Implementing a reliable and ordered multicast protocol

February 2020

Reliable and Ordered Multicaster (ROM)

This protocol is based on a fixed sequencer algorithm, ring leader election algorithm, reliable multicast algorithm and causal-total ordering.

One node is elected as the sequencer by ring leader election (RLE). The node with the highest cpu capacity is elected as the sequencer. The sequencer is responsible for ordering messages. If the sequencer fails, a new leader election is initiated and the node with the highest local sequence number is elected as the new sequencer. RLE is optimized by only forwarding the highest value, either cpu capacity (for the first leader election) and later local sequence number, to its neighbor and thus reducing message overhead.

How does the protocol satisfy the reliability requirements?

A reliable multicast is one that satisfies *integrity*, *validity* and *agreement*. This report goes on to discuss each property separately and additionally describe the ordering mechanism of the protocol and node failures management. Furthermore, we assume that processes are connected by reliable channels with no membership service. Moreover, each message has a initial sender attached to the message and is assumed to be unique and of a certain type to simplify identification and ensure authenticity.

How does the protocol satisfy *Integrity*

Each correct process delivers a message m at most once by storing each delivered message. Each node makes sure the set of delivered messages does not contain the received message m before delivering m.

How does the protocol satisfy *Validity*

Each correct process that multicasts message m will eventually deliver m. To multicast a message m, a node p adds m to a HashSet queue and sends m to the sequencer. Upon receiving m, the sequencer attaches a sequence number to the message m and broadcasts m to the network. Upon receiving message m from sequencer, node p delivers message m and removes it from its queued messages. Furthermore, upon receiving message m from sequencer, each node multicasts m to the rest of the network in case p sends m to sequencer who then attaches a sequence number to the message m and sends m to $q \neq p$ and crashes. Node q will then, upon receiving message m from sequencer, broadcast m to the rest of the network and p will eventually deliver m. Moreover, if sequencer fails upon sending message m, the message will be resent when a new sequencer has been elected.

How does the protocol satisfy *Agreement*

If a correct node delivers message m, then all other correct nodes will eventually deliver m. Considering that all messages are sent to the elected sequencer, who will upon delivery multicast m to rest of the network, implies that all other nodes will eventually deliver m, assuming each node satisfies *integrity* and *validity*.

How does the protocol satisfy the ordering requirements?

To broadcast a message m, a node sends m to the sequencer. If the multicast of a message m casually precedes the multicast of message m', then the sequencer does not deliver m' unless it has previously delivered m, and instead queues it for delivery until m is delivered. Upon delivering message m, the sequencer then attaches a sequence number to the message m and broadcasts m to the network. The latter delivers messages according to the expected sequence numbers from the sequencer and thus satisfying total order.

To guarantee causal ordering, each node keeps a variable for labeling the order of sent messages. The order is attached to each message upon sending it to the sequencer and then incremented. The sequencer keeps track of what message number is expected from each node. Furthermore, each node keeps track of what sequence number they expect from the sequencer by a local sequence number variable. Upon delivery, each node increments the local sequence number and updates the next expected message from the initial sender of the message. Thus, if the sequencer fails, each node has an updated list of next expected message for the other nodes and can therefore be elected as the new sequencer.

How does the protocol deal with crashing processes?

Messages are saved in a queued in case no sequencer is elected or sequencer fails. Upon electing sequencer, messages are resent and removed from queue upon delivery. Nodes that fail are removed from the ring topology initiated upon leader election. All requests for forwarding message to neighbor during leader election are saved in set of requests. If node crashes during leader election, the message is resent to the requested nodes (neighbor's) successor.