Fault Tolerance

- Basic concepts
- Process resilience
- Reliable client-server communication
- Reliable group communication

Basic Concept: Dependability

**Availability**
- Readiness for usage

**Reliability**
- Continuity of service delivery

**Safety**
- Very low probability of catastrophes

**Maintainability**
- How easy can a failed system be repaired
Terminology

**Failure:**
When a component is not living up to its specifications, a failure occurs

**Error:**
That part of a component’s state that can lead to a failure

**Fault:**
The cause of an error

---

Terminology

**Fault prevention:**
prevent the occurrence of a fault

**Fault tolerance:**
build a component so that it can meet its specifications in the presence of faults (i.e., mask the presence of faults)

**Fault removal:**
reduce the presence, number, seriousness of faults

**Fault forecasting:**
estimate the present number, future incidence, and the consequences of faults
### Failure Models

<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></td>
</tr>
<tr>
<td>Receive omission</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td>Send omission</td>
<td>A server fails to receive incoming 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></td>
</tr>
<tr>
<td>Value failure</td>
<td>The server's response is incorrect</td>
</tr>
<tr>
<td>State transition failure</td>
<td>The value of the response is wrong</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) A → B → C

(b) A1 → V1 → B1 → V4 → C1 → V7
    A2 → V2 → B2 → V5 → C2 → V8
    A3 → V3 → B3 → V6 → C3 → V9

Triple modular redundancy
Process Resilience

Protection against faulty processes by replicating and distributing computations in a group

Flat groups:
Good for fault tolerance as information exchange immediately occurs with all group members; however, may impose more overhead as control is completely distributed (hard to implement).

Hierarchical groups:
All communication through a single coordinator » not really fault tolerant and scalable, but relatively easy to implement.

Flat Groups versus Hierarchical Groups

![Diagram of Flat and Hierarchical Groups]
Group agreement

when a group can mask any $k$ concurrent member failures, it is said to be **k-fault tolerant**

Assume crash/performance failure semantics
a total of $k + 1$ members are needed to survive $k$ member failures.

Assume arbitrary failure semantics
a total of $2k + 1$ members are needed to survive $k$ member failures.

Agreement in Faulty Systems (1)

The Byzantine generals problem for 3 loyal generals and 1 traitor

a) The generals announce their troop strengths
   (in units of 1 kilosoldiers)

b) The vectors that each general assembles based on (a)

b) The vectors that each general receives in step 3
Agreement in Faulty Systems (2)

- The same as in previous slide, except now with 2 loyal generals and one traitor.

Reliable Communication

- **Error detection:**
  - Framing of packets to allow for bit error detection
  - Use of frame numbering to detect packet loss

- **Error correction:**
  - Add so much redundancy that corrupted packets can be automatically *corrected*
  - Request retransmission of lost, or last $N$ packets
Reliable Communication

What can go wrong?
1: Client cannot locate server
2: Client request is lost
3: Server crashes
4: Server response is lost
5: Client crashes

Server Crashes

At-least-once semantics: server guarantees to carry out an operation at least once, no matter what
At-most-once-semantics: server guarantees to carry out an operation at most once
Server Crash Scenarios

<table>
<thead>
<tr>
<th>Reissue strategy</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>MPC</td>
<td>PMC</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Never</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>ZERO</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>ZERO</td>
</tr>
<tr>
<td>Only when ACKed</td>
<td>DUP</td>
<td>DUP</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Only when not ACKed</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>OK</td>
</tr>
</tbody>
</table>

M→P: send message then process
P→M: process then send message

Reliable Multicast

**Basic model:** multicast channel *c* with two groups:

- **The sender group** *SND(c)* processes that *submit* messages to channel *c*
- **The receiver group** *RCV(c)* processes that can receive messages from channel *c*

**Simple reliability:** If process *P* in *RCV(c)* at the time message *m* was submitted to *c*, and *P* does not leave *RCV(c)*, then *m* should be delivered to *P*
Basic Reliable-Multicasting Schemes

Message transmission

Report feedback

Nonhierarchical Feedback Control

Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.
Hierarchical Feedback Control

The essence of hierarchical reliable multicasting

a) Each local coordinator forwards the message to its children

b) A local coordinator handles retransmission requests

Atomic Multicast

- process group G can change
- all members of G agree on the current group membership.
- message is delivered only to the non-faulty members of G

Virtually synchronous multicast