eip: TBD

title: Alternative ETH2.0 - A Non-Sharding Approach

author: Qi Zhou (@qizhou)

discussions-to: https://ethereum-magicians.org/t/eip-alternative-eth2-0-a-non-sharding-approach/4857

status: Idea

review-period-end: TBD

type: Standards Track

category: Informational

created: 2020-10-15

requires: None

Simple Summary

Propose an alternative ETH2.0 without sharding to satisfy near-future performance needs

Motivation

Due to technical challenges and uncertainties of ETH2.0 with sharding, a fully functional ETH2.0 network may take years to complete, while the blockchain industry is hungry to have the Ethereum network with greater scalability. The EIP proposes an alternative ETH2.0 without sharding - a single Ethereum chain with minimum modifications to support greater scalability. The performance goal is that after the upgrade, the network can support 1000+ TPS with 100M+ (or 1B) accounts on a commodity PC with decent storage (~ 500 GB) and network connections (~ 1Gbps). We hope that with the upgrade, the network is able to serve the near-future blockchain needs in 5-10 years.

Note that it differs from 1.x in that it requires significant changes in consensus and ledger model. Further, it does not conflict with existing ETH2.0 sharding work, which can be promoted to ETH3.0.

Specification

As of UPGRADE_BLOCK_NUMBER

, we will re-genesis the network and start the ETH2.0 without sharding with the following components:

- A fast- finality PoS consensus such as Tendermint or LMD-GHOST/Casper; and
- A ledger model with flat KV as the ledger instead of MPT and replacing trie hash with the hash of KV operations of all TXs in a block in the header; and
- A periodic snapshot of the state for every BLOCKS_PER_SNAPSHOT

blocks (BLOCKS_PER_SNAPSHOT

is large) and including snapshotted state hash (trie hash) in the headers of each k * BLOCKS_PER_SNAPSHOT

block;

- An automatic pruning for finalized blocks that are covered by a block with snapshot hash.

Fast-Finality PoS Consensus

The proposed ETH2.0 will adopt a fast-finality PoS consensus such as Tendermint or LMD-GHOST/Casper. In the following, we will use a modified Tendermint (a.k.a., the proposed consensus engine) to illustrate the idea, but the specification should be also applied to other consensuses. The proposed consensus engine has the following main features:

- PoS: A list of validators will be chosen according to the amount of ETH staked (including delegated). Users are free to choose any validator to delegate its stake.
- Fast finality: The consensus engine will not propose the next block until the current block is finalized. This means the chain is forkless

and we do not have to deal with complicated chain re-org logic.

- Epoch-based: The validator sets will be not changed until the blocks with heights k * BLOCKS_PER_EPOCH

are reached. The blocks with heights k * BLOCKS_PER_EPOCH

are called pilot blocks

, and the blocks in [k * BLOCKS_PER_EPOCH

, (k + 1) * BLOCKS_PER_EPOCH

- 1] are called the blocks in epoch k

. The purpose of the epoch is to quickly bootstrap a client and save storage, which will be discussed in the following sections.

Ledger Model Without Maintaining MPT

The proposed ETH2.0 will adopt a flat KV storage as its state, where the key-value pairs of the storage are like:

- addr -> account

for per-account data

- addr + "/" + storage_slot -> storage_data

for the smart contract storage

Note that, unlike the current ETH1.0 model, which maintains multiple versions

of the state, the proposed ETH2.0 only maintains a single version

of the KV-paired state of the latest finalized block. Since the proposed consensus engine (modified TM) is forkless

, we will only commit KV writes to the storage when a new block is finalized by the engine and thus without worrying about undo the KV operations due to chain re-org.

When an unverified block is received from a peer, a tentative execution of the TXs of the block must cache the results in memory

, where the results are a list KV operations like

- update(addr, account)

(for balance/nonce update, create a new account, etc)

- delete(addr, account)

(for contract suicide)

- update(addr + "/" + storage_slot, storage_data)

(for SSTORE)

- delete(addr + "/" + storage_slot, storage_data)

(fro SSTORE with zero value)

E.g., a block with a balance transfer from addr A to addr B, will result in the list as

[update(A, accountA), update(B, accountB)]

.

The hash of the KV operation list will replacestate root hash in the block header, and the hash value in the header of an incoming block must match the local hash value after the tentative execution. This makes sure that, as long as the block is verified and finalized, all non-Byzantine nodes will have exactly the same state without maintaining an MPT.

Periodic Snapshot of the State

When committing the TXs of a finalized block with height k * BLOCKS_PER_SNAPSHOT

, the node will use the underlying DB snapshot feature, e.g.,

GetSnapshot()

call in goleveldb()

which will return a snapshot of the KV storage with very low cost (the cost is at a similar level of Get()

)

After the block is committed and the underlying KV storage is snapshotted, a background thread will start to calculate the hash of the state based on the KV snapshot without interfering with the consensus engine (i.e., propose/execute/finalize/commit a new block). The hash of the state of the block with height k * BLOCKS_PER_SNAPSHOT

will be included in the header of the next snapshot block, i.e., the block with height (k + 1) * BLOCKS_PER_SNAPSHOT

- that means we have the time for BLOCKS_PER_SNAPSHOT

blocks to calculate the hash.

Note that we do not specify which trie to calculate the hash and according to the Turbo-Geth result, the MPT itself may be good enough - it just takes a couple of hours to calculate the hash (see stage 6 and 7 in here). Assuming the BLOCKS_PER_SNAPSHOT

will last at least a few days or weeks or even a month, the extra computation cost of the snapshotting should be negligible.

Automatic Pruning for Finalized Blocks

When TPS is high, the data of the blocks can cost a significant amount of storage. To reduce the storage cost, the proposed ETH2.0 without sharding will automatically prune the finalized blocks that are covered by a block with snapshot hash. Suppose the latest snapshot block is K * BLOCKS_PER_SNAPSHOT

, which contains the hash of the state of the snapshot block (K - 1) * BLOCKS_PER_SNAPSHOT

. The pruning algorithm will remove all the blocks (including headers but excluding pilot block headers) before (K - 1) * BLOCKS_PER_SNAPSHOT

. This allows a new client to recover the state of the latest block by downloading/verifying the state at (K - 1) * BLOCKS_PER_SNAPSHOT

and then replaying the TXs of the rest non-pruned blocks without touching the pruned blocks.

Rationale

Performance Analysis

We discuss the potential bottlenecks of the proposal in the following areas:

- Networking: We hope that with the consensus (and possible optimization on P2P), the bandwidth can be increased from 60kbps to 600kbps or more. This means that if a TX size is 150 bytes, the network can deliver a maximum throughput of 4K TPS.
- IO/CPU performance: Using flat KV can significantly increase the performance compared to MPT, especially the state is large, e.g., with 100M+ accounts or more (likely 1B). Further, unlike MPT whose performance degrades with larger state, the performance degradation of flat KV is almost negligible with larger state. Our performance test result shows that flat KV can easily deliver more than 1000 TPS with 100M+ accounts (see below for more details).

Storage Cost

A client of ETH2.0 without sharding can maintain the following data:

- Headers of pilot blocks. Assuming `BLOCKS_PER_EPOCH' is large, the cost should be negligible.
- Blocks from (K - 1) * BLOCKS_PER_SNAPSHOT

to tip

. Assuming BLOCKS_PER_SNAPSHOT

lasts for one week, and each TX data in the block costs 200 bytes, the cost will be 1000 (TPS) * 200 (bytes per TX) * 86400

(sec per day) * 7 * 2 (up to 2x BLOCKS_PER_SNAPSHOT

from (K - 1) * BLOCKS_PER_SNAPSHOT

to the latest block) ~ 224G

- State of the latest block: Suppose each account costs 200 bytes, 1B accounts will take about 200GB storage.

Sum up together, the storage cost should be ~ 500GB.

Fast Bootstrapping a New Node

When a new node joins the network, it can quickly become a full node by running the following steps:

- Download all headers of pilot blocks. Since validator set changes only happen for pilot blocks, this allows the new node to quickly discover the validator set and become a light node.
- Download (K - 1) * BLOCKS_PER_SNAPSHOT

block (where K * BLOCKS_PER_SNAPSHOT

is the latest snapshot block) and obtain the state hash.

- Download/Verify the state at block (K - 1) * BLOCKS_PER_SNAPSHOT

.

- Download/Replay all blocks from (K - 1) * BLOCKS_PER_SNAPSHOT

to tip

.

Benefits vs Sharding Approach

- Maximum backward compatibility with existing dApps including composability; and
- Should be much easier to implement.

Implementation

We are testing the performance of each component with Tendermint and flat KV. The early result on flat KV is very promising - without consensus cost and using Ethereum mainnet data with ~100M accounts, we could achieve 4.5x performance improvement with about 2500 TPS. Note that our implementation of the flat KV based EVM can completely replay existing mainnet TXs including all smart contract TXs. We will soon open-source the code and more results.

Migration

Since the proposed ETH2.0 makes significant changes of consensus and ledger model, an in-place upgrade of Geth to support the proposed ETH2.0 may be difficult. To simplify the upgrade process, we have the following options:

- As of UPGRADE_BLOCK_NUMBER

, we will stop ETH1.0 and start ETH2.0 without sharding with genesis state == state of stopped ETH1.0; or

- Similar to migration from ETH1.0 to Beacon chain, users can lock ETH in ETH1.0 forever and obtain ETH in the proposed ETH2.0 mainnet.

Timeline

By reusing existing components such as Tendermint and the support from the Ethereum community, we hope that a stable testnet achieving the aforementioned performance goal is available in a year.

Security Considerations

Upon receiving a new block, a node can immediately detect if the sender peer has the same state as local by comparing the hash of the KV operations of all TXs in the block header with the hash of local execution results.

Thanks

Part of the flat KV idea is from TurboGeth.

Copyright