Localized causal broadcast

Sergei Volodin, EPFL MSc student

Implements:
LocalizedCausalBroadcast, **instance** *lcrb*.

Uses:
UniformReliableBroadcast, **instance** *urb*.

upon event <*lcrb, Init*> **do**
V-send := $[0]^N$;
V-recv := $[0]^N$;
lsn := 0;
pending := $\emptyset$;

upon event <*lcrb, Broadcast* | *m*> **do**
W := *V-send*;
W[rank(self)] := lsn;
lsn := lsn + 1;
trigger <*urb, Broadcast* | *self, (DATA, W, m)*>;

upon event <*urb, Deliver* | *p, (DATA, W, m)*> **do**
pending := *pending* $\cup \{(p, W, m)\}$
while exists $(p', W', m') \in$ *pending* such that $W' \leq$ *V-recv* **do**
pending := *pending* $\setminus \{(p', W', m')\}$
V-recv[rank(p')] := *V-recv*[rank(p')] + 1;
if *p'* $\in$ *locality*(self) **then**
V-send[rank(p')] := *V-send*[rank(p')] + 1;
trigger <*lcrb, Deliver* | *p', m*>;

1. **LCURB1: Validity:** If a correct process p broadcasts a message m, then p eventually delivers m.
2. **LCURB2: No duplication:** No message is delivered more than once.
3. **LCURB3: No creation:** If a process delivers a message m with sender s, then m was previously broadcast by process s.
4. **LCURB4: Uniform agreement:** If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.
5. **LCURB5: Causal delivery:** 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.

We say that a message m_1 may have potentially caused another message m_2 , denoted as $m_1 \rightarrow m_2$, if any of the following relations apply:

1. Some process p broadcasts m_1 before it broadcasts m_2
2. Some process p delivers m_1 **from** $p' \in$ *locality*(p) and subsequently broadcasts m_2
3. There exists some message m such that $m_1 \rightarrow m$ and $m \rightarrow m_2$

Proof. We argue that the algorithm implements $LCURB$. We note that ATTA the meaning of *V-recv* is the number of messages currently delivered from other processes. The meaning of *V-send* is the current number of dependencies for a newly sent message and this is respected by the algorithm because the newly sent message must depend on a message received from some other process only if that process is inside the set of dependencies. In case the process is not in the set of dependencies, the send vector clock is not incremented because the newly sent messages don't depend on the received message.

We note that a set of dependencies of a message $dep(m)$ has a following property: if a message $m' \in dep(m)$, then $dep(m)$ contains all previous messages sent by the sender of m' because of the FIFO ordering of the relation $\rightarrow$. Therefore, the set of dependencies is 1-to-1 mapped to a vector clock. Therefore we send, together with a message, the current set of dependencies of a message W=*V-send* represented by a vector clock.

1. **LCURB1**. Suppose that a correct process p broadcasts m. ATTA message was broadcast with the current value *V-send'*. We note that ATTA *V-send'* $\leq$ *V-recv'* and also both of these vectors' entries can only increase. Therefore, when instance *urb* would deliver (m, *V-send'*), it would be true that *V-send'* $\leq$ *V-send* $\leq$ *V-recv*, and, therefore, the message would be *crb*-delivered

2. **LCURB2, LCURB3** Since every *crb* delivery corresponds to a previously *urb*-delivered, we have no duplication/creation by properties of *urb*

3. **LCURB5**. Suppose that a process has delivered a message m with vector clock W. Therefore, by the meaning of W this message has sequence numbers of dependencies sent by i: $1..W(i)$. According to the algorithm, before delivery it must be that $W \leq V$-recv. This implies that the process has delivered all the dependencies of m.

4. **LCURB4**. Suppose that any process p has delivered m. We prove that correct process p' would also deliver m. Since p has delivered m, by **LCURB5** it has delivered every $m' \rightarrow m$. Therefore, they were *urb*-delivered. Therefore by agreement property of *urb* we have p' also delivering m and all $m' \rightarrow m$. ATTA this would imply that p' would deliver m.

□