16-124-836
Marcel Zauder

8.1 Total-Order Broadcast using Consensus

(a) Consensus does not sort *deterministically* before *tob-delivering*

A consensus decides on a set of messages to deliver. A set normally has in many practical implementations no determinsistic order and therefore all processes could deliver the same set but in a different order which violates the total order property.

(b) Modifying algorithm s.t. we do not require the sorting of decided payload messages

To create a total-order broadcast that does not require the sorting of decided payload messages we can change the implementation s.t. each process proposes a single message to the consensus. The consensus will the agree on delivering one message at a time. This guarantees that all processes deliver the messages in the same sequence.

8.2 Atomic Register as a Replicated State Machine

```
upon event ⟨nnar, Init⟩ do
  $val = \emptyset$
  $readCount = 0$
  $writeCount = 0$

upon event ⟨nnar, READ⟩ do
  trigger ⟨tob, Broadcast | READ⟩

upon event ⟨nnar, WRITE | $v$⟩ do
  trigger ⟨tob, Broadcast | [WRITE, $v$]⟩

upon event ⟨tob, DELIVER | $p$, READ⟩ do
  trigger ⟨pl, Send | $p$, [ReadAck, $val$]⟩

upon event ⟨tob, DELIVER | $p$, [WRITE, $v$]⟩ do
  $val = v$
  trigger ⟨pl, Send | $p$, WriteAck⟩

upon event ⟨pl, DELIVER | $p$, WriteAck⟩ do
  $writeCount = writeCount + 1$
  if $writeCount = |\Pi|$ then
    $writeCount = 0$
    trigger ⟨nnar, WriteReturn⟩

upon event ⟨pl, DELIVER | $p$, [ReadAck, $v$]⟩ do
  $readCount = readCount + 1$
  if $readCount = |\Pi|$ then
    $readCount = 0$
    trigger ⟨nnar, ReadReturn | $v$⟩
```

We can ensure that that this implementation terminates because of the validity and agreement properties of the total order-broadcast. These properties say that any message broadcasted by any correct process is eventually delivered by itself, hence both the WRITE and READ messages will eventually arrive.
Furthermore the property of a total-order broadcast that any set of messages can be linearized to the same linear sequence across mulitple processes leads to atomicity property s.t. each process eventually has the same sequence of delivered messages.
Therefore it follows that any process deliver message r_1 in response to a READ and then returns its locally stored value, any subsequent READ operation with message r_2 - delivered after r_1 - is ensured to return the same or a newer value.

8.3 Replicated Register with Local Read

Because total-order broadcast only guarantees, that any process has the same history of delivered messages $(m_1 \cdots m_{n-1})$ when delivering message m_n, but does not provide any guarantees what the message sequence of other processes is at that moment or will be in the future. Therefore a local read might break atomicity, i.e.:

1. p nnar-writes x, broadcasts $[WRITE, x]$
2. p tob-delivers $[WRITE, x]$, sets $val := x$, nnar-write-returns
3. p nnar-reads, reads locally, returns $val = x$
4. q nnar-reads, reads locally, returns $val = \perp$
5. q tob-delivers $[WRITE, x]$, sets $val := x$

The total-order property holds, as the tob-delivered messages of both processes are $([WRITE, x])$. This implies that both state machines are in sync, but the atomicity property of a regular register is violated because the second read returned the earlier value that the read preceeding it.