Synchronization

Consistency
Overview

1. Distributed Synchronization
2. Consistency
Overview

1. Distributed Synchronization
   - Centralized lock server
   - Lamport's algorithm
   - Leader election
   - Deadlocks

2. Consistency
First approach:
- Centralized lock server
- Maintains queue of lock requests

Assessment
- Lock server is a **Performance bottleneck**
- Single point of failure
- Failure of client holding lock is **critical**

[Veríssimo and Rodrigues 2001]
Multipeer Mutual Exclusion

Next approach:
- **Replicate** state of central server
- All processes become clients *and* servers

Assessment
- Interleaving of LOCK-requests may lead to
  - Inconsistent states
  - **Deadlock**
- One solution:
  - **Totally ordered** multicast

[Veríssimo and Rodrigues 2001]
**Total Order + Mutual Exclusion**

### Prerequisites

- FIFO channels
- Logical clocks

-_msgs are time-stamped by logical clocks_

- Logical clocks advanced whenever msgs sent or received
Lamport's Algorithm I

When a process needs resource:
  ▶ Send **LOCK-request** to all other processes

When a process releases resource:
  ▶ Send **UNLOCK-msg** to all other processes

When process receives LOCK request:
  ▶ Insert request in waiting queue
  ▶ Queue ordered according to request time stamps
  ▶ Send **ACK-msg** to all other processes
Lamport's Algorithm II

As soon as ACK is received from all other processes:
- Mark request as **stable**

A process p is granted resource if
- Received UNLOCK-request **and**
- Request of p is
  - At head of request queue **and**
  - Marked as stable

**Optimization:**
- Process holding resource defers ACK-msgs until release
- Can thus avoid UNLOCK-msgs
Leader Election

- **Advantage of centralized server:**
  - Simplicity

- **Disadvantage:**
  - Server crashes $\Rightarrow$ System becomes unavailable

- **Distributed leader algorithm:**
  - Elect server at run-time
  - System survives failures of server

- Similar to mutual exclusion:
  - Mutual-exclusion alg “elects” the process that is granted resource
Leader Election Algorithm

**Approach**
- Every process requests lock
- First process granted access becomes leader

**Optimizations**
- As soon as leader is elected:
  - Processes can abort lock request
- If LOCK-request is received from other processor p:
  - May support p's election
Bully Algorithm

- Processes concurrently send *LOCK-request*
- All requests have same time stamp
- Lexically first active process is granted lock
**Observation:**

- Only lexically higher ranking processes can be elected

![Diagram with arrows and nodes representing process interactions]

[Veríssimo and Rodrigues 2001]

**However:**

- Still need synchronous msgs to deal with faulted nodes!
Overview

1. Distributed Synchronization
2. Consistency
   - Snapshot protocol by Chandy and Lamport
   - Distributed consensus
   - Atomic broadcasts
   - Agreement on membership
   - View synchrony
Consistency

System state is **consistent**:  
- No integrity constraint imposed on system specification is violated

**Examples**:  
- Absence of deadlock  
- Token is still rotating

To assess consistency:  
- Must take *snapshot* of global system state

**Global system state**:  
- Set of local states at given execution points
**Ad-Hoc Snapshots**

- **Cut** in the space-time diagram:
  - Segment intersecting the timelines of all processes

---

[Verissimo and Rodrigues 2001]
Example: Sum of Accounts

Cut 1: (weakly) consistent
Cut 2: inconsistent

[Veríssimo and Rodrigues 2001]
Inconsistent Cut

Cut is consistent $\iff$ Events in cut are concurrent

[Verissimo and Rodrigues 2001]
Snapshot Protocols

Brute force approach for obtaining consistent cut:
- **Stop** whole system
- Record system state
- **Continue** system

This not untypical for commercial systems

More sophisticated approach:
- **Snapshot protocol**
Protocol by Chandy and Lamport I

**Assumptions:**
- FIFO channels
- Special control message \textit{MARKER}

Global snapshot includes
- Local state of each process
- State of each channel

Processes capture states of their incoming channels
Protocol by Chandy and Lamport II

1. Process P initiates snapshot:
   - P saves its state
   - P sends MARKER through all outgoing channels
2. If P receives MARKER on channel C:
   - P records state of C as those msgs received after snapshot initiation
3. If P has received MARKER on all channels:
   - Snapshot is complete
Protocol by Chandy and Lamport III

4. If process $Q \neq P$ receives MARKER for first time, on channel $C_Q$:
   - $Q$ saves state of $C_Q$ as empty
   - $Q$ saves own state
   - $Q$ sends MARKER through all channels

5. If process $Q \neq P$ receives MARKER on channel $C \neq C_Q$:
   - $Q$ records state of $C$ as those msgs received after snapshot initiation
**Distributed Consensus**

**Consensus:**
- Set of processes agrees on a *single value*
- Value depends on *initial values* of each participant
- Excludes trivial solution (constant value)

**Example**

**Scenario:**
- Items arrive on conveyor belt
- Items must be serviced by one of several machines

**Need for consensus:**
- *Which machine picks which item?*
Example: Dispatch Items to Machines

- **Machine is free when item arrives:**
  - Machine proposes itself to service item

- **Machine is busy when item arrives:**
  - Machine waits to become free

- **Machine is busy and receives proposal:**
  - Machine supports other proposal
  - Machine does not change its mind later

- **Once proposals have been collected from everybody:**
  - Deterministic voting function
  - E.g., lexically first proposal
The problem (again):

How to make this fault tolerant?
**Group Membership Service**

- **Group membership**
  - Create groups
  - Join/Leave groups

- **Group view**
  - Information about currently active processes
  - **Accuracy:**
    - Provided information reflects physical scenario
  - **Consistency:**
    - Provided information is consistent at all processes
Agreement on Membership

When there is membership change:
- Want to deliver new view to all participants

**Assumption**
- Views are totally ordered
- View $V^{i+1}$ delivered after $V^i$ has been delivered to all

**Approach**
- When new process wants to join:
  - Send msg to members of $V^i$
  - Members make proposals for $V^{i+1}$
  - Perform consensus protocol on proposals
Example: Load Balancing

- Tasks pick load share based on view
- View $V^0 = [q, s]$
Atomic Broadcast

**Atomic broadcast:**
- Reliability + total ordering

Atomicity wrt faults

Atomic broadcast as *consensus problem*:
- Must agree on whether msg was delivered
- Must agree on ordering wrt other msgs

Simple approach:
- Processes buffer all received msgs
- When view changes:
  - Reach consensus on what msgs to deliver, in which order

Improvement:
- Periodic triggering of consensus
View Synchrony

Message $m$ is delivered to $p$ in view $V^i$:
- $m$ is delivered after delivery of $V^i$
- $m$ is delivered before delivery of $V^{i+1}$

View synchronous ordering:
- If $m$ is delivered to $p$ in view $V^i$, then
  - $m$ is delivered to all $q \in V^i$ in view $V^i$ as well

This also called virtual synchrony

May again use consensus protocol
**View-Synchronous View Change**

When installing $V^{i+1}$:
- Reach agreement on which msgs to deliver in $V^i$
- May lead to delay in delivery of new view

[Veríssimo and Rodrigues 2001]
Summary – Distributed Synchronization

- Distributed synchronization even harder than centralized synchronization
  - Must obtain consistent snapshot of system
- Centralized lock server
  - Simple
  - Single point of failure
- Distributed lock server
  - Replicates system state
  - Still not fault tolerant
- Leader election
  - Make sure only (initially) active node becomes lock server
Summary – Consistency

- **System state is consistent if no integrity constraint imposed on system specification is violated.**
- To assess consistency, we must take a snapshot or cut of global system state.
- If we take an ad-hoc snapshot of a system, we risk to obtain an inconsistent cut.
- The snapshot protocol by Chandy and Lamport obtains a consistent cut using MARKER msgs.
Summary – Consensus

The system-wide agreement on some value constitutes the distributed consensus problem

**Examples:**
- Agreement on membership
- Atomic broadcasts
- View synchrony
To Go Further

**Overview of Synchronization:**
- [Veríssimo and Rodrigues 2001], Chapter 2

**Lamport's Ordering Algorithm:**