Turbo-Geth / Silkworm

Past and Future?

Turbo-Geth

Started as a curiosity project in December 2017 (during the CryptoKitties craze)

Initially explored alternatives to the trie-based database model

Received a small ($25k) grant from Ethereum Foundation in March 2018

Used as a platform to get state analytics for State Rent research in Q1-Q2 2019

Used as a platform to perform Stateless Ethereum backtesting in Q3-Q4 2019

Considered conceptually sound (by me) before Devcon5 in Osaka

Turbo-Geth

At Devcon5, I proposed a one-year moratorium for EIPs to convert all implementation to the similar data model. It was not taken upon due to skepticism, and general lack of initiative of “core developers” as a group.

The skepticism was about the way to efficiently compute and update state root hash.

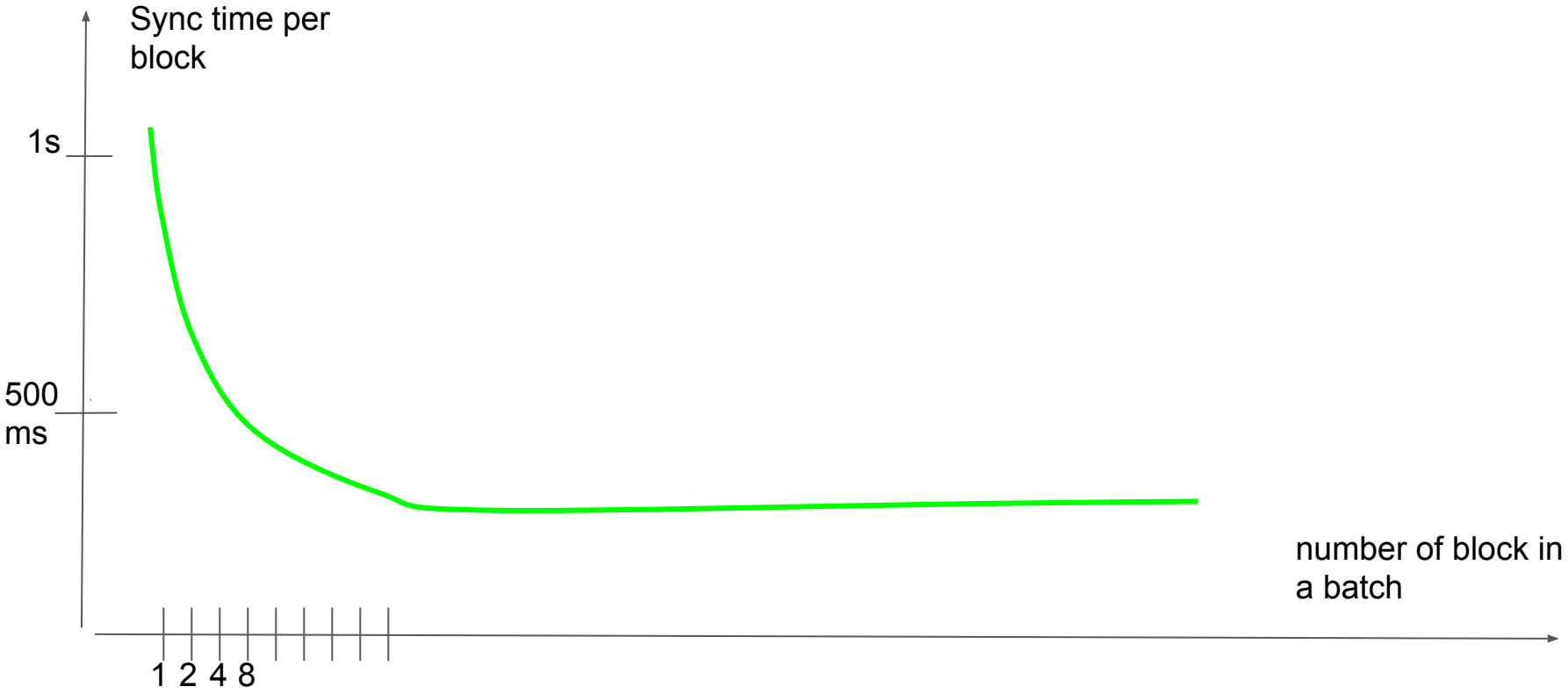
At EthCC 2020, in March 2020, the solution was presented - extra data structure called “Intermediate Hashes”, fully implemented over the next few months

Turbo-Geth

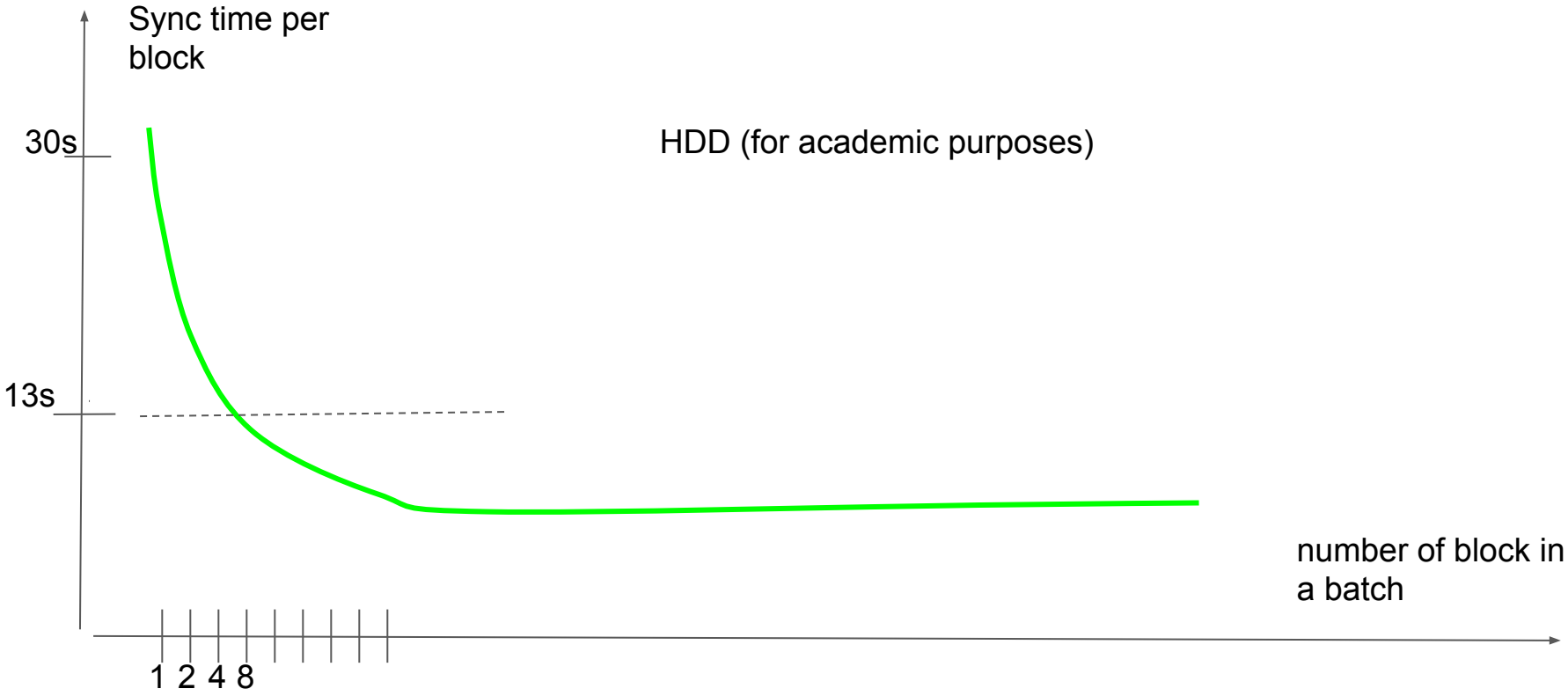
Idea of staged sync was born out of the observation of the measurements of per-table write churn. Solution to the churn was to insert data in pre-sorted order. Observation was made at the end of 2019, but first experimental implementation - in Feb 2020, confirming big performance advantage

Staged sync was a very significant change of the architecture (but not of the data model) carried out March-July 2020. It is responsible for large improvements in the sync times (10x).

Turbo-Geth



Turbo-Geth


Sync time per block
30s
13s
HDD (for academic purposes)
number of block in a batch
1 2 4 8

Turbo-Geth

In August 2020, a way of reducing state representation from 50Gb to 10Gb was discovered.

In September 2020, the “Intermediate Hashes” were made more fine-grained, improving the performance of calculating state root hash 4x (from 200ms to 50ms), while also reducing its size from 7Gb to 2.5Gb

Currently working on proper indexing of logs

Turbo-Geth

What is the catch?

Mostly, there is no catch, because current implementations aren’t on the efficient frontier

There are still couple of “untied ends” though:

1. Merkle proofs of something in the very far away history are inefficient to produce (recent history is OK). Can be remediated by introducing intermediate state snapshots if required (they are relatively small)
2. Some consensus algos are hard to make work with staged sync, and should ideally be co-designed

Silkworm

Idea of creating Apache 2.0 licensed, modular Ethereum implementation in C++ existed from beginning of 2019, as we watched “Aleth” project effectively being abandoned.

But it was not a good time.

In May-June 2020, we decided it was a good time. 4 things happened:

1. We switched from BoltDB to LMDB (database written in C), this can ensure DB compatibility between Turbo-Geth and Silkworm

Silkworm

2. Staged sync architecture **naturally** split the implementation into relatively independent components, interacting mostly via the records in the database (or via pages in memory if it happening within one DB transaction). This means that creation of C++ implementation can happen one component at a time.

3. Earlier experiments with EVM one (using EVMC interface) showed the large overhead of inter-languages interfaces, exacerbated by the EVMC's dual interface.

4. We felt we have enough experience to be able to pull this off in a reasonable timeframe (1 year instead of 5-10 years), with some help from experts.

Future

Starting Silkworm also opened us up to the idea of migrating the implementation to other languages (like Rust), one component at a time.

I believe that Ethereum 1 can scale at least 10x even without sharding. There will be 3 main challenges:

1. High gas block limit causes DOS attacks. Turbo-geth's safe limit is likely to be more than 10x higher than of other implementations. Silkworm likely even more.

Future

2. High gas block limit will allow potentially large blocks. This, in turn, creates two problems:

a) Block propagation. This can be addressed by pre-consensus (effectively trading off transaction latency for transaction throughput)
b) Block downloading and storage. This can be addressed by using specialised storage networks like BitTorrent (this work is under-way).