Causal broadcast with vector clocks

Large Scale Distributed Systems

Objectives

Implement a causal broadcast algorithm based on vector clocks. Use Maelstrom to help in debugging tentatative implementations towards a correct version.

Software

Have a JVM and node.js installed. Download the version of Maelstrom made available in a self-contained jar, which defines a *workload* named *cbcast*, used with `-w cbcast`.

Tasks

The *cbcast* workload aims to test a causal broadcast involving a set of nodes, where each node broadcasts messages to others. The workload is to be used with a single client process per node, to issue node events, here the `cbcast` request, that has a `message` field. As Maelstrom does not allow messages which are not replies to some request, `cbcast` should be replied to by the list of events that have ocurred since the previous client-issued event, here the list of messages that were "delivered" to the node, in the `messages` field of the `cbcast_ok` message. The algorithm should then accumulate in a list the messages that it "delivers", to use as reply to the next `cbcast` request, resetting the list then.

1. Study the code of a trivially wrong implementation, in `wrong-causal-broadcast.py`, which "delivers" each received message immediately.
2. Run the code in Maelstrom and observe the incorrect behavior, analysing execution histories and message diagrams, using some nodes, but only one client per node, e.g., with:
   `--node-count 3 --concurrency 1n`
   To see incorrect behaviour we need to induce variable message latency, e.g. with:
   `--latency-dist uniform --latency 500`
3. Modify the code to implement the causal broadcast algorithm based on vector clocks. Use Maelstrom to help in making the implementation correct, observing histories and message diagrams involving several nodes.