Goliath: a horizontally-scalable storage system for crypto.

Liam Zebedee

Abstract

The need for scalable storage systems in crypto infrastructure remains largely unaddressed.

Goliath is a novel horizontally-scalable storage network for crypto inspired by Google's File System (GFS), adapted for decentralized environments. Using Goliath, users can deploy clusters for storing blockchain state and history, decentralized AI models and their outputs, and confidential data stores. Clusters are read-write, permissioned by public keys and ZK proofs, and can grow in size dynamically.

1 Introduction.

1.1 Unbundling Ethereum.

The Ethereum blockchain was invented in 2015, a monolithic bundling of many inventions: Indexes (logs), execution (EVM), sequencing (mempool), consensus (PoW + GHOST), data availability (coinbase). Since then, every aspect of this stack is being torn out and spun into its own protocol.

Overall, Ethereum represents a maturing ecosystem, wherein modules have diverged into more specialized subcomponents. For example, the Lyra Protocol began as a DeFi protocol on Ethereum L1, and then later spun out as an optimistic rollup L2, enabling them to scale with cheaper transcation fees, and then replaced their data availability with Celestia for even more savings passed on to the user.

Curiously, there is an aspect missing from this stack: storage. Every Ethereum node, every Ethereum L2 sequencer, every blockchain requires storage for its apps - both for the raw state of the database (e.g. ERC20 balances) and for the history of the chain itself (what is used to sync). And storage so far has remained the exact same - operators must manually manage and allocate storage via attaching HDD's.

Component	Ethereum (2015)	Ethereum protocols (2024)
Indexes	Logs	Subgraphs, Shadow, Custom indexers
Execution	EVM	Parallel EVM rollups, WebAssembly (Arbitrum AnyTrust), custom ZK-proven VM's (Starknet, Scroll)
Sequencing	Mempool	Decentralized sequencers, threshold-encrypted mempools
Data Availability	Coinbase	Celestia, EigenDA, Danksharding
Security models	Replicated state machines	Optimistic rollups, Validity rollups / ZK-proofs
Economic/validator distribution	Ethereum social layer	Eigenlayer, Symbiotic

Table 1.1: Unbundling the Ethereum monolith.

We have been tracking the development of Ethereum since 2015, and have been interested in innovating in this space. This whitepaper serves as a research exploration into creating something new - a cryptonetwork for scalable storage.

1.2 A storage cryptonetwork.

Crypto infrastructure demands a scalable storage system in many areas of the stack. Syncing a blockchain is extremely expensive and slow, indexing and downloading from many peers from scratch for every node. After a blockchain synchronises, every node must keep a full copy of state on their storage device, ruling out devices with low storage capacities from participating. There is a huge demand for decentralized storage systems that exist independently of blockchains - for example for indexers, for hosting decentralized AI models, confidential data stores [5].

While there have been products, no solution has really succeeded as a modular component in the same way Celestia has for data availability. BitTorrent is designed primarily for *data sharing*, cannot be used to write data, and is not a cryptonetwork where you can pay for storage as $ETH pays for blockspace. Filecoin has succeeded in building an economic model, but its product has failed to capture the market. There are many potential reasons for Filecoin's lack of adoption - the API to read/write storage is cumbersome, it requires asynchronously interacting with an orderbook (storage deals) rather than in Ethereum where users purchase blockspace synchronously from a proto-AMM (the progressive gas auction), the product does not provide guarantees of read/write speeds. Although Filecoin failed, its underlying technology came close - many prominent crypto companies use IPFS to verifiably host data, such as Zora - but instead of paying to put the data on Filecoin, they run their own IPFS nodes.

While we don't claim to know what the market wants, we believe a storage network that is encapsulated so users can deploy their own clusters is key. Rather than a single monolithic shared environment, users should be able to deploy their own storage cluster, with their own nodes, with different replication and mirroring factors, economic settings that reward data publishers, read/write quotas, and so on.

1.3 Conceptual design.

Our design for this system comes from Google's distributed systems, namely GFS [6] and Bigtable [4].

Key contributions . We rebuild the Google File System within a byzantine trust model [8] as a cryptonetwork. Instead of Chubby and Paxos, we use a blockchain on Tendermint. The master server is replaced by the blockchain node, which performs deterministic and non-deterministic roles (such as checking liveness via heartbeats). Chunkservers, which provide the storage to clients, perform the same function as in GFS - though are regularly issued data availability challenges in the style of Celestia and Filecoin. The total storage capacity of the network is tracked by the master node,

and sold via a custom-built automated market maker. Users of Goliath storage clusters interact via ECDSA wallets, and are able to read and write files and directories in an S3-compatible interface.

2 Design.

Our intuition for building Goliath is through reading the Google distributed systems literature, and noting the analogues.

Firstly, we will briefly explain Google File System. Then we will give an overview of the components and their analogues in the crypto space. Then we will sketch the design for the Goliath system.

2.1 Google File System.

Google File System (GFS) [6] is a scalable, distributed file system designed to handle large amounts of data across numerous machines. It organizes files into fixed-size chunks, typically 64 megabytes each, which are stored on multiple chunkservers to ensure reliability and availability. A single master server manages metadata, such as the namespace, access control, and the mapping of files to their respective chunks and chunkservers. The key insight of GFS is that the master can fit all the metadata in memory, allowing it to efficiently oversee and coordinate the entire system. By distributing chunks across many chunkservers, GFS can scale storage capacity horizontally as more servers are added, balancing the load and enhancing performance.

In order to elect a server as the master process, GFS uses Chubby [3]. Chubby is a strongly-consistent highly-available lock file service developed by Google. Chubby allows application developers to store small files locked to a single network process via the Paxos [10] consensus algorithm. In GFS, a master server is identified by its lock on a Chubby file. If the master process crashes, the lock is released, and another server can take the lock and assume the role of the master. The orchestration of such servers is done by a separate system, Borg.

The analogue for these Google systems is clear. The Paxos consensus algorithm can be replaced by a byzantine fault tolerant algorithm such as Tendermint [2]. The Chubby lock file service, a small stateful program, can be replaced by a smart contract on a blockchain. Finally, in order to verify the chunkservers are indeed storing data - we can introduce data availability sampling, whereby nodes provide a random proof that they are hosting a subset of data.

Google	**Crypto analogue**
Paxos consensus	Tendermint consensus
Chubby lock file service	Smart contract
GFS master	Blockchain node
GFS chunkserver	Chunkserver with data availability proofs

Table 2.1: Google/crypto distributed systems analogues.

Non-deterministic computation. One interesting difference to Ethereum L2's is that Goliath makes use of *non-deterministic* computation affordances in Cosmos blockchains. Namely, the master process is a Tendermint blockchain, though unlike Ethereum smart contracts, we perform out-of-protocol functions that are usually reserved for the role of *keepers* within a crypto context - an example of this is connecting to chunkservers, performing regular heartbeats to verify their liveness.

2.2 System overview.

1. **Goliath cluster**. A Goliath cluster consists of a Tendermint blockchain representing the master process, and a set of worker nodes referred to as chunkservers, which perform the data storage.
2. **Master process**. The master process is implemented in Go, as a Tendermint blockchain. It stores the file index (file paths, replication factor, chunks, permissions), chunk servers (address, chunk sets, heartbeat, capacities), read/write leases (locks), token balances (accounts).

It runs the functions to manage chunkservers (indexing their chunks, heartbeats to check status, assigning a primary, data availability checks), allocating chunks to chunkservers (assignment, replication), pricing and selling storage (automated market maker), and allocating leases for file writes to users.

3. **Chunkserver process**. The chunk server process is implemented in Go, as a standalone process. It stores chunks. It follows commands given by the master blockchain, including replicating chunks, serving reads to users, answering data availability checks. It verifies leases issued by the master and then authorises writes. There is one chunkserver deemed the primary which determines write order, the rest follow.
4. **Client library**. The client library is used by users to read and write data to a Goliath cluster. Control flow begins at the master, where users acquire a write lease which ensures writes are consistent. Users then interact directly with the chunkserver primary to upload chunk data, which is then replicated on secondaries, and finally committed. Chunkservers are cached inside the client library to reduce load on the master.

2.3 Tokenomics.

Goliath functions similarly to a blockchain, but instead of selling blockspace, we sell storage. The master keeps track of the total capacity of the storage system. Users pay transaction fees to write data. The price paid is a function of an automatic market maker, modelled after Uniswap v2 [1], which models the storage supply. Price increases when storage utilization is high, and decreases when it is low. Nodes are incentivised to come online and saturate demand for storage through automatic "surge pricing" incentives.

3 Product.

Goliath can be used to deploy horizontally-scalable storage clusters for cryptonetworks. The chief benefit of a cryptonetwork is its longetivity and open-source alignment; a cryptonetwork is public infrastructure that is robust to political change.

We discuss a couple of use cases for Goliath below:

Permissioning Operators can setup a Goliath cluster and permission file system read/write access for different users based on cryptographic identities, tokens, or even ZK proofs which show the validity of data. For example, the Ethereum blockchain could use Goliath to store its block history, by setting a write permission check that requires the node to provide a ZK proof of consensus.

Data collaboration Goliath can be used for archiving and mirroring datasets. The file system will be designed so that nodes can selectively mirror subsets of / the entire file system. Operators get the best of both worlds - they can deploy clusters on their own hardware, whilst also running an open protocol which allows users to mirror any and all of the dataset. This is similar to a data storage consortium.

Publish, prove, get paid. Goliath will allow people to get paid anytime someone downloads/uses their data. For example, an open-source blockchain indexer could spend compute power building blockchain indexes and publish them to Goliath. Anytime someone wants to read data (for example, all ERC20 addresses that have interacted with a contract), they can pay a small fee and access this prebuilt index. This not only improves the customer experience, since they don't have to wait for their indexer to run, it also deduplicates work in the marketplace, by allowing one node to publish, prove, and get paid. ZK proofs play an important role here - allowing people to "try before you buy" through validating the data is valid and not junk. Encrypted data can be trustlessly sold using schemes such as fair data exchange (FDE) [12] from a16z.

Onchain AI agents. In 2040 there will exist a computer platform where fully autonomous Turing-complete processes (RISC-V executables) can run and pay for their compute, networking and storage purely via crypto. They will have highly-developed reasoning systems that various high-level human corporations interact with via distributed training algorithms [11] [13]. Why will they use AI's over humans? There will be no comparison - AI agents will work deeper and longer. They will sit on

problems for days, weeks, and months in "system 2" mode. Unlike humans, who require NDA's to prevent leakage of proprietary information, AI's can be run as pure algorithms inside encrypted secure multi-party computation circuits [9]. They will be provably trustworthy and reliably neutral due to cryptography and alignment. These AI intelligence softwares will be deployed autonomously for the same reasons that blockchains are useful - an unbiased global layer will make the cost of coordination much lower. Just as hedge funds develop highly-sophisticated trading strategies which make markets more efficient, thus producing more accurate signals for the market supply-demand process, so will there be firms developing highly-sophisticated AI models which will be used by firms to improve their economy too. Agents will be colocated with public information (price feeds, AMM pools, prediction markets), jointly operated with private information using MPC technology, and produce private/public outputs. These AI models may take actions of their own - perhaps using the joint private information of two firms to buy/sell assets in markets. They may even produce products of their own - for example crunching the datasets of two firms, producing a ZK proof of a solution to a mutual problem without revealing the solution itself. The choice to buy this solution (thus paying the "hedge fund" who developed it) being enacted again over MPC, only proceeding with public and irreversible payment via a blockchain [12].

Databases Crypto lacks database systems which can be used to replicate very basic web2 websites - for example, forum boards, imageboards, chat systems, and even further onto products like Spotify and Github, which are simply relational interfaces to data. In order to build these decentralized database systems, we need to first have the storage systems in place to actually persist data. Goliath can be used as a foundation to build decentralized databases, just as GFS is used as a foundation for Google Bigtable and Spanner.

Rebuilding the original WWW vision. The original vision for the web was a shared workspace where people could visit websites, edit documents, and link out to each other. Part of what makes this hard is that users don't want to run their computer all the time, and providing edit access to everyone is hard when all computers have different specifications. Goliath provides a storage backend which can make this possible. By linking Goliath to an open naming system like ENS, and modifying the browser to access ENS content through Goliath, users can publish websites and openly permission read/write access to other users. We believe this is important for science, that anyone is able to share their knowledge and collaborate seamlessly using open-source software systems.

4 Implications

Based purely on technical analogues, it appears that Goliath could theoretically work similar to Google File System, with minimal modifications. Google reports a sample GFS cluster with 72 TB of storage capacity, distributed across 372 nodes, with 20 MB of metadata required on the master. At a modest 12.5 MB/s downlink per node, this cluster supports a 30 MB/s aggregate write throughput, and read rates of 583 MB/s.

These statistics are not intended to portray an academic reflection, however are enticing as a comparison point to existing crypto storage systems.

We look to geohot's "The World Computer" [7] as the anchor point for future systems. While existing blockchains are focused on scaling execution throughput (*gas per second*, *gigagas*, etc.), we need to be scaling storage too.

References

[1] H. Adams, N. Zinsmeister, and D. H. Robinson. Uniswap v2 core. 2020. URL https://api.semanticscholar.org/CorpusID:221835652.

[2] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on bft consensus, 2019. URL https://arxiv.org/abs/1807.04938.

[3] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, 2006. URL http://labs.google.com/papers/bigtable.html.

[5] F. Collective. Suave specifications - confidential data store. URL https://github.com/flashbots/suave-specs/blob/main/specs/rigil/confidential-data-store.md.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. doi: 10.1145/945445.945450. URL http://doi.acm.org/10.1145/945445.945450.

[7] G. Hotz. The world computer. URL https://geohot.github.io/blog/jekyll/update/2024/02/26/the-worlds-computer.html.

[8] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. URL http://dblp.uni-trier.de/db/journals/toplas/toplas4.html#LamportSP82.

[9] Y. Lindell. Secure multiparty computation (MPC). Cryptology ePrint Archive, Paper 2020/300, 2020. URL https://eprint.iacr.org/2020/300.

[10] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, apr 1980. ISSN 0004-5411. doi: 10.1145/322186.322188. URL https://doi.org/10.1145/322186.322188.

[11] M. Ryabinin, T. Dettmers, M. Diskin, and A. Borzunov. Swarm parallelism: Training large models can be surprisingly communication-efficient, 2023. URL https://arxiv.org/abs/2301.11913.

[12] E. N. Tas, I. A. Seres, Y. Zhang, M. Melczer, M. Kelkar, J. Bonneau, and V. Nikolaenko. Atomic and fair data exchange via blockchain. Cryptology ePrint Archive, Paper 2024/418, 2024. URL https://eprint.iacr.org/2024/418.

[13] B. Yuan, Y. He, J. Q. Davis, T. Zhang, T. Dao, B. Chen, P. Liang, C. Re, and C. Zhang. Decentralized training of foundation models in heterogeneous environments, 2023. URL https://arxiv.org/abs/2206.01288.