CSE 486/586 Distributed Systems
Reliable Multicast --- 2

Steve Ko
Computer Sciences and Engineering
University at Buffalo

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 T1 and T2
• FIFO-related messages F1 and F2
• Causally related messages C1 and C2
• 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

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 Sequencer

1. Algorithm for group member p
   On initialization: t_p = 0.
   To TD-multicast message m to group g
   B-multicasting g. (sequence(m), t_g) — i.e., i ≥ t_g
   On B-deliver(m, i) with g = group(p)
   Phase on, i > 0 to hold-back queue.
   On B-deliver(m, i) with g = group(p)
   wait until on, i > 0 to hold-back queue and t_g ≤ i.
   TD-action: m, i (after delivering it from the hold-back queue)
   t_g = i + 1;

2. Algorithm for sequence of g
   On initialization: t_g = 0
   On B-deliver(m, i) with g = group(p)
   B-multicasting g. (sequence(m), i) — i ≥ t_g
   t_g := i + 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?

**CSE 486/586 Administrivia**

- Please start PA2 if you haven’t.
- AWS codes will be distributed on UBLearns.
  - Will post setup instructions.
- Come talk to me!

**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**

Showing the process id only when necessary

**Proof of Total Order**

- For a message $m_i$ consider the first process $p$ that delivers $m_i$.
- At $p$, when message $m_i$ is at head of priority queue and has been marked deliverable, let $m_j$ be another message that has not yet been delivered (i.e., is on the same queue or has not been seen yet by $p$)
  - Since queue ordered by increasing priority
  - Final priority of $m_j$ is greater than proposed priority of $m_i$
  - Due to "max" operation at sender
- Suppose there is some other process $p'$ that delivers $m_j$ before it delivers $m_i$. Then at $p'$
  - Final priority of $m_j$ is greater than proposed priority of $m_i$
  - Since queue ordered by increasing priority
- 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 \( p_i \) (\( i = 1, 2, \ldots, N \))

On initialization

\[
V^0_i[j] = 0 \quad (j = 1, 2, \ldots, N);
\]

The number of group-g messages from process \( j \) that have been seen at process \( i \) so far.

To C0-multicast message \( m \) to group \( g \)

\[
V^0_i[j] := V^0_i[j] + 1;
\]

B-multicast(\( g, V^0_i, m \)).

On B-deliver(\( V^0_i, m \)) from \( p_j \) with \( g = \text{group}(m) \)

place \( V^0_i[j] \) in hold-back queue;

wait until \( V^0_i[j] = V^0_i[j] + 1 \) and \( V^0_i[k] \leq V^0_i[k] \) (\( k \neq j \));

C0-deliver \( m \); // after removing it from the hold-back queue

\[
V^0_i[j] := V^0_i[j] + 1;
\]

Example: Causal Ordering Multicast

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).