Motivation

Reusing Ethereum mainnet security and extending Ethereum scalability are the major goals of Ethereum L2 solutions. Most existing L2 solutions such as rollups focus on scaling Ethereum computation power, a.k.a., higher transactions per second. Meanwhile, with the popularity of dApps such as NFT/DeFi/etc, the demand for storing a large amount of data reusing Ethereum mainnet security has grown dramatically.

For example, one strong storage demand comes fromon-chain NFTs, where not only

the token of an NFT contract is owned by the users, but also

the on-chain images belong to the users. In contrast, storing the images on a 3rd party (e.g., ipfs or centralized servers) introduces additional trust, which can be easily and frequently broken (e.g., a lot of images of old NFT projects using ipfs are now unavailable).

Another demand is the front end of dApps, which are mostly hosted by centralized servers (with DNS). This means that the websites can be easily censored (happening for Tornado Cash). Further issues include DNS hijacking, website hacking, or server crash.

By reusing Ethereum mainnet security, all the aforementioned problems can be immediately solved. However, if everything is stored on-chain, the cost will be extremely high - for example, storing 1GB data using SSTORE will cost 1GB / 32 (per SSTORE) * 20000 (gas per SSTORE) * 10e9 (gas price) / 1e18 (Gwei to ETH) * 1500 (ETH price) = $10M! The cost can be reduced to 1/3x using a contract code, but it is still far more expensive than other storage solutions (S3/FILECOIN/AR/etc).

Goals

With L2 and data availability technologies, we believe that we can achieve an Ethereum storage scaling solution with the following goals:

- Increase the capacity to PB or more assuming that each node has a few TB disks
- Reduce the storage cost to 1/100x or 1/1000x vs SSTORE
- Similar KV CRUD semantics as SSTORE (a few limitations will apply, see below)
- Reuse Ethereum mainnet security on block re-organization, storage cost settlement, and censorship-resistant

Solution

How? The current cost of storing a large amount of data on the Ethereum mainnet comes in two parts

- Upload cost (calldata)
- Storage cost (SSTORE)

For the upload cost, with DA, especially danksharding, we are expecting the upload cost will dramatically be decreased in a near future - e.g., the current draft of EIP-4844 will have 1 gas per byte (i.e., ~128KB per BLOB given a BLOB size is ~128KB) and minimal gas price to be 1. The throughput of uploading can be further increased to ~10x by danksharding, which should further reduce the cost of uploading data.

For the storage cost, our solution is to build a permissionless L2 data retention network. The L2 network contains the following components

- a storage contract

deployed on Ethereum mainnet, which offers KV CURD semantics such as put()/get()*/delete() besides verify(). The storage contract does not store the full values of the keys - only commitments (e.g., KZG commitment of BLOBs) are stored, while the corresponding data is available thanks to DA. The storage contract will also accept proof of storage and efficiently verify that data is stored in the data nodes

in an L2 data retention network. (get()* is only available in data nodes, see below)

- data nodes

run a special client of Ethereum (a modified version of geth and a consensus client), which synchronizes the latest state of Ethereum (and thus all commitments of the values of KV pairs). Further, the data nodes serve additional functions as * accept the configuration of which parts (i.e., shards) of BLOBs will host

- synchronize the BLOBs of interest by joining the L2 data retention network
- copy the BLOBs from the Ethereum mainnet DA network if the corresponding commitments in the storage contract are updated/appended in the storage contract
- generate proof of storage, submit it to Ethereum mainnet, and collect storage fee
- serve storage_contract.put() in JSON-RPC eth_call() method
- accept the configuration of which parts (i.e., shards) of BLOBs will host
- synchronize the BLOBs of interest by joining the L2 data retention network
- copy the BLOBs from the Ethereum mainnet DA network if the corresponding commitments in the storage contract are updated/appended in the storage contract
- generate proof of storage, submit it to Ethereum mainnet, and collect storage fee
- serve storage_contract.put() in JSON-RPC eth_call() method

Note that running a data node is completely permissionless - as long as a data provider has sufficient disk space, it could run a node, synchronize the BLOBs from L2 network, copy the BLOBs from DA if the commitments on L1 change, and prove the retention to L1.

Benefits vs Existing Solutions

Besides reusing Ethereum mainnet security, EthStorage can offer the additional benefits:

- Rich storage semantics (KV CRUD)

. FILECOIN/AR mostly works for static files, which lack efficient update/delete operations - i.e., the users have to pay twice to update existing data. Thanks to DA and smart contracts, EthStorage can offer full KV CRUD semantics similar to SSTORE.

- Programmability

. The storage can be programmable by smart contracts, which can easily enable new features easily such as multi-user access control or data composability.

- Atomicity with application logic and storage logic

. Current dweb using ENS generally requires two steps: 1, uploading the data to an external storage network; 2, storing the contenthash on ENS. With DA and EVM, EthStoage can complete both application logic and storage logic in a single transaction, which is more friendly to users (also widely found in Web2 applications, e.g., Twitter/FB/etc).

- Zero-onboarding cost

. EthStorage is built on top of Ethereum, the storage cost is also paid by ETH, and thus, the storage operations can be done just by ETH wallet like Metamask - users do not have to learn new token/wallet/address.

Key Problems to Solve

- Proof of storage on large dynamic datasets with data redundancy: Enabling data redundancy in a decentralized way is a key challenge, especially the dataset changes constantly.
- Efficient recurrent storage payment: Submitting a valid proof of storage on-chain will reward the prover (or data providers) with a storage fee (in ETH). We need an efficient storage rental/payment model to ensure sufficient redundancy of the storage data perpetually.
- Rarity data discovery and token incentive to encourage auto re-replication: When some of the data nodes are off, we need to encourage other nodes to join the data retention network and replicate the data
- Efficient verification on-chain: We will explore some techniques, especially zero-knowledge proof to reduce the cost of verification.

Other Questions

- Q: What is the difference with DA?

A: Current Ethereum DA is expected to expire the data in a few weeks or months (See Proto-Danksharding FAQ). EthStorage is expected to store the data permanently given some well-known assumptions on storage cost (e.g., storage cost over ETH price constantly drops every year). Further, EthStorage offers full KV CRUD semantics on-chain (note that read is limited to only data nodes).

- Q: What is the access protocol to read the storage EthStorage (similar to ipfs://)?

A: The data can be retrieved by calling eth_call

on a data node, which will search and retrieve the corresponding BLOBs in L2 network. Further, from an end-user perspective, we could use web:// access protocol to browse the BLOBs hosted by a smart contract, whose content can be dynamic.