Hi all,

This is perhaps an obvious idea, but I haven't seen it discussed, so I'm interested to get your thoughts.

TLDR

: With some protocol changes, it would be feasible to run a highly scalable full node backed by a cluster of servers. The number of full nodes would drop as the resource requirements increased, but validators who don't run full nodes could still participate, delegating block production to other nodes.

I'm still wrapping my head around the stateless client ideas, so I won't try to draw any comparisons, except pointing out that a scheme like this keeps the protocol simpler while making clients more complex. Arguably a good tradeoff since client bugs are easier to fix than protocol flaws.

Validator and block producer roles

Validators and block producers would share mining rewards at a negotiated ratio. Validators wouldn't select producers solely based on the ratio, though; they would also account for historical performance (has the producer effectively maximized gas rewards?) and reliability (has the producer failed to produce valid blocks several times?). New producers can bootstrap their reputation by selecting themselves when it's their turn to be the validator.

At the start of each round, the validator irreversibly declares which block producer they are delegating to. They also decide which parent block the producer must extend. Since most validators will run SPV clients, they will occasionally select an invalid block, but the partnered block producer can correct the mistake by sending a fraud proof and recommending a valid block.

A validator who makes more than one declaration for the same block height would be penalized. This mitigates the nothing-at-stake problem, and also discourages validators from delegating to multiple producers just in case one fails, which would waste resources.

Protocol changes

The protocol would let each server declare a range of shards that it's interested in. Servers would not bother to sync with peers with nonoverlapping shard ranges.

A block header would contain the hash of each block shard. Block shards would simply be arrays of transactions. The grouping need not be specified; block producers could use whatever grouping is convenient.

Full node implementations

A simple approach would be to designate a server for each shard index. The server which owns a certain shard index would be responsible for downloading and verifying shards with that index. For a full node which participates in block production, it would also create and broadcast shards for that index.

As the size of a cluster grows, individual server failures may become common, so some fault tolerance is desirable. A simple fault-tolerant approach is to assign 3 servers which compete to validate and create the same shards. When a server finishes creating a shard, it would write it to a datastore such as CockroachDB, where writes will block until consensus is achieved on which shard proposal is accepted. Once a shard proposal is accepted, all 3 servers will start broadcasting the accepted shard.

Since shards are processed concurrently, some care must be taken to avoid nondeterminism. Read operations like BALANCE

would read the state of the parent block. Write operations like SSTORE

would acquire a lock before performing the write. If another shard has already acquired the lock, it's an illegal conflict, and the entire block is deemed invalid.

Implications of a low node count

Low node counts can be conducive to eclipse or Sybil attacks, but they can always be detected by examining the block rate or PoS participation rate. I think this is an important security practice regardless of node count, since an attacker who has access to the cable outside my home can simulate an arbitrary number of nodes with any IPs of his choosing.

The centralization of block production makes censorship more of a possibility. Still, it only takes a single identity-agnostic block producer to thwart any censorship effort.

As running full nodes becomes increasingly expensive, more individuals and small companies would need to resort to SPV verification with fraud proofs. For extra security, there's always the option of outsourcing full verification to a third party who runs a full node. If someone doesn't want to trust any single entity, they can query multiple verification services and make sure they agree.

51% fork attacks would be difficult, since validators would declare which parent block the block producer must extend. A block producer may attempt to convince a validator (either in or outside the protocol) to select an inferior parent, but the validator should only change their decision if a fraud proof is provided. Validators could be bribed to select an inferior parent without a fraud proof, but we assume that corrupting 51% of validators is impractical.