Testing path for a potential gas limit increase

Thanks to Barnabé, Andrew, Sam and Marius for the review and discussions

Introduction

As the topic of Gas limit increase has surfaced recently, I thought I'd spend some time compiling thoughts from various places and try to enumerate a path forward to increase the limit safely.

In its current form, Ethereum consists of 2 layers, the Execution Layer (EL) that is responsible for handling transactions, state, history, building blocks and the Consensus Layer (CL) that is responsible for consensus. The CL consensus engine is currently a combination of Casper the Friendly Finality Gadget (Casper FFG) and the LMD-GHOST fork choice algorithm. This consensus engine is strictly time dependent and introduces a concept of Slots and Epochs, there are tasks that need to be completed at certain periodic intervals by the validators. Two such tasks that we will consider are called Block proposals and Attestations. A block proposal specifies what the assigned validator thinks the canonical slot needs to be and the attestation specifies the vote of each validator for said proposed slot. Each interval (slot) is 12 seconds long and the specification says that the validator must attest as soon as they see a valid block or at the 4s mark into the slot. This introduces the concept of a deadline for the attestation task, i.e, under ideal conditions we'd want every validator on the network to attest to the canonical slot with a valid block before the 4s mark.

On receiving information about the proposed block, the CL triggers a engineAPI call called forkchoiceUpdated

(fcu). If the EL has all the data it requires, then it verifies the block and responds with VALID

. Until this verification is complete, the CL cannot proceed with its duties, implying that the verification needs to ideally be complete within the 4s mark. There are a couple of variables at play with regards to this verification time: It depends on when the CL sends it the fcu

, how big the block being verified is and if the EL already has all the data required for verification. A higher gas limit would be one factor that affects this verification time.

The presence of Blobs also adds some complexity to this systems as a slot isn't valid until all the blobs proposed in it are fetched from the network and verified. The CLs do this in parallel, i.e once the block is received the fcu

is issued, the block is then imported into forkchoice once the blobs have been deemed valid and available. The trigger for both tasks are handled by the CL, since they are done in parallel we would need to analyze which aspect takes the longest. If the blob fetch and verification takes longer, then the assumption is that on this one metric we can potentially afford to increase the EL gas limit. If the block verification takes longer, then on this one metric alone we can potentially increase the blob count. However, we do need to ensure that in any case the tasks are complete (ideally) in under 4s across the network.

While we mainly spoke about CL considerations till this point, the EL also has some independent considerations. These mainly have to do with the size of the state, the access duration and the growth for the foreseeable future. Péter from the Geth team articulates these arguments in depth in his talk here. The arguments largely boil down to increasing difficulty in managing access to data as time passes. We would need to collect metrics as to the rate of growth per month of the state at the current gas limit and attempt to predict the rate of growth with an increased gas limit.

Generalizing the constraints, we can classify them under two categories:

- Bust constraints: For a given slot we want the processing time to always be lower than the worst case the network can handle
- Long term constraints: We want to be sure that with certain network parameters the clients are viable for the forseeable future

So we need to ensure that we collect metrics for both types of constraints before we make any decision on changing the gas limits.

TLDR: There's a time budget we have, gas limit factors into this time budget. Consider the CL, EL and combined node perspectives.

Testing

We currently have 5 supported CLs and 5 supported ELs. Since there is no limitation on what combination users can run, we need to ensure that any gas limit decisions would perform well in any of these 25 combinations.

We'd ideally like to start testing by figuring out interesting metrics. Since the gas change would affect various layers of the stack, we'd likely need to monitor:

- Client performance related to attestation and committee duties

- Missed slots/reorg rates
- iops per node
- rate of account growth
- rate of trie size growth per client
- disk usage growth per node
- RAM/CPU increase per node

… at a bare minimum. There are probably a lot more metrics that EL client devs would be interested in surfacing as well.

It would be prudent to start with local testing and then graduate to shadowforks (Which would represent more real network loads).

Initial Local testing

My suggestion would be to use kurtosis as a local testing tool to orchestrate test networks with arbitrary ELs and CLs as participants. Kurtosis supports both a Docker and Kubernetes backend, Kubernetes would be the preferred approach here as we can isolate a EL/CL pair from the effects of others on the same host. One has to be careful to use local volumes, as network volumes introduce their own latency complications.

Kurtosis supports the ability to pass a participant arbitrary flags, this is useful as the ELs allow you to override the gas limit with a runtime flag (Flags similar to --miner.gaslimit

). Some CLs contain flags similar to --suggested-gas-limit

on the validator, others might need a value change and recompilation. (Hint, if you have a branch with changes and need a Dockerfile, the ethpandaops github action can build it for you here).

Once you have the flags/images and kurtosis config ready, you can spin up a new network with kurtosis

and specify the tooling it should spin up as well (Such as tx_spammer

and goomy_blob

). This tooling will ensure that your blocks are being filled up for the test.

Running the kurtosis run with mev_mode: full

would also spin up the entire mev workflow, allowing us to analyse the delays it adds to our expectations. Note that the default mev workflow that is spun up is purely greedy, this may not be the strategy run in mainnet by relays.

This would also be a good time to figure out the data collection and metrics aspects. While some/most metrics are available via the clients themselves, they might not be standard or visible in a single dashboard. It might be easier to fetch the data via an exporter or from the host directly vianode exporter or similar tools. Ideally whatever metric collection approach is chosen, is independent of kubernetes - as we currently don't support shadowforking with kurtosis on kubernetes. The shadowforks are usually instrumented with ansible on various hosts.

Shadowforks

Once the gas limit change has been vetted on local tests and we have a better idea of what kind of metrics are needed, we can graduate to shadowforks. These networks fork away from the canonical chain, so they inherit the entire chain and for a limited duration continue to import the transaction load. This would mean that we could genuinely see how the network would perform with a higher gas limit.

Shadowforks currently aren't automated in kurtosis, the work for that is still underway. The best approach would be to follow this guide. The base testnet functionality can be foundhere, This will allow you to spin up nodes on a cloud provider and deploy a network onto them. Most of the shadowfork functionality is performed with various ansible playbooks found here.

We also support running the entire mev workflow with ansible, this can be foundhere. MEV adds a lot of changes to the default pathways, so it would be prudent to test this on the shadowfork as well.

Conclusion

This two step approach would allow us to collect data in an agile manner and to vet our proposed changes on a production-like network. This however would still not include some externalities like MEV-relay strategies and DVT clusters. It also wouldn't include the edge case of an attacker finding a transaction that can trigger extremely slow evm execution. But this systematic approach would lead to a reproducible system we can use for future discussion of gas limit increases as well.