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

Steve Ko
Computer Sciences and Engineering
University at Buffalo

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 model: causal-total ordering, FIFO-total ordering.

Example: FIFO Multicast

Physical Time

Accept: 1 = 0 + 1
Accept: 2 = 1 + 1
Reject: 1 < 1 + 1

Sequence Vector

Accept: Buffer 2 = 0 + 1
Accept: Buffer 2 = 1 + 1
Accept: Buffer 2 = 1 + 1

 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: \( i_p = 0 \).
   
   To TO-multicast message \( m \) to group \( g \): \( B\)-multicast \( m \).
   
   On \( B\)-deliver \( \langle n, i \rangle \) with \( g = \text{group}(p) \):
   
   Place \( \langle n, i \rangle \) in hold-back queue.
   
   On \( B\)-deliver \( \langle little \( c \), 0 \rangle \) with \( g = \text{group}(m) \):
   
   wait until \( \langle n, i \rangle \) in hold-back queue and \( S = i + 1 \).
   
   TO-deliver \( m \) after deleting it from the hold-back queue.
   
   \( i_p = i + 1 \).

2. Algorithm for sequencer of \( g \)
   
   On initialization: \( i_0 = 0 \).
   
   On \( B\)-deliver \( \langle n, i \rangle \) with \( g = \text{group}(p) \):
   
   \( B\)-multicast \( \langle \text{"little"}, i, i + 1 \rangle \).
   
   \( i_0 = i + 1 \).

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?
**ISIS algorithm for total ordering**

- No central sequencer
  - Achieves decentralization
  - Distributed doesn’t mean decentralized.
- Every sender acts as a sequencer.
- Since there is no single sequencer that determines a number, it requires agreement on sequence numbers.
  - Agreement is very important for decentralization.
- Thus, each sender does not pick a sequence number alone.
  - Otherwise, two different senders can pick the same number.
- Each sender receives proposals for a sequence number every time.
  - Among the proposals, the sender picks a number.
- How to propose a number?
  - Need a way to guarantee that a higher number is picked among all numbers assigned as sequence numbers already or potentially assigned as sequence numbers.
  - Each message receiver pick a number that is the highest among all the numbers that it has ever seen, i.e., all previous proposals and actual message sequence numbers.
- How to pick a sequence number out of all proposals?
  - Among all proposals, pick the highest number.

**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.
  - Reorder the delivery queue based on the priorities.
  - Deliver any deliverable messages at the front of priority queue.

**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?
  - Use process numbers as a tie-breaker.
  - For a proposal, always use the following format: X.Y
    - X is the proposed number and Y is the process id.
  - P1 has proposed 2 for M1 → The proposal for M1 is now 2.1.

**CSE 486/586 Administrivia**

- PA2-B is due on 3/15.
  - Right before Spring break.
- Midterm is on 3/13.
- Come up with a schedule that works.
We don't dictate when events are happening

Example: ISIS algorithm

P1

P2

P3

Example: ISIS algorithm

P1

P2

P3

Example: ISIS algorithm

P1

P2

P3

A

Example: ISIS algorithm

P1

P2

P3

A

Example: ISIS algorithm

P1

P2

P3

A

Example: ISIS algorithm

P1

P2

P3

A

Example: ISIS algorithm

P1

P2

P3

A

Example: ISIS algorithm

We don't dictate when events are happening.

P1: A

P2: C

P3: B

A: 1.1
B: 1.2
C: 2.1

A: 2.2
C: 3.2

A: 2.3
B: 1.3
C: 3.3
We don't dictate when events are happening.
Example: ISIS algorithm

We don't dictate when events are happening.
Example: ISIS algorithm

We don't dictate when events are happening

P1 A
 B
P2 B
P3

C:3.3  A:2.3  B:3.1
B:1.2  A:2.3  C:3.3
A:2.3  B:3.1  C:3.3

CSE 486/586
Example: ISIS algorithm

We don't dictate when events are happening.
Example: ISIS algorithm

We don’t dictate when events are happening

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 have happened before, i.e., messages that the sender had delivered before the multicast message.

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 $m_i$ be another message that has not yet been delivered (i.e., is on the same queue or has not been seen yet by $p$).
  - $\text{finalpriority}(m_i) \geq \text{proposedpriority}(m_i)$
  - $\text{finalpriority}(m_i) > \text{finalpriority}(m_i)$
  - a contradiction!

Due to the max operation at sender

Since queue ordered by increasing priority

Causal Ordering

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

On initialization

\[ V^g_i(j) = \text{initial} \quad 0 \leq j \leq 1, \ldots, Nk \]

To CO-multicast message $m$ to group $g$

\[ V^g_i(i) = V^g_i(i) + 1; \]

\[ B\text{-multicast}(g, <V^g_i, m>); \]

On B-deliver($<V^g_i, m> $) from $p_j$, with $g = \text{group}(m)$

- place $<V^g_i, m>$ in hold-back queue;
- wait until $V^g_j(j) = V^g_i(j) + 1$ and $V^g_i(k) \leq V^g_i(k) (k \neq j)$;
- CO-deliver $m_i$ after removing it from the hold-back queue

\[ V^g_i(j) = V^g_i(j) + 1; \]
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).