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 T₃,
• FIFO-related messages F₁ and F₂
• Causally related messages C₁ and C₃
• 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

<table>
<thead>
<tr>
<th>Time</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Physical Time

Sequence Vector

(Do NOT be confused with vector timestamps)

Accept* = Deliver

Total Ordering Using a Sequencer

1. Algorithm for group member p
On initialization: r_p = 0;
To TD-multicast message m to group g,
B-end(m, x, y, z, w) := (sequence(m), r_p)
On r_d(ℓ, i) with g = group(p)
Place ℓ, i in hold-back queue.
On r_d(ℓ, i) with g = group(p)
wait until ℓ, i in hold-back queue and r_p = r_p;
TD-deliver m, i (after deleting it from the hold-back queue)
r_p = r_p + 1;

2. Algorithm for sequencer
On initialization: r_p = 0;
On r_d(ℓ, i) with g = group(p)
B-end(m, x, y, z, w) := (sequence(m), r_p)
ℓ := ℓ + 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.
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?

CSE 486/586 Administrivia

- PA2-B will be released today.

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)

$$ \text{finalpriority}(m_i) \geq \text{proposedpriority}(m_j) > \text{finalpriority}(m_j) $$

Due to “max” operation at sender

Since queue ordered by increasing priority

- Suppose there is some other process $p'$ that delivers $m_j$ before it delivers $m_i$. Then at $p'$,

$$ \text{finalpriority}(m_j) \geq \text{proposedpriority}(m_i) > \text{finalpriority}(m_i) $$

Due to “max” operation at sender

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 CO-multicast message \( m \) to group \( g \)
\[
V^1_i[j] = V^0_i[j] + 1;
\]
B-multicast(\( g, \langle V^1_i, m \rangle \))

On B-deliver(\( V^1_i, m \)) from \( p_j \) with \( g = \text{group}(m) \)
place \( V^1_i[j] \) in hold-back queue;
wait until \( V^1_i[j] = V^1_i[j+1] \) and \( V^1_i[k] \leq V^1_i[k] \) (\( k \neq j \));
\( \text{CO-deliver} \ m; \) // after removing it from the hold-back queue
\[
V^1_i[j] = V^1_i[j+1];
\]

Example: Causal Ordering Multicast

Physical Time

Acknowledgements

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