TLDR

: By changing the state-root included in the ExecutableData in theExecutable beacon chain proposal to the pre-state rather than the post-state, we can get rid of the execution bottleneck in block verification/propagation. This means it is ok for the eth1 payload execution to take several seconds on average, which would degrade the beacon chain in the current (post-state) proposal. This means we can likely increase the block gas limit to 50-100M almost immediately post merge, without any security compromises.

Introduction

In theory, a Proof of Stake based system should be able to accommodate a longer block execution time than a Proof of Work based system, because it does not need to account for “block jitter”: All blocks are exactly equally spaced in time, so it is ok to exploit that time for execution; whereas the jitter makes a Proof of Work system degrade somewhat already when block execution only requires a fraction of the average block time, because some blocks will arrive earlier.

However, the current Executable beacon chain proposal does not really allow us to make use of this freedom: It requires full execution of the Eth1 payload before it can be decided whether a block is valid. Since validity is a precondition for attesting to a block, and attestations are supposed to be created 1/3 block time (4s) after a block is published, this really does not leave much time for Eth1 payload execution; much more than 0.5s-1s on average will be difficult because it will interfere with block propagation. At 20M gas per second target, this does not leave much room for a gas limit increase post-merge.

Committing to pre-state roots rather than post-state roots has been suggested in the past to improve scalability (e.g. see Near Sharding Design, section 3.5). It means that Eth1 payload validation will not be necessary at all to validate the current block; it is only necessary to validate the pre-state of the next block, which comes 12s later and thus leaves a lot of time for execution.

I argue that in the current context, we should max out what we can do on the single-sharded EVM and this proposal can, in my prediction, allow us to increase the gas limit 5-10x within weeks of the merge, so in around 12 months. Having this could easily make or break Ethereum, since it will probably be another year or two from then for the effect of full sharding to be felt in gas prices. The proposal also does not depend on any other scaling solutions being deployed and thus benefits even those applications that cannot make use of rollups in the near future.

Proposal

The executable beacon chain proposal suggests adding the ExecutableData

data structure to the beacon state:

class ExecutableData(Container): coinbase: bytes20 # Eth1 address that collects txs fees state_root: bytes32 gas_limit: uint64 gas_used: uint64 transactions: [Transaction, MAX_TRANSACTIONS] receipts_root: bytes32 logs_bloom: ByteList[LOGS_BLOOM_SIZE]

We need to make one change to this data structure: We will remove the coinbase

variable and just immediately use the proposer validator balance for it (this is necessary as we will need to be able to charge it; It would be possible to charge to an Eth1 address as well, but would require that all validators maintain a well funded Eth1 address in order to be able to propose blocks (which is very capital inefficient if you only get to propose a block every few weeks). We would need to add a signature that proves that a validator is the owner of the Eth1 address in question

). Further, we will change all variables except for transactions

to refer to the state at the end of the execution of the previous

ExecutableData

block, which we clarify by adding the pre

keyword:

class NewExecutableData(Container): pre_state_root: bytes32 pre_gas_limit: uint64 pre_gas_used: uint64 pre_receipts_root: bytes32 pre_logs_bloom: ByteList[LOGS_BLOOM_SIZE] transactions: [Transaction, MAX_TRANSACTIONS]

Validating the NewExecutableData

means checking that all pre_

variables have the proposed value after the previous

block’s NewExecutableData.transactions

have been executed.

All tips (non-basefee part of gas) are sent directly to the validator’s (Eth2) balance.

Handling invalid transactions

There are a couple of reasons why a transaction can be invalid in Eth1, and this list may not be exhaustive:

- Invalid signature
- Invalid Nonce
- Not enough balance to pay for gas
- Block gas limit exceeded

Note that this is different from runtime errors, e.g. running out of gas: The latter do not make a transaction invalid, they just revert all state changes except for the gas charged. However, for invalid transactions, we cannot even necessarily charge the sender, since they may not have intended for it to be included or may not be responsible for its failure. So these transactions need to be charged to the proposer instead: The EIP1559 BASEFEE times basic transaction cost (21k gas plus charge per byte used).

This means that there is nothing in transactions

that can make a block invalid, and thus there is no need to check it before propagating/attesting to a block. Only when building/checking the next block do we need to have executed all transactions.

Note that, except for the block gas limit, all of these can be checked by the proposer quite cheaply without actually executing the transactions. This suggests that some proposers might get away with composing blocks without executing transactions, and just staying safely below the block gas limit will make sure they won’t be out of pocket

Maximum gas limit

There are three limits on how much execution time we can allow with this proposal:

1. The next proposer needs to be able to assemble their own block – they do

need enough time for full execution even if they don’t commit to the post-state, as they don’t want to include any invalid transactions

1. We cannot use 100% of the available time between two blocks for execution, otherwise it is literally impossible to ever sync with a chain
2. DOS attacks. But it turns out they are probably more benign overall post-merge, because long execution time will only lead to orphaned beacon blocks, but can still have the same attestation rates; so only throughput will be decreased, but not security.

1 and 2 both lead to a (somewhat aggressive) maximum of targetting a little less than 50% slot time for Eth1 execution. For example, if we did target 5s at 20M gas/s, we could increase the gas limit to 100M gas (50M EIP1559-target), which is a lot more than is possible now. A major downside is that it will obviously make syncing much harder, so it would be essential that good sync protocols are implemented that can yield a state quite close to the tip.