Causal Broadcast

Seif Haridi
haridi@kth.se
**Motivation**

- Assume we have a chat application
  - Whatever written is **reliably broadcast** to group
- If you get the following output, is it ok?
  
  ![Chat messages]

  - Cosmin’s message **caused** Lars’s message,
    - Lars’s message **caused** Paris’s message
Motivation (2)

• Does uniform reliable broadcast remedy this?

[d]
Motivation (3)

- Causal reliable broadcast solves this
  - Deliveries in causal order!

- Causality is same as happened-before relation by Lamport!
Cause-effect relations in message passing systems

- An event $e_1$ may potentially have caused another event $e_2$ if the following relation, called, \textit{happens-before} and denoted by $e_1 \rightarrow e_2$ holds
Happens-before relation

- e1 and e2 occurs at the same process p, and e1 occurs before e2
- e1 is the transmission of a message m at process p and e2 is the reception of the same message at process q
- There exist some event e’ such that e1 → e’ and e’ → e2
Happens-before relation

\[ e_1 \quad e_2 \]

\[ e_1 \quad e'_2 \]
Intuitions (1)

• So far, we did not consider ordering among messages; In particular, we considered messages to be independent
• Two messages from the same process might not be delivered in the order they were broadcast
• A message m1 that causes a message m2 might be delivered by some process after m2
Intuitions (2)

- Causal broadcast means
  - Causality between broadcast events is preserved by the corresponding delivery events

- If broadcast(m1) happens-before broadcast(m2), any delivery(m2) cannot happen-before a delivery(m1)
Causality of Messages

- Let $m_1$ and $m_2$ be any two messages:
  $m_1 \rightarrow m_2$ ($m_1$ causally precedes $m_2$) if

- **C1 (FIFO order).**
  - Some process $p_i$ broadcasts $m_1$ before broadcasting $m_2$

- **C2 (Network order).**
  - Some process $p_i$ delivers $m_1$ and later broadcasts $m_2$

- **C3 (Transitivity).**
  - There is a message $m'$ such that $m_1 \rightarrow m'$ and $m' \rightarrow m_2$
Causality

- **C1 (FIFO order).**
  - Some process $p_i$ broadcasts $m_1$ before broadcasting $m_2$.

![Diagram showing causality with processes $p_1$, $p_2$, $p_3$ and messages $m_1$, $m_2$.]
Causality (2)

- **C2 (Network order)**.
  - Some process $p_i$ delivers $m_1$ and later broadcasts $m_2$
Causality (3)

- **C3 (Transitivity).**
  - There is a message $m'$ such that $m_1 \rightarrow m'$ and $m' \rightarrow m_2$
Specification of causal reliable broadcast
Causal Broadcast Interface

• **Module:**
  • Name: CausalOrder (co)

• **Events**
  • Request: \langle co Broadcast \mid m \rangle
  • Indication: \langle co Deliver \mid src, m \rangle

• **Property:**
  • \textbf{CB}: If node \( p_i \) delivers \( m_1 \), then \( p_i \) must have delivered every message causally preceding \((\rightarrow)\) \( m_1 \) before \( m_1 \)
Causal Broadcast Interface

- If node $p_i$ delivers $m_1$, then $p_i$ must have delivered every message causally preceding ($\rightarrow$) $m_1$ before $m_1$

- Is this useful? How can it be satisfied? [d]

  - It is only safety. Satisfy it by never delivering!
Different Causalities

- **Property:**
  - \( CB \): If node \( p_i \) delivers \( m_1 \), then \( p_i \) must deliver every message causally preceding \((\rightarrow)\) \( m_1 \) before \( m_1 \)
  
  - \( CB' \): If \( p_j \) delivers \( m_1 \) and \( m_2 \), and \( m_1 \rightarrow m_2 \), then \( p_j \) must deliver \( m_1 \) before \( m_2 \)

- What is the difference? [d]
Different Causalities

- **Property:**
  - $\textbf{CB}$: If node $p_i$ delivers $m_1$, then $p_i$ must deliver every message causally preceding ($\rightarrow$) $m_1$ before $m_1$
  - $\textbf{CB'}$: If $p_j$ delivers $m_1$ and $m_2$, and $m_1 \rightarrow m_2$, then $p_j$ must deliver $m_1$ before $m_2$

- What is the difference? [d]

- Indeed, CB implies CB’
Reliable Causal Broadcast Interface

- **Module:**
  - Name: `ReliableCausalOrder (rco)`

- **Events**
  - Request: `⟨rco Broadcast | m⟩`
  - Indication: `⟨rco Deliver | src, m⟩`

- **Property:**
  - `RB1-RB4` from regular reliable broadcast
  - `CB`: If node $p_i$ delivers $m$, then $p_i$ must deliver every message causally preceding ($→$) $m$ before $m$
Uniform Reliable Causal Broadcast

- **Module:**
  - Name: UniformReliableCausalOrder (urco)

- **Events**
  - Request: \( \langle \text{urco Broadcast} | m \rangle \)
  - Indication: \( \langle \text{urco Deliver} | \text{src}, m \rangle \)

- **Property:**
  - **URB1-URB4**: from uniform reliable broadcast
  - **CB**: If node \( p_i \) delivers \( m \), then \( p_i \) must deliver every message causally preceding (\( \rightarrow \)) \( m \) before \( m \)
Idea reuse…

- Reuse RB for CB

  - Use **reliable broadcast** abstraction to implement **reliable causal broadcast**

  - Use **uniform reliable broadcast** abstraction to implement **uniform causal broadcast**
Implementation of causal reliable broadcast
Towards an implementation

● Main idea
  ● Each broadcasted message carries a **history**
  ● Before delivery, ensure causality

● First algorithm
  ● History is set of all **causally preceding** messages
Fail-Silent No-Waiting Causal Broadcast

- Each message $m$ carries an ordered list of causally preceding messages in $\text{past}_m$

- Whenever a node rb-Delivers $m$
  - co-Deliver causally preceding messages in $\text{past}_m$
  - co-Delivers $m$
    - Avoid duplicates using delivered
Execution (direct override)
Execution (indirect override)
Fail-silent Causal Broadcast Impl

- **Implements:**
  - ReliableCausalOrderBroadcast (rco)

- **Uses:** ReliableBroadcast (rb)

- **upon event** \(\langle \text{Init} \rangle \) **do**
  - delivered := \(\emptyset\); past := nil

- **upon event** \(\langle \text{rco Broadcast} \mid m \rangle \) **do**
  - trigger \(\langle \text{rb Broadcast} \mid (\text{DATA}, \text{past}, m) \rangle\)
  - past := append(past, \((p_i, m)\))

Append this message to past history
Fail-silent Causal Broadcast Impl (2)

- **upon event** $\langle \text{rb Deliver} \mid \text{pi}, (\text{DATA}, \text{past}_m, m) \rangle$ do
  - if $m \notin \text{delivered}$ then
    - forall $(s_n, n) \in \text{past}_m$ do
      - if $n \notin \text{delivered}$ then
        - trigger $\langle \text{rco Deliver} \mid s_n, n \rangle$
        - delivered := delivered $\cup \{n\}$
        - past := append(past, $(s_n, n)$)
  - trigger $\langle \text{rco Deliver} \mid \text{pi}, m \rangle$
  - delivered := delivered $\cup \{m\}$
  - past := append(past, $(\text{pi}, m)$)

in ascending order
deliver preceding messages
append to history
deliver current message
append to history
Correctness

- RB1-RB4 follow from use of RB
  - No creation and no duplication still satisfied
  - Validity still satisfied
    - Some messages might be delivered earlier, never later
  - Agreement directly from RB
Correctness

• RB1-RB4 follow from use of RB
  • No creation and no duplication still satisfied
  • Validity still satisfied
    • Some messages might be delivered earlier, never later
Correctness

- RB1-RB4 follow from use of RB
  - Agreement directly from RB
  - If correct process $p_k$ delivers all correct processes deliver
    - all processes will deliver because of RB agreement either immediately or included in the past $m$ of previous message $m$
Correctness of CB

- If process $p_i$ delivers $m$, then $p_i$ must deliver every message causally preceding ($\rightarrow$) $m$ before $m$
- This property is an invariant of each execution (or prefix of)
- $P$ is an invariant if $P(E)$ holds for all executions $E$
- If $P(E)$ is an invariant,
  - $P$ hold for all $s_0$ in the set of initial states
  - If $P$ holds in execution (prefix) $E$ with final state $s_n$ then $P$ holds after extending $E$ with any transition step $(s_n, e_{n+1}, s_{n+1})$
Correctness CB

- Each message carries its causal past
  - Each delivery of a message m makes sure that its causal past is delivered before m
- CO by *induction* on prefixes of executions
  - It is true for empty executions (initial state $s_0$)
  - Assume it is true for all deliveries of a prefix
    - Then it is true for any extension with one more event
Improving the algorithm

- Disadvantage of algorithm is that the message size (bit complexity) grows
- Useful idea
  - Garbage collect old messages
- Implementation of GC
  - Acknowledge causal delivery of every message m to all processes
  - Use perfect failure detector P
    - Determine with P when all correct nodes got message m
    - Delete m from past when all correct processes got m
Improving the algorithm

● We use \( P \)
● Use FIFO reliable broadcast
● It is possible to trim \( \text{Past} \) ?
Causal Broadcast Algorithm using FIFO Broadcast
Causal Broadcast Interface

- **Module:**
  - Name: FIFO-ReliableBroadcast (frb)

- **Events**
  - Request: \( \langle \text{frb Broadcast} | m \rangle \)
  - Indication: \( \langle \text{frb Deliver} | \text{src, m} \rangle \)

- **Property:**
  - **FIFO delivery**: if \( p_i \) broadcasts message \( m_1 \) before it broadcasts message \( m_2 \), then no correct process delivers \( m_2 \) unless it has already delivered \( m_1 \)
  - RB1-RB4
Idea of using FIFO reliable broadcast

- Assume we use fifo-rb instead rb
- In the no-waiting algorithm
  - Each process $p_i$ rb-broadcasts the message $\text{append}(\text{past}_m, m)$
  - Assume two consecutive broadcasts by $p_i$
    - $\text{append}(\text{past}_{m_1}, m_1)=l_1$ and then $\text{append}(\text{past}_{m_2}, m_2)=l_2$
    - Each correct process delivers $l_1$ before $l_2$ by FIFO delivery
    - But $l_1$ is a prefix of $l_2$ so $p_i$ needs to only broadcast $l_2 - l_1$
  - Each $p_i$ needs to keep track only of messages between to consecutive broadcasts
Fail-silent Causal Broadcast Impl

- Implements:
  - ReliableCausalOrderBroadcast (rco)
- Uses: FIFO-ReliableBroadcast (frb)
- upon event \langle \text{Init} \rangle\ do
  - delivered := ∅; \( l := \text{nil} \)
- upon event \langle \text{rco Broadcast} | m \rangle\ do
  - trigger \langle \text{frb Broadcast} | \text{(DATA, append}(l, m)\rangle
  - \( l := \text{nil} \)
  - reset \( l \) to store only new deliveries
Fail-silent Causal Broadcast Impl (2)

- upon event $\langle \text{frb Deliver} | \text{pi,(DATA, } l_m) \rangle$ do
  - for all $(s_n,n) \in l_m$ do
  - if $n \not\in \text{delivered}$ then
    - trigger $\langle \text{rco Deliver} | s_n, n \rangle$
    - delivered $:= \text{delivered} \cup \{n\}$
    - if $(s_n,n) \not\in l$ then
      - append $(l, (s_n,n))$
  - Can we trim the delivered set? [d]
Fail-Silent Waiting Algorithm
Towards another implementation

- Main idea
  - Each broadcasted message carries a history
  - Before delivery, ensure causality

- First & Second algorithms
  - History is set of all causally preceding messages

- Third algorithm [d]
  - History is a vector timestamp
Fail-Silent Waiting Causal Broadcast

- Represent past history by vector clock (VC)

- Slightly modify the VC implementation
  - At process $p_i$
    - $VC[i]$: number of messages $p_i$ coBroadcasted
    - $VC[j]$, $j \neq i$: number of messages $p_i$ coDelivered from $p_j$
Fail-Silent Waiting Causal Broadcast

- Upon CO broadcast $m$
  - Piggyback VC and RB-broadcast $m$
  - $VC_m[r]$ is the number messages causally preceding $m$ from $r$

- Upon RB delivery of $m$ with attached $VC_m$
  - compare $VC_m$ with local $VC_i$
  - Only deliver $m$ once $VC_m \leq VC_i$
  - Do Not deliver if $VC_m > VC_i$ or $VC_m \neq VC_i$
Upon RB delivery of m with attached \( VC_m \) compare \( VC_m \) with local \( VC_i \)

- Only deliver \( m \) once \( VC_m \leq VC_i \)
- Do Not deliver if \( VC_m > VC_i \) or \( VC_m \neq VC_i \)
Execution

\[(0,0,0)\]

\[b(m_1)\]

\[d(m_1)\]

\[b(m_2)\]

\[d(m_2)\]

\[d(m_1)\]

\[d(m_2)\]

\[d(m_2)\]

\[d(m_1)\]

\[(1,0,0)\]

\[m_1(0,0,0)\]

\[m_2(1,0,0)\]

\[m_1(0,0,0)\]

\[m_2(1,0,0)\]

\[m_1(0,0,0)\]

\[m_2(1,0,0)\]

\[m_1(0,0,0)\]

\[m_2(1,0,0)\]

\[d(m_2)\]

\[d(m_1)\]

\[hold\ m_2\]

\[(1,0,0)\]

\[(2,0,0)\]

\[(0,0,0)\]

\[(1,0,0)\]

\[p1\]

\[p2\]

\[p3\]
Fail-Silent Waiting Causal Implementation

- **Uses:** ReliableBroadcast (rb)
- **upon event** ⟨Init⟩ do
  - `forall` `pi ∈ Π` `do` `VC[pi] := 0`
  - `sn := 0`
  - `Pending := ø`
- **upon event** ⟨rco Broadcast⟩m do
  - `W = copy(VC)`
  - `W[self] := sn`
  - `trigger` ⟨rbBroadcast⟩(DATA, W, m)⟩
  - `sn := sn + 1`
  - `send m` with `VC`
  - `Increase sn` for next broadcast
Fail-Silent Waiting Causal Impl. (2)

- **upon event** \( \langle \text{rbDeliver} | p_j, (\text{DATA}, \text{VC}_m, m) \rangle \) **do**
  - pending := pending \( \cup (p_j, (\text{DATA}, \text{VC}_m, m)) \)
  - deliver-pending()

- **proc** deliver-pending()
  - **while exists** \( x = (s_m, (\text{DATA}, \text{VC}_m, m)) \in \text{pending} \) s.t. \( \text{VC}_m \leq \text{VC} \) **do**
    - pending := pending \( \setminus (s_m, (\text{DATA}, \text{VC}_m, m)) \)
    - \( \text{VC}[s_m] := \text{VC}[s_m] + 1 \)
    - **trigger** \( \langle \text{rcoDeliver} | s_m, m \rangle \)
Correctness

- **Validity**
  - $m$ is co-cast by a correct pi with $VC_m$ equal $VC_i$ at send time or higher only at $VC_i[i]$ by outstanding earlier co-cast not delivered yet.
  - By rb-cast validity $m$ is eventually rb-delivered at $p_i$ as well as earlier co-casts.
  - At delivery time $VC_i$ can only increase, so
  - Eventually $VC_m \leq VC_i$ and $m$ is co-delivered

upon event $\langle rco \text{ Broadcast} | m \rangle$
do
\begin{align*}
W &= \text{copy}(VC) \\
W[\text{self}] &:= sn \\
\text{trigger} \langle rbBroadcast|(DATA, W, m) \rangle \\
\text{sn} &:= sn + 1
\end{align*}
Correctness

• Agreement
  • Assume m is co-delivered at correct pi
  • pi co-delivered all message causally before m
  • Every correct process rb-delivered m and all causally preceding messages (agreement of RB)
  • Hence every correct process co-deliver m
Correctness

- **Causal Order**
  - Assume p rb-delivers m, VC_m from q
  - VC_m[r] is the number messages causally preceding m from r
  - VC at p stores the number of messages co-delivered from each process
  - For some r, VC_m[r] > VC[r] implies there is at least one message from r that is causally before m, which is not co-delivered at p
  - P waits to deliver m until VC_m[r] ≤ VC_i[r], for all r
  - Hence m is not delivered until all causally preceding messages are delivered
Orderings of Broadcast
Possible execution?

• Delivery order isn’t same!
  ❑ What is wrong? [d] Nothing, there is no causality.
Other possible orderings

- Other common orderings
  - Single-source FIFO order
  - Total order
  - Causal order
Single-Source FIFO order

- Intuitively
  - Msgs from same node delivered in order sent
- For all messages $m_1$ and $m_2$ and all $p_i$ and $p_j$,
  - if $p_i$ broadcasts $m_1$ before $m_2$, and if $p_j$ delivers $m_2$,
    then $p_j$ delivers $m_1$ before $m_2$

- Caveat
  - This formulation doesn’t require delivery of both messages
Total Order

• Intuitively
  • Everyone delivers everything in exact same order

• For all messages $m_1$ and $m_2$ and all $p_i$ and $p_j$,
  • if both $p_i$ and $p_j$ deliver both messages, then they deliver them in the same order

• Caveat
  • This formulation doesn’t require delivery of both messages
  • Everyone delivers same order, maybe not send order!
 Execution Example (1)

single-source FIFO? yes

totally ordered? no

causally ordered? yes
Execution Example (2)

single-source FIFO? no
totally ordered? yes
causally ordered? no
Execution Example (3)

single-source FIFO? yes
totally ordered? no
causally ordered? no
Hierarchy of Orderings

- Stronger implies weaker ordering (→)

- best-effort → reliable → uniform reliable
- FIFO best-effort → reliable FIFO → uniform reliable FIFO
- causal best-effort → reliable causal → uniform reliable causal