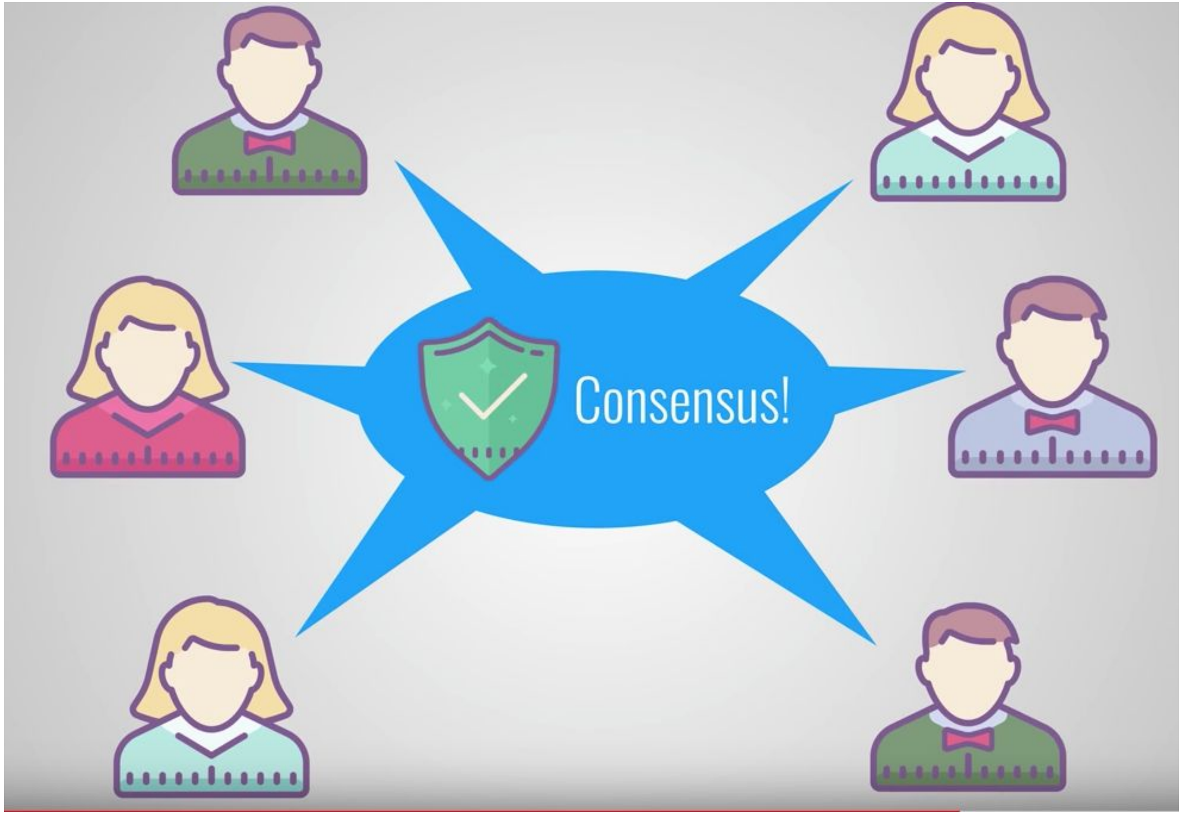
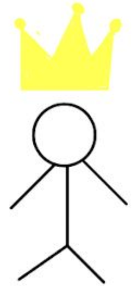
Transport Paxos To Cloud Environment

Paxos Algorithm

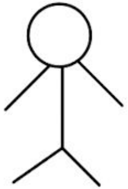
What is Paxos?



Roles in Paxos

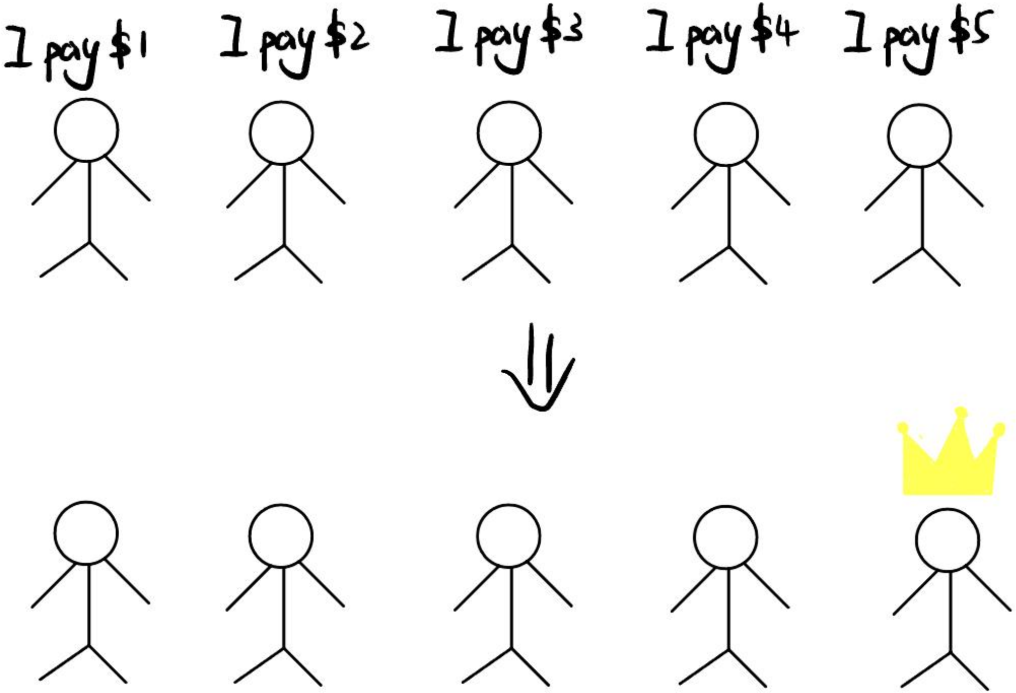


Leader: collects information from learners and makes decision on which value

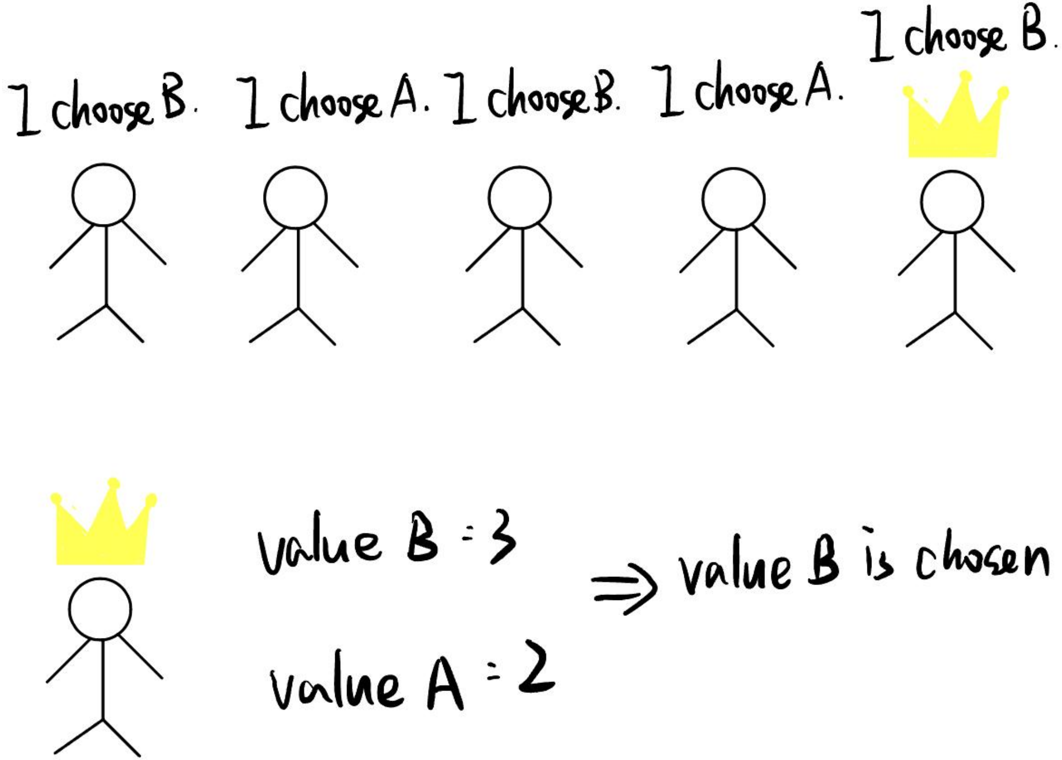


Learner: vote for values and learn the decision from the leader

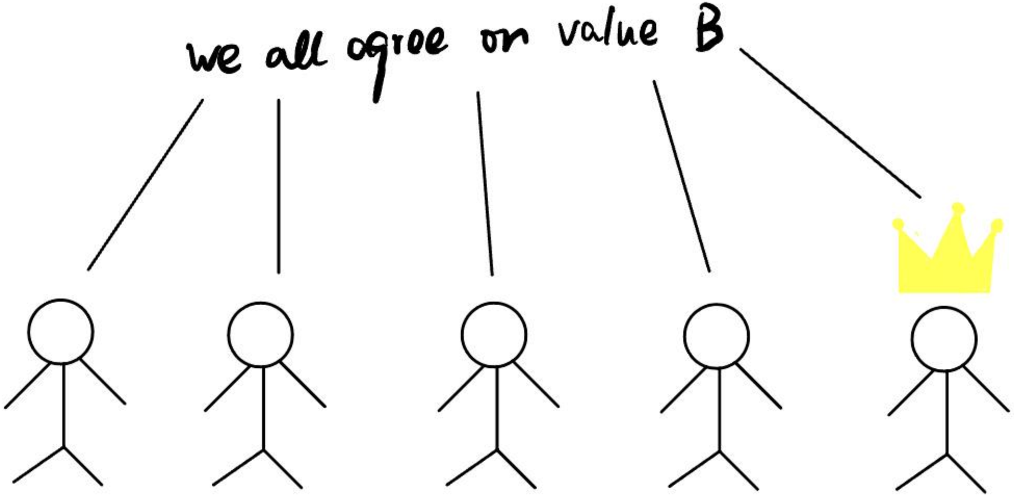
Leader Election(first consensus)



Leader Makes Decision(second consensus)



Learner learn the decision



My Implementation

Application: key/value store between client and servers

Client: put(key, value) and get(key)

Server: reply putOK or value of the key

Learner: vote for client request

Leader: decide which request execute first and store changes to database

Why servers vote for different client request?

The answer is unreliable network.

Example: delayed message

Round 1: every server votes for request 1, however, one vote message is delayed.

Round 2: other servers vote for request 2, however, the delayed message is arrived.

Leader will consider delayed message as a vote in round 2

Paxos Client

1. broadcast command to all server
2. only accept the first correct result from one server, and ignore repeated result

Paxos Server

Phase 1: leader election

1. every server start with a ballot number
2. every server broadcasts their own ballot numbers to all servers
3. server keep track of received ballot number
4. server only vote for the one with ballot number that is higher than its current recorded ballot number
5. server with >50% votes become the leader(majority calculation)

Paxos Server

Phase 2: stable leader and decision notification

1. leader broadcast command to all servers
2. leader and other servers execute the command from the client
3. leader send the result back to the client

Unreliable Network

How paxos system handle delayed or lost message?

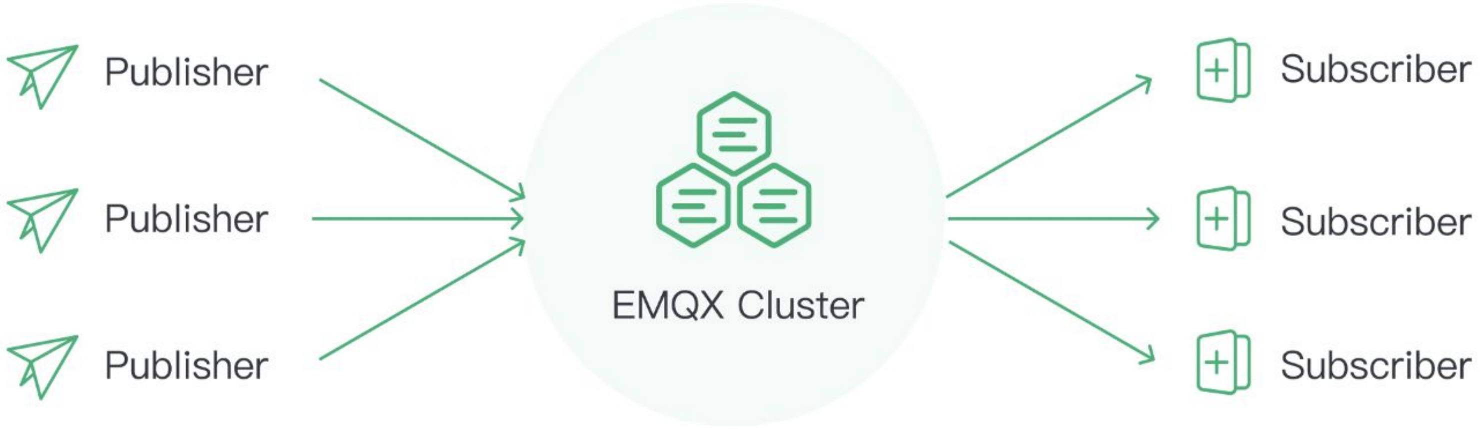
Answer: resend mechanism

1. For almost every type of message, there is a timer.
2. When time up, the message will be resended until it get desired reply

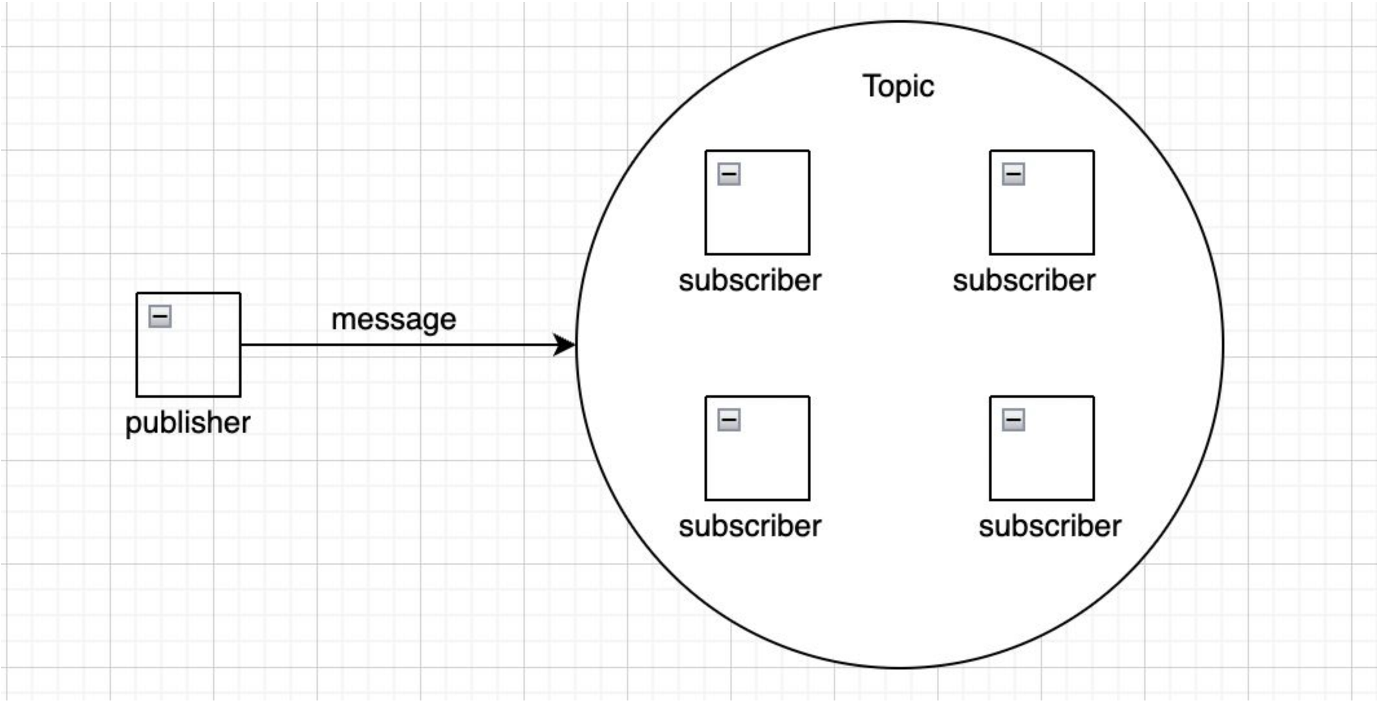
EMQX

Paxos Communication

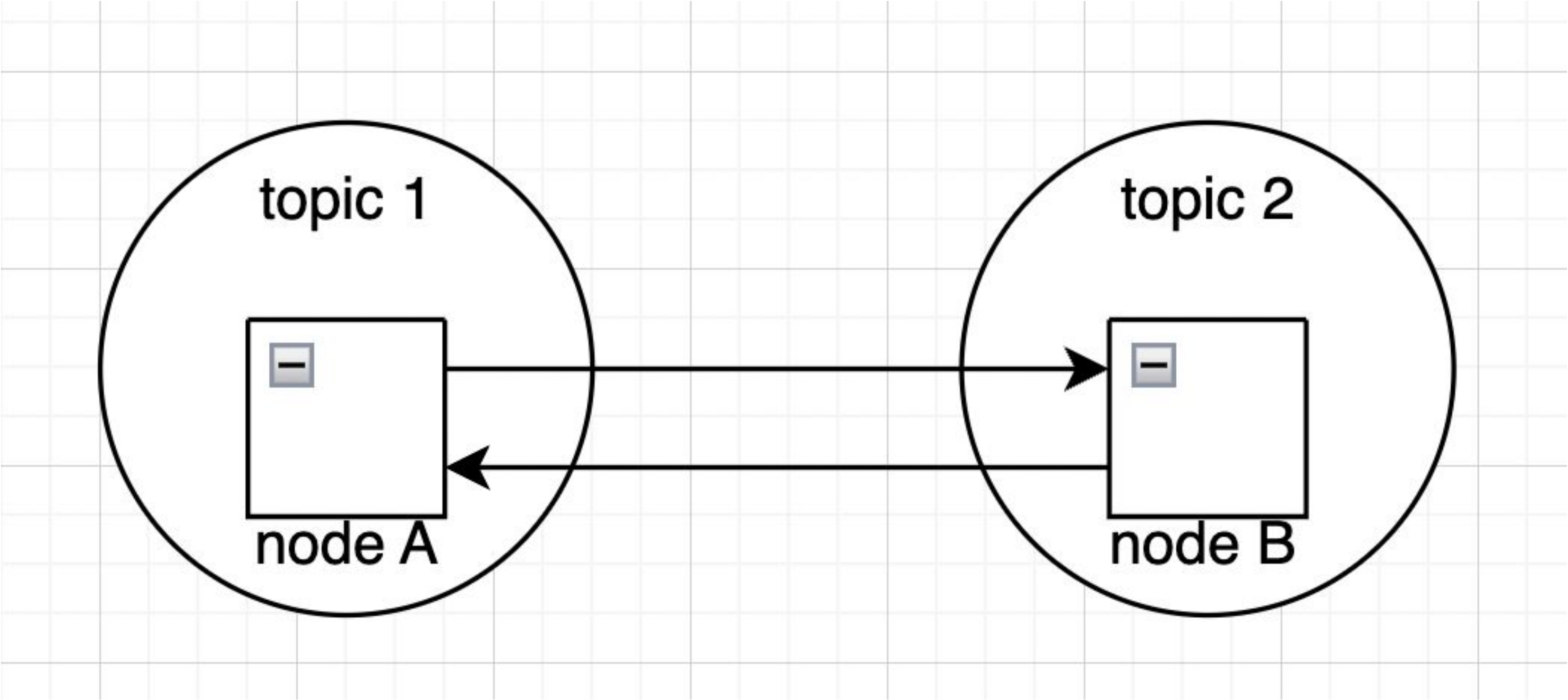
EMQX (MQTT-Message Queuing Telemetry Transport)



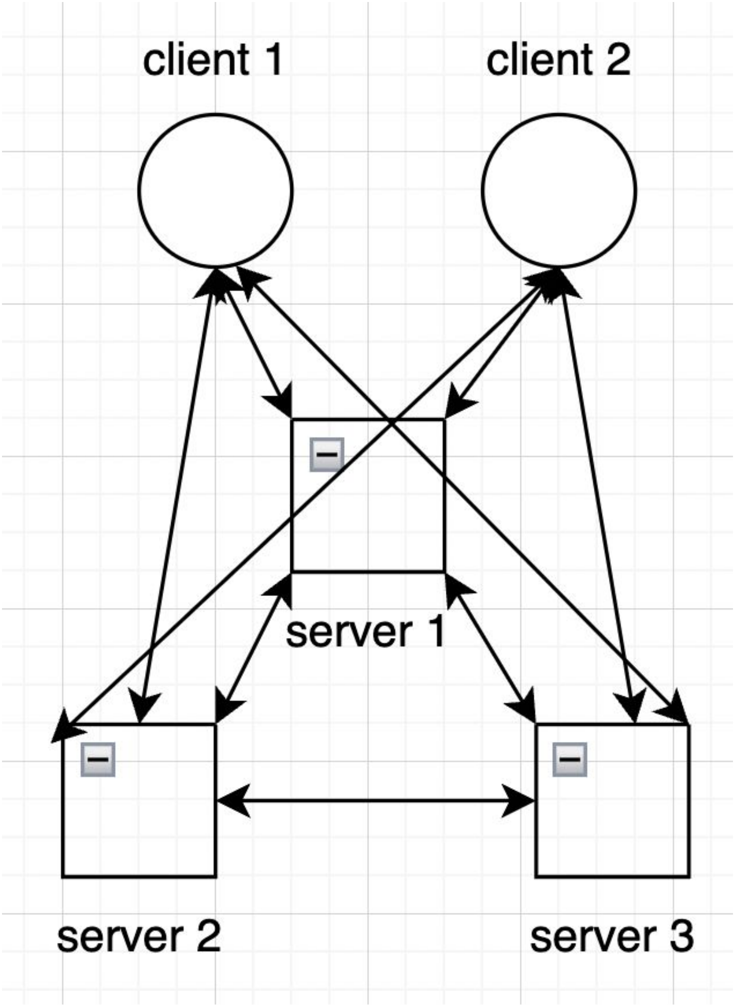
EMQX topic


Topic
subscriber
subscriber
message
publisher
subscriber
subscriber

My Design of Node Communication

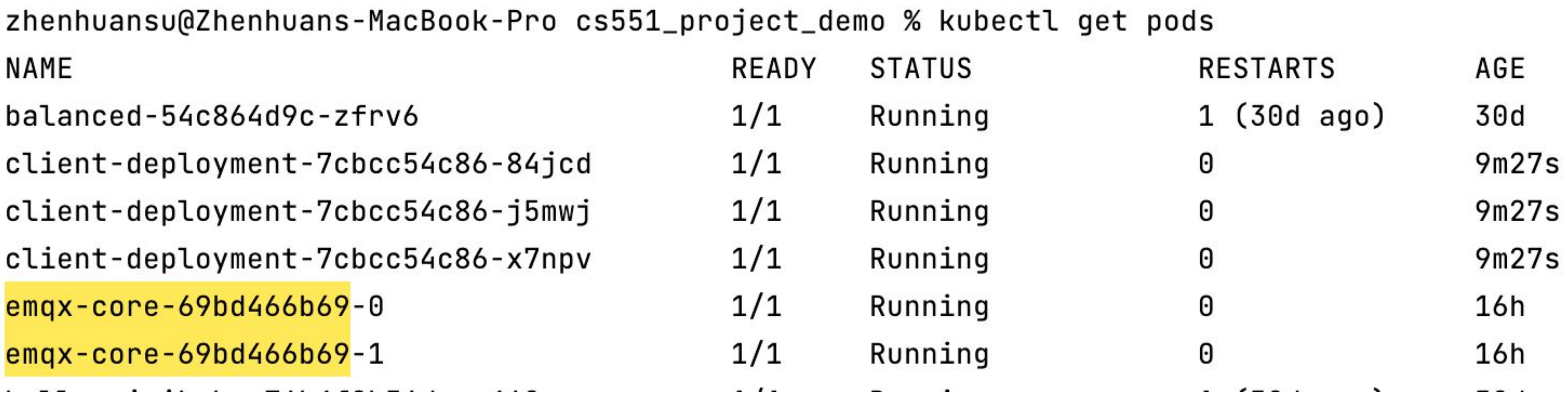


Example of a simple paxos system



Kubernetes

EMQX in K8s

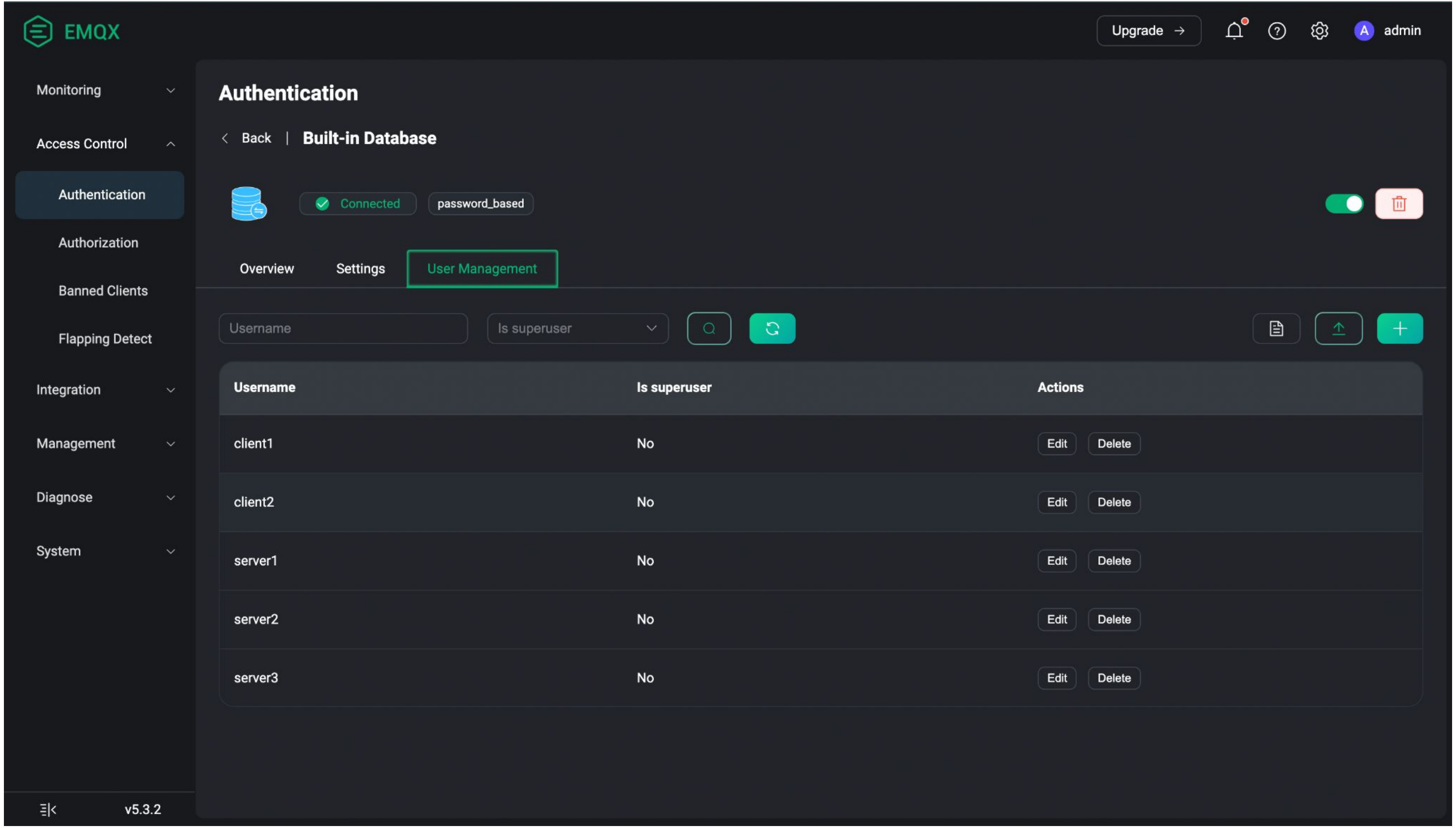


```
zhenhuansu@Zhenhuans-MacBook-Pro cs551_project_demo % kubectl get pods
NAME                                 READY   STATUS      RESTARTS       AGE
balanced-54c864d9c-zfrv6             1/1     Running     1 (30d ago)    30d
client-deployment-7cbcc54c86-84jcd   1/1     Running     0              9m27s
client-deployment-7cbcc54c86-j5mwj   1/1     Running     0              9m27s
client-deployment-7cbcc54c86-x7npv   1/1     Running     0              9m27s
emqx-core-69bd466b69-0               1/1     Running     0              16h
emqx-core-69bd466b69-1               1/1     Running     0              16h
```

Deployment of client and server

1. every client or server node is inside of one pod
2. All of pods communicate based on multiple topics through EMQX cluster

EMQX dashboard



Authentication

< Back | **Built-in Database**

Connected   password_based

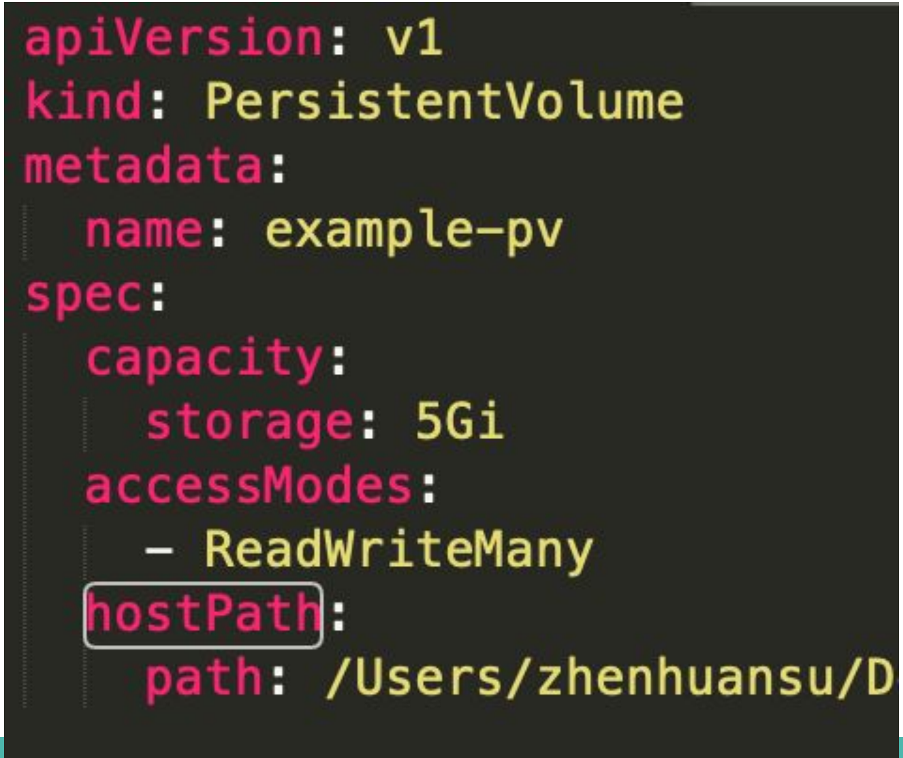
Username	Is superuser	Actions
client1	No	Edit Delete
client2	No	Edit Delete
server1	No	Edit Delete
server2	No	Edit Delete
server3	No	Edit Delete

Shared Files among Pods

1. I have multiple testcases with different configuration of node size
2. every server needs to know the the size of server to calculate majority number
3. every server also need to know if other servers connect to the EMQX cluster so it can start to run Paxos algorithm

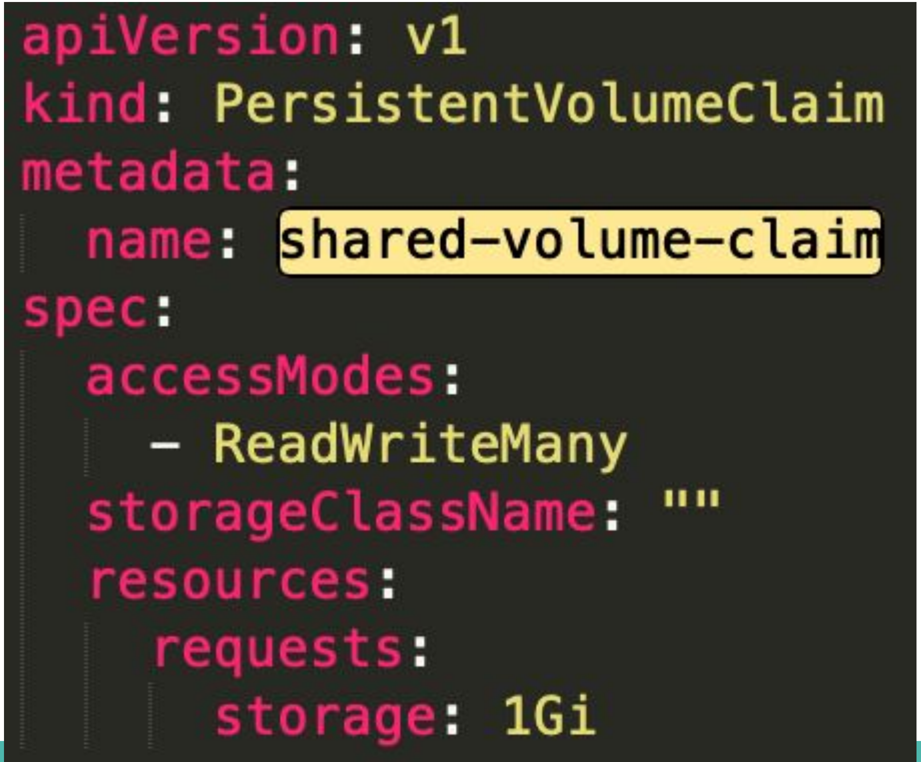
Persistent Volume

Persistent Volume



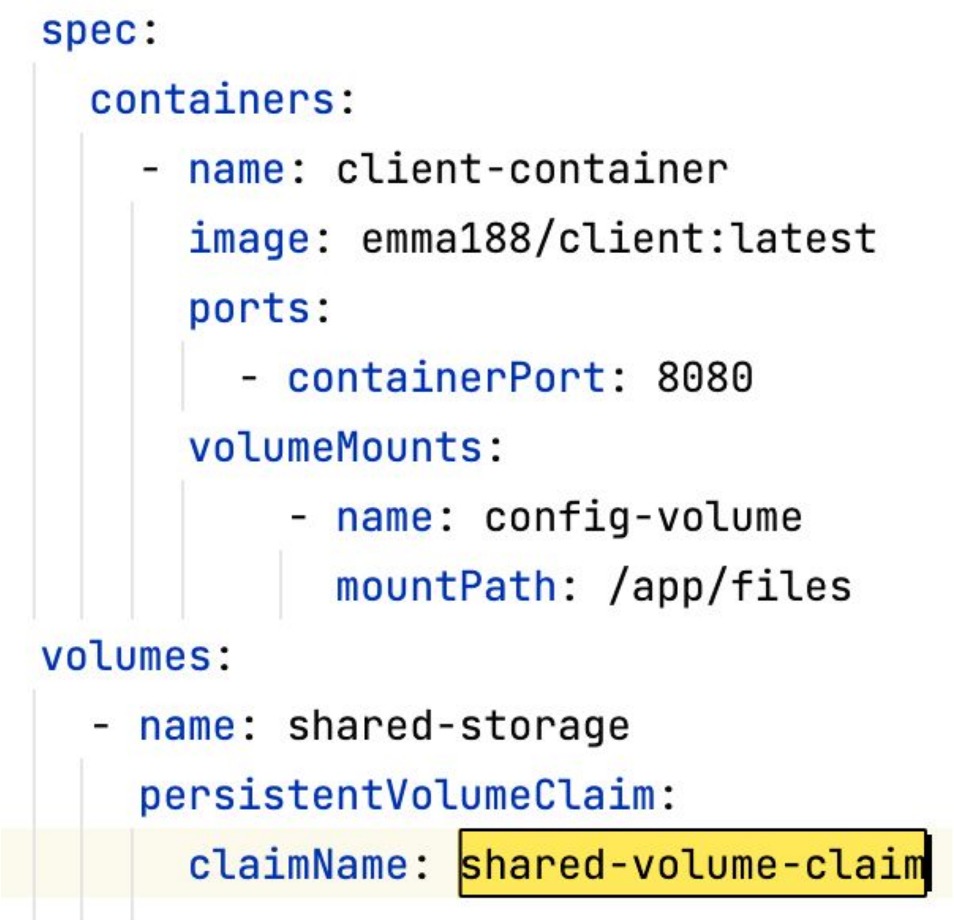
```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: /Users/zhenhuansu/D
```

Persistent Volume Claim



```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: shared-volume-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Gi
```

Pod Volume

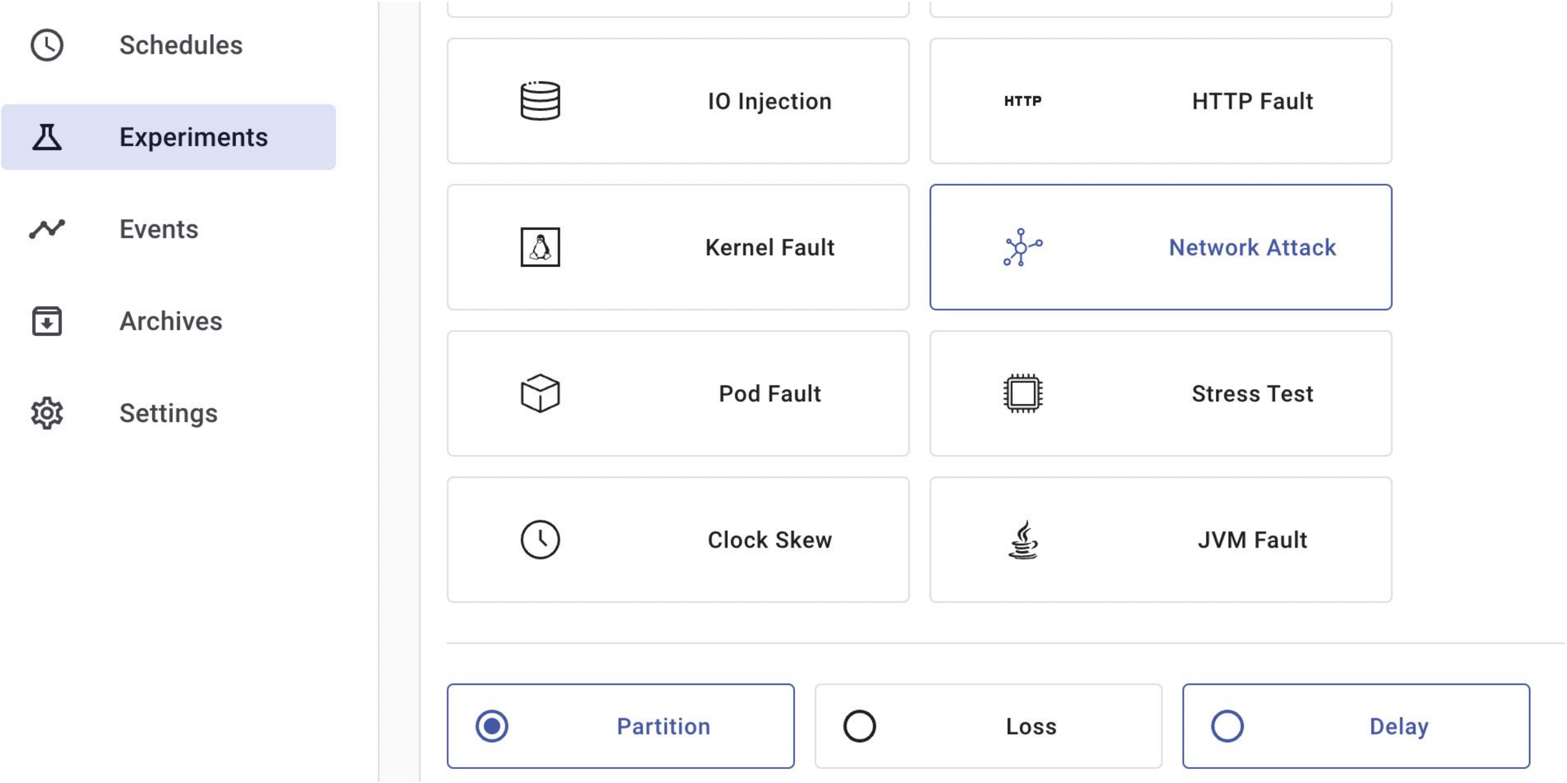


```
spec:
  containers:
    - name: client-container
      image: emma188/client:latest
      ports:
        - containerPort: 8080
      volumeMounts:
          - name: config-volume
            mountPath: /app/files
volumes:
  - name: shared-storage
    persistentVolumeClaim:
      claimName: shared-volume-claim
```

Chaos Mesh

unreliable network

Chaos Mesh experiment



What is the principle of network failure in chaos mesh?

Chaos Controller: Chaos Mesh defines a custom resource called NetworkChaos that allows users to specify chaos experiments related to network behavior.

Chaos Daemon: Chaos Daemon running on each targeted node receives the instructions from the Chaos Controller. It uses iptables rules to manipulate network traffic.

Complexity of Paxos in terms of number of messages

N is the number of servers

Phase 1: O(N x N) every node send its ballot number to each other for leader election

Phase 2: O(N) leader send command to other servers.

This is only under the reliable network condition

My Implementation

I have 6 types of message and 5 timers

Example:

HeartBeatMessage: the message that leader broadcast to other servers to let them know that leader is still working.

HeartBeatTimer: leader broadcast HeartBeatMessage to all the servers every 100 millisecond.

My test case

Test goal: test paxos system under bad network condition

Configuration: 7 servers and 5 client

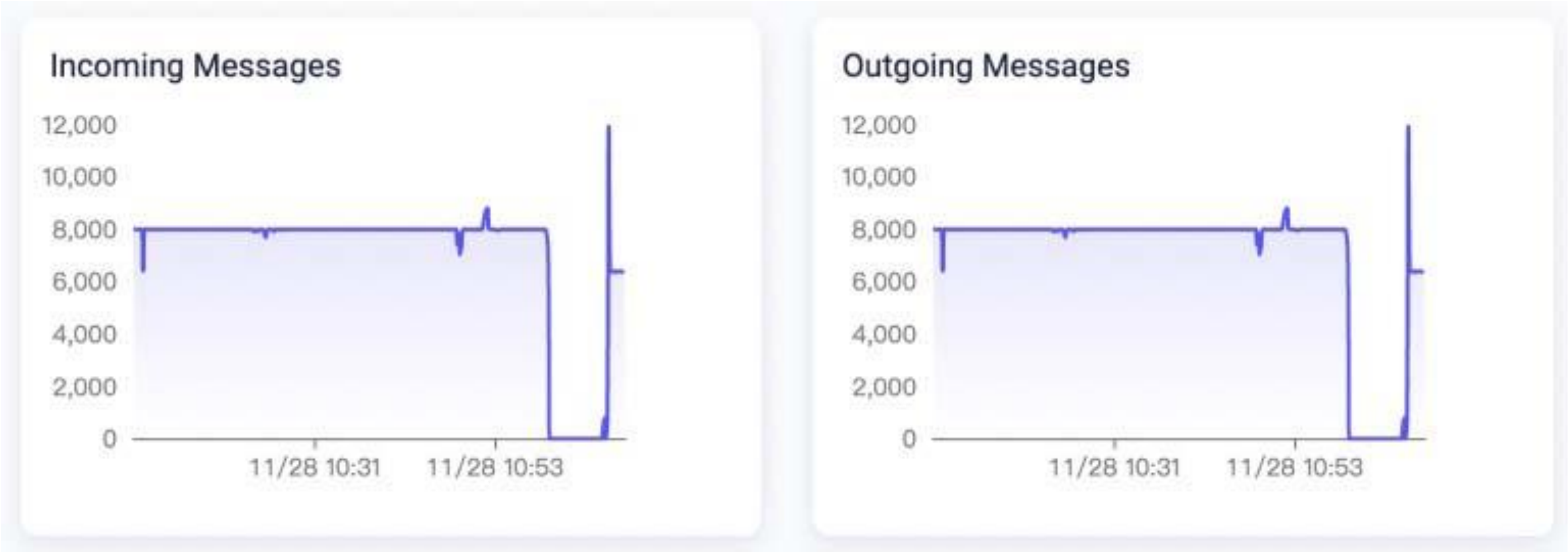
Network condition: Message loss: 20%, Message delay: 10 ms, Multiple network partitions

Details:

client: constantly send their command

server: randomly split servers into 2 partition groups in size of 4 and 3. Do the repartition 5 times.

Result


Incoming Messages
12,000
10,000
8,000
6,000
4,000
2,000
0
11/28 10:31
11/28 10:53
Outgoing Messages
12,000
10,000
8,000
6,000
4,000
2,000
0
11/28 10:31
11/28 10:53

Question?