Distributed Algorithms

Fall 2019

Reliable & Causal Broadcast
3rd exercise session, 07/10/2019

Matteo Monti <matteo.monti@epfl.ch>
Athanasiros Xygkis <athanasiros.xygkis@epfl.ch>
Reliable broadcast

Specification:

- **Validity**: If a *correct* process broadcasts $m$, then it eventually delivers $m$.
- **Integrity**: $m$ is delivered by a process at most once, and only if it was previously broadcast.
- **Agreement**: If a correct process delivers $m$, then all correct processes eventually deliver $m$. 
Algorithm: Lazy Reliable Broadcast

**Implements:**
ReliableBroadcast, instance \( rb \).

**Uses:**
BestEffortBroadcast, instance \( beb \);
PerfectFailureDetector, instance \( P \).

**upon event** \( \langle rb, \text{Init} \rangle \) do

\[
\begin{align*}
\text{correct} & := \emptyset; \\
from[p] & := [0]^N;
\end{align*}
\]

**upon event** \( \langle rb, \text{Broadcast} \mid m \rangle \) do

\[
\begin{align*}
\text{trigger} & \langle beb, \text{Broadcast} \mid [\text{DATA}, \text{self}, m] \rangle;
\end{align*}
\]

**upon event** \( \langle beb, \text{Deliver} \mid p, [\text{DATA}, s, m] \rangle \) do

\[
\begin{align*}
\text{if } m & \notin from[s] \text{ then} \\
\text{trigger} & \langle rb, \text{Deliver} \mid s, m \rangle; \\
from[s] & := from[s] \cup \{m\}; \\
\text{if } s & \notin \text{correct} \text{ then} \\
\text{trigger} & \langle beb, \text{Broadcast} \mid [\text{DATA}, s, m] \rangle;
\end{align*}
\]

**upon event** \( \langle P, \text{Crash} \mid p \rangle \) do

\[
\begin{align*}
\text{correct} & := \text{correct} \setminus \{p\}; \\
\forall m \in from[p] & \text{ do} \\
\text{trigger} & \langle beb, \text{Broadcast} \mid [\text{DATA}, p, m] \rangle;
\end{align*}
\]

**Strong accuracy:**
No correct process is ever suspected:
\[
\forall F, \forall H, \forall t \in T, \forall p \in \text{correct}(F), \forall q : p \notin H(q, t)
\]

**Strong completeness:**
Eventually, every faulty process is permanently suspected by every correct process:
\[
\forall F, \forall H, \exists t \in T, \forall p \in \text{crashed}(F), \forall q \in \text{correct}(F'), \forall t' \geq t : p \in H(q, t')
\]

Where:
- \( \text{crashed}(F) \) is the set of crashed processes.
- \( \text{correct}(F) \) is the set of correct processes.
- \( H(p, t) \) is the output of the failure detector of process \( p \) at time \( t \).
Exercise 1

Implement a reliable broadcast algorithm without using any failure detector, i.e., using only \textit{BestEffort-Broadcast(BEB)}. 
Exercise 1 (Solution)

Use a step of all-to-all communication.

In particular, very process that gets a message relays it immediately.

Recall that in the original algorithm, processes were relaying messages from a process p only if p crashes.

Upon initialization do
  delivered := {}  

Upon RB-broadcast(m) do
  send(m) to Π \ {p}
  RB-deliver(m)

Upon BEB-receive(m) from q do
  if not m ∈ delivered
    send (m) to Π \ {p, q}
    RB-deliver(m)
  delivered := delivered ∪ m

Agreement: Before RB-delivering m, a correct process p forwards m to all processes. By the properties of perfect channels and the fact that p is correct, all correct processes will eventually receive m and RB-deliver it.
Exercise 2

The reliable broadcast algorithm presented in class has the processes continuously fill their different buffers without emptying them.

A. from
B. delivered

Modify it to remove (i.e. garbage collect) unnecessary messages from the buffers:
A. The *from* buffer is used only to store messages that are relayed in the case of a failure. Therefore, messages from the *from* buffer can be removed as soon as they are relayed.

B. Messages from the *delivered* array cannot be removed. Consider this scenario: If a process crashes and its messages are retransmitted by two different processes, then a process might RB-deliver the same message twice if it empties the *delivered* buffer in the meantime. This is a violation of the "no duplication property".
Uniform reliable broadcast

Specification:

- **Validity**: If a *correct* process broadcasts $m$, then it eventually delivers $m$.
- **Integrity**: $m$ is delivered by a process at most once, and only if it was previously broadcast.
- **Uniform Agreement**: If a *correct* process delivers $m$, then all correct processes eventually deliver $m$. 
Algorithm: All-Ack Uniform Reliable Broadcast

**Implements:**
- UniformReliableBroadcast, instance urb.

**Uses:**
- BestEffortBroadcast, instance beb.
- PerfectFailureDetector, instance P.

**upon event** (urb, Init) do
- delivered := ∅;
- pending := ∅;
- correct := \( \Pi \);
- for all \( m \) do \( \text{ack}[m] := ∅; \)

**upon event** (urb, Broadcast | \( m \)) do
- pending := pending ∪ \{\( (self, m) \)\};
- trigger (beb, Broadcast | [DATA, self, m]);

**upon event** (beb, Deliver | \( p \), [DATA, s, m]) do
- \( \text{ack}[m] := \text{ack}[m] \cup \{p\}; \)
- if \( (s, m) \not\in \text{pending} \) then
  - pending := pending ∪ \{(s, m)\};
- trigger (beb, Broadcast | [DATA, s, m]);

**upon event** (P, Crash | \( p \)) do
- correct := correct \( \setminus \{p\}; \)

**function** candeliver(\( m \)) returns Boolean is
- return \((\text{correct} \subseteq \text{ack}[m]); \)

**upon exists** \( (s, m) \in \text{pending} \) such that \( \text{can deliver}(m) \wedge m \not\in \text{delivered} \) do
- delivered := delivered ∪ \{m\};
- trigger (urb, Deliver | s, m);
Exercise 3

What happens in the reliable broadcast and uniform reliable broadcast algorithms if the:

A. accuracy, or
B. completeness

property of the failure detector is violated?
Exercise 3 (Solution 1/2)

Reliable broadcast:

1. Suppose accuracy is violated. Then, the processes might be relaying messages when this is not really necessary. This wastes resources but does not impact correctness.

2. Suppose completeness is violated. Then, the processes might not be relaying messages they should be relaying. This may violate agreement. For instance, assume that only a single process \(p_1\) BEB-delivers (hence RB-delivers) a message \(m\) from a crashed process \(p_2\). If a failure detector (at \(p_1\)) does not ever suspect \(p_2\), no other correct process will deliver \(m\) (agreement is violated).
Exercise 3 (Solution 2/2)

Uniform Reliable broadcast:

Consider a system of three processes $p_1$, $p_2$ and $p_3$. Assume that $p_1$ URB-broadcasts the message $m$.

1. Suppose accuracy is violated. Assume that $p_1$ falsely suspects $p_2$ and $p_3$ to have crashed. $p_1$ eventually URB-delivers $m$. Assume that $p_1$ crashes afterwards. It may happen that $p_2$ and $p_3$ never BEB-deliver $m$ and have no knowledge about $m$ (uniform agreement is violated).

2. Suppose completeness is violated. $p_1$ might never URB-deliver $m$ if either $p_2$ or $p_3$ crashes and $p_1$ never detects their crash. Hence, $p_1$ would wait indefinitely for $p_2$ and $p_3$ to relay $m$ (validity property violation).
Causal Broadcast

Definition (Happens-before):

We say that an event $e$ happens-before an event $e'$, and we write $e \rightarrow e'$, if one of the following three cases holds (is true):

1. (e and $e'$ are executed by the same process)
   \[ \exists p_i \in \Pi \ s.t. \ e = e^r_i, \ e' = e^s_i, \ r < s \]

2. (e and $e'$ are send/receive events of a message respectively)
   \[ e = \text{send}(m, \ast) \land e' = \text{receive}(m) \]

3. (i.e. $\rightarrow$ is transitive)
   \[ \exists e'' \ s.t. \ e \rightarrow e'' \rightarrow e' \]
Causal Broadcast

Specification:

It has the same specification of reliable broadcast, with the additional ordering constraint of causal order.

More precisely (causal order):

\[ \text{broadcast}_p(m) \rightarrow \text{broadcast}_q(m') \Rightarrow \text{deliver}_r(m) \rightarrow \text{deliver}_r(m') \]

Which means that:

If the broadcast of a message \( m \) happens-before the broadcast of a message \( m' \), then no correct process delivers \( m' \) unless it has previously delivered \( m \).
Exercise 4

Can we devise a broadcast algorithm that does **not** ensure the causal delivery property **but only** (in) its non-uniform variant:

No correct process $p_i$ delivers a message $m_2$ unless $p_i$ has already delivered every message $m_1$ such that $m_1 \rightarrow m_2$?
Exercise 4 (Solution)

No! Assume that some algorithm does not ensure the causal delivery property but ensures its non-uniform variant. Assume also that $m_1 \rightarrow m_2$.

This means that a correct process has to deliver $m_1$ before delivering $m_2$, but a faulty process is allowed to deliver $m_2$ and not delivering $m_1$.

However, a process doesn’t know that is faulty in advance (i.e. before it crashes). So, no algorithm can “treat faulty processes in a special way”, i.e., a process has to behave correctly until it crashes.

Reminder (Causal delivery property): For any message $m_1$ that potentially caused a message $m_2$, i.e., $m_1 \rightarrow m_2$, no process delivers $m_2$ unless it has already delivered $m_1$. 
Exercise 5

Suggest a memory optimization of the garbage collection scheme of the following algorithm:

No-Waiting Causal Broadcast

**Implements:**
CausalOrderReliableBroadcast, `instance crb`.

**Uses:**
ReliableBroadcast, `instance rb`.

```plaintext
upon event (crb, Init) do
  delivered := \emptyset;
  past := [];

upon event (crb, Broadcast | m) do
  trigger (rb, Broadcast | [DATA, past, m]);
  append(past, (self, m));

upon event (rb, Deliver | p, [DATA, m, past, m]) do
  if m \notin delivered then
    forall (s, n) \in m, past do // by the order in the list
      if n \notin delivered then
        trigger (crb, Deliver | s, n);
        delivered := delivered \cup \{n\};
      if (s, n) \notin past then
        append(past, (s, n));
  trigger (crb, Deliver | p, m);
  delivered := delivered \cup \{m\};
  if (p, m) \notin past then
    append(past, (p, m));
```

---

Garbage-Collection of Causal Past in the “No-Waiting Causal Broadcast”

**Implements:**
CausalOrderReliableBroadcast, `instance crb`.

**Uses:**
ReliableBroadcast, `instance rb`;
PerfectFailureDetector, `instance P`.

// Except for its ⟨ Init ⟩ event handler, the pseudo code on the left is // part of this algorithm.

```
upon event (crb, Init) do
  delivered := \emptyset;
  past := [];
  correct := II;
  forall m do ack[m] := \emptyset;

upon event (P, Crash | p) do
  correct := correct \setminus \{p\};

upon exists m \in delivered such that self \notin ack[m] do
  ack[m] := ack[m] \cup \{self\};
  trigger (rb, Broadcast | [ACK, m]);

upon event (rb, Deliver | p, [ACK, m]) do
  ack[m] := ack[m] \cup \{p\};

upon correct \subseteq ack[m] do
  forall (s', m') \in past such that m' = m do
    remove(past, (s', m));
```
Exercise 5 (Solution)

When removing a message $m$ from the past, we can also remove all the messages that causally precede this message — and then recursively those that causally precede these. This means that a message stored in the past must be stored with its own distinct past.
Exercise 6

Can we devise a Best-effort Broadcast algorithm that satisfies the causal delivery property, *without* being a causal broadcast algorithm, i.e., without satisfying the *agreement* property of a reliable broadcast?
Exercise 6 (Solution 1/2)

No! Assume that some broadcast algorithm ensures the causal delivery property and is not reliable but best-effort; define an instance co of the corresponding abstraction, where processes co-broadcast and co-deliver messages.

The only way for an algorithm to be best-effort broadcast but not reliable broadcast is to violate the agreement property: there must be some execution of the algorithm where some correct process p co-delivers a message m that some other process q does not ever co-deliver. This is possible in a best-effort broadcast algorithm, in fact this can only happen if the process s that co-broadcasts the message m is faulty (and crashes during the broadcast of m).
Exercise 6 (Solution 2/2)

Assume now that after co-delivering m, process p co-broadcasts a message m’. Given that p is correct and that the broadcast is best-effort, all correct processes, including q, will co-deliver m’. Given that m precedes m’ (in causal order), q must have co-delivered m as well, a contradiction.

Hence, any best-effort broadcast that satisfies the causal delivery property satisfies agreement and is, thus, also a reliable broadcast.

In a nutshell: