Distributed Algorithms

Fall 2019

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

Matteo Monti <matteo.monti@epfl.ch>
Athanasiou Xygkis <athanasios.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

**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 *BestEffort-Broadcast*(BEB).
Exercise 2

The reliable broadcast algorithm presented in Slide 3 has the processes continuously fill their different buffers without emptying them.

Modify it to remove (i.e. garbage collect) unnecessary messages from the buffers:

A. from, and
B. delivered
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 := P;`
  `forall m do 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
  `ack[m] := ack[m] ∪ {p};`
  `if (s, m) ∉ pending then`
  `  pending := pending ∪ {(s, m)};`
  `  trigger { beb, Broadcast | [DATA, s, m] };`

**upon event** `{ P, Crash | p }` do
  `correct := correct \ {p};`

**function** `canDeliver(m)` **returns** Boolean is
  `return (correct ⊆ ack[m]);`

**upon exists** `{ (s, m) ∈ pending such that canDeliver(m) ∧ m \∈ delivered }`
  `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?
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. $\exists p_i \in \Pi \text{ s.t. } e = e_i^r, e' = e_i^s, r < s$ (i.e. $\rightarrow$ is transitive)
2. $e = \text{send}(m, \star) \land e' = \text{receive}(m)$ (e and e’ are send/receive events of a message respectively)
3. $\exists e'' \text{ s.t. } e \rightarrow e'' \rightarrow e'$

(e and e’ are executed by the same process)
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 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.

 upon event ( crb, Init ) do
  delivered := ∅;
  past := [];

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

 upon event ( rb, Deliver | p, [DATA, mpast, m ] ) do
  if m /∈ delivered then
    forall (s, n) ∈ mpast do // by the order in the list
      if n /∈ delivered then
        trigger ( crb, Deliver | s, n );
        delivered := delivered ∪ {n};
      if (s, n) /∈ past then
        append(past, (s, m));
      trigger ( crb, Deliver | p, m );
      delivered := delivered ∪ {m};
      if (p, m) /∈ 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 := ∅;
  past := [];
  correct := IT;
  forall m do ack[m] := ∅;

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

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

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

 upon correct ⊆ ack[m] do
  forall (s', m') ∈ past such that m' = m do
    remove(past, (s', m));
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?