Fault Tolerance Basic Concepts

- Being fault tolerant is strongly related to what are called dependable systems
- Dependability implies the following:
  1. Availability
  2. Reliability
  3. Safety
  4. Maintainability

### 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>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>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>

Figure 8-1. Different types of failures.

### Failure Masking by Redundancy

![Diagram](https://via.placeholder.com/150)

Figure 8-2. Triple modular redundancy.
**Flat Groups versus Hierarchical Groups**

![Diagram](image)

Figure 8-3. (a) Communication in a flat group. (b) Communication in a simple hierarchical group.

---

**Agreement in Faulty Systems (1)**

Possible cases:
1. Synchronous versus asynchronous systems.
2. Communication delay is bounded or not.
3. Message delivery is ordered or not.
4. Message transmission is done through unicasting or multicasting.

---

**Agreement in Faulty Systems (2)**

![Diagram](image)

Figure 8-4. Circumstances under which distributed agreement can be reached.

---

**Agreement in Faulty Systems (3)**

![Diagram](image)

Figure 8-5. The Byzantine agreement problem for three nonfaulty and one faulty process. (a) Each process sends their value to the others.
RPC Semantics in the Presence of Failures

Five different classes of failures that can occur in RPC systems:
1. The client is unable to locate the server.
2. The request message from the client to the server is lost.
3. The server crashes after receiving a request.
4. The reply message from the server to the client is lost.
5. The client crashes after sending a request.
Server Crashes (2)

Three events that can happen at the server:
- Send the completion message (M),
- Print the text (P),
- Crash (C).

Server Crashes (3)

These events can occur in six different orderings:
1. M → P → C: A crash occurs after sending the completion message and printing the text.
2. M → C (→ P): A crash happens after sending the completion message, but before the text could be printed.
3. P → M → C: A crash occurs after sending the completion message and printing the text.
4. P → C (→ M): The text printed, after which a crash occurs before the completion message could be sent.
5. C (→ P → M): A crash happens before the server could do anything.
6. C (→ M → P): A crash happens before the server could do anything.

Server Crashes (4)

<table>
<thead>
<tr>
<th>Client Release strategy</th>
<th>Strategy M → P</th>
<th>Strategy P → M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPC</td>
<td>MC(P)</td>
</tr>
<tr>
<td>Always</td>
<td>DUP</td>
<td>OK</td>
</tr>
<tr>
<td>Never</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Only when ACKed</td>
<td>DUP</td>
<td>OK</td>
</tr>
<tr>
<td>Only when not ACKed</td>
<td>OK</td>
<td>OK</td>
</tr>
</tbody>
</table>

OK = Text is printed once
DUP = Text is printed twice
ZERO = Text is not printed at all

Basic Reliable-Multicasting Schemes

Figure 8-8. Different combinations of client and server strategies in the presence of server crashes.

Figure 8-9. A simple solution to reliable multicasting when all receivers are known and are assumed not to fail. (a) Message transmission. (b) Reporting feedback.
Nonhierarchical Feedback Control

Figure 8-10. Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.

Hierarchical Feedback Control

Figure 8-11. The essence of hierarchical reliable multicasting. Each local coordinator forwards the message to its children and later handles retransmission requests.

Virtual Synchrony (1)

Figure 8-12. The logical organization of a distributed system to distinguish between message receipt and message delivery.

Virtual Synchrony (2)

Figure 8-13. The principle of virtual synchronous multicast.
Message Ordering (1)

Four different orderings are distinguished:
- Unordered multicasts
- FIFO-ordered multicasts
- Causally-ordered multicasts
- Totally-ordered multicasts

Message Ordering (2)

<table>
<thead>
<tr>
<th>Process P1</th>
<th>Process P2</th>
<th>Process P3</th>
<th>Process P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>sends m1</td>
<td>receives m1</td>
<td>receives m3</td>
<td>sends m3</td>
</tr>
<tr>
<td>sends m2</td>
<td>receives m3</td>
<td>receives m1</td>
<td>sends m4</td>
</tr>
<tr>
<td></td>
<td>receives m2</td>
<td>receives m2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>receives m4</td>
<td>receives m4</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8-14. Three communicating processes in the same group. The ordering of events per process is shown along the vertical axis.

Message Ordering (3)

<table>
<thead>
<tr>
<th>Process P1</th>
<th>Process P2</th>
<th>Process P3</th>
<th>Process P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>sends m1</td>
<td>receives m1</td>
<td>receives m3</td>
<td>sends m3</td>
</tr>
<tr>
<td>sends m2</td>
<td>receives m3</td>
<td>receives m1</td>
<td>sends m4</td>
</tr>
<tr>
<td></td>
<td>receives m2</td>
<td>receives m2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>receives m4</td>
<td>receives m4</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8-15. Four processes in the same group with two different senders, and a possible delivery order of messages under FIFO-ordered multicasting

Implementing Virtual Synchrony (1)

<table>
<thead>
<tr>
<th>Multicast</th>
<th>Basic Message Ordering</th>
<th>Total-Ordered Delivery?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable multicast</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>FIFO multicast</td>
<td>FIFO-ordered delivery</td>
<td>No</td>
</tr>
<tr>
<td>Causal multicast</td>
<td>Causal-ordered delivery</td>
<td>No</td>
</tr>
<tr>
<td>Atomic multicast</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>FIFO atomic multicast</td>
<td>FIFO-ordered delivery</td>
<td>Yes</td>
</tr>
<tr>
<td>Causal atomic multicast</td>
<td>Causal-ordered delivery</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 8-16. Six different versions of virtually synchronous reliable multicasting.
Implementing Virtual Synchrony (2)

Figure 8-17. (a) Process 4 notices that process 7 has crashed and sends a view change.

Implementing Virtual Synchrony (3)

Figure 8-17. (b) Process 6 sends out all its unstable messages, followed by a flush message.

Implementing Virtual Synchrony (4)

Figure 8-17. (c) Process 6 installs the new view when it has received a flush message from everyone else.

Two-Phase Commit (1)

Figure 8-18. (a) The finite state machine for the coordinator in 2PC. (b) The finite state machine for a participant.
Two-Phase Commit (2)

<table>
<thead>
<tr>
<th>State of Q</th>
<th>Action by P</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMIT</td>
<td>Make transition to COMMIT</td>
</tr>
<tr>
<td>ABORT</td>
<td>Make transition to ABORT</td>
</tr>
<tr>
<td>INIT</td>
<td>Make transition to ABORT</td>
</tr>
<tr>
<td>READY</td>
<td>Contact another participant</td>
</tr>
</tbody>
</table>

Figure 8.19. Actions taken by a participant P when residing in state READY and having contacted another participant Q.

Two-Phase Commit (3)

Actions by coordinator:
write START_2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
  wait for any incoming vote;
  if timeout {
    write GLOBAL_ABORT to local log;
    multicast GLOBAL_ABORT to all participants;
    exit;
  } record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT {
  write GLOBAL_COMMIT to local log;
  multicast GLOBAL_COMMIT to all participants;
} else {
  write GLOBAL_ABORT to local log;
  multicast GLOBAL_ABORT to all participants;
}

Two-Phase Commit (5)

actions by participant:
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
  write VOTE_ABORT to local log;
  exit;
} if participant votes COMMIT {
  write VOTE_COMMIT to local log;
  send VOTE_COMMIT to coordinator;
  wait for DECISION from coordinator;
  if timeout {
    multicast DECISION_REQUEST to other participants;
    wait until DECISION is received; /* remain blocked */
    write DECISION to local log;
  } if DECISION == GLOBAL_COMMIT
    write GLOBAL_COMMIT to local log;
  else if DECISION == GLOBAL_ABORT
    write GLOBAL_ABORT to local log;
} else {
  write VOTE_ABORT to local log;
  send VOTE_ABORT to coordinator;
}

Figure 8.21. (a) The steps taken by a participant process in 2PC.
Three-Phase Commit (1)

The states of the coordinator and each participant satisfy the following two conditions:

1. There is no single state from which it is possible to make a transition directly to either a COMMIT or an ABORT state.
2. There is no state in which it is not possible to make a final decision, and from which a transition to a COMMIT state can be made.

Three-Phase Commit (2)

![Finite state machine for the coordinator in 3PC](image)

![Finite state machine for a participant](image)

Recovery – Stable Storage

![Stable storage](image)

![Crash after drive 1 is updated](image)

![Bad spot](image)
**Checkpointing**

- Initial state
- Recovery line
- Checkpoint
- Message sent from P2 to P1
- Inconsistent collection of checkpoints
- Failure

**Figure 8-24. A recovery line.**

**Independent Checkpointing**

- Initial state
- Checkpoint
- Failure

**Figure 8-25. The domino effect.**

**Characterizing Message-Logging Schemes**

- Q crashes and recovers
- m1
- m2
- m3
- m2 is never replayed, so neither will m3
- m3

**Figure 8-26. Incorrect replay of messages after recovery, leading to an orphan process.**