Reliable One-to-One Communication

• How to handle communication failures?
  – Use reliable transport protocols (TCP) or handle at the application layer
• RPC semantics in the presence of failures (discussed in lecture 4)
  – Client unable to locate server
  – Lost request messages
  – Lost reply messages
  – Server crashes after receiving request
  – Client crashes after sending request

Reliable One-to-Many Communication

• Reliable multicast
  – Lost messages => need to retransmit
• ACK-based schemes
  – ACK explosion problem: Sender can become bottleneck
• NACK-based schemes
  – Use only NACKs
    • How long should sender keep a message in buffer?
    • NACK explosion can occur
  – Feedback suppression
    • When a receiver missed a message, it multicasts a NACK to the rest of the group
    • The first NACK suppresses the others
Atomic Multicast

- Atomic multicast: a guarantee that a message is delivered to either all processes or to none at all
  - Replicated database example
- Problem: how to handle process crashes?
- Solution: group view
  - Each multicast message is uniquely associated with a group of processes
    - View of the process group when message was sent
    - All processes in the group should have the same view
- Virtually synchronous reliable multicast
  - A message multicast to group view \( G \) is delivered to each nonfaulty process in \( G \).
  - If the sender crashes during the multicast, the message may either be delivered to all remaining processes, or ignored by each of them.

Implementing Virtual Synchrony

(a) Process 4 notices that process 7 has crashed, sends a view change.
(b) Process 6 sends out all its unstable messages, followed by a flush message.
(c) Process 6 installs the new view when it has received a flush message from everyone else.
Distributed Commit

• Atomic multicast is an example of a more general problem, known as distributed commit
  – All processes in a group perform an operation or not at all
  – Examples:
    • Reliable multicast: operation = delivery of a message
    • Distributed transactions: operation = commit of a transaction

• Problem of distributed commit
  – All or nothing operations in a group of processes

• Possible approaches
  – Two-phase commit (2PC)
  – Three-phase commit (3PC)

Two-Phase Commit

• Coordinator process coordinates the operation

• Two phases
  – Voting phase: processes vote on whether to commit
  – Decision phase: actually commit or abort

(a) The finite state machine for the coordinator. (b) The finite state machine for a participant.
Recovering from a Crash

- Timeout is used when a process is waiting for a message from another process.
- Upon timeout:
  - The coordinator in WAIT state will send Global_Abort to all participants.
  - A participant in INIT state will abort the transaction and send a Vote_Abort message to the coordinator.
  - A participant in Ready state will contact another process Q and examine Q’s state.
    - If all participants are in Ready state, they will block until the coordinator recovers.

<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>

Implementing 2PC

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;
}

Outline of the steps taken by the coordinator in a two-phase commit protocol.
Implementing 2PC

Three-Phase Commit

- A participant in Ready or Precommit state will contact another process Q and examine Q’s state
  - If all participants are in Ready state, the transaction should be aborted
  - If all participants are in Precommit state, the transaction should be committed.