CSE 486/586 Distributed Systems

Reliable Multicast --- 2

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Reliable Multicast --- 2

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

<table>
<thead>
<tr>
<th>Time</th>
<th>Physical Time</th>
<th>Sequence Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(1, 0, 0)</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>(2, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(2, 1, 0)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>(2, 2, 0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>Buffer</th>
<th>Accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1 = 0 + 1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2 = 1 + 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2</th>
<th>Buffer</th>
<th>Accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1 = 0 + 1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 = 1 + 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P3</th>
<th>Buffer</th>
<th>Accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1 = 0 + 1</td>
</tr>
</tbody>
</table>

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: \( s_p = 0 \).
   - \( B\)-multicast: \( g \times \text{sequence}(g) \times s_p \times i \geq 0 \).
   - On \( B\)-deliver \( (o, i) \) with \( g \times \text{group} \( (p) \)
     - Send \( o, i \) to hold-back queue.
   - On \( B\)-deliver \( (o, i) = \text{sequence}(g, i, S) \) with \( g \times \text{group}(p) \)
     - Wait until \( o, i \) is in hold-back queue and \( S = s_p \).
     - \( B\)-deliver \( o, i \) (after delivering it from the hold-back queue)
     - \( s_p = S + 1 \).

2. Algorithm for sequencer of \( g \)
   - On initialization: \( s_p = 0 \).
   - On \( B\)-deliver \( (o, i) \) with \( g \times \text{group}(p) \)
     - \( B\)-multicast: \( g \times \text{sequence}(g, i, s_p + 1) \).
     - \( s_p := s_p + 1 \).
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 Adminstrivia

- PA2-B is due on 3/17.
- Midterm is on 3/15.

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 \( p \))
- \( \text{finalpriority}(m'_j) \geq \text{proposedpriority}(m'_j) \)
- \( \text{finalpriority}(m'_j) \) Since queue ordered by increasing priority
- Suppose there is some other process \( p' \) that delivers \( m_j \) before it delivers \( m_i \). Then at \( p' \),
  - \( \text{finalpriority}(m_j) \) \( \geq \) \( \text{proposedpriority}(m_j) \)
  - 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 $p_i$ ($i = 1, 2, ..., N$)

On initialization:
$$v_{i}[j][g] = 0 \
\quad (j = 1, 2, ..., N);$$

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 $g$:
$$v_{i}[i][g] := v_{i}[i][g] + 1;\nB$\text{-multicast}(g, <v_{i}[i][g]>);$$

On $B$-deliver($<v_{i}[i][g]>$ from $p_j$ with $g = \text{group}(m)$):
- place $v_{i}[i][g]$ in hold-back queue;
- wait until $v_{i}[j][g] = v_{i}[i][g] + 1$ and $v_{i}[j][g] \leq v_{i}[k][g] \quad (k \neq j);$;

CO-deliver $m$; // after removing it from the hold-back queue
$$v_{i}[i][g] := v_{i}[i][g] + 1;$$

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).