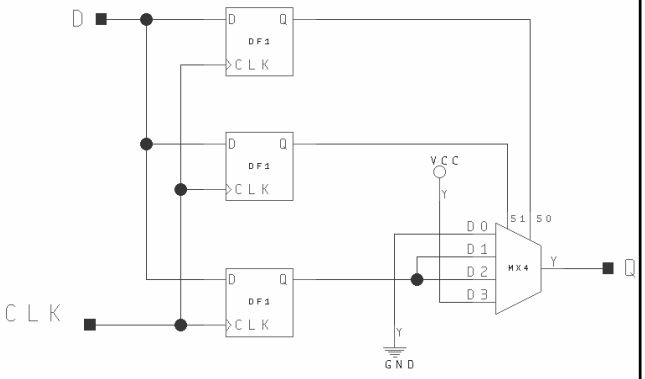
Passive Hardware Redundancy

- 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

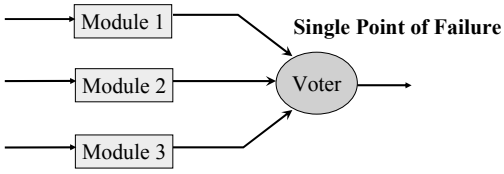


TMR/Voter Structures

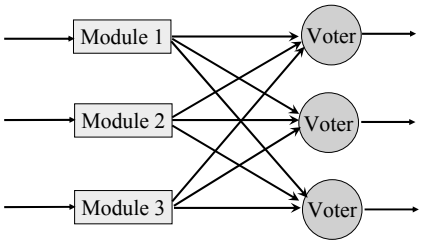


Fault-Tolerance capability

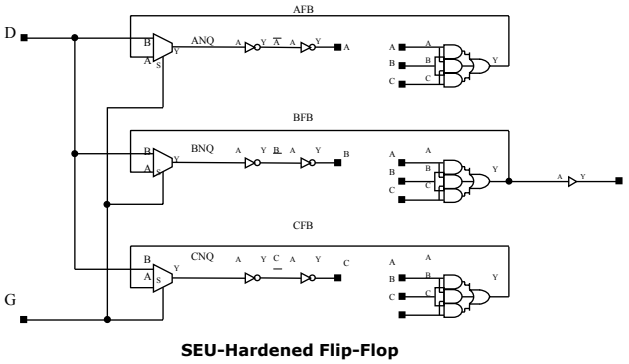
- Assuming perfect voter, how many module faults can the TMR technique tolerate?
- What if 2 modules fail the same way?
- Does TMR technique provide fault detection capability?
- How about imperfect voter?
- Performance impacts from the voter in the TMR technique



TMR with Triplicated Voters

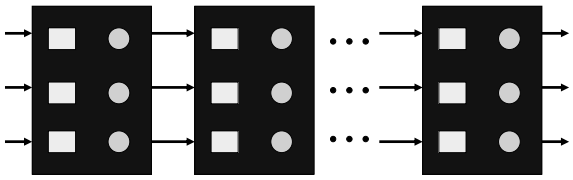


Static Redundancy Example



SEU-Hardened Flip-Flop

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

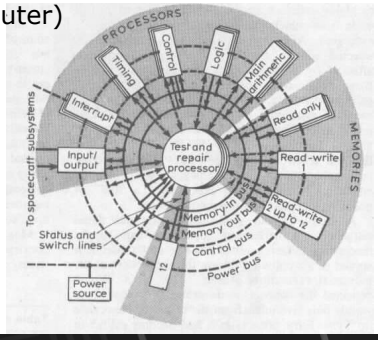
Passive Hardware Redundancy

- 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)

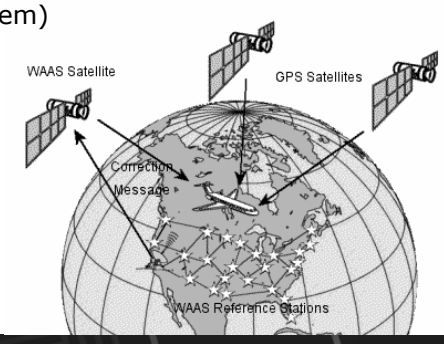
Example Systems Using TMR Technique

- JPL STAR (Self-Testing And Repairing computer)

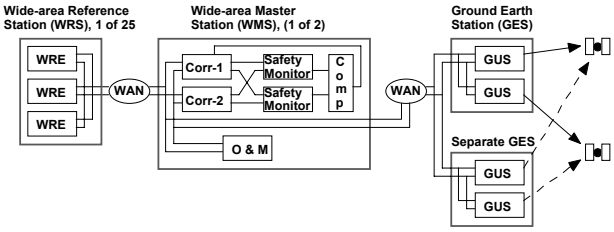


Example Systems Using TMR Technique

- FAA WAAS (Wide Area Augmentation System)



WAAS Block Diagram



Active Hardware Redundancy

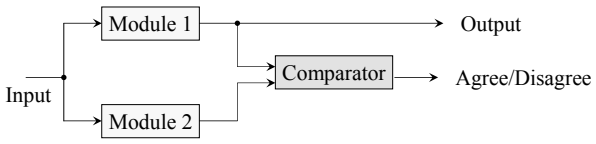
- Achieve fault tolerance by detecting the existence of faults and performing some action to remove the faulty parts
- Require the system be reconfigured to tolerate faults
- 3 steps: fault detection, fault location, and fault recovery

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



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

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.
- 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.

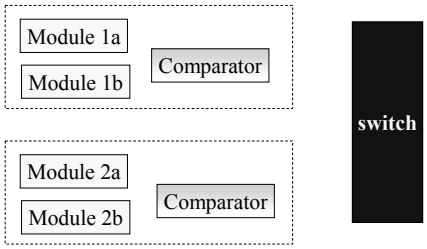
Hot & Cold Standby Sparing

- 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.

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

Pair-and-a-spare scheme



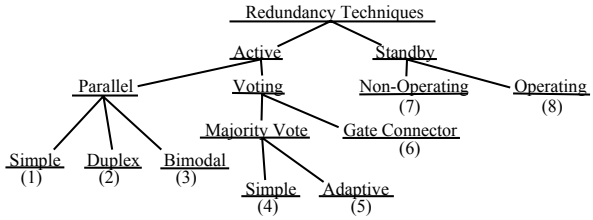
http://www.stratus.com/

Types of Redundancy

NASA Office of Logic Design - klabs.org

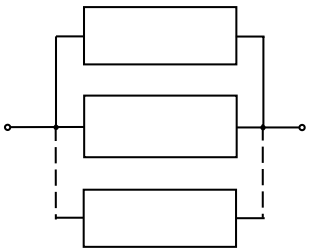
- Classified on how the redundant elements are introduced into the circuit
- Choice of redundancy type is application specific
- Active or Static Redundancy
  - External components are not required to perform the function of detection, decision and switching when an element or path in the structure fails.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

Redundancy Techniques



Simple Parallel Redundancy

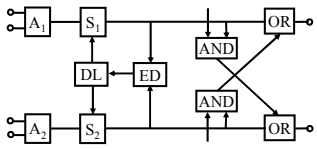
Active - Type 1



In its simplest form, redundancy consists of a simple parallel combination of elements. If any element fails open, identical paths exist through parallel redundant elements.

Duplex Parallel Redundancy

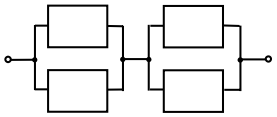
Active - Type 2



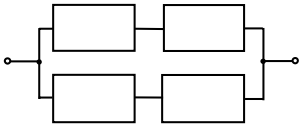
This technique is applied to redundant logic sections, such as A1 and A2 operating in parallel. It is primarily used in computer applications where A1 and A2 can be used in duplex or active redundant modes or as a separate element. An error detector at the output of each logic section detects noncoincident outputs and starts a diagnostic routine to determine and disable the faulty element.

Bimodal Parallel Redundancy

(a) Bimodal Parallel/ Series Redundancy



(b) Bimodal Series/ Parallel Redundancy

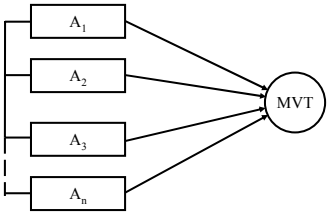


Active - Type 3

A series connection of parallel redundant elements provides protection against shorts and opens. Direct short across the network due to a single element shorting is prevented by a redundant element in series. An open across the network is prevented by the parallel element. Network (a) is useful when the primary element failure mode is open. Network (b) is useful when the primary element failure mode is short.

Simple Majority Voting

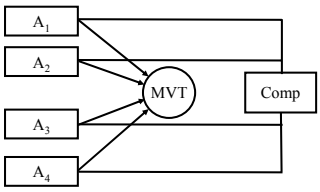
Active - Type 4



Decision can be built into the basic parallel redundant model by inputting signals from parallel elements into a voter to compare each signal with remaining signals. Valid decisions are made only if the number of useful elements exceeds the failed elements.

Adaptive Majority Voting

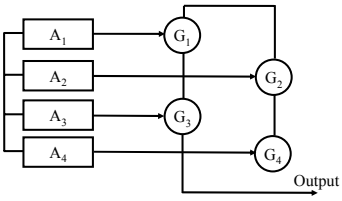
Active - Type 5



This technique exemplifies the majority logic configuration discussed previously with a comparator and switching network to switch out or inhibit failed redundant elements.

Gate Connector Voting

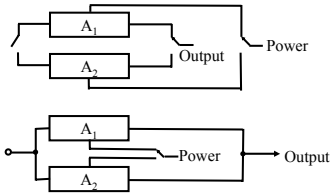
Active - Type 6



Similar to majority voting. Redundant elements are generally binary circuits. Outputs of the binary elements are fed to switch-like gates which perform the voting function. The gates contain no components whose failure would cause the redundant circuit to fail. Any failures in the gate connector act as though the binary element were at fault.

Non-Operating Redundancy

Standby - Type 7



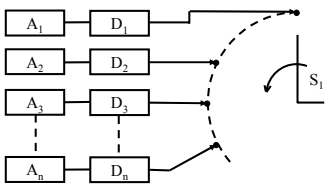
A particular redundant element of a parallel configuration can be switched into an active circuit by connecting outputs of each element to switch poles. Two switching configurations are possible.

1) The element may be isolated by the switch until switching is completed and power applied to the element in the switching operation.

2) All redundant elements are continuously connected to the circuit and a single redundant element activated by switching power to it.

Operating Redundancy

Standby - Type 8



In this application, all redundant units operate simultaneously. A sensor on each unit detects failures. When a unit fails, a switch at the output transfers to the next unit and remains there until failure.

Hybrid Hardware Redundancy

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

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
- For passive redundancy, 5 modules are needed to tolerate 2 faulty modules.

Hybrid hardware redundancy

- Self purging redundancy
  - initially start with NMR
  - purge one unit at at time till arrive at 3MR
    - can tolerate more faults initially compared to NMR with spare
    - cost of the switch - higher?
    - How does it compare to sift-out redundancy?
- Triple-duplex redundancy
  - combines duplication-with-compare and TMR

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

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

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

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

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

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)

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

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

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

Software Fault Tolerance (SFT)

- Single version software FT techniques (mostly in self-checking software)
- Multiple version software FT techniques (design diversity)
  - recovery blocks (RcB),
  - N-version programming (NVP),
  - and N self-checking programming (NSCP).
- Multiple data representation techniques (data diversity)
  - retry blocks (RtB)
  - and N-copy programming (NCP)
- Temporal diversity
- Environment diversity

Dynamic fault tolerance in software

- 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

- Environmental detection
  - hardware (illegal instruction) or OS/RTS (null pointer)
- Application detection: checks that can be detected by the application
  - replication checks (as in NVP)
  - timing checks: watchdog timers (which must be reset by sw); deadline misses reported by the environment
  - reversal checks: with 1-1 mapping between input and output
  - coding checks using redundant data (e.g., checksum)
  - reasonablesness checks: on state, on value of an object, checked through assertions
  - structural checks: integrity of data objects (e.g., #elements in a list), redundant pointers, status information
  - dynamic reasonableness checks: e.g., correlation between consecutive outputs.

2 - Damage Assessment & Containment

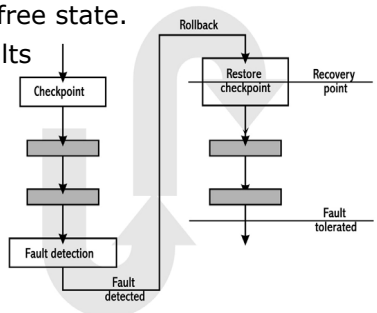
- Necessary due to the delay between fault and error
- Goal of containment is to minimize damage caused by a faulty component
  - "firewalling"
- Assessment closely related to containment techniques used
- Two techniques: modular decomposition, atomic actions
- Modular decomposition provides static damage containment
  - allows data to flow through
  - however, the static structure of the software system is lost at run-time
- Atomic actions provide dynamic damage containment
  - an activity is atomic if there are no interactions between the activity and the system for the duration of the activity
  - allow moving system from one consistent state to another.

3 Error 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
- 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)

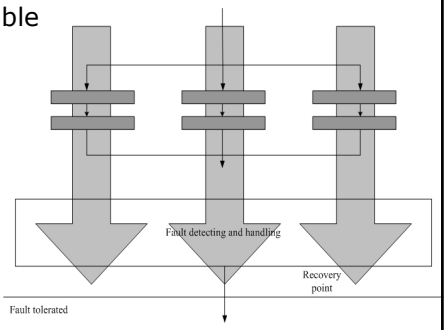


Forward Recovery

- 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

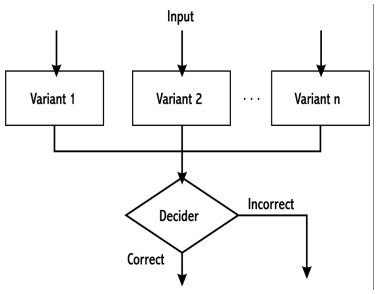
- Even with recovery, the error may recur. Need to eradicate the fault from the system
- Automatic treatment of faults is very application specific
- Make some assumptions. For instance:
  - all faults are transient
- Fault treatment in two stages
  - Fault location
  - System repair
- Fault location
  - 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.

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

Design Diversity

- Higher cost

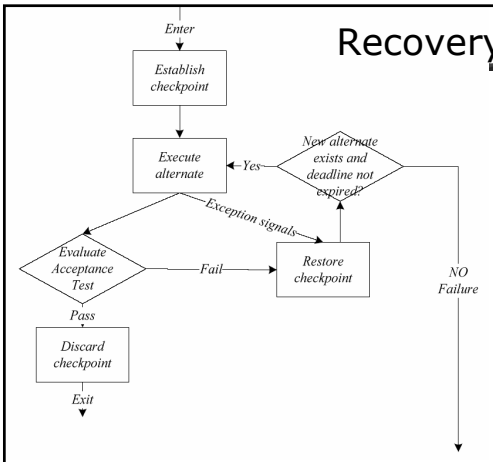


SFT Techniques Using Design Diversity

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

Recovery Blocks



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

Recovery Blocks

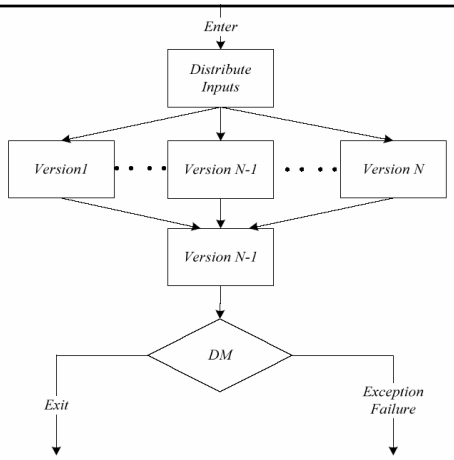
- A language level support for backward error recovery
  - blocks in the normal programming language sense, but
  - at the entrance to the block is an automatic recovery point and
  - at the exit an acceptance test to test that the system is an an acceptable state
  - if the acceptance test fails, the 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

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).
- Several experimental implementations
  - S. Srivastava, "Concurrent Pascal with backward error recovery: Implementation", Software Practice and Experience
  - Purtilo, Jalote, IEEE Trans on SW Engg, 1991
  - Garg, Gupta, CASES 2001.

NVP



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

N-Version Programming

- Built off TMR/NMR in hardware
- 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, ...

NVP

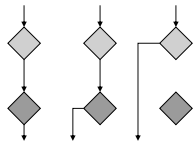
- Controlled by a driver process
  - invokes each of the versions
  - waiting for the versions to complete
  - comparing and acting on the results
- Problem: assumes programs run to completion!
  - So the versions must actually interact (with the driver program)
    - Comparison Points: points in the 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

Interaction of Drivers and Versions

- This interaction is specified as consisting of 3 components
  - Comparison vectors
    - Data structures that represent the outputs (or votes) produced by the versions plus any attributes
    - E.g., in an ATC, values to be compared are aircraft positions, an attribute may be the associated radar reading (recent or calculated)
  - Comparison status indicators
    - From driver to versions: indicate the actions that each version must perform as a result of driver's comparison
    - E.g., continuation, termination, continuation after changing a comparison vector (to a majority value)
  - Comparison points
    - Define the granularity of fault tolerance
- If different programming languages have been used for versions, then a RTOS will typically provide the means of communication and synchronization

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.

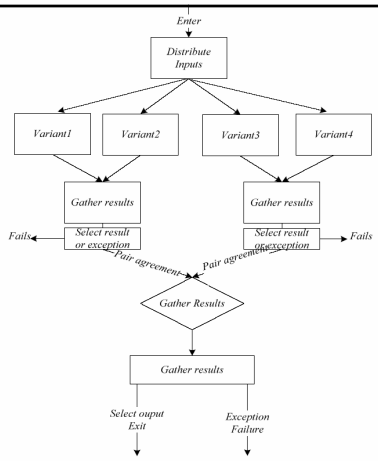
The problem will reappear even when using inexact comparison (just have to be near a threshold value).

And what happens when there are multiple solutions?

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.

NSCP

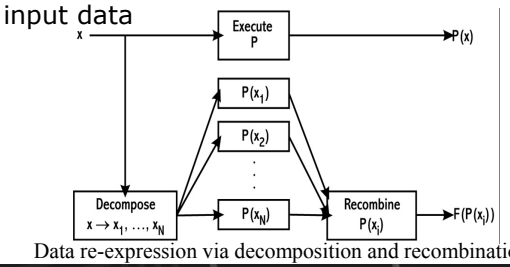


N Self-Checking Programming

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

Data Diversity

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

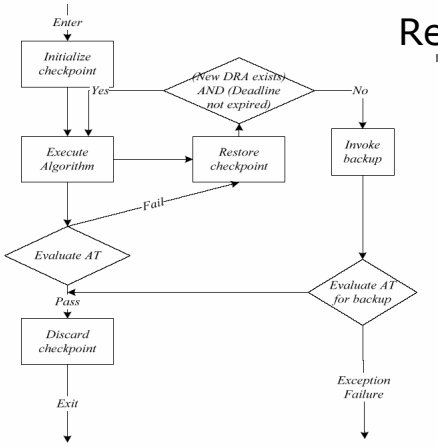


Data re-expression via decomposition and recombination

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

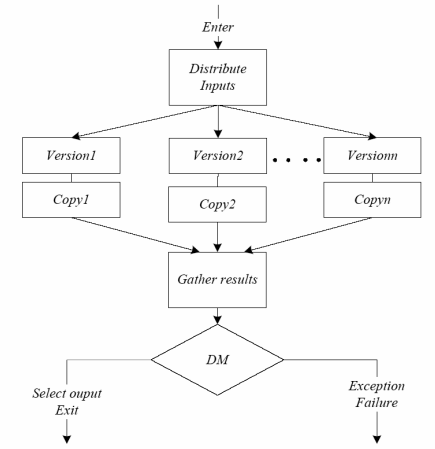
Retry Blocks



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

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.

Adjudicators

- Voter
- AT
- Hybrid, other

What's new

- Modifications to Traditional Techniques
  - Adaptive N-version Systems
  - Fuzzy Voting
  - Abstraction
  - Parallel Graph Reduction
- New Concepts (Not Classifiable in Data or Design Diversity)
  - Rejuvenation

Modifications to Traditional Techniques

- Adaptive N-version systems
- Fuzzy voting
- Abstraction
- Parallel Graph Reduction

Why Adaptive

- Defective components should be removed from the voting process
- The voting procedure can be adaptively modified and tailored to the fault state of the overall system

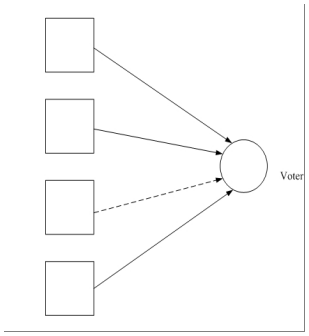
An Adaptive Approach for n-Version Systems

- 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.

Voting schemes

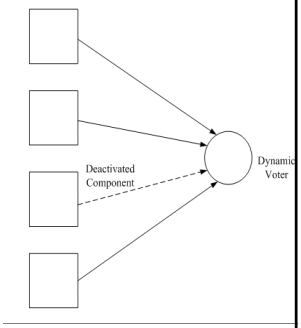
- Static voting:

  fixed number of versions



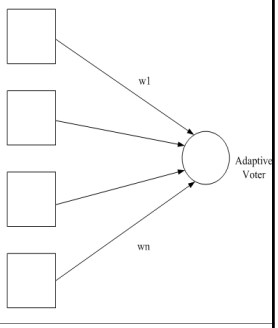
Voting schemes

Dynamic voting: defective components are removed from the voting process



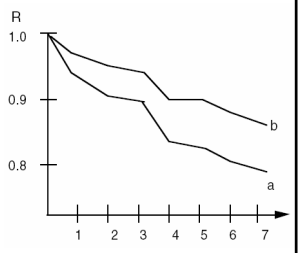
Voting schemes

Adaptive voting: versions have dynamically changeable weight factors



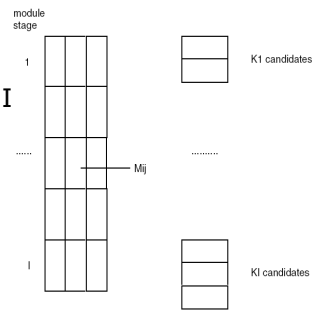
Example

- Cumulated reliability of 7 components of a software system for a satellite control application
  - a) classical 3-version system
  - b) 3-version system with adaptive weight factors



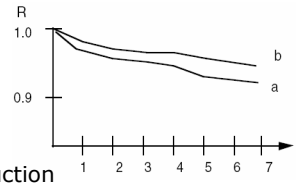
Structure of a Modular n-Version System

- Solve I tasks
- Mij:
- Version j of module I
- i=1,..I; j=1,…n.



Example

- Cumulated reliability of 7 components of a software system for a satellite control application
  - a) 3-version system with modular construction of the versions
  - b) 3-version system with modular construction of the version and additional adaptive adjustment of module weights



Modifications to Traditional Techniques

- Adaptive N-version systems
- Fuzzy voting
- Abstraction
- Parallel Graph Reduction

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

Fuzzy Equality Equation

- Traditional Equality Equation

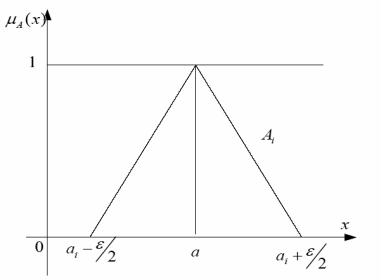
$$r_{i,j} = \begin{cases} 1, & if \mid x_i - x_j \mid \leq \varepsilon \\ 0, & otherwise \end{cases}$$

- Fuzzy Equality Equation

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

Output of Fuzzy Sets (Triangular Shape)

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



Fuzzy Equivalence

- A fuzzy relation is a fuzzy equivalence relation if and only if all three properties of fuzzy relations are satisfied:
  - reflexivity,
  - Symmetry,
  - and transitivity.
- Without a satisfied transitivity property the equivalence relation further leads to incorrect classification.

Modifications to Traditional Techniques

- Adaptive N-version systems
- Fuzzy voting
- Abstraction
- Parallel Graph Reduction

Why Abstraction Improve Fault Tolerance

- Reduce the cost of fault tolerance
- Improve its ability to mask software errors.

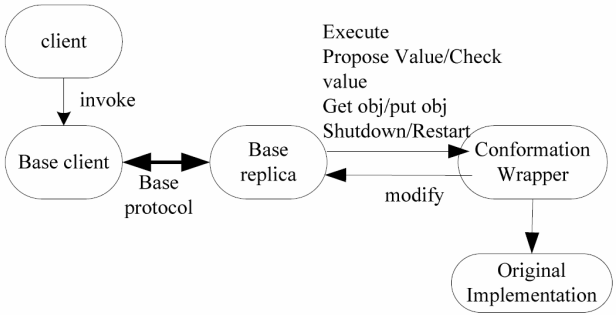
An Example of Abstraction: BASE

- Byzantine fault tolerance (BFT) with Abstract Specification Encapsulation (BASE)

How BASE Works

- BASE reduces cost because it enables reuse of off-the-shelf service implementations.
- It improves availability because each replica can be repaired periodically using an abstract view of the state stored by correct replicas, and because each replica can run distinct or nondeterministic service implementations, which reduces the probability of common mode failures.

BASE Function Calls and Upcalls.



Modifications to Traditional Techniques

- Adaptive N-version systems
- Fuzzy Voting
- Abstraction
- Parallel Graph Reduction

Parallel Graph Reduction

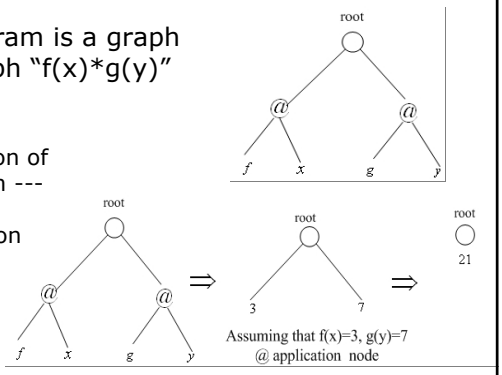
- Fault-tolerance of functional programs in parallel computing

Why Parallel Graph Reduction

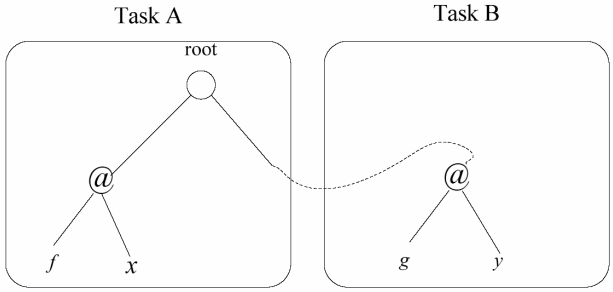
- Reduce time overhead of fault tolerance by taking advantage of referential transparency.

An Example of Graph Reduction

- A program is a graph ---Graph "f(x)*g(y)"
- Execution of program --- Graph Reduction

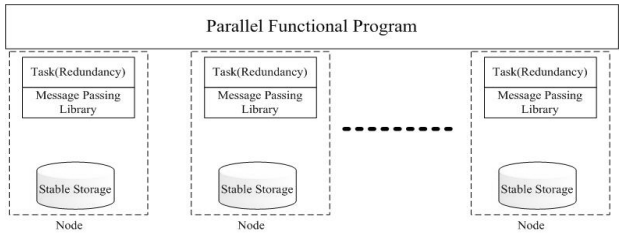


An Example of Parallel Graph Reduction



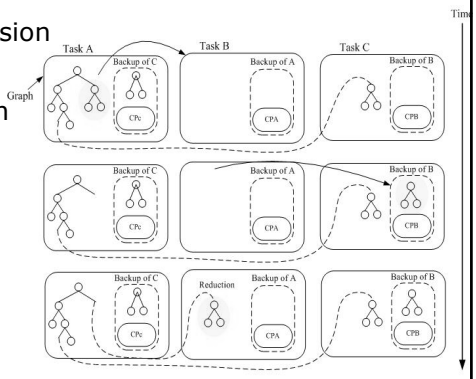
System Architecture

- Each subgraph is assigned to each node and reduced in parallel
- A task is executed in a node and its backup is stored in another node



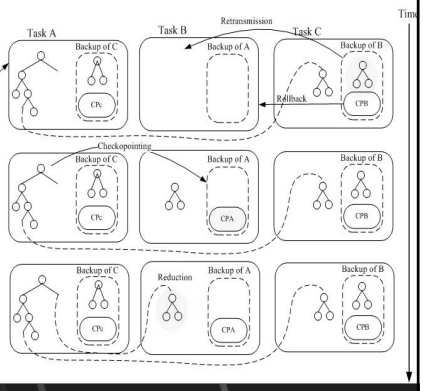
Backup Procedure

- Transmission
- Backup
- Reduction



Error Recovery

- Rollback
- Retransmission
- Checkpointing
- Reduction

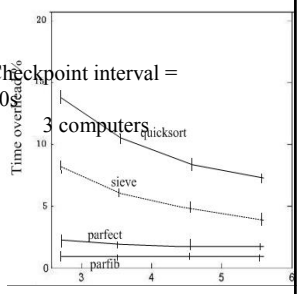


Time overhead vs. Number of Tasks

- Benchmark Programs:
- Par(allel)fact(orial)
- Par(allel)fib(onacci)
- Quicksort
- Sieve(searching for
- Prime numbers by
- Using the sieve of
- Eratosthenes)

Checkpoint interval = 20s

3 computers



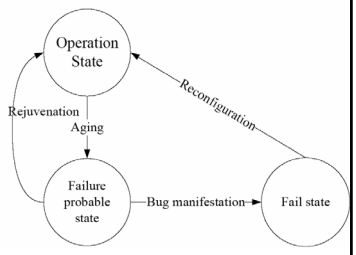
Overhead = time to take checkpoint and backup

New Techniques

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

Reconfiguration and Rejuvenation

- Complementary ways
- Reconfiguration
  - Reactive
  - Analogy
    - Event-driven interrupts
- Rejuvenation
  - Proactive
  - Analogy
    - Polling resources



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.

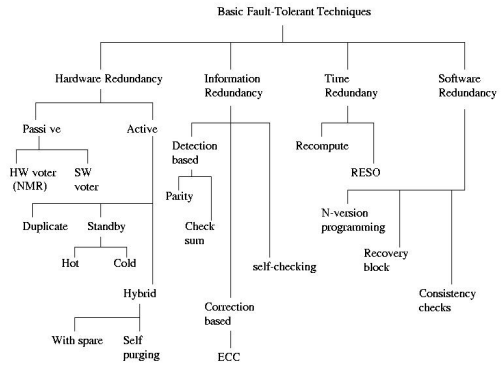
Discussion

- 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.

Summary

- An example to define the scope and list methods
- Hardware redundancy
  - passive, active, and hybrid
- Information redundancy
  - coding method and self-checking
- Time redundancy
  - re-execution, re-transmission, and RESO concept
- Software redundancy
  - consistency checks, assertion check, N-version programming, capability checks, recovery block, and N-self checking

A summary chart of all techniques



Questions?

Tallinn University of Technology

Gert Jervan

Department Of Computer Engineering
Tallinn University of Technology
Estonia