Fault Tolerance in Distributed Systems

A system's ability to tolerate failure-1

- Reliability: the likelihood that a system will remain operational for the duration of a mission
  - requirement might be stated as 0.999999 availability for a 10 hr mission -> the probability of failure during the mission must be at most 10^{-6}
  - Very high reliability is most important in critical applications - space shuttle, industrial control, in which failure could mean loss of life

A system's ability to tolerate failure-2

- Availability: expresses the fraction of time a system is operational
  - a 0.999999 availability means the system is not operational at most one hour in a million hours
  - a system with high availability may in fact fail -> its recovery time and failure frequency must be small enough to achieve the desired availability
  - high availability is important in airline reservations, telephone switching etc, in which every minute of downtime translates into revenue loss

Importance of Design

- A good fault- tolerant system design requires a careful study of failures, causes of failures, and system responses to failures
- Such a study should be carried out in detail before the design begin and must remain part of the design process

Requirement Specification-1

- Planning to avoid failures is most important
- A designer must analyze the environment and determine the failures that must be tolerated to achieve the desired level of reliability
- To optimize fault tolerance, it is important to estimate actual failure rate for each possible failure

Requirement Specification-2

- Failure types:
  - Some are more probable than others
  - Some are transient, others permanent
  - Some occur in hardware, others in software

Design-1

- Design systems that tolerate faults that occur while system is in use
- Basic Principle - Redundancy
  - Spatial - redundant hardware
  - Informational - redundant data structures
  - Temporal - redundant computation

Design-2

- Redundancy costs money and time
  - Must optimize the design by trading off amount of redundancy used against the desired level of fault tolerance
  - Temporal redundancy usually requires re-computation and results in a slower recovery from failure
  - Spatial has faster recovery but increases hardware costs, weight, and power requirements

Design-3

- Commonly Used Techniques for Redundancy
  - Modular redundancy
    - Uses multiple, identical replicas of hardware modules and a voter mechanism
    - The outputs from the replicas are compared, and correct output is determined - majority vote
    - Can tolerate most hardware faults in a minority of the hardware modules

Design-4

- N- Version Programming
  - Write multiple versions of a software module
  - Outputs from these versions are received and correct output is determined via voting mechanism
  - Each version is written by different team, with the hope that they will not contain the same bugs
  - Can tolerate software bugs that affect a minority of versions
  - Cannot tolerate correlated fault - reason for failure is common to two (or more) modules eg two modules share a single power supply, failure of which causes both to fail

Error- Control Coding-1

- Replication is expensive
- For certain applications - RAM, Buses, error correcting codes can be used
  - Hamming or other codes
- Checkpoints and rollbacks
- A checkpoint is a copy of an application's state saved in some storage that is immune to the failures under consideration
- A rollback restarts the execution from a previously saved checkpoint

Error- Control Coding-2

- When a failure occurs, the application's state is rolled back to the previous checkpoint and restarted from there.
- Can be used to recover from transient as well as permanent hardware failures
- Can be used for uniprocessor and distributed applications

Recovery Blocks

- Uses multiple alternates to perform the same function
- One module is primary others are secondary
- When primary completes execution, its outcome is checked by an acceptance test
- If the output is not acceptable, a secondary module executes and so on until either an acceptable output is obtained or alternates are exhausted
- Can tolerate software failures because alternates are usually implemented with different approaches (software algorithms)

Dependability Evaluation

- Once a system has been designed, it must be evaluated to determine if it meets reliability and dependability objectives

- Two dependability approaches:
- Use an Analytical Model
  - can help developers to determine a system's possible states and probabilities of transitions among them
  - can be difficult to analyze models accurately
- Injecting Faults
  - Various types of faults can be injected to determine various dependability metrics

Dependability Evaluation-2

- In distributed systems a transaction based Service can accept occasional failures followed by a lengthy recovery procedure
- A Realtime Service - Process Control
  - may have inputs that are readings taken from sensors
  - may have outputs to actuators that are used to control a process directly or to activate alarms so that humans can intervene in the process
  - due to strict timing requirements, recovery must be achieved within a very small time limit e.g. air traffic control, monitoring patients, controlling reactors

Dependability Evaluation-2

- A Fault- Tolerant Service
  - For a service to perform correctly, both the effect on a server's resources and the response sent to the client must be correct
  - Correct behavior must be specified
  - Failure Semantics - the ways in which the service can fail, must be specified
  - Can detect a fault, thus,
    - fails predictably
    - masks fault from its users
    - operates in the presence of faults in services on which it depends

Dependability Evaluation-2

- Characteristics of Faults (Failures)
  - Omission Failure
  - A server omits to respond to a request
  - Response Failure
  - Value failure - returns wrong value
  - State transition failure - has wrong effect on resources
  - Timing Failure- any response that is not available to a client within a specified real time interval
  - Server Crash Failure: a server repeatedly fails to respond to requests until it is restarted
  - Amnesia- crash - a server starts in its initial state, having forgotten its state at the time of the crash, ie loses the values of the data items
  - Pause- crash - a server restarts in the state before the crash
  - Halting- crash - server never restarts

Failure Examples

- UDP service
  - has omission failures because it occasionally looses messages
  - does not have value failures because it does not transmit corrupt messages.
    - UDP uses checksums to mask the value failures of the underlying IP by converting them to omission failures

Failure Examples

- RPC Call Semantics
  - At least- once masks omission failures by re-transmitting request messages, but sometimes executes a remote procedure more than once which may lead to a response failure
  - At most- once provides a service without response failures
- Multimedia Workstation
  - Server supplies digital audio containing spoken information, the speech will become incomprehensible if data is not delivered rapidly enough to maintain the speech in real time - Timing failure
- How is the failure of a Server Detected ?
- How can Amnesia- crash behavior be avoided

Failure Examples

- Hierarchical Masking
  - A server depends on lower level services, if possible, faults are masked at the lower level eg a Request- Reply protocol at a lower level will mask omission failures
- Group Masking
  - A service can be made fault tolerant by implementing it as a group of servers each running on a different computer
    - If some of servers fail, the remaining can still provide the service
    - If servers do not exhibit value failure, any output can be accepted
    - If servers exhibit value failure, then a majority vote must be taken to determine the correct output
    - If servers exhibit timing failure, accept first available output