Distributed Algorithms

Fall 2019

Causal & Total Order Broadcast
4th exercise session, 14/10/2019

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

Exercise 1

Would it make sense to add the total-order property to the best-effort broadcast?

Exercise 1 (Solution)

Total order property: Let m_1 and m_2 be any two messages and suppose p and q are any two correct processes that *deliver* m_1 and m_2. If p delivers m_1 before m_2, then q delivers m_1 before m_2.

The resulting abstraction would not make much sense in a failure-prone environment, as it would not preclude the following scenario:

Assume that a process p broadcasts several messages with best-effort properties and then crashes. Since it is not guaranteed that all correct processes will receive the same set of messages, we might end up having some processes delivering all those messages (in the same order), whereas some other correct processes might end up not delivering any message.

Exercise 2

What happens in our "Consensus-Based Total-Order Broadcast" algorithm, if the set of messages delivered in a round is not sorted deterministically after deciding in the consensus abstraction, but before it is proposed to consensus?

What happens in that algorithm if the set of messages decided on by consensus is not sorted deterministically at all?

Consensus-Based Total-Order Broadcast algorithm

upon event $\langle$ *tob*, *Init* $\rangle$ **do**
    unordered := $\emptyset$;
    delivered := $\emptyset$;
    round := 1;
    wait := FALSE;

upon event $\langle$ *tob*, *Broadcast* | m $\rangle$ **do**
    trigger $\langle$ *rb*, *Broadcast* | m $\rangle$;

upon event $\langle$ *rb*, *Deliver* | p, m $\rangle$ **do**
    if $m \notin$ *delivered* **then**
        unordered := *unordered* $\cup \{(p, m)\}$;

upon *unordered* $\neq \emptyset \wedge$ *wait* = FALSE **do**
    wait := TRUE;
    Initialize a new instance *c.round* of consensus;
    trigger $\langle$ *c.round*, *Propose* | *unordered* $\rangle$;

upon event $\langle$ *c.r*, *Decide* | *decided* $\rangle$ **such that** r = *round* **do**
    // by the order in the resulting sorted list
    forall $(s, m) \in$ *sort*(*decided*) **do**
        trigger $\langle$ *tob*, *Deliver* | s, m $\rangle$;
    delivered := *delivered* $\cup$ *decided*;
    unordered := *unordered* $\setminus$ *decided*;
    round := *round* + 1;
    wait := FALSE;

Exercise 2 (Solution 1/2)

Messages not sorted deterministically after the decision but sorted prior to the proposal

If the deterministic sorting is done prior to proposing the set for consensus, instead of *a posteriori* upon deciding, the processes would not agree on a set but on a sequence of messages. But if they TO-deliver the messages in the decided order, the algorithm still ensures the total order property.

Exercise 2 (Solution 2/2)

Messages not sorted deterministically neither a priori nor a posteriori

If the messages, on which the algorithm agrees in consensus, are never sorted deterministically within every batch (neither *a priori*, not *a posteriori*), then the total order property does not hold.

Even if the processes decide on the same batch of messages, they might TO-deliver the messages within this batch in a different order. In fact, the total order property would be ensured only with respect to batches of messages, but not with respect to individual messages. We thus get a coarser granularity in the total order.

Exercise 3

The "Consensus-Based Total-Order Broadcast" algorithm transforms a consensus abstraction (together with a reliable broadcast abstraction) into a total-order broadcast abstraction.

Describe a transformation between these two primitives in the other direction, that is, implement a (uniform) consensus abstraction from a (uniform) total-order broadcast abstraction.

Exercise 3 (Solution)

Given a total-order broadcast primitive TO, a consensus abstraction is obtained as follows:

When a process proposes a value *v* in consensus, it TO-broadcasts *v*. When the first message is TO-delivered containing some value *x*, a process decides *x*.

Since the total-order broadcast delivers the same sequence of messages at every correct process, and every TO-delivered message has been TO-broadcast, this abstraction implements consensus.

```
upon init do
        decided := false
end

upon propose(v) do
        TO-broadcast(v)
end

upon TO-deliver(v) do
        if not decided then
                decided := true
                decide(v)
        end
end
```