Single-version software fault tolerance techniques

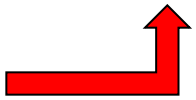
(redundancy applied to a single version of software to detect errors and recover)

Software faults

Bohrbugs
permanent design faults, deterministic in nature
identified during the testing and debugging phase

Heisenbugs
temporary internal faults (intermittent faults)
They are essentially permanent faults whose conditions of activation occur rarely or are not easily reproducible.
For example faults at boundaries between various software components with timinig dependences.
They are state dependent and input dependent faults.
(extremely difficult to identify through testing)

Basis to implement fault tolerance
- software architecture (modularization)
- system closure principle
- self-checking and self-protection principle



Software architecture
(basis to implement fault tolerance)

1) Modularization
   add to modules error detection capability
2) Hierarchy and connectivity of components
   used to analyse error propagation
3) Partitioning
   functional independent modules + control modules (that coordinate the execution)
   provide isolation between functionally independent modules
   error confinement
4) Temporal structuring of the activity between interacting components
   atomic action: activity in which the components interact with each other and
   there is no interaction with the rest of the system for the duration of the activity
   provide a framework for error confinement and recovery
   (if a failure is detected during an atomic action, only the participating components can be affetcted)

Error confinement areas, with boundary at interfaces between components

System closure fault tolerance principle

no action is permissible unless explicitly authorized (mutual suspicion)

1. Each component is only granted the capabilities needed to execute its function

2. Each component examines each request or data item from other components before acting on it
For example, each software module checks legality and reasonableness of each request received

3. A capability disabled by an error disables a valid action
(it does not result in an undesirable action)

Error detection and confinement
Added overhead, need for providing: signalling back to requestor and own strategy for dealing with erroneous requests

Self-protection and self-checking principles

Software system: a set of communicating components

Component (self-protection): protect itself by detecting errors in the information received by other interacting components

Component (self-checking): able to detect internal errors and take appropriate actions to prevent the propagation to other components

Error detection checks

Reasonableness checks: use known sematic properties of data
(acceptable range of variables, rate of change, acceptable transitions, probable results…) Based on the design requirements of a module

Reversal checks: inverse computation
use the output to compute the corresponding inputs

assume the specified function of the system is to compute a mathemathical function, output = F(input)
if the function has an inverse function F', such that F'(F(x))=x,
we can compute F'(output) and verify that F'(output) = input

Coding checks: use coding in the representation of information
technique developed for hardware can be used for software
basically in data communication (in which the content of the data is not changed)

Error detection checks

Structural checks: use known properties of data structures

lists, trees, queues can be inspected for a number of elements

(redundant data structure could be added, extra pointers, embedded counts, …)

Timing checks: watchdog timers

check deviations from the acceptable module behaviour

Run-time checks:

error detection mechanism provided in hardware (dived by 0, overflow, underflow, …) can be used to detect design errors

Error recovery

Exception handling

exceptions are signalled by the error detection mechanism

catch() clauses implement the appropriate error recovery

Three classes of exceptions

interface exceptions
(invalid service request, triggered by the self-protection mechanism, handled by the module that requested the serice)

internal local exceptions
(an error in the internal operations of the module, triggered by the error detection mechanism of the module, handled by the module)

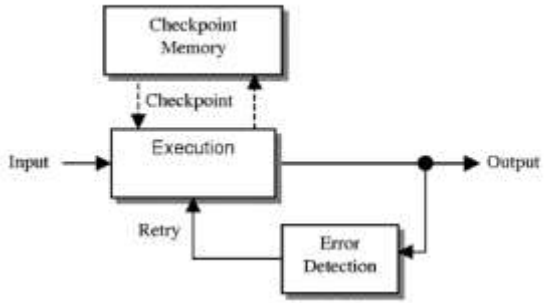
failure exceptions
(detected error, not handled by the fault processing mechanism. Tell the module requesting the service that the service had a failure)

Error confinement is essential to design effective exception handlers

Checkpointing and restart recovery mechanism

Most of the faults at this stage are *Heisenbugs,* hence these faults result in transient failures, i.e., failures which may not recur if the software is restarted.

Restart is usually enough to successful completion of the execution of the module

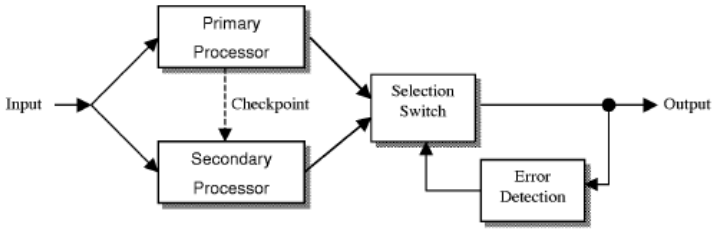


W. Torres-Pomales
Software fault tolerance: A tutorial
NASA,/TM-2000-210616, 2000

Checkpointing and restart recovery mechanism

- **Static**
  restart from predetermined states
  (initial state or intermediate state, ..)
- **Dynamic**
  restart from checkpoints created during
  the execution of the module (backword error recovery)

Process pair



Process pair:

two processors
uses the same version of the software
the primary processor sends checkpoints to the other
error detection:
the secondary process takes the role of primary and starts from the checkpoint

C/C++ language: checkpoint libraries

Redundancy at code level

1. Duplication implemented in a compiler

   RECCO: a REliable c/c++ Code COmpiler for dependable applications

   - duplicate variables (code analysis to find important variables - read variables, variables keeping a value for a long time, lifetime of variables)

   - duplicate instructions - selective instruction duplication (e.g., instruction that are executed more frequently)

   Covered faults: data errors, memory instruction in memory errors

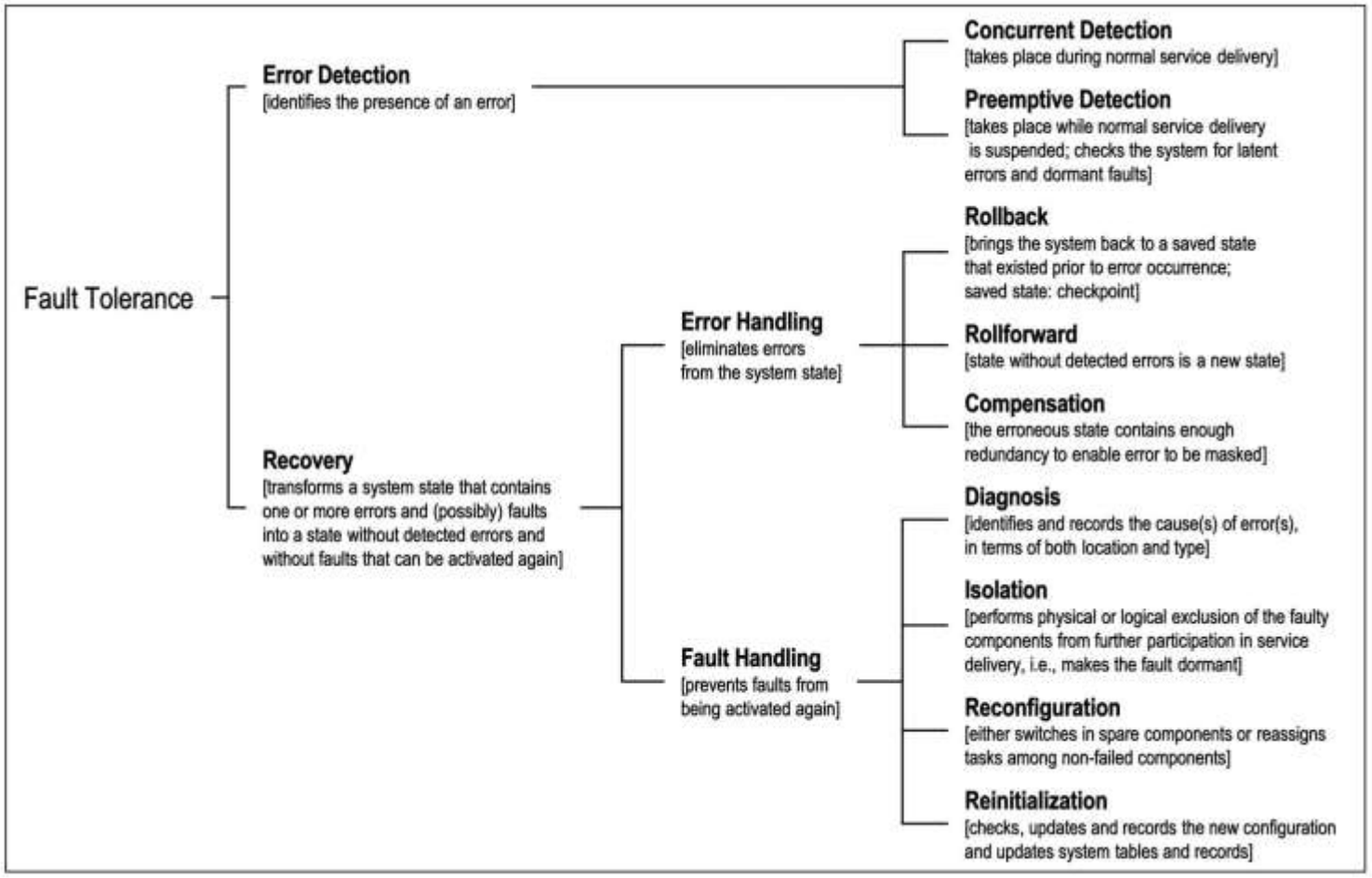
2. Add information to the Control Flow Graph, and check conditions at run-time

   Covered faults: control flow errors

3. ……

Organisation of fault tolerance

From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004



Error Recovery

Forward recovery

transform the erroneous state in a new state from which the system can operate

Example:
real-time control systems, an occasional missed response to a sensor input is tolerable The system can recover by skipping its response to the missed sensor input

- Fundamental in case of interaction with the outside world
(Computing systems must talk to the outside world)
If these actions are not correct, the computer may still limit (or undo) the damage by a compensating action

Forward recovery would help return the system to a consistent state by sending the environment a message informing it to disregard previous output from the program

Example:
A cash dispensing machine does not give the right money
forward recovery: tell the bank and ask the bank for the money

Backward recovery

bring the system back to a state prior to the error occurrence

- Checkpointing
  (coordinated checkpointing, overhead)

Backward and forward recovery can be combined if the error persists

Fault handling

Fault location

1. can the error detection mechanism identify the faulty component/task with sufficient precision?
   - LOG and TRACES are important
   - diagnostic checks

2. What if diagnostic information / testing components are themselves damaged?

3. System level diagnosis:

   A system is a set of nodes:
   - who tests whom is described by a testing graph
   - checks are never 100% certain

   Suppose A tests B.
   If B is faulty,
   A has a certain probability (we hope close to 100%) of finding out.
   But if A is faulty too,
   it might conclude B is OK; or says that C is faulty when it isn't

Fault treatment

1. Faulty components could be left in the system
   - faults can add up over time

2. Reconfigure faulty components out of the system
   - physical reconfiguration
     turn off power, disable from bus access, ..
   - logical reconfiguration:
     don’t talk, don’t listen to it

3. Excluding faulty components will in the end exhaust available redundancy
   -insertion of spares
   -reinsertion of excluded component after thorough testing, possibly repair

4. Newly inserted components may require:
   - reallocation of software components
   - bringing the recreated components up to current state

Observations

Fault tolerance uses replication for error detection and system recovery

Fault tolerance relies on the independency of redundancies with respect to the process of fault creation and activations

When tolerance to physical faults is foreseen, the channels may be identical, based on the assumption that hardware components fail **independently**

When tolerance to design faults is foreseen, channels have to provide identical service through separate designs and implementation (through **design diversity**)

Fault masking will conceal a possibly progressive and eventually fatal loss of protective redundancy.

Practical implementations of masking generally involve error detection (and possibly fault handling), leading to masking and error detection and recovery.