16-124-836
Marcel ZAUDER

9.1 Flooding Consensus

(a) Can one reduce the number of rounds for some process?

The number of processes is $N = 2$, we can only consider to reduce the number of rounds in either p or q to at least $N - 1 = 1$ rounds. Therefore we can establish the following execution, where the number of rounds of p is reduced to 1:

- p proposes x, broadcasts $[PROPOSAL, 1, \{x\}]$
- q proposes y, broadcasts $[PROPOSAL, 1, \{y\}]$
- p beb-delivers $[PROPOSAL, 1, \{x\}]$ and $[PROPOSAL, 1, \{y\}]$.
- As p has received messages from all healthy processes, and $r = N - 1 = 1$ it decides x.
- p crashes. q detects p as having crashed.
- q beb-delivers $[PROPOSAL, 1, \{y\}]$. As it has received messages from all healthy processes, it proceeds to round $r = 2$.
- q broadcast and beb-delivers $[PROPOSAL, 2, \{y\}]$.
- As q has received messages from all healthy processes, and $r = N = 2$, it decides y.

Because the two processes have decided on different values, this implementation violates the *uniform agreement* property (the same would hold when lowering the rounds of q because of symmetry). Therefore it is not possible to lower the rounds of any process.

(b) Is this algorithm correct if the failure detector is not perfect but only eventually perfect?

We can establish the following implementation:

- p proposes x, broadcasts $[PROPOSAL, 1, \{x\}]$
- q proposes y, broadcasts $[PROPOSAL, 1, \{y\}]$
- p beb-delivers both messages, proceeds to round $r = 2$, broadcasts $[PROPOSAL, 2, \{x, y\}]$
- q suspects p of having crashed
- q beb-delivers $[PROPOSAL, 1, \{y\}]$, proceeds to round $r = 2$, broadcasts $[PROPOSAL, 2, \{y\}]$
- p beb-delivers $[PROPOSAL, 2, \{x, y\}]$ and $[PROPOSAL, 2, \{y\}]$, decides x
- q beb-delivers $[PROPOSAL, 2, \{y\}]$, decides y.

Again the two process have decided on different values and the usage of an eventually perfect failure detector does not lead to a correctly working algorithm because of the resulting problems shown in the example.

9.2 Leader-Driven Consensus

(a) Why does this algorithm require a majority of correct processes?

To establish a quorum a majority of correct processes is required. This follows from the property that two quorum must overlap in at least one process and therefore the size of any quorum must be greater than $\frac{N}{2}$ - which is only guaranteed to be achieved when the majority of processes is correct.

(b) Which property is violated if there is no majority of correct processes?

If there is no majority of correct processes then the *lock-in* property can be violated which leads to a violation of the *uniform agreement* property.

(c) Execution example (4 processes)

We consider a system with four processes (p, q, r, s with ranks 1, 2, 3, 4). We relax the requirement to $F = \frac{N}{2}$ (which also effects the quorum size). Therefore no majority of correct processes is required. The folowing execution can be established:

- An epoch starts for all processes, with p as a leader and epoch timestamp 1
- p proposes x
- p broadcasts a read
- p and q respond with their state $(0, \perp)$
- As neither process knows of a previously-decided value, p broadcasts a write for value x
- p and q store $(1, x)$ and respond with an accept
- p receives the two accepts, and broadcasts a decide for value x
- p and q receive the message and decide x
- p and q crash
- A new epoch starts, with r as leader and epoch timestamp 3
- r proposes z.
- r broadcasts a read
- r and s respond with their state $(0, \perp)$
- As neither process knows of a previously-decided value, r broadcasts a write for value z
- r and z store $(3, z)$ and respond with an accept
- r receives the two accepts, and broadcasts a decide for value z
- r and s receive the message and decide z

Because processes p and q are deciding differently than r and s (deciding on the value x instead of y) this execution violates the *uniform agreement* property.

9.3 Leader-Driven Consensus, *optimized*

The epoch consensus needs the READ and STATE messages to establish the *lock-in* property. This is especially the case when a leader from an earlier epoch has decided on any value, the leader of the current epoch must decide on the same value, as well.
If we are speaken of the first epoch, there is no epoch "predating" it and therefore there cannot be an earlier decision. Because this implies that all processes are guaranteed to have the state with which they were initialised with, this set of messages can be left out.