Solana whitepaper

- Users send txns to the cluster
  - Every cluster has a leader node. These are the nodes user talks to, to do PoH
  - Leader nodes generate these hashes, which we'll call txns for now.
  - These txns combine to form a block.
  - Verifier nodes will check to ensure whatever the leader node is generating is actually a valid block.
  - Once verified, they send the end state of verification back to the leader which will decide to commit this block to the ledger or not.

How is a leader selected?

↓

a node chosen from a cluster of nodes to process transactions from users.

- Whichever validator has the most sol staked, will become the new leader.

- Leader schedule: list of identities of all slot leaders.
- This schedule is recomputed locally & periodically. It assigns slot leaders for a duration of time called an epoch.
- To generate leader schedule, we take verifier nodes that are actively verifying the txns and these nodes would be set into an active set, and then from this active set of nodes, we would sort it by the nodes that have the greatest amount of SOL staked into it. From there, leader node would generate an arbitrary seed and this seed is used to arbitrarily pick and assign the nodes in a certain epoch, for the leader schedule.

Proof of History algorithm

PoH is an algorithm that creates a time ordering to allow validator nodes to determine the order of incoming blocks, that the leader generates.

Solana uses PoS which is further enhanced by PoH's algorithm.

PoW with BTC - we have computers randomly trying to guess a hash, so that they can be the first node to produce a block in the btc network, and the block is distributed across every node in the btc n/w.

Problem here is it takes time to propagate those blocks to each validator node. Btc solves this by adjusting difficulty and time needed to solve, allowing validators lot more time to recieve the blocks validated and converge on a certain order.

PoH solves this by creating a verifiable time ordering. All the validator nodes know the order of blocks even if they recieve them in incorrect order.

Hashing is a way to transform a given input to a different output. Deterministic in nature.

How PoH uses hashing?

"Sample text" $\xrightarrow{SHA256}$ hash1

hash1 $\xrightarrow{SHA256}$ hash2 ∞ infinitely

hash2 $\longrightarrow$ hash3, and so on

from text to hash1 to hash2, we can create a concept of time, because we know hash2 must occur after hash1.

The leader node keeps on hashing one after another independently.

Whenever a leader node recieves a transaction from the user, we just take the hash of that transaction, combining it with whatever hash the leader node is currently on, say hash100, and use that to generate our new output which is hash101, and then we use hash101 as the input for hash function to continue generating.

So now, by doing this, we have solidified that transaction1 has occured during the generation of hash 100 giving us proof that transaction one have occured befor transaction 2, for exam

Verifying PoH

The hashes in the final state, we send it to validators to ensure that the block that the leader

generated is valid.

The validator nodes would go through & recalculate all the hashes following the same mechanism that was used to generate it, and then it would calculate the final state which is the final hash that we recieve and then send it back to the leader. The leader recieves a majority of the same final state that it has generated, it will commit the block it has proposed to the ledger.

Reaching block consensus (PoS)

In order to commit the block, we need something called super Majority - 2/3rd of all staked SOL - should support the final state

CAP theorem for distributed systems

In distribute systems, one of the nodes might go down; or the leader node might go down.

CAP theorem solves this.

C - Consistency: All clients see the same view of data, even right after update or delete

A - Availability: All clients find a replica of the data, even in case of partial node failures

P - Partitioning: The system continues working, even in presence of partial network failure.

In CAP theorem, we have to essentially pick 2 of these 3. Partitioning is mandatory - 2 options now - CP or AP.

Solana says, consistency is almost always picked over availability in an event of a partition. So, Solana uses CP.

How Solana handles network failures?

Dealing with Partitions - we always need to ensures that our network is available.

What if we have a n/w issue and not all availability nodes can vote?

Solana

* Solana slows down the block generation process and wait for the nodes to recover.

In the white paper, there were 3 states that we proposed:-

First state → No. of verifier nodes that we have available in the whole n/w is greater than 2/3rd of the n/w → When this happens, we have a very quick unstaking process using a low timeout

2nd Scenario → No. of validator nodes we have is less than 2/3rd of total n/w but greater than 1/2 the n/w → We no longer have super majority, so we can no longer reach consensus we just have to have a longer timeout & have the leader generate more hashes and try to wait for the nodes to come back.

3rd State → No. of validator nodes is less than 1/2 the n/w → Similar to previous state. We have to wait even longer for the validator nodes to come back.

Vertical Scaling vs. Horizontal Scaling

Vertical Scaling: When we hit a CAP limit to the no. of txns we can process, we just add more RAM, CPU & GPU to our servers.

Horizontal Scaling: Instead of increasing CPU, just add more servers. Problem is that, we now have multiple leaders using PoH and this breaks PoH, because each server acts independently of each other. We no longer know the actual ordering of blocks that are being produced when the txn comes in.

Solana's hypothetical solution, each leader will take a different letter to combine with hash and synchronise it every now & then.