CC @JustinDrake @vlad as this builds on a lot of ideas you've worked on.

In the sharding spec as it currently exists, and in the current ethereum blockchain, consensus on transaction ordering and state calculation are tightly coupled together. A block contains both a set of transactions, and a post-state root, so it represents a claim about both what the current transaction history is and the state after executing this history. This is convenient in many ways, but also has some significant drawbacks, which we will get into in this post.

The purpose of this post is to explore a different paradigm, one where the consensus process happens only on transaction ordering, and then a separate

process exists to incentivize calculating state roots; that is, making claims of the form "the post-state root of the block with hash X is Y"; correct claims are rewarded and incorrect claims are penalized. The state execution process is at all times aware of the state roots at all block heights that it has already processed, so the state execution process itself is responsible for calculating incentive payouts/penalties for these claims.

Note that this implies that state execution is not a consensus game; even if 90% of executors say that the state root is Y1, from the point of any node that actually has done the calculations themselves (or received a SNARK proving them, or went through a truebit game proving them), if that node sees that the actual root is Y2 != Y1, then the majority of the executors get penalized and their result is thrown out.

Some old research

This is a rabbit hole that I actually went down two years ago, which you can still findn this Casper PoC 1 repo. The idea there was even more radical: not only is state execution a separate process, but consensus on the block at each height

is a distinct and separate process, so for example block 378224 could conceivably finalize before block 378223.

I used a quadratic scoring rule to incentivize claims on state; any validator could bet that the state root at height H is Y, and specify odds U

(eg. odds 5 means that the state root is 5 times more likely to be Y than not, so in general odds U

means probability $\frac{U}{U+1}$

). If a validator makes a correct claim, they are rewarded U

, and if they make a false claim they are penalized $\frac{U^2}{2}$

; the maximum U

is the one where the penalty for loss is a validator's entire deposit. The purpose of this mechanism is to allow validators to make state commitments at various probability levels before the transaction history is even finalized, giving partial info to light clients as quickly as possible.

That said, at this point I favor keeping block consensus chain-based, and allowing claims on state roots with only a single (very high) level of confidence; we can avoid subjecting executors to uncertainty about consensus by making the claims conditional

on the result of the block consensus process, so an executor submitting a claim (H, X, Y) would be incentivized as follows:

- Reward of R if the block hash at height H (which represents the entire history up until that point) is X and the state root is Y
- Large penalty of -D if the block hash at height H is X but the state root is NOT Y
- No reward and no penalty if the block hash at height H is not X.

The nice thing about the separation between execution and consensus is that it allows clients to take advantage of alternative ways of learning the state, like verifying SNARK proofs, playing different kinds of truebit games, calculating it yourself, having a "trust the executors by default but if one of N trusted service providers tells me something's fishy execute it yourself" setup, etc.

Consequences for sharding

This has several kinds of consequences for sharding. First of all, at least theoretically it's possible to validate without having any state calculation logic whatsoever. However, realistically validators will want to learn the state so that they know what transactions they can include that would give them rewards. A collator must have (approximate) knowledge of the up-to-date state

for this reason. They would also still want to have witnesses with transactions so that they can fully execute the transactions and know what their end state is. The infrastructure would thus end up being very similar in many respects, and if this was the end off the story it’s not clear separating the two has large benefits.

However, this is not the end of the story. It is highly desirable that in the sharding system we have a notion of cross-shard transactions. The usual way that cross-shard transactions are conducted is simple:

1. An operation on shard A creates a receipt on shard A (note: receipts are calculated as part of the state calculation process, so think of receipts as being kind of part of the state)
2. The receipt on shard A gets confirmed
3. An operation on shard B incorporates a proof that the receipt on shard A was confirmed, and performs some execution based on this.

However, there is a risk: what if, after this happens, shard A has a large reorg (ie. many blocks get reverted) but shard B does not, and in the new main chain of shard A the original receipt no longer exists? Then, you have a “dangling effect” in shard B without a cause, which is very dangerous and could lead to things like money being created out of nowhere.

We could simply have the reorg on shard A trigger a reorg on shard B, but that would be dangerous as it would be a DoS vulnerability: a small number of attackers reorging shard A could conceivably reorg every

shard, if there is much cross-shard communication going on. The “dependency cone” of A will likely grow quickly. To prevent this, we can only go for the dumb solution: wait for the receipt on shard A to finalize, so that reorgs are simply not possible.

But separating state execution gives us another way out: if shard A does a reorg, then we don’t reorg any transactions

on shard B, but rather we simply let the executors recalculate the state roots. Any operations on shard B that actually do depend on activity on shard A would have their consequences reorg’ed, but any operation on shard B that is not

part of the dependency cone of the receipt would be left alone. Furthermore, it should be possible to calculate ahead of time that some operation on shard B is not part of the dependency cone of something in shard A simply by looking at the access lists of transactions (the access lists would be extended so that transactions can also access historical receipts on other shards), and so users would have private knowledge that their operation on shard B is safe and sound without waiting for confirmation from the global state root. With this kind of approach, we could allow cross-shard transactions to happen very quickly, possibly even allowing transactions to reference receipts from the most recent collation in some other shard.

Statefulness and statelessness

As discussed above, validators in this model do need to be stateful, though it’s more acceptable for them to rely on partial guarantees, as if they use a wrong state root the only bad thing that will happen is that they will lose some fee revenue as they’ll include some transactions that are invalid (and will in this model be no-ops); there’s less systemic risk to the protocol if this happens sometimes, because there will be (stateful) executors on each shard and various kinds of interactive verification games can clean up any mess even if 90% of the players are dishonest.

Validators could

use “see what the executors say about the state 25 blocks ago and execute the state after that point” as a heuristic for determining state, though it would be likely more optimal to use a hybrid strategy like “do that for the last 5 blocks, then download the last 100 blocks without witnesses to verify that they’re valid and available”.

In this model, transaction senders would need to propagate transactions with witnesses for validators’ benefit, though blocks could be broadcasted without witnesses if we rely on executors to be stateful. Note that this would once again imply that state size control is an issue, and so rent or other kinds of storage fees may be optimal.