Last Time

- How do a group of processes communicate?
- Multicast
  - One-to-many: “Local” broadcast within a group \( g \) of processes
- What are the issues?
  - Processes crash (we assume crash-stop)
  - Messages get delayed
- B-multicast
- R-Multicast
  - Properties: integrity, agreement, validity
- Ordering
  - Why do we care about ordering?

Recap: Ordering

- Totally ordered messages \( T_1 \) and \( T_2 \)
- FIFO-related messages \( F_1 \) and \( F_2 \)
- Causally related messages \( C_1 \) and \( C_2 \)
- Total ordering does not imply causal ordering.
- Causal ordering implies FIFO ordering
- Causal ordering does not imply total ordering.
- Hybrid mode: causal-total ordering, FIFO-total ordering.

Example: FIFO Multicast

- Physical Time
- Buffer
- Sequence Vector (do NOT be confused with vector timestamps)
- Total ordering does not imply causal ordering.
- Causal ordering implies FIFO ordering
- Causal ordering does not imply total ordering.
- Hybrid mode: causal-total ordering, FIFO-total ordering.

Totally Ordered Multicast

- Using a sequencer
  - One dedicated “sequencer” that orders all messages
  - Everyone else follows.
- ISIS system
  - Similar to having a sequencer, but the responsibility is distributed to each sender.

Total Ordering Using a Sequerencer

1. Algorithm for group member \( p \)
   - On initialization: \( s_p = 0 \)
   - To TD-multicast message \( m \) to group \( g \)
     - \( s_{\text{sender}} \) \( \leq \) \( s_{\text{receiver}} \) \( + \) unique message id
     - On TD-deliver \( m \) \( \langle \text{to} \rangle \) \( \text{with} \) \( \approx \text{to} \) \( \text{group} \( g \)
       - Use \( s_p \) to hold-back queue
     - On TD-deliver \( m \) \( \langle \text{to} \rangle \) \( \text{with} \) \( g \) \( \approx \text{group} \( g \)
       - \( s_p \) \( \approx \) \( s \) \( + \) \( 1 \)
       - TD-deliver \( m \) \( \langle \text{to} \rangle \) \( \text{after delivering it from the hold-back queue} \)
       - \( s_p \) \( = \) \( s \) \( + \) \( 1 \)

2. Algorithm for sequencer of \( g \)
   - On initialization: \( s_p = 0 \)
   - On TD-multicast \( m \) \( \langle \text{to} \rangle \) \( \text{with} \) \( g \) \( \approx \text{group} \( g \)
     - \( s_{\text{sender}} \) \( < \text{output} \) \( \langle \text{to} \rangle \) \( \approx g \)
     - \( s_{\text{sender}} \) \( = \) \( s \) \( + \) \( 1 \)
ISIS algorithm for total ordering

• Sender multicasts message to everyone
• Reply with proposed priority (sequence no.)
  – Larger than all observed agreed priorities
  – Larger than any previously proposed (by self) priority
• Store message in priority queue
  – Ordered by priority (proposed or agreed)
  – Mark message as undeliverable
• Sender chooses agreed priority, re-multicasts message with agreed priority
  – Maximum of all proposed priorities
• Upon receiving agreed (final) priority
  – Mark message as deliverable
  – Deliver any deliverable messages at the front of priority queue

Notice any (small) issue?

Problematic Scenario

• Two processes P1 & P2 at their initial state.
• P1 sends M1 & P2 sends M2.
• P1 receives M1 (its own) and proposes 1. P2 does the same for M2.
• P2 receives M1 (P1’s message) and proposes 2. P1 does the same for M2.
• P1 picks 2 for M1 & P2 also picks 2 for M2.
• Same sequence number for two different msgs.
• How do you want to solve this?

Example: ISIS algorithm

Proof of Total Order

• For a message m, consider the first process p that delivers m.
  • At p, when message m is at head of priority queue and has been marked deliverable, let m be another message that has not yet been delivered (i.e., is on the same queue or has not been seen yet by p)
  \[
  \text{finalpriority}(m) \geq \text{proposedpriority}(m) > \text{finalpriority}(m')
  \]
  • Suppose there is some other process p' that delivers m before it delivers m. Then at p',
  \[
  \text{finalpriority}(m) \geq \text{proposedpriority}(m) > \text{finalpriority}(m')
  \]
  • a contradiction!
Causally Ordered Multicast

- Each process keeps a vector clock.
  - Each counter represents the number of messages received from each of the other processes.
- When multicasting a message, the sender process increments its own counter and attaches its vector clock.
- Upon receiving a multicast message, the receiver process waits until it can preserve causal ordering:
  - It has delivered all the messages from the sender.
  - It has delivered all the messages that the sender had delivered before the multicast message.

Causal Ordering

Algorithm for group member $p_i (i = 1, 2, ..., N)$

On initialization,
$$V_i^{(j)}[j] = \mathbf{0} \forall j = 1, 2, ..., N;$$
the number of group-$g$ messages from process $j$ that have been seen at process $i$ so far.

To CO-multicast message $m$ to group $g$ $V_i^{(j)}[i] := V_i^{(j)}[i] + 1$;

B-multicast($g, < V_i^{(j)}, m>$);

On $B$-deliver($V_i^{(j)}, m$) from $p_j$ with $g = \text{group}(m)$

place $< V_i^{(j)}, m >$ in hold-back queue;

wait until $V_i^{(j)}[j] = V^{(j)}[j] + 1$ and $V_i^{(j)}[k] \leq V^{(j)}[k] (k \neq j)$;

CO-deliver $m$; // after removing it from the hold-back queue

Example: Causal Ordering Multicast

Physical Time

Summary

- Two multicast algorithms for total ordering
  - Sequencer
  - ISIS
- Multicast for causal ordering
  - Uses vector timestamps

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).