Distributed Algorithms

Fall 2020

Reliable & Causal Broadcast
1st exercise session, 28/09/2020

Matteo Monti <matteo.monti@epfl.ch>
Jovan Komatovic <jovan.komatovic@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 \( \langle \text{rb, Init} \rangle \) do**
\[
\text{correct} := \text{II}; \\
\text{from}[p] := [0]^N;
\]

**upon event \( \langle \text{rb, Broadcast} \mid m \rangle \) do**
\[
\text{trigger} \ (\text{beb, Broadcast} \mid [\text{DATA, self, m}]);
\]

**upon event \( \langle \text{beb, Deliver} \mid p, [\text{DATA, s, m}] \rangle \) do**
\[
\text{if } m \not\in \text{from}[s] \text{ then} \\
\text{trigger} \ (\text{rb, Deliver} \mid s, m); \\
\text{from}[s] := \text{from}[s] \cup \{m\}; \\
\text{if } s \not\in \text{correct then} \\
\text{trigger} \ (\text{beb, Broadcast} \mid [\text{DATA, s, m}]);
\]

**upon event \( \langle P, Crash \mid p \rangle \) do**
\[
\text{correct} := \text{correct} \setminus \{p\}; \\
\text{forall } m \in \text{from}[p] \text{ do} \\
\text{trigger} \ (\text{beb, Broadcast} \mid [\text{DATA, p, m}]);
\]

**Strong accuracy:**
No correct process is ever suspected:
\[
\forall F, \forall H, \forall t \in T, \forall p \in \text{correct}(F), \forall q : p \not\in H(q, t)
\]

**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 2

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

A. from, and
B. delivered

Modify it to remove (i.e. garbage collect) unnecessary messages from the buffers:
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 \( \mathcal{P} \).

upon event \( \langle \text{urb}, \text{Init} \rangle \) do
\[ \text{delivered} := \emptyset; \]
\[ \text{pending} := \emptyset; \]
\[ \text{correct} := \Pi; \]
\[ \text{forall } m \text{ do } \text{ack}[m] := \emptyset; \]

upon event \( \langle \text{urb}, \text{Broadcast} \mid m \rangle \) do
\[ \text{pending} := \text{pending} \cup \{(\text{self}, m)\}; \]
\[ \text{trigger} \langle \text{beb}, \text{Broadcast} \mid \text{[DATA, self, m]} \rangle; \]

upon event \( \langle \text{beb}, \text{Deliver} \mid p, \text{[DATA, s, m]} \rangle \) do
\[ \text{ack}[m] := \text{ack}[m] \cup \{p\}; \]
\[ \text{if } (s, m) \notin \text{pending} \text{ then} \]
\[ \text{pending} := \text{pending} \cup \{(s, m)\}; \]
\[ \text{trigger} \langle \text{beb}, \text{Broadcast} \mid \text{[DATA, s, m]} \rangle; \]

upon event \( \langle \mathcal{P}, \text{Crash} \mid p \rangle \) do
\[ \text{correct} := \text{correct} \setminus \{p\}; \]

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

upon exists \( (s, m) \in \text{pending} \) such that \( \text{can-deliver(m)} \land m \notin \text{delivered} \) do
\[ \text{delivered} := \text{delivered} \cup \{m\}; \]

trigger \( \langle \text{urb}, \text{Deliver} \mid s, m \rangle; \)
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 4

Implement a uniform reliable broadcast algorithm without using any failure detector, i.e., using only $BestEffort-Broadcast(BEB)$.
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):

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

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

\[
\exists e'' \ s.t. \ e \rightarrow e'' \rightarrow e' \quad (\text{i.e. } \rightarrow \text{ is transitive})
\]
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 \( m \) \textit{happens-before} the broadcast of a message \( m' \), then no process delivers \( m' \) unless it has previously delivered \( m \).
Exercise 5

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 6

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** (\( \text{crb}, \text{Init} \)) do
\[
\text{delivered} := \emptyset;
\]
\[
\text{past} := [];
\]

**upon event** (\( \text{crb}, \text{Broadcast} \mid m \)) do
\[
\text{trigger (} \text{rb, Broadcast} \mid \text{DATA, past, m} \text{)};
\]
\[
\text{append} (\text{past}, \text{(self, m)}) ;
\]

**upon event** (\( \text{rb, Deliver} \mid p, \text{DATA, mpast, m} \)) do
\[
\text{if } m \not\in \text{delivered} \text{ then}
\]
\[
\text{forall } (s, n) \in \text{mpast} \text{ do} \quad \| \text{by the order in the list}
\]
\[
\text{if } n \not\in \text{delivered} \text{ then}
\]
\[
\text{trigger (} \text{crb, Deliver} \mid s, n \text{)};
\]
\[
\text{delivered} := \text{delivered} \cup \{n\};
\]
\[
\text{if } (s, n) \not\in \text{past} \text{ then}
\]
\[
\text{append} (\text{past}, (s, n)) ;
\]
\[
\text{trigger (} \text{crb, Deliver} \mid p, m \text{)} ;
\]
\[
\text{delivered} := \text{delivered} \cup \{m\};
\]
\[
\text{if } (p, m) \not\in \text{past} \text{ then}
\]
\[
\text{append} (\text{past}, (p, m)) ;
\]

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

**Implements:**
CausalOrderReliableBroadcast, instance \( crb \).

**Uses:**
ReliableBroadcast, instance \( rb \);
PerfectFailureDetector, instance \( \mathcal{P} \).

// Except for its \( \langle \text{Init} \rangle \) event handler, the pseudo code on the left is
// part of this algorithm.

**upon event** (\( \text{crb}, \text{Init} \)) do
\[
\text{delivered} := \emptyset;
\]
\[
\text{past} := [];
\]
\[
\text{correct} := \mathcal{I};
\]
\[
\text{forall } m \text{ do } \text{ack}[m] := \emptyset;
\]

**upon event** (\( \mathcal{P}, \text{Crash} \mid p \)) do
\[
\text{correct} := \text{correct} \setminus \{p\};
\]

**upon exists** \( m \in \text{delivered} \) such that \( \text{self} \not\in \text{ack}[m] \) do
\[
\text{ack}[m] := \text{ack}[m] \cup \{\text{self}\};
\]
\[
\text{trigger (} \text{rb, Broadcast} \mid \text{ACK, m} \text{)} ;
\]

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

**upon correct \( \subseteq \text{ack}[m] \) do
\[
\text{forall } (s', m') \in \text{past} \text{ such that } m' = m \text{ do}
\]
\[
\text{remove} (\text{past}, (s', m));
\]
Exercise 7

Can we devise a Best-effort Broadcast algorithm that satisfies the causal delivery property, \textit{without} being a causal broadcast algorithm, i.e., without satisfying the \textit{agreement} property of a reliable broadcast?
Exercise 8

In the “Waiting Causal Broadcast”, we say that $V \leq W$ if, for every $i = 1, \ldots, N$, it holds that $V[i] \leq W[i]$.

Question: Why do we not use “$<$” instead of “$\leq$”?

Algorithm 3.15: Waiting Causal Broadcast

```plaintext
Implement:
   - CausalOrderReliableBroadcast, instance crb.

Uses:
   - ReliableBroadcast, instance rb.

upon event (crb, Init) do
   $V := [0]^N$;
   $lsn := 0$;
   $pending := \emptyset$;

upon event (crb, Broadcast | m) do
   $W := V$;
   $W[\text{rank}(\text{self})] := lsn$;
   $lsn := lsn + 1$;
   trigger (rb, Broadcast | [DATA, W, m]);

upon event (rb, Deliver | p, [DATA, W, m]) do
   pending := pending $\cup \{ (p, W, m) \}$;
   while exists $(p', W', m')$ $\in$ pending such that $W' \leq V$
do pending := pending $\setminus \{ (p', W', m') \}$;
   $V[\text{rank}(p')] := V[\text{rank}(p')] + 1$;
   trigger (crb, Deliver | $p'$, $m'$);
```