Fault Tolerance

- Failure models
- Agreement in presence of faults
- Reliable communication
- Distributed commit
- Failure recovery

Fault Tolerance

- Single-machine systems
  - Failures are all or nothing
    - OS crash, disk failures
- Distributed systems: multiple independent nodes
  - Partial failures are possible (some nodes fail)
- Goal: automatically recover from partial failures
  - Important issue since probability of failure grows with number of independent components (nodes) in the system
  - $\text{Prob}(\text{failure}) = \text{Prob}(\text{any one component fails})$
Basic Concepts

• Need to build *dependable systems*
• Requirements for dependable systems
  – Availability: system should be available for use at any given time
    • 99.999% availability (five 9s) => 5 minutes of downtime per year
  – Reliability: system should run continuously without failure
  – Safety: temporary failures should not result in a catastrophe
    • Example: computing systems controlling an airplane, nuclear reactor
  – Maintainability: a failed system should be easy to repair

Basic Concepts

• Fault tolerance: system should provide services in the presence of faults
  – Transient faults: occur once and then disappear
  – Intermittent faults: come and go
  – Permanent faults: continue to exist until the faulty component is replaced
Failure Models

- A system is said to fail when it cannot adequately provide the services it was designed for.

<table>
<thead>
<tr>
<th>Type of failure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>A server halts, but is working correctly until it halts</td>
</tr>
<tr>
<td>Omission failure</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td>Receive omission</td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td>Send omission</td>
<td>A server fails to send messages</td>
</tr>
<tr>
<td>Timing failure</td>
<td>A server's response lies outside the specified time interval</td>
</tr>
<tr>
<td>Response failure</td>
<td>A server's response is incorrect</td>
</tr>
<tr>
<td>Value failure</td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td>State transition failure</td>
<td>The server deviates from the correct flow of control</td>
</tr>
<tr>
<td>Arbitrary failure</td>
<td>A server may produce arbitrary responses at arbitrary times</td>
</tr>
</tbody>
</table>

Failure Masking by Redundancy

(a)

(b)
Failure Masking and Replication

- **$K$-fault tolerant**: system can survive $k$ faults and yet function
- Assume processes fail silently (crash failure)
  - Need $(k+1)$ redundancy to tolerant $k$ faults
- **Byzantine failures**: processes run even if sick
  - Produce erroneous, random or malicious replies
  - Need $(2k+1)$ redundancy to tolerate $k$ Byzantine faults

Agreement in Faulty Systems

- Two perfect processes with unreliable channel
  - Need to reach agreement
- Two-army problem: Two armies waiting to attack
  - Each army coordinates with a messenger
  - Messenger can be captured by the hostile army
  - Can generals reach agreement?
  - Property: Two perfect processes can never reach agreement in presence of unreliable channel
- Byzantine generals problem: Can $N$ generals reach agreement with a perfect channel?
  - $M$ generals out of $N$ may be traitors
Byzantine Generals Problem

- The Byzantine generals problem for 3 loyal generals and 1 traitor. (a) The generals announce their troop strengths. (b) The vectors that each general assembles based on (a). (c) The vectors that each general receives in step 3.

Byzantine Generals Problem

- The same as in previous slide, except now with 2 loyal generals and one traitor.
- Property: With k faulty processes, agreement is possible only if 2k+1 processes function correctly out of 3k+1 total processes. [Lamport 82]
  - Need more than two-thirds processes to function correctly