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
- 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.
- Newsgroup servers
  - Multicast to each other in a reliable and ordered manner.

The Interface
What: Properties to Consider

- **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
- Liveness and safety are used in many other CS contexts.

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

What: Reliable Multicast Goals

- **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
- **Agreement**: If a correct process delivers message \(m\), then all the other correct processes in \(g\) will eventually deliver \(m\).
  - Property of "all or nothing."
- **Validity**: If a correct process multicasts (sends) message \(m\), then it will eventually deliver \(m\) itself.
  - Guarantees liveness to the sender.
- Validity and agreement together ensure overall liveness: if some correct process multicasts a message \(m\), then, all correct processes deliver \(m\) too.

Reliable Multicast Overview

- Keep a history of messages for at-most-once delivery
- Everyone repeats multicast upon a receipt of a message for agreement & validity.
  
  Why?

Reliable R-Multicast Algorithm

On initialization

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

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

\[\text{B-multicast}(g, m);\]

\((p \not\in g \text{ is included as destination})\)

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

\[\text{if } (m \not\in \text{Received}):\]

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

\[\text{if } (q \not= p):\]

\[\text{B-multicast}(g, m);\]

\[\text{R-deliver}(m)\]

On initialization

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

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

\[\text{B-multicast}(g, m);\]

\((p \not\in g \text{ is included as destination})\)

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

\[\text{if } (m \not\in \text{Received}):\]

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

\[\text{if } (q \not= p):\]

\[\text{B-multicast}(g, m);\]

\[\text{Agreement}\]

\[\text{R-deliver}(m)\]

\[\text{Validity}\]
CSE 486/586 Administrivia

- PA2 is out.
- New TA: Yavuz Yılmaz
  - Office hours: W 12pm – 3pm

Ordered Multicast Problem

Each process delivers received messages independently.

The question is, what ordering does each process use?

Three meaningful types of ordering
- FIFO
- Causal
- Total

FIFO Ordering

- Preserving the process order
- The message delivery order at each process should preserve the message sending order from every process.
  - For example,
    - P1: m0, m1, m2
    - P2: m3, m4, m5
    - P3: m6, m7, m8
  - FIFO? (m0, m3, m6, m1, m4, m7, m2, m5, m8)
    - Yes!
  - FIFO? (m0, m4, m6, m1, m3, m7, m2, m5, m8)
    - No!

Causal Ordering

- Preserving the happened-before relations
- The message delivery order at each process should preserve the happened-before relations across all processes.
  - For example,
    - P1: m0, m1, m2
    - P2: m3, m4, m5
    - P3: m6, m7, m8
    - Cross-process happened-before: m0 → m4, m5 → m8
  - Causal? (m0, m3, m6, m1, m4, m7, m2, m5, m8)
    - Yes!
  - Causal? (m0, m4, m1, m7, m3, m6, m2, m5, m8)
    - No!

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, m1, m2, 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, m1, m2, m4, m5, m3, m6, m0, m8
    - P3: m7, m1, m2, m4, m5, m3, m6, m0, m8

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) → multicast(g,m’) then any correct process that delivers m’ will have already delivered m.
  - Typically, → 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 $C_2$
- 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.

Display From Bulletin Board Program

<table>
<thead>
<tr>
<th>Item</th>
<th>From</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>A. Hanlon</td>
<td>Mach</td>
</tr>
<tr>
<td>24</td>
<td>G. Joseph</td>
<td>Microkernels</td>
</tr>
<tr>
<td>25</td>
<td>A. Hanlon</td>
<td>Re: Microkernels</td>
</tr>
<tr>
<td>26</td>
<td>T.L. Heames</td>
<td>RPC performance</td>
</tr>
<tr>
<td>27</td>
<td>M. Walker</td>
<td>Re: Mach</td>
</tr>
</tbody>
</table>

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.
  - When a message is received, if message # is:
    » as expected (next sequence), accept
    » higher than expected, buffer in a queue
    » lower than expected, reject

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 $S = R_q^g + 1$.

Hold-back Queue for Arrived Multicast Messages

Example: FIFO Multicast

(Do NOT be confused with vector timestamps)
*Accept* = Deliver

<table>
<thead>
<tr>
<th>Physical Time</th>
<th>Sequence Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>Accept 1 = 0 + 1</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Buffer 2 = 0 + 1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>Buffer 2 = 0 + 1</td>
</tr>
</tbody>
</table>

Display From Bulletin Board Program

Bulletin board: os.interesting
Item: From Subject
23: A. Hanlon Mach
24: G. Joseph Microkernels
25: A. Hanlon Re: Microkernels
26: T.L. Heames RPC performance
27: M. Walker Re: Mach

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