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 \( C_3 \).
- 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

(Do NOT be confused with vector timestamps)

*Accept* = Deliver

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 \( x \)
   - On initialization: \( i_x = 0 \);
   - To \( \text{TO-multicast} \) message \( m \) to group \( g \)
     \[ \text{TO-multicast}(x, m, z, i_x, i_x > t) \]
   - On \( \text{B-deliver} \) \( m; \) with \( g = \text{group}(m) \)
     - Place \( \langle m, z \rangle \) in hold-back queue.
     - On \( \text{B-deliver} \) \( m; \) with \( g = \text{group}(m) \)
       - Wait until \( \langle m, z \rangle \) in hold-back queue and \( z < t_i \)
     - \( \text{TO-deliver} 1 \) \( m \) (after deleting it from the hold-back queue)
     \[ t_x = t_x + 1 \]

2. Algorithm for sequencer of \( g \)
   - On initialization: \( i_x = 0 \);
   - On \( \text{B-deliver} \) \( m; \) with \( g = \text{group}(m) \)
     - \( \text{multicast} (\langle \text{"seq"}, x, i_x \rangle) \)
     - \[ i_x = i_x + 1 \]
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.

Sequence number requirement
- It should always be strictly increasing (otherwise, there's no ordering).
- I.e., a sequence number should always be greater than the previous sequence number.
- How do we ensure that?
  - Reminder: a sequence number is picked among proposals.
  - Each receiver proposes a higher number that it has never proposed or agreed to use as a sequence number.
  - A sender picks the highest number among all proposals.
- How to always propose a higher number?
  - A receiver looks at two things.
    - The last agreed sequence number assigned to a message
    - The last proposed number by the receiver
  - A receiver takes the max of the two, adds one, and proposes that number.

A Walk-Thru with Two Processes
- Assume:
  - P1 has proposed up to 8 so far.
  - P2 has proposed up to 5 so far.
  - The last message's sequence number was 4.
- Q: why would something like this happen?
  - Multiple messages sent around the same time and network delays.
  - P1 sends a message to P1 & P2.
  - P1 proposes 9 (to P1).
  - P2 proposes 6 (to P1).
  - P1 picks 9 as the sequence number.
  - P1 announces that the sequence number is 9.
  - Sequence numbers of the last two messages: 4 & 9

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
- Notice any (small) issue?

CSE 486/586 Administrivia
- PA2-B is due on 3/13.
  - Right before Spring break
- PA1 re-grading office hours
  - Tuesdays 1pm - 4pm
  - Wednesdays 2pm - 5pm
  - Thursdays 9am - 12pm
  - Fridays 9am - 12pm
- Midterm is on 3/11.
  - During class, not in the evening
**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.

**Example: ISIS algorithm**

We don't dictate when events are happening

```
P1 A

P2

P3
```

```
P1 A

P2 A:1.1

P3 B
```

```
P1 A

P2 B:1.2

P3 A:2.2
```

```
P1 A

P2 B:1.2

P3 B:1.3
```
Example: ISIS algorithm
We don’t dictate when events are happening

\begin{itemize}
\item P1
\item P2
\item P3
\end{itemize}

\begin{itemize}
\item A
\item C
\item B
\end{itemize}

\begin{itemize}
\item A:1.1
\item B:1.2
\item B:1.3
\item C:1.2
\item C:1.3
\end{itemize}
Example: ISIS algorithm

We don't dictate when events are happening

P1 A

P2 C

P3 B

A:2.3 B:1.2 C:3.1
A:2.2 C:3.2
A:2.3 C:3.3

A:1.3 B:1.2 C:3.2
A:2.2 C:3.3
A:2.3 C:3.3

A:2.3 B:1.3 C:2.1
A:2.1 B:1.3 C:2.3
A:2.3 B:1.3 C:2.3
We don't dictate when events are happening.
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>

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

We don't dictate when events are happening

P1
A

P2
C

P3
B

A:2.3  B:3.1  C:3.3

B:1.2  A:2.3  C:3.3

A:2.3  B:3.1  C:3.3

B:3.1  A:2.3  C:3.3

A:2.3  B:3.1  C:3.3

B:3.1  A:2.3  C:3.3

A:2.3  B:3.1  C:3.3

B:3.1  A:2.3  C:3.3

A:2.3  B:3.1  C:3.3

B:3.1  A:2.3  C:3.3

A:2.3  B:3.1  C:3.3

B:3.1  A:2.3  C:3.3

A:2.3  B:3.1  C:3.3

B:3.1  A:2.3  C:3.3
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 $m_1$ be another message that has not yet been delivered (i.e., is on the same queue or has not been seen yet by $p$).
  
  \[
  \text{finalpriority}(m_1) \geq \text{proposedpriority}(m_1) > \text{finalpriority}(m_i)
  \]
  
- Suppose there is some other process $p'$ that delivers $m_2$ before it delivers $m_1$. Then at $p'$,
  
  \[
  \text{finalpriority}(m_1) \geq \text{proposedpriority}(m_1) > \text{finalpriority}(m_2)
  \]
  
- a contradiction!

Causally Ordered Multicast

- Each process keeps a vector of message clocks.
  - Each counter represents the number of messages received from each of the other processes.
  - This works just like vector clocks, but with messages.
- 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, i.e., until it has delivered all the messages that have happened before:
  - 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**
  \( v_{i}^{g}(j) = 0 \) \( (j = 1, 2, ..., N) \).

- **To CO-multicast message** \( m \) to group \( g \):
  \[ v_{i}^{g}(i) = v_{i}^{g}(i) + 1; \]
  - \( B\)-multicast \( g, \langle v_{i}^{g}, m \rangle \).

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

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).