Paxos Made Simple

Key Takeaways

- Paxos is a protocal to reach consensus in a distributed system with unreliable network and machines.
- This paper talks about the algorithm of Paxos and how it can be derived naturally from properties of consensus of distributed systems.

The Consensus Algorithm

- Paxos is a protocal to solve consensus in a network of unreliable processes.
- The processes can operate at arbitrary speed, may fail by stopping, and may start. On the other hand, we assume non-Byzantine model, i.e. the processes will not try to sabotage the information.
- The processes communicate with each other through messages sent across network. The network is not reliable: the messages are sent asynchroneously and may take arbitrarily to deliver, and the messages can be lost, reordered, or duplicated. The messages are not corrupted, as the failure model is non-Byzantine.

Roles:

- Proposer: A proposer advocates a client request and attempt to convince the acceptors to agree on the request by sending the proposal to a set of acceptors.
- Acceptors: they act as the fault-tolerant memory of the voting system.
- Learners: Learners take action after the consensus has been reached by the acceptors. For example, if the clients request some action, the learners execute the request and send a response to the client.

System Design Logic

- Suppose the system only proposes one value, then this value should be accepted. Therefore, an acceptor must accept the first proposal that it receives. This serves as starting conditions for each acceptor who has not seen any proposal yet, and the acceptor must accept a proposal when it has no history.
- Only a single value can be chosen, and this chosen value must have been proposed by any proposer. Since each acceptor must accept the first proposal they have seen, then if we assume each acceptor can only accept a single value, then it is possible that we have two proposals each accepted by half of the acceptors and the system get stuck. Hence the system needs to allow each acceptor to accept multiple values.
- Since multiple proposals can be accepted, this means multiple proposals can be chosen. In order to ensure only a single value is chosen, we need to ensure the chosen proposals have the same value. Suppose each proposal has a sequence number n and a proposal value v. Suppose we can guarantee that if a proposal with value v is chosen, then every higher numbered proposal that is chosen also has value v. Then we can guarantee all chosen proposals have the same value v.
- The above condition can be made more stringent by assuming if a proposal with value v is chosen, then every higher numbered proposal accepted (instead of chosen) by any acceptor needs to have value v.
- If we allow multiple proposers, then it is likely that proposal 1 and proposal 2 propose different values. Some acceptors that have not seen any value will accept proposal 1 and some acceptors that have not seen any value will accept proposal 2, since they are required to accept any proposal if they haven't seen any. This violates our setup that any accepted proposals by any acceptor needs to have the same value v as a chosen v. Therefore, we need to have the requirement that if a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer needs to have value v.
- In order to ensure the above requirement, we maintain the following invariant: for any value v and proposal number n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors so that either (a) no acceptor in S has accepted any proposal numbered less

than n, or (b) v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S.

- In order to maintain the above invariant, for a majority set S, the proposal wants to know whether the acceptor has accepted or will accept any proposal numbered less than n, and whether its value v is the value of the highest-number proposal among all less than n accepted by those in S previously or in the future. It is easy to know what is already accepted, but it is hard to predict whether it will be the case in the future. Instead of predicting the future, the proposer asks the acceptors to promise there won't be any acceptance of proposal with number less than n.
- Therefore, a proposer will choose a proposal number n and sends a request to each of some set of acceptors, asking it to respond with a promise never to accept a proposal with number less than n, and the proposal with the highest number less than n that it has accepted. This request from proposer to acceptor is called a prepare request.
- The proposer needs to know the previously accepted value with number less than n because it can only propose value the same as the value accepted among all proposes with number less than n.
- After the proposer receives the requested responses from a majority of the acceptors, it can issue a proposal with number n and value v, where v is the value of the highest-numbered proposal among the responses, or is any value selected by the proposer if the responders has not accepted any proposals. This message is called a request message.
- An acceptor can ignore a message without hurting the safety property, hence we only need to specify when the acceptor needs to respond to a message.
- An acceptor can accept a proposal numbered n if and only if it has not responded to a prepare request having a number greater than n, since it would have promised another proposer that it will not accept such proposal.
- An acceptor only needs to remember the highest-number proposal that it has accepted and the number of the highest-numbered prepare request to which it has responded. It must remember this information even if it fails then restarts, hence this information is stored to stable storage to maintain the information that the acceptor must remember. An acceptor records its intended response in stable storage before actually sending the response.

Algorithm

- A proposer selects a proposal number n and sends a prepare request with number n to a majority of acceptors.
- If an acceptor receives a prepare request with number n greater than than any prepare request to which it has responded, then it respondsto the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.
- If the proposer receives a response to its prepare requests from a majority of acceptors, then it sends an accept request to each of the acceptors for a proposal numbered n with value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals existing.
- If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n.

Learning a Chosen Value

- After consensus has been reached by acceptors, the learners must learn about the final agreement.
- The simplest method is for each learner to ask each acceptor about the accepted value, but this takes too many messages.
- A better way is to get a distinguished learner among all learners. The acceptors send the accepted value to the distinguished learner which then forwards the value to each learner. This is possible because it is assumed the failures are non-Byzantine.

Distinguished Proposer

- There could be cases when we have two proposers, each trying to send a proposal to each acceptor, but they values increase alternatively so all previous proposals are ignored and no progress can be made.
- In order to solve the above problem, a distinguished proposer must be selected as the only one trying to issue proposals.

Implementing a State Machine

- A distributed system can be implemented as a collection of clients issuing commands to a central server, and the server can be described as a deterministic state machine that performs client commands in some sequence.
- A collection of servers are used for fault-tolerance. Since the state machine is deterministic, all servers will produce the same sequences of states and outputs if they all execute the same sequence of commands.
- Paxos consensus algorithm is used to ensure the sequence of commands executed by all servers are the same. Each command is determined by a single Paxos process.
- Each server plays all the rols, including proposer, acceptor, and learner. A single server is elected as the distinguished proposer (leader).
- The clients send commands to the leader, and the leader decides for each position of command sequence which command should be put there through Paxos.
- The application of Paxos to a sequence of consensus agreement is called multi-Paxos.