# Table of Contents

- Introduction and Motivation
- Theoretical Foundations
- Distributed Programming Languages
- Distributed Operating Systems
- Distributed Communication
- Distributed Data Management
- Reliability
- Applications
- Conclusions
- Appendix
Type of Faults

- **Types of faults:**
  - Hardware faults
  - Software faults
  - Communication faults
  - Timing faults

- Schneider’s classification:
  - Omission failure
  - Failstop failure (detectable)
  - Crash failure (undetectable)
  - Crash and link failure
  - Byzantine failure
Redundancy

- **Hardware redundancy**: extra PE's, I/O's
- **Software redundancy**: extra version of software modules
- **Information redundancy**: error detecting code
- **Time redundancy**: additional time used to perform a function
Fault Handling Methods

- Active replication
- Passive replication
- Semi-active replication
Building Blocks of Fault-Tolerant Design

- **Stable storage** is a logical abstraction for a special storage that can survive system failure.

- **Fail-stop processors** do not perform any incorrect action and simply cease to function.

- An **atomic action** is a set of operations which are executed indivisibly by hardware. That is, either operations are completed successfully and fully or the state of the system remains unchanged (operations are not executed at all).
Domino Effect

- Storage of checkpoints
- Checkpointing methods

An example of domino effect.
Focus 21: Byzantine Faults

Several divisions of the Byzantine army camp outside an enemy city. Each division commanded by its own general. Generals from different divisions communicate only through messengers. Some of the generals may be traitors. After observing the enemy, the generals must decide upon a common battle plan.
Two Requirements

- All loyal generals decide upon *the same plan* of action
- A small number of traitors *cannot* cause the loyal generals to *adopt a wrong plan*

Note
- Loyal generals may start with different decisions, but end up the same decision.
- The final decision must come from at least one loyal general’s initial decision.
Theoretical result
- Consensus is reachable if \( n \geq 3m + 1 \), where \( n \) is the total number of generals and \( m \) is the number of traitors.

(m+1)-round of consensus algorithm
- At the first round, each node, including traitors, broadcasts its initial decision
- At the (i+1)th round, each node broadcasts all messages received at ith round
**Agreement Protocol**

<table>
<thead>
<tr>
<th>P1*</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First round:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-, v₂, v₃, v₄)</td>
<td>(v₁², -, v₃, v₄)</td>
<td>(v₁¹, v₂, -, v₄)</td>
<td>(v₁³, v₂, v₃, -)</td>
</tr>
<tr>
<td><strong>Second round:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(v₁², -, v₃, v₄)</td>
<td>(v₁¹, v₂, -, v₄)</td>
<td>(v₁¹, -, v₃, v₄)</td>
<td>(v₁², -, v₃, v₄)</td>
</tr>
<tr>
<td>(v₁¹, v₂, -, v₄)</td>
<td>(v₁³, v₂, v₃, -)</td>
<td>(v₁³, v₂, v₃, -)</td>
<td>(v₁¹, v₂, -, v₄)</td>
</tr>
<tr>
<td>(v₁³, v₂, v₃, -)</td>
<td>(-, v₂⁴, v₃⁴, v₄⁴)</td>
<td>(-, v₂⁵, v₃⁵, v₄⁵)</td>
<td>(-, v₂⁶, v₃⁶, v₄⁶)</td>
</tr>
<tr>
<td>(d₁, d₂, d₃, d₄)</td>
<td>(v₁⁷, v₂, v₃, v₄)</td>
<td>(v₁⁷, v₂, v₃, v₄)</td>
<td>(v₁⁷, v₂, v₃, v₄)</td>
</tr>
</tbody>
</table>

An algorithm for reaching agreement.
No-Agreement Among Three Processes

Cases leading to failure of the Byzantine agreement.
Extended Agreement Protocols

- Boolean values or arbitrary real values for the decisions.
- Unauthenticated or authenticated messages.
- Synchronous or asynchronous.
- Completely connected network or partially connected networks.
- Deterministic or randomized.
- Byzantine faults or fail-stop faults.
- Non-totally decentralized control system and, in particular, hierarchical control systems.
Reliable Communication

- Acknowledgement: acknowledge the receipt of each packet.
- TCP: transport protocol for reliable point-to-point comm.
- Negative acknowledgement
  - Signal for a missing packet.
  - Pros: better scalability (without positive acknowledgement).
  - Cons: sender is forced to keep each packet in the buffer forever.
Reliable Group Communication

- **Feedback suppression**: multicast or broadcast each positive (or negative) acknowledge.
- Combination of positive and negative acknowledgements
Example 26: Combination of Positive and Negative Acknowledgements in Broadcasting.

Let A be a packet and a (a) the positive (negative) acknowledgement for A.

\[ A, B_a, C_b, D_b, E_c, F_{cd}, C_b, G_{def} \]

1. Message A is sent first, acknowledged by the sender of B, which is in turn acknowledged by the senders of C and D.

2. The sender of E acknowledges C and the sender of F acknowledges the receipt of D but a negative acknowledgment of C.

3. Some node (not necessarily the original sender) retransmits C.

4. The sender of G acknowledges both E and F but sends a negative acknowledgment of D (after receiving F).
Different Types of Reliable Multicasting

- Reliable multicast: no message ordering
- FIFO multicast: FIFO-ordered delivery
- Causal multicast: causal-ordered delivery
- Atomic multicast: reliable multicast + total-ordered delivery
- FIFO atomic multicast: FIFO multicast + total-ordered delivery
- Causal atomic multicast: Causal multicast + total-ordered delivery
Focus 22: Total-Ordered Multicasting

- **Total-ordered multicasting**
  - Each transfer order (message) can be assigned a global sequence number.
  - There exists a global sequence.

- **Sequencer**
  - The sender sends message to a sequencer
  - The sequencer allocates a global sequence number to the message.
  - The message is delivered by every destination based on the order.
Implementations of Sequencer

- Privilege-based (token circulated among the senders)
- Fixed sequencer (a fixed third party)
- Moving sequencer (token circulated among the third-party nodes)
Multicast with Total Order

Multicast with total order

seq(m_1) < seq(m_2)

Multicast without total order

Neither seq(m_1) < seq(m_2) nor seq(m_2) < seq(m_1)
Focus 23: Birman’s Virtual Synchrony

- Virtual synchrony: reliable multicast with a special property.
- View: a multicast group.
- View change: (a) a new process joins, (b) a process leaves, and (c) a process crashes.
- Each view change is multicast to members in the group.
- Special property: each view change acts as a barrier across which no multicast can pass. (Application: distributed debugging.)
Focus 23 (Cont’d)

Virtual synchrony.
Implementing a Virtual Synchronous Reliable Multicast

- Message received versus message delivered.
- If message $m$ has been delivered to all members in the group, $m$ is called **stable**.
- Point-to-point communication is reliable (TCP).
- Sender may crash before completing the multicasting. (Some members received the message but others did not.)

Message receipt versus message delivery.
Implementing a Virtual Synchronous Reliable Multicast (Cont’d)

- At group view $G_i$, a view changed is multicast.
- When a process receives the view-change message for $G_{i+1}$, it multicasts to $G_{i+1}$ a copy of unstable messages for $G_i$ followed by a flush message.
- A process installs the new view $G_{i+1}$ when it has received a flush message from everyone else.

Virtual synchrony.
Reliable Process

Active model

Passive model
Exercise 7

1. Use a practical example to illustrate the differences among faults, errors, and failures.

2. Illustrate the correctness of the agreement protocol for authenticated messages using a system of four processes with two faulty processes. You need to consider the following two cases:
   - The sender is healthy and two receivers are faulty (the remaining receiver is healthy).
   - The sender is faulty and one receiver is faulty (the remaining receivers are healthy).

3. In Byzantine agreement protocol $k + 1$ rounds of message exchanges are needed to tolerate $k$ faults. The number of processes $n$ is at least $3k + 1$. Assume $P_1$ and $P_2$ are faulty in a system of $n = 7$ processes.
   - (a) Show the messages $P_3$ receives in first, second, and third round.
   - (b) Demonstrate the correctness of the protocol by showing the final result vector (after a majority voting) for $P_3$.
   - (c) Briefly show that result vectors for other non-faulty processes are the same.