State machine replication

Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. CSUR 1990

State machine replication

- What is the system model? (What is our world?)
  - *Deterministic* state machine (with replicas): sm_i. Same seq of commands => same state.
  - Clients: submit commands to the state machine
    - Why are clients distinct from replicas?
    - Clients may be non-deterministic, less stateful.
    - Clients don't have to be as powerful as replicas nor as reliable ($ argument: many cheap clients, and a few expensive replicas)
  - Output device. Generality — separate the (passive) consumer/receiver of the state machine output. (Something that needs the notification of command execution). Actuators.
- What types of failures do we care about?

State machine replication

- Challenges with achieving a deterministic state machine. (Depends on what is a state machine)
  - Concurrency (for perf)
  - Correctness (correct implementation)
  - Synchronization of code between replicas: they have to run the same state machine.

State machine replication

- What types of failures do we care about?
  - Fail-stop failures (weaker: non-paranoid). Failure in which a device either works or doesn't. Observable by other components (disinfect from halting failures, where observability is not provided).
  - Byzantine failures (stronger: paranoid). Some # of processes can have *arbitrary* behaviour. Dangerous!
    - Includes software bugs, hardware failures, cosmic rays
    - *"A horse that is electrocuted and falls onto the power cables, disconnecting the data center"*
    - Includes malicious (attacker) behaviour: hacking!
  - In general failures are detected by "failure detectors", which is a large research area.
- What about the network?
  - Generally FIFO (first in first out): ordering on the wire (fairly unrealistic)
  - In reality, msgs may be dropped, no set route (msgs can take different paths)

State machine replication

- Why do we want replication? (Another benefit of distributed state?)
  - Fault tolerance
  - Better performance (maybe), e.g., use replica closer to me (in network distance)

SMR+ failure model

- Assume that at most t replicas can fail; design a system to withstand this number of failures.
- Fail-stop: *need at least t + 1 replicas.* Need only one replica to be working (the other t can fail). I trust this last remaining replica to work correctly.
- Byzantine: need at least 2t + 1 replicas. You don’t know which t are the byz replicas. Use voting to determine the “true” behaviour. *This works because t + 1 are correct, and t+1 is always a majority in a set of 2t + 1.*

State machine replication

- **Agreement**: All replicas receive all requests (as a set)
  - Why is the paper quiet on how to achieve agreement?
  - Network is FIFO… it's also reliable.
- **Order**: Replicas execute requests in same relative order (this order is indep. of client order)
  - *If a client issues a, then b; SMR executes a before b.*
  - Many ways to achieve it!
  - Assume that we have IDs on requests; ***request stability****:* a request is stable if once no request from a correct client bearing a lower id can be delivered to the replica.
  - Order can be achieved: **if a replica next executes <u>the stable</u> request with the <u>smallest unique identifier</u>**

State machine replication

- Lamport-clock based ordering/stability algorithm. Order / stability is achieved with logical clocks.
- **The bad**
    - Must wait for all clients to send a message *after* some request (to determine it's stability)
    - If a client has no request to send… they must send a no-op (communicate timestamp to replica)
    - All-to-all connectivity. High bandwidth. *Every client needs to know every replica.*
    - *If I have many many clients (10^6), this is a terrible approach. Doesn't scale in number of clients.*
- **The good?**
    - No communication between replicas (they don't need to know each other)
    - It's very simple. Replicas are trivial. Clients are more complex.

State machine replication

- What about physical clocks? Same structure, but instead of lamport clocks, use assumptions about clock speed.
  - Hard bounds on estimate of propagation delay (latency between nodes)
  - Hard bounds on estimate on the clock synchronization between clients
  - Then formulate stability as a mathematical formula that constraints what timestamp can appear in the future given a timestamp in the past.

State machine replication

- Replica coordination as another approach to derive order/stability.
- This is the more tradition RSM technique
- **Paxos, Raft:** solve both agreement and ordering simultaneously in an *async network.* (Both suffer from the FLP result, which means that they are not always live & safe).