Distributed Systems

CS425/ECE428

Feb 19 2021

Instructor: Radhika Mittal

Acknowledgements for some of materials: Indy Gupta and Nikita Borisov
Logistics

• HW1 is due tonight at 11:59pm.

• HW2 has been released.
  • You should be able to solve the first two questions right away.
  • You should (hopefully) be able to solve all questions after Wednesday’s class.

• MP1 will be released on Wednesday (Feb 24th).
  • Please reach out to me if you are changing groups for MP1, so that we can accordingly reassign the VM clusters.
Logistics

- Midterm on March 8\textsuperscript{th}, Monday, 7pm-8:50pm.
  - CBTF will proctor the exam via Zoom.
- Please sign up with CBTF if you have not already done so.
- Conflict and DRES accommodation requests will be dealt with using the CBTF portal.
- It is a closed-book exam (no websites, no textbooks).
  - You are allowed one physical double-sided cheat sheet (could be typed or hand-written).
- Your answers can be hand-written or typed.
  - If you are typing your answers, use of any online editors (e.g. Google Docs) is \textbf{not} allowed.
  - You can use offline text editors (e.g. Microsoft Word, textEdit, vim, notepad, etc).
- You must submit your responses on Gradescope within 1 hour 50 mins of the start of your exam.
- Syllabus includes everything covered up to (and including) “Multicast”.
Recap: Global snapshot

- State of each process (and each channel) in the system at a given instant of time.
- Difficult to capture global state at same instant of time.
- Capture consistent global state.
  - If captured state includes an event $e$, it includes all other events that happened before $e$.
- Chandy-Lamport algorithm captures consistent global state.
Recap: Global snapshot

• **Global system properties (or predicates):** defined for a captured global state. Two categories:
  • Liveness, e.g. has the algorithm terminated?
    • Must be true for **some** state reachable from initial state for all linearizations.
  • Safety, e.g. the system is not deadlocked.
    • Must be true for **all** states reachable from initial state for all linearizations.

• Chandy-Lamport algorithm can capture **stable global properties:**
  • once true, stays true forever afterwards (for stable liveness)
  • once false, stays false forever afterwards (for stable non-safety)
Today’s agenda

- **Multicast**
  - Chapter 15.4

- **Goal**: reason about desirable properties for message delivery among a group of processes.
Communication modes

• **Unicast**
  - Messages are sent from exactly one process to one process.

• **Broadcast**
  - Messages are sent from exactly one process to all processes on the network.

• **Multicast**
  - Messages broadcast within a group of processes.
  - A multicast message is sent from any one process to a group of processes on the network.
Where is multicast used?

- Distributed storage
  - Write to an object are multicast across replica servers.
  - Membership information (e.g., heartbeats) is multicast across all servers in cluster.

- Online scoreboards (ESPN, French Open, FIFA World Cup)
  - Multicast to group of clients interested in the scores.

- Stock Exchanges
  - Group is the set of broker computers.

- ......
Communication modes

• **Unicast**
  • Messages are sent from exactly one process to one process.
    • *Best effort:* if a message is delivered it would be intact; no reliability guarantees.
    • *Reliable:* guarantees delivery of messages.
    • *In order:* messages will be delivered in the same order that they are sent.

• **Broadcast**
  • Messages are sent from exactly one process to all processes on the network.

• **Multicast**
  • Messages broadcast within a group of processes.
  • A multicast message is sent from any one process to the group of processes on the network.
  • *How do we define (and achieve) reliable or ordered multicast?*
What we are designing in this class?

One process $p$

Application (at process $p$)

$\text{multicast}(g,m)$

$\text{deliver}(m)$

MULTICAST PROTOCOL

Incoming messages
What we are designing in this class?

One process $p$

Application (at process $p$)

$\text{multicast}(g,m)$

$\text{deliver}(m)$

MULTICAST PROTOCOL

Incoming messages
Basic Multicast (B-Multicast)

• Straightforward way to implement B-multicast:
  • use a reliable one-to-one send (unicast) operation:
    \[ \text{B-multicast}(\text{group } g, \text{ message } m): \]
    for each process \( p \) in \( g \), send \((p,m)\).
    \[ \text{receive}(m): \text{B-deliver}(m) \text{ at } p. \]

• Guarantees: message is eventually delivered to the group if:
  • Processes are non-faulty.
  • The unicast “send” is reliable.
  • Sender does not crash.

• Can we provide reliable delivery even after sender crashes?
  • What does this mean?
Reliable Multicast (R-Multicast)

• **Integrity**: A correct (i.e., non-faulty) process $p$ delivers a message $m$ at most once.
  • Assumption: no process sends exactly the same message twice

• **Validity**: If a correct process multicasts (sends) message $m$, then it will eventually deliver $m$ itself.
  • Liveness for the sender.

• **Agreement**: If a correct process delivers message $m$, then all the other correct processes in group($m$) will eventually deliver $m$.
  • 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 Multicast (R-Multicast)

- **Integrity**: A *correct* (i.e., non-faulty) process $p$ delivers a message $m$ at most once.
- **Assumption**: no process sends exactly the same message twice
- **Validity**: If a *correct* process multicasts (sends) message $m$, then it will eventually deliver $m$ itself.
- **Liveness for the sender**.
- **Agreement**: If a *correct* process delivers message $m$, then all the other *correct* processes in group($m$) will eventually deliver $m$.
- **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.

What happens if a process initiates B-multicasts of a message but fails after unicasting to a subset of processes in the group?

Agreement is violated! R-multicast not satisfied.
Implementing R-Multicast

MULTICAST PROTOCOL

Application (at process p)

multicast(g,m)

deliver(m)

Incoming messages
Implementing R-Multicast

Application (at process $p$)

$R$-multicast$(g,m)$

$R$-deliver$(m)$

$B$-multicast$(g,m)$

$B$-deliver$(m)$

Incoming messages
Implementing R-Multicast

On initialization

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

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

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

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

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

- **Integrity**: A *correct* (i.e., non-faulty) process $p$ delivers a message $m$ at most once.
  - Assumption: no process sends *exactly* the same message twice
- **Validity**: If a *correct* process multicasts (sends) message $m$, then it will eventually deliver $m$ itself.
  - Liveness for the sender.
- **Agreement**: If a *correct* process delivers message $m$, then all the other *correct* processes in group($m$) will eventually deliver $m$.
  - 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.
Ordered Multicast

• Three popular flavors implemented by several multicast protocols:
  1. FIFO ordering
  2. Causal ordering
  3. Total ordering
1. FIFO Order

- Multicasts from each sender are delivered in the order they are sent, at all receivers.
- Don’t care about multicasts from different senders.

- More formally
  - 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.
FIFO Order: Example

*M1:1* and *M1:2* should be delivered in that order at each receiver. Order of delivery of *M3:1* and *M1:2* could be different at different receivers.
2. Causal Order

• Multicasts whose send events are causally related, must be delivered in the same causality-obeying order at all receivers.

• More formally
  • If $\text{multicast}(g,m) \rightarrow \text{multicast}(g,m')$ then any correct process that delivers $m'$ will have already delivered $m$.
  • $\rightarrow$ is Lamport’s happens-before
  • $\rightarrow$ is induced only by multicast messages in group $g$, and when they are delivered to the application, rather than all network messages.
Causal Order: Example

M3:1 \rightarrow M3:2, M1:1 \rightarrow M2:1, M1:1 \rightarrow M3:1 and so should be delivered in that order at each receiver.

M3:1 and M2:1 are concurrent and thus ok to be delivered in any (and even different) orders at different receivers.
To be continued in next class

- More on causal ordering
- Total ordering
- Implementing of FIFO/Causal/Total ordering
Summary

- Multicast is an important communication mode in distributed systems.

- Applications may have different requirements:
  - Reliability
  - Ordering: FIFO, Causal, Total
  - Combinations of the above.