CSE 486/586 Distributed Systems
Reliable Multicast --- 1

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time
- Global states
  - A union of all process states
  - Consistent global state vs. inconsistent global state
- The "snapshot" algorithm
  - Take a snapshot of the local state
  - Broadcast a "marker" msg to tell other processes to record
  - Start recording all msgs coming in for each channel until receiving a "marker"
  - Outcome: a consistent global state

Today’s Question
- How do a group of processes communicate?
- Unicast (best effort or reliable)
  - One-to-one: Message from process p to process q.
  - Best effort: message may be delivered, but will be intact
  - Reliable: message will be delivered
- Broadcast
  - One-to-all: Message from process p to all processes
  - Impractical for large networks
- Multicast
  - One-to-many: "Local" broadcast within a group g of processes (e.g., m processes out of n total processes)
- What are the issues?
  - Processes crash (we assume crash-stop)
  - Messages get delayed

Why: Examples
- Akamai’s Configuration Management System (called ACMS)
  - A core group of 3-5 servers,
  - Continuously multicast to each other the latest updates.
  - After an update is reliably multicast within this group, it is then sent out to all the (1000s of) servers Akamai has all over the world.
- Air Traffic Control System
  - Commands by one ATC need to be ordered (and reliable) multicast out to other ATC’s.
- News group servers
  - Multicast to each other in a reliable and ordered manner.

The Interface
Application (at process p)
  - send
  - deliver
  - multicast

MULTICAST PROTOCOL
  - Incoming messages
  - receive

One process p
  - send
  - multicast
  - deliver
  - receive

CSE 486/586
**Basic Multicast (B-multicast)**

- A straightforward way to implement B-multicast is to use a reliable one-to-one send (unicast) operation:
  - B-multicast\(g, m\): for each process \(p\) in \(g\), send\(p, m\).
  - receive\(m\): B-deliver\(m\) at \(p\).

**Guarantees?**

- All processes in \(g\) eventually receive every multicast message...
- ... as long as the sender doesn’t crash
- This guarantee is not so good.

**What guarantees do we want?**

---

**What: Properties to Consider**

- Often times, a distributed system cares about at least two categories of properties.
- **Liveness:** guarantee that something good will happen eventually
  - For the initial state, there is a reachable state where the predicate becomes true.
  - “Guarantee of termination” is a liveness property
- **Safety:** guarantee that something bad will never happen
  - For any state reachable from the initial state, the predicate is false.
  - Deadlock avoidance algorithms provide safety
- It is important to think about liveness and safety in your system & context.
  - Liveness and safety are used in many other CS contexts.

---

**Reliable Multicast Overview**

- Keep a history of messages for at-most-once delivery
- Everyone repeats multicast upon a receipt of a message.
  - Why? For agreement & validity.
  - Even if the sender crashes, as long as there is one process that receives, it’s all good since that process is going to repeat.

---

**What: Reliable Multicast Goals**

- These are refined from liveness and safety categories for the context of reliable multicast.
- **Integrity:** A correct (i.e., non-faulty) process \(p\) delivers a message \(m\) at most once.
  - “Non-faulty”: doesn’t deviate from the protocol & alive
- **Validity:** If a correct process multicasts \(m\), then it will eventually deliver \(m\).
  - Property of “all or nothing.”
- **Agreement:** If a correct process delivers message \(m\), then all the other correct processes in group\(m\) will eventually deliver \(m\).
  - Property of “all or nothing.”
- **Validity and agreement together ensure overall liveness:** if some correct process multicasts a message \(m\), then, all correct processes deliver \(m\) too.

---

**Reliable R-Multicast Algorithm**

On initialization

\[\text{Received} := \{\}\;\]

For process \(p\) to R-multicast message \(m\) to group \(g\)

\[\text{B-multicast}(g, m);\]
\(p \in g\) is included as destination

On \(\text{B-deliver}(m)\) at process \(q\) with \(g = \text{group}(m)\)

\[\text{tf (} m \notin \text{Received});\]
\(\text{Received} := \text{Received} \cup \{m\};\)
\(\text{if } (q \neq p);\)
\(\text{B-multicast}(g, m);\)
\(\text{R-deliver}(m)\)

Integrity

On \(\text{R-deliver}(m)\) at process \(q\) with \(g = \text{group}(m)\)

\[\text{tf (} m \notin \text{Received});\]
\(\text{Received} := \text{Received} \cup \{m\};\)
\(\text{if } (q \neq p);\)
\(\text{B-multicast}(g, m);\)
\(\text{Agreement}\)
\(\text{R-deliver}(m)\)

Validity
CSE 486/586 Administrivia

- PA1 grading is done.
  - Grades will be posted today after my office hours.
  - Will accept re-grading requests from next week, just during that week.
  - Come to see a TA during the following hours and only the following hours:
    » Tuesdays 1pm - 4pm
    » Wednesdays 2pm - 5pm
    » Thursdays 9 am - 12 pm
    » Fridays 9am - 12pm
  - Bring your laptop for re-grading. If you don’t have a laptop, write a private Piazza post and ask what to do.
- PA2A due this Friday

Ordered Multicast Problem

- Assume a delivery mechanism: deliver as soon as you receive
  - What is the order of delivery at each process?
  - Will this mess up anything?

Example: Bulletin Board

<table>
<thead>
<tr>
<th>Bulletin board: #os.interesting</th>
<th>Item</th>
<th>From</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>23 A.Hanlon Mach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 G.Joseph Microkernels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 A.Hanlon Re: Microkernels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 T.L’Heureux RPC performance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 M.Walker Re: Mach</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Authors are message senders.
- The delivery order determines the display order.
- What is the ideal ordering that you want?
- What are the important orderings that you must have?

FIFO Ordering

- Preserving the process (sender) order
- The message delivery order at each receiving process should preserve the message sending order from each sender. But each process can deliver in a different order overall.
- For example,
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8
- Now, each process will receive & deliver all, from m0 to m8.
- FIFO?
  - P1: m0, m3, m6, m1, m4, m7, m2, m5, m8
  - P2: m0, m4, m6, m1, m3, m7, m2, m5, m8
  - P3: m6, m7, m8, m0, m1, m2, m3, m4, m5

Causal Ordering

- Preserving the happened-before relations
- The message delivery order at each receiving process should preserve the happened-before relations across all processes. But each process can deliver in a different order overall.
- For example,
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8
- Cross-process happened-before: m0 → m4, m5 → m8
- Causal?
  - P1: m0, m3, m6, m1, m4, m7, m2, m5, m8
  - P2: m0, m4, m1, m7, m3, m6, m2, m5, m8
  - P3: m0, m1, m2, m3, m4, m5, m6, m7, m8
Total Ordering

- Every process delivers all messages in the same order.
- For example:
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8

- Total?
  - P1: m7, m1, m2, m4, m5, m3, m6, m0, m8
  - P2: m7, m2, m1, m4, m5, m3, m6, m0, m8
  - P3: m7, m1, m2, m4, m5, m3, m6, m0, m8

- Total?
  - P1: m7, m1, m2, m4, m5, m3, m6, m0, m8
  - P2: m7, m2, m1, m4, m5, m3, m6, m0, m8
  - P3: m7, m1, m2, m4, m5, m3, m6, m8, m0

Ordered Multicast

- FIFO ordering: If a correct process issues multicast\((g, m)\) and then multicast\((g, m')\), then every correct process that delivers \(m'\) will have already delivered \(m\).
- Causal ordering: If multicast\((g, m) \rightarrow\) multicast\((g, m')\) then any correct process that delivers \(m'\) will have already delivered \(m\).
- Typically, \(\rightarrow\) defined in terms of multicast communication only.
- Total ordering: If a correct process delivers message \(m\) before \(m'\) (independent of the senders), then any other correct process that delivers \(m'\) will have already delivered \(m\).

Total, FIFO and Causal 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.

Display From Bulletin Board Program

- Bulletin board: os.interesting
- Item
  - From
  - Subject
  - A.Hanlon
  - Mach
  - G.Joseph
  - Microkernels
  - T.L'Heureux
  - RPC performance
  - M.Walker
  - Re: Mach

What is the most appropriate ordering for this application?
(a) FIFO (b) causal (c) total

Providing Ordering Guarantees (FIFO)

- Look at messages from each process in the order they were sent:
  - Each process keeps a sequence number for each of the other processes.
    - E.g., in a system with 3 processes, P1 keeps \((x, y, z)\): \(x\) for P1, \(y\) for P2, & \(z\) for P3 (note: this is not a vector clock)
    - Each of \(x, y, \& z\) indicates the sequence # of the last message from the corresponding process, delivered by P1.
  - When a message is received, if message # is:
    - as expected (next sequence), accept
    - higher than expected, buffer in a queue
    - lower than expected, reject

Hold-back Queue for Arrived Multicast Messages
Implementing FIFO Ordering

- \( S_p^g \): the number of messages \( p \) has sent to \( g \).
- \( R_q^g \): the sequence number of the latest group-\( g \) message \( p \) has delivered from \( q \).

- For \( p \) to FO-multicast \( m \) to \( g \):
  - \( p \) increments \( S_p^g \) by 1.
  - \( p \) "piggy-backs" the value \( S_p^g \) onto the message.
  - \( p \) B-multicasts \( m \) to \( g \).

- At process \( p \), upon receipt of \( m \) from \( q \) with sequence number \( S \):
  - \( p \) checks whether \( S = R_q^g + 1 \). If so, \( p \) FO-delivers \( m \) and increments \( R_q^g \).
  - If \( S > R_q^g + 1 \), \( p \) places the message in the hold-back queue until the intervening messages have been delivered and \( S = R_q^g + 1 \).

Example: FIFO Multicast

Summary

- Reliable Multicast
  - Reliability
  - Ordering
  - R-multicast

- Ordered Multicast
  - FIFO ordering
  - Total ordering
  - Causal ordering

- Next: continue on multicast

Acknowledgements

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