LOAD AVS: A Horizontally Scalable Decentralized Hot Cache Storage Network

draft v2.2
(load://0x943a06973895dd4c7ebe7b0a397ac40fd1094df2805cd967aa7cb7ec05c02e12/0)

April 10, 2025

Rani Elhusseini rani@decent.land

Benjamin Brandall ben@decent.land

Mykyta Rykov nik@hns.is

1. Abstract

In this paper we describe LOAD AVS: a horizontally scalable decentralized storage network for temporary data. LOAD AVS aims to serve as a scalable hot cache layer in the EVM ecosystem, built on top of EigenLayer's AVS infrastructure. The network achieves this through a partition-based architecture where independent segments operate in parallel, each maintaining a sovereign state while contributing to overall network capacity. At its core, LOAD AVS utilizes pBFT consensus paired with Proof of Custody for continuous data replication verification. LOAD AVS implements dynamic pricing derived from global NVMe SSD market rates, ensuring competitive and realistic onchain storage costs. By leveraging EigenLayer's economic security stack and implementing a standalone parallel partition mechanism, LOAD AVS provides a cost-efficient, load-balanced, and scalable temporary storage solution, providing an alternative to slow, expensive storage layers available for EVM developers today.

---

2. Introduction

Storage systems form the backbone of modern digital infrastructure, with an increasing demand for decentralized solutions that can match the performance and reliability of centralized systems while offering enhanced security and availability guarantees. Decentralized data storage, often underpinned by or otherwise compatible with blockchain technology, represents a paradigm shift from traditional centralized architectures. These systems enable users to store, access, and share files in a distributed manner, fundamentally improving security, availability, and scalability in data handling. Unlike centralized storage, where data management relies on single-point infrastructure, decentralized storage distributes data across independently incentivized network nodes, eliminating third-party dependencies for data management and retention.

The evolution of decentralized storage systems can be traced through several significant developments. Early implementations in peer-to-peer networks demonstrated the feasibility of distributed data management, with BitTorrent (2001) establishing foundational principles for scalable peer-to-peer file sharing [1]. The InterPlanetary File System (IPFS) later introduced content-addressed storage, while academic implementations like Freenet advanced the theoretical framework for distributed storage systems [2].

Contemporary decentralized storage systems can be categorized into two primary architectures. Traditional decentralized systems, including distributed file systems like HDFS, GlusterFS, and Ceph Storage, focus on data distribution and redundancy mechanisms. These systems operate alongside peer-to-peer networks such as BitTorrent, eDonkey, and Gnutella, which primarily address file sharing without incorporating economic incentives or cryptographic proofs.

The emergence of blockchain technology has catalyzed innovation in decentralized storage, introducing economic models and enhanced security guarantees. Networks like Filecoin implement proof-of-replication systems [3], while platforms such as Storj and Sia have established onchain storage marketplaces. Notable innovations include Arweave's permanent storage through endowment mechanisms [4] and Ocean Protocol's tokenized data marketplace approach [5].

3. Problem

3.1 General

The general problem the crypto industry faces at the time of writing is the lack of a cost efficient, decentralized and secure data storage solution that is adjacent to the EVM tech stack. Many solutions have emerged while focusing mainly on small data needs, serving for improving the L2 experience, such as EIP-4844 [5] blobs and non-Ethereum data availability solutions, while the general problem of EVM-adjacent and cost-effective storage for arbitrarily large data remains unsolved.

Today's decentralized storage landscape shows a clear gap between demand and viable solutions:

1. **Cost inefficiency:** While solutions such as EthStorage [6] tackle this problem with strong decentralization and data replication, they remain cost-prohibitive for mass adoption. For example, storing 1 GB of data on EthStorage costs 4.43 ETH yearly.

2. **Limited scope**: Current solutions primarily focus on Layer 2 data needs and small-scale storage requirements. This narrow focus leaves a significant gap for applications requiring larger data storage capacities.

3. **Technical integration**: Most existing solutions lack seamless integration with data storage industry standards, creating additional complexity and overhead for developers and users.

This combination of high costs, limited scope, and integration challenges creates a significant barrier to the widespread adoption of decentralized storage solutions in the EVM ecosystem, particularly for applications requiring substantial storage capacity.

3.2 Load Network Adjacent

Load Network [7] in its design will be pruning history, keeping only 1 month history (last 2,592,000 blocks) due to balancing high performance with the network's hardware constraints. Although pruned, the history will not be lost as it will be permanently archived on Arweave. Since the network history data archived on Arweave will lose its data availability guarantees once the Load Network layer 1 network prunes it, there will be a need for a data storage solution that offers data availability guarantees of large data chunks for the pruned Load Network data.

This positions LOAD AVS as a complementary layer that extends Load Network’s data availability guarantees beyond the network's pruning horizon, while maintaining the performance benefits of the original pruning design. LOAD AVS fits as an effective and aligned solution to extend the longevity of Load Network data availability after pruning, on demand, and for specified period.

3.3 General utility for any blockchain

3.3.1 Improved time-flexible DA with hot cache

LOAD AVS can store data pruned from blockchain nodes, ensuring that the high-throughput DA [31] needed by high-bandwidth networks remains available for a longer period with strong availability guarantees. This ensures that data is accessible and provable in a way that is both secure and economically incentivized by EigenLayer.

3.3.2 Economic security

By building on EigenLayer, LOAD AVS leverages the economic security provided by stETH holders who stake assets to secure the layer. This eliminates the reliance on a chain’s native network token price, greatly improving the robustness of the security models of new chains and mitigating the risk of price manipulation attacks.

3.3.4 Complement to Arweave

LOAD AVS does not replace Arweave as the permanent archive but serves as a complementary layer. While Arweave guarantees permanence, LOAD AVS guarantees the availability and security of recent data for real-time validation, acting as a fast-access storage layer until the data is no longer required for immediate use.

4. Protocol design

4.1 Design principles

4.1.1 KISS (Keep it simple, stupid)

LOAD AVS protocol design and implementation follow the KISS principle, focusing on minimalism, simplicity, efficiency, and modularity. The main node implementation of LOAD AVS - described in section 4.3 - will be written in Rust to leverage the language's built-in features such as type and memory safety, concurrency, and high performance

4.1.2 Strong and low-volatility incentives

Unlike other decentralized data storage protocols that base their incentives and pricing on volatile protocol tokens, LOAD AVS takes a different approach by pricing storage in US dollars. The price will be derived from real-time worldwide average pricing of hard storage (e.g., SSDs, NVMe, etc.). As LOAD AVS aims for hot cache storage, there's no need to adjust the incentives design for years-long storage market changes, which are expected to keep evolving at a high rate. Focusing on short-term storage and real-time, USD-based pricing and incentives allows LOAD AVS to easily adapt to storage market-related changes (hardware pricing) and the global market pricing base (the state of USD dominance).

4.2 LOAD AVS as an Actively Validated Service (AVS)

4.2.1 Introduction to EigenLayer Actively Validated Services (AVS)

4.2.1.1 Actively Validated Services (AVS)

An Actively Validated Service (AVS) is a service built externally to EigenLayer [6] that requires active validation by a set of Operators. An AVS deploys its service manager to interact with EigenLayer core contracts, enabling Operator registration to Operator Sets, slashing, and rewards distribution. Once registered, Operators agree to run the AVS's off-chain code. The LOAD AVS network is an AVS built upon these principles.

4.2.2.2 AVS Operator

An entity that registers an Operator address on EigenLayer to receive Staker delegations and operate AVS infrastructure. These Operators (equivalent to LOAD AVS Nodes in LOAD AVS terminology) allocate their delegated stake across Operator Sets created by their chosen AVS - in this case, the LOAD AVS Network.

4.2.2.3 AVS Operator Set

A distinct grouping of Operators, established by an AVS, that secures specific service tasks using allocated staked assets. These assets may be exclusively reserved for securing that particular set.

4.2.2.4 Staker

An individual address that supplies assets directly to EigenLayer. This address can be an EOA wallet or a smart contract controlled by either an individual or institution.

4.2.2.5 Restaker

An entity that restakes Native or LST ETH within the EigenLayer protocol.

4.3 LOAD AVS Node Components And Design

4.3.1 LOAD AVS Heap

LOAD AVS Heap is a P2P data sharing network that enables user communication with LOAD AVS network operators.

Data transmission over Ethereum [7] is cost-inefficient and not scalable, which is one of the main reasons for building LOAD AVS. Additionally, communicating through data references on the cloud compromises the network's decentralization at its data ingress point.

To address these issues, LOAD AVS Heap serves as a P2P distributed file system, providing a cost-efficient (free) content-addressable method for data ingress communication with LOAD AVS operators.

LOAD AVS Heap achieves these goals by operating as a private IPFS network that exclusively handles LOAD AVS P2P data communication. LOAD AVS Heap has a data retention period of 48 hours, which keeps the data pinned in the P2P data sharing protocol for 2 LOAD AVS network epochs.

4.3.2 LOAD AVS Optimistic Gateway

LOAD AVS Gateway is an optimistic cache for data that passes through data ingress voting (pBFT consensus over data acceptance in the network). Users and developers can use the gateway to optimistically access LOAD AVS-stored data without running an operator or needing to connect with the specific operator gateway storing their data. Gateways can implement their own content management

policies, complying with their jurisdiction's requirements, data retention rules, and optimistic caching latency targets.

4.3.3 LOAD AVS Decentralized Gateways

LOAD AVS decentralized gateways function as partition-specific data serving endpoints, individually operated by partition operators. Each gateway serves objects contained within its operator's partition.

While gateway operation is optional, operators are incentivized to run gateways through Proof of Data Serving (LOAD) rewards, detailed in section 8.3. This incentive structure promotes distributed data availability [31], efficient object retrieval across the network, and less reliance on the LOAD AVS Optimistic Gateway.

4.3.4 LOAD AVS Smart Contracts

A set of smart contracts required to form an AVS based on EigenLayer AVS design specifications [8]. These handle tasks processing, governance, consensus management, slashing, and rewards distribution.

4.3.5 LOAD AVS Operators

LOAD AVS Operators are LOAD AVS Nodes running off-chain AVS components that form the core logic of the network. They are responsible for data storage, proof submission, and voting.

4.3.6 LOAD AVS Aggregator

After LOAD AVS Nodes provide BLS-signed responses to storage requests (AVS tasks), the LOAD AVS Aggregator combines the operators' multiple signatures into a single aggregated BLS signature. It then submits this signature onchain by interacting with the AVS smart contracts.

4.4 LOAD AVS Data Retention

4.4.1 Data retention periods

LOAD AVS network data retention periods are measured in seconds. Time durations used throughout this paper refer to the following conversions:

- 1 day = 86,400 seconds
- 1 month = 31 days = 2,678,400 seconds
- 1 year = 365.25 days = 31,557,600 seconds

A partition is considered active as long as there is at least one active operator (hence, one replica per object).

4.4.2 Maximum storage period

For protocol to be flexible against movements in the market of storage mediums, maximum storage duration has to be introduced, avoiding the need for dynamic endowment adjustment of prices on current storage deals based on volatility in storage medium markets. This limitation cannot be too low, as it would decrease the usability of LOAD AVS and its applications for various use cases.

As the storage medium market is relatively stable, we set this parameter to 1 year (365 days) - a reasonable timeframe to assume no major surges in storage medium prices will occur.

4.4.3 Minimum data retention duration

The minimum data retention period for all operators is set to 30 days. When users submit a data storage transaction, they will be charged for 30 days regardless of whether they plan to store data for less time. This minimum storage period ensures protocol integrity against potential abuse (DDoS) and protects operators' resources

5. LOAD AVS Partitions

5.1 Partitions and operators

LOAD AVS Network implements horizontal scaling, where the network's full ledger consists of partitions [9], each responsible for its assigned data. Each partition contains 1-4 operators, with each operator allocating resources of 500GB SSD NVMe, resulting in partition sizes of 500-2000 GB. Therefore, data stored across the LOAD AVS Network and assigned to a single partition will have 1-4 replicas.

$$R(d) = \min(n, 4)$$

where:

$R(d)$ = number of replicas for data piece d

n = number of active operators in partition p

$$1 \leq n \leq 4$$

Partitions are capped at 2 TB maximum total size. Scaling network storage ingress requires adding more partitions to the network

5.2 LOAD AVS Network Maximum Capacity

The LOAD AVS Network's maximum capacity is determined by the total number of partitions multiplied by the maximum partition size (2 TB). Available storage ingress capacity represents the difference between the network's maximum capacity and currently used storage space.

Maximum Network Capacity (C_{max}):

$$C_{max} = N_p \times S_p$$

where:

C_{max} = Network maximum capacity

N_p = Total number of partitions

S_p = Maximum partition size (2 TB)

Available Storage Capacity ($C_{available}$):

$$C_{available} = C_{max} - S_{used}$$

where:

$C_{available}$ = Available storage ingress capacity

S_{used} = Total used storage space

When the network reaches 70% of maximum storage capacity, it will automatically create a new partition and increase the network capacity to make zero-space scenarios unlikely and prevent periods of network unusability.

5.3 Partition Bucket-Object storage

5.3.1 LOAD AVS Buckets

To store data (objects) on the LOAD AVS Network, users must first create a bucket in their preferred partition. The partition choice typically depends on several factors: operator policies, geographic location, available data capacity, and terms offered by partition operators - all of which the bucket inherits.

This design implements a flat structure of buckets [10]. The object storage system uses this flat structure along with metadata and unique identifiers for each object, making it efficient to locate specific objects among potentially billions of stored items.

5.3.2 Bucket Structure

Each bucket on LOAD AVS has unique identifiers that define its identity and location within the network. Buckets contain unique names, access control (managed by the bucket creator/admin), location address (partition ID), and object placeholders. Bucket management occurs onchain through the AVS smart contracts.

```
struct Bucket {
    bytes32 name;         // Unique name of the bucket (auto-assigned as bytes32)
    uint32 type;         // bucket type identifier (1 or 2)
    uint32 fab_size;     // > 0 when type == 1 (represent max bytes)
    uint32 credits;      // unused credits
    address renter;      // Creator/admin of the bucket
    address[] admins;     // List of admins who can manage access
    uint32 location;      // Location on LOAD AVS Network (partition id)
    mapping(string => Object) objects;    // Placeholder for objects in the bucket
}

function generateUniqueBucketName() internal view returns (bytes32) {
    return keccak256(
        abi.encodePacked(
            msg.sender,
            block.timestamp,
            block.number,
            tx.gasprice
        )
    );
}
```

Bucket names are 32-byte arrays, which translate to 66 characters in hexadecimal string representation (including the 0x prefix).

5.3.3 Bucket Types

Buckets [11] have different types that are immutably determined at bucket creation.

5.3.3.1 Fixed Allocation Bucket (FAB)

FAB Buckets (type 1) function like renting fixed partition space, regardless of actual bucket usage (number of objects stored). Users rent storage space in the partition where the bucket is allocated. FAB's maximum size must be less than or equal to the partition size.

5.3.3.2 Ghost Buckets

Ghost Buckets (type 2) are prunable after 6 epochs (~12 hours) of inactivity (when they have no objects). Operators are incentivized to prune these buckets after reaching the pruning epoch [12] to free up space for active data egress. Ghost Buckets have dynamic sizing and operate on a pay-as-you-go model.

5.3.4 Bucket Deletion

Buckets are deleted under two conditions:

5.3.4.1 Owner-Initiated Deletion When a bucket owner explicitly requests deletion, triggering:

- Immediate bucket removal
- Credit refund of remaining balance to renter

5.3.4.2 Zero-Operator Deletion When a partition reaches zero active operators:

- All partition buckets are automatically deleted
- Full refund of delegated credits to respective renters

5.3.4 LOAD AVS Objects

LOAD AVS Objects are stored within user buckets. Minimal Object metadata must exist onchain as operators will download the actual Object data from the LOAD AVS Heap and vote on its correct data seeding and structure:

```
struct Object {
    bytes32 hid;        // Heap ID in bytes32 format
    uint40 timestamp;   // Creation timestamp (40 bits = until year 2078)
    address owner;      // Object creator/owner
}
```

The Object data structure on LOAD AVS Heap is defined as:

```
#[derive(
          Debug,  Default,  Serialize,  Deserialize,  PartialEq,  BorshSerialize,
BorshDeserialize, Clone,
)]

pub struct Tag {
    pub name: String,
    pub value: String,
}

#[derive(
          Debug,  Default,  Serialize,  Deserialize,  PartialEq,  BorshSerialize,
BorshDeserialize, Clone,
)]
pub struct ObjectMetadata {
    pub content_type: String,           // MIME type of the content
    pub size: u64,                      // Size in bytes
    pub created_at: u32,                // Ethereum Blockheight
    pub tags: Option<Vec<Tag>>,   // Custom metadata tags
}


#[derive(
          Debug,  Default,  Serialize,  Deserialize,  PartialEq,  BorshSerialize,
BorshDeserialize, Clone,
)]
pub struct Object {
    pub name: String,                   // Object name/key
    pub bucket_name: String,            // Reference to parent bucket
    pub metadata: ObjectMetadata,       // Current object metadata
    pub data_location: u32,             // Bucket Partition location
    pub data: Vec<u8>,                  // raw object data
}
```

5.3.5 HTTP API

Object data within buckets is accessible through an HTTP API [13] (via LOAD AVS Gateways), following standard bucket-object storage API conventions [14][15].

6. Storage Economics and Pricing Model

6.1 LOAD AVS Payment Tokens

The LOAD AVS Network processes payments using USD-pegged cryptocurrencies (e.g., USDC [16], USDT [17]). The network's accepted USD stablecoins (which may vary over time) are used for user deposits (storage credits) and storage payments (rewards [18]) to operators.

6.2 LOAD AVS Deposits

The deposit system reduces gas costs by enabling storage deal smart contracts to track balances locally, rather than receiving protocol-accepted stablecoins [19] each time a user initiates a storage deal.

6.3 Price Discovery Mechanism

The network implements a dynamic pricing model based on hardware costs and operational parameters. Storage costs are derived from real-time SSD (NVMe) market prices through web3 smart contract oracles [20]. All payments and calculations are performed in USD-pegged LOAD AVS supported stablecoins to ensure price stability and predictability.

Let Ω represent the network's state space, where:

$$\Omega = \{\chi, \alpha, r, \iota\}$$

where:

$\chi \in \mathbb{N}$: Drive capacity (GB)

$\alpha \in \mathbb{R}+$: Hardware cost (USD)

$r \in [1,4]$: Replication factor

$\iota \in \mathbb{R}+$: Incentive rate (USD/GB/epoch)

6.3.1 Oracle Integration

The system employs a permissionless price update mechanism with the following constraints:

- Update frequency: τ blocks (≈ 1 hour)
- Price aggregation: USD-denominated
- Data sources: web3 decentralized oracle network aggregating retail hardware prices

The hardware cost oracle feed F(t) at time t is defined as:

$$F(t) = \text{median}(\{h(t-k) \mid k \in [0, 24h]\})$$

where $h(t)$ is the spot hardware price at time t

6.4 Economic Model

6.4.1 External Incentives

The protocol implements an inflation-based incentive mechanism ι (in LOAD tokens), distributed per GB-epoch to maintain competitive pricing while ensuring operator sustainability. This mechanism allows the network to:

- Subsidize operator costs beyond upload fees
- Maintain competitive pricing compared to services with fewer replicas
- Incentivize network usage through GB/epoch rewards

6.4.2 Storage Price Function

The fundamental pricing function P: $\mathbb{R}+ \times \mathbb{R}+ \rightarrow \mathbb{R}+$ is defined as:

$$P(s,t) = \frac{\alpha \cdot s \cdot r \cdot t}{\chi \cdot \tau} - \iota \cdot s$$

Subject to:

$s > 0$: Storage size in GB

$t \geq t_{min}$: Storage duration

$\tau = 31{,}557{,}600$: Target period in seconds (1 year)

$r \leq 4$: Maximum replication factor

6.5 Network Economic Constraints

The network operates under the following economic constraints:

$\forall p \in P : p \geq 0$ (Non-negative pricing)

$$\sum_{i=1}^{n} s_i \leq N \cdot \chi$$ (Network capacity constraint)

where:

s_i: Individual storage allocations

N: Number of active partitions

6.6 Storage Cost Simulation

Given the pricing formula defined in section 6.4.2, we simulate the storage costs using the following parameters:

α = \$50 (SSD drive cost)

χ = 500 GB (Drive capacity)

r = 4.0 (Replication factor)

τ = 31,557,600 (Target period in seconds)

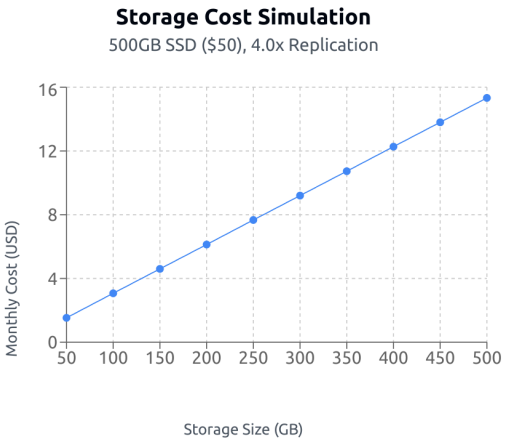
t = 2,592,000 (Time unit in seconds)

ι = 0.001 (Incentive rate USD/GB/month, paid in LOAD tokens)

The simulation demonstrates the linear relationship between storage size and monthly cost. For a given storage size s, the monthly cost follows our pricing function P(s,t). Key observations from the simulation:

- Cost for 1GB storage per month: \$0.0307
- This includes 4.0x replication for data reliability

The graph below illustrates the relationship between storage size and monthly cost, demonstrating the linear scaling of our pricing model while maintaining cost efficiency through hardware resource optimization and replication factor considerations.



This simulation validates that our pricing model achieves both economic sustainability and market competitiveness while ensuring data reliability through replication [21].

7. Operator Economics and Hardware Lifecycle

7.1 Initial Investment Analysis (4 Operator Nodes, 1 Partition)

Hardware Costs:

- 4 × 500GB NVMe SSDs @ \$50 each = \$200 initial investment
- Supporting infrastructure costs not included (AVS staking, cloud, compute, network, onchain fees, etc.)

7.2 Revenue Calculation

Using our pricing formula P(s,t) defined in section 6.4.2 with full capacity utilization:

Monthly Revenue per Node = P(500, t) = \$11.695 per 500GB

Total Monthly Revenue (4 nodes) = \$46.78

7.2.1 Break-even Analysis

Break-even Period = Initial Investment / Monthly Revenue

\$200 / \$46.78 = 4.28 months

7.2.2 SSD Longevity Analysis [22]

7.2.2.1. TBW (Terabytes Written) Calculations:

Typical NVMe SSD (500GB) specifications:

- TBW rating: ~400 TBW
- DWPD (Drive Writes Per Day): 0.4
- Daily Write Limit = 500GB × 0.4 = 200GB per day

7.2.2.2. Write Amplification Factors:

Daily writes considering factors:

- User data writes
- Replication overhead (4x)
- Garbage collection (~1.1x)
- System metadata (~1.05x)

Total Write Amplification = Base writes × 4 × 1.1 × 1.05

7.2.2.3. Estimated Lifespan Calculation:`

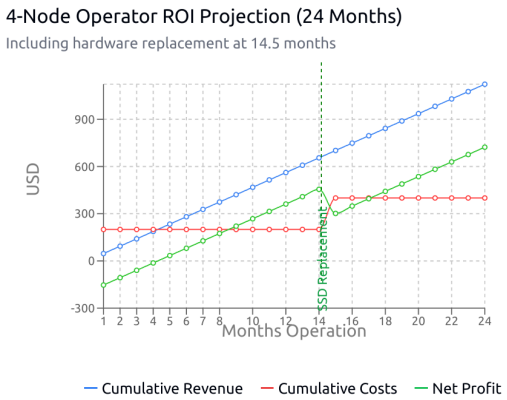
Given 400 TBW rating:

Maximum Data Written = 400,000 GB

Daily Write Load (with amplification) = 200GB × 4 × 1.1 × 1.05 = 924GB

Theoretical Lifespan = 400,000 / 924 = 433 days (~1.19 years)

7.3 ROI Projections



7.3.1 Key Economic Indicators

7.3.1.1. Initial 14.5-month cycle (before first SSD replacement):

- Total Revenue: $888.82
- Initial Investment: $200
- Net Profit before replacement: $688.82
- ROI: 344.41%

7.3.1.2. Risk Factors:

- Network utilization variations
- Hardware failure before TBW limit
- Market price fluctuations
- Network incentive adjustments
- Ethereum network fees
- AVS staked assets associated risks
- Oracle attacks [23]

The analysis suggests that operators can achieve ROI within ~4.28 months under optimal conditions, with substantial profit potential over the hardware lifecycle. However, operators should plan for hardware replacement at approximately 14.5-month intervals and maintain reserves for unexpected replacements.

8. Cryptographic storage proofs

To establish a mechanism providing us with reasonable assumptions about honest operation of storage operators, mentioned operators will be periodically challenged by The Challenger to submit a proof of work [24]. Specifically, at regular intervals T, every operator must provide a proof comprising the hash of the partition, the operator's address, and a nonce. The computational effort for generating this proof is constrained to approximately $T/3$ seconds on an average CPU, assuming a statistically average system at the time.

The allocation of $T/3$ time serves multiple purposes:

1. **Energy and computational efficiency**. Limiting the computation time conserves both computing resources and energy consumption.
2. **Equity among hardware capabilities**. By restricting the proof generation time, advantages held by faster CPUs or specialized hardware such as ASICs [25] are mitigated. Even if an ASIC can compute the hash in less time (such as $T/5$), all operators are still required to submit their proofs within the T interval.
3. **Incentivization of cost-effective hardware usage**. This constraint incentivizes the use of less expensive CPUs, promoting the utilization of superior storage mediums over high-performance compute units.

This protocol ensures that, with the most cost-effective configuration, operators must retain the data on local storage devices for at least one-third of the designated time period. Relying on network storage and generating hashes during data retrieval would significantly reduce the time available to produce the proof. Such a reduction limits the ability of dishonest operators to consistently generate valid proofs within the required timeframe, as it would necessitate excessive bandwidth usage. Consequently, the most viable strategy for operators is to maintain the data on local drives.

Even if an operator attempts to delete the data after computing the hash, this approach is suboptimal. Re-downloading the entire partition at each interval would demand substantial bandwidth, which is more efficiently allocated to transferring new data to the network, giving new fees and incentive to operators doing it.

In the pessimistic scenario where an operator successfully removes the data during the two-thirds of the period when proof computation is not required, the advantage of such dishonest behavior remains ambiguous. Assuming the

availability of free or high-speed bandwidth, the space freed on the drive must eventually be replenished with the deleted data after the two-thirds period elapses. Given that re-downloading a potentially large partition is time-consuming, the window during which the operator can access the stored data remains minimal, thereby limiting the practicality of such a storage strategy.

Storage proof verification cannot be efficiently performed via external observers (i.e. onchain smart contracts) without direct dataset access, as obtaining dataset hashes would be impossible. The responsibility of verifying storage proofs lies with other operators, using majority vote to determine proof validity via pBFT [5] consensus. The system implements an optimistic model where only invalid proofs trigger voting, while valid proofs pass without vote. The Challenger aggregates operator proofs and raises disputes when invalid proofs are detected.

8.1 Incentives of participation in storage proof system

Incentives need to be in place to ensure that operators are motivated to participate in the validation/submission process:

- **Incentives for validating proofs**. A portion of the slashed penalties from invalid proofs should be distributed among validators who agree on the invalidity of a proof.
- **Disincentives for False Positive Votes**. Operators who incorrectly vote to invalidate a proof, without reaching a majority consensus, must incur penalties. This discourages dishonest attempts to undermine valid proofs. After each epoch with valid Proof of Storage voting, the system distributes object bucket rewards (stablecoins) to partition operators on a pro-rata basis.

8.2 Proof-optimized hashing algorithm: Proof of Probabilistic Chunk-Sampling Hash (PoPCSH)

Usual hashing algorithms must account for every bit of information to generate the final output of a hash function. While this characteristic is crucial in use cases of hashing functions where hash is responsible for data integrity, it may be not the best choice when it comes down to proving storage of a large blob of data via PoW.

Not only is taking full data computationally suboptimal, it also forces operators to store all the content of the partition, which may affect operation of storage nodes in problematic regions.
LOAD AVS uses a hashing function that would consider only ϱ percentage of the data (ϱ = 25%). This way, operators have the ability to generate proof of storage having just ϱ of data, although the time to generate such proof increases to keep storing the whole partition the most optimal strategy.

The hashing function is multi-round and stateful: Picking small chunks scattered among whole data, updating state of hash and thus next chunk to hash.

Described hashing function can be represented in pseudocode as follows:

```
Function diskHash(D, ρ, k, S)
    Given:
        D: original dataset
        ρ: selection ratio, % (0 < ρ ≤ 100)
        k: chunk size
        S: seed

    N ← |D| / k
    M ← ⌈ρN⌉ - 1
    D_0 ← first k elements of D
    H_0 ← hash(D_0 ‖ S)

    for m = 1 to M do
        j_m ← H_{m-1} mod N
        D_j ← j_m-th chunk of D (D[j_m·k : (j_m+1)·k])
        H_m ← hash(H_{m-1} ‖ D_j ‖ S)

    return H_M
```

8.3 Simplified Proof of Data Sharing (SPoDS)

SPoDS represents the third, optional challenge performed by the Challenger on operators running LOAD AVS Gateways alongside their Operator Nodes. The "Simplified" designation reflects the straightforward proving mechanism: random checks by the Challenger verify correct data serving from LOAD AVS Gateways.

8.3.1 Incentives

Gateway operators receive LOAD tokens as incentive for maintaining data availability. This creates an additional revenue stream beyond basic storage rewards.

8.3.2 Future Development

Future versions of this paper will expand SPoDS specifications and detail upgrades to the full PoDS system.

9. AVS Storage Visualized

9.1 Bucket Creation

The storage process begins when a renter (which can be a user, AI agent, smart contract, DePIN node, or other entity) creates a bucket on the LOAD AVS smart contract. Two bucket types are available:

- **Fixed Allocation Bucket (FAB):** Pre-allocated fixed storage space
- **Ghost Bucket:** Dynamic storage allocation with automatic pruning capability

9.2 Object Submission

Renters construct and submit valid objects to the LOAD AVS Network through one of two pathways:

- via the Sequencer
- Direct interaction with LOAD AVS smart contracts

9.3 Storage Workflow Validation Process

LOAD AVS Operators perform a pBFT consensus vote to validate:

- Object data structure validity
- Correct data seeding
- Sufficient renter credit balance
- Available network storage capacity

The Aggregator collects signatures from operators and aggregates them using BLS aggregation. When a response passes the pBFT quorum threshold (67%), the aggregator posts the aggregated response to the AVS smart contract, optimizing onchain interactions.

9.4 Capacity Management

If the LOAD AVS Network reaches maximum partition capacity across all partitions:

- New object submissions are automatically rejected
- A new partition is created
- Three new operators are allowed to join the new partition
- Normal validation process resumes

9.5 Gateway Caching

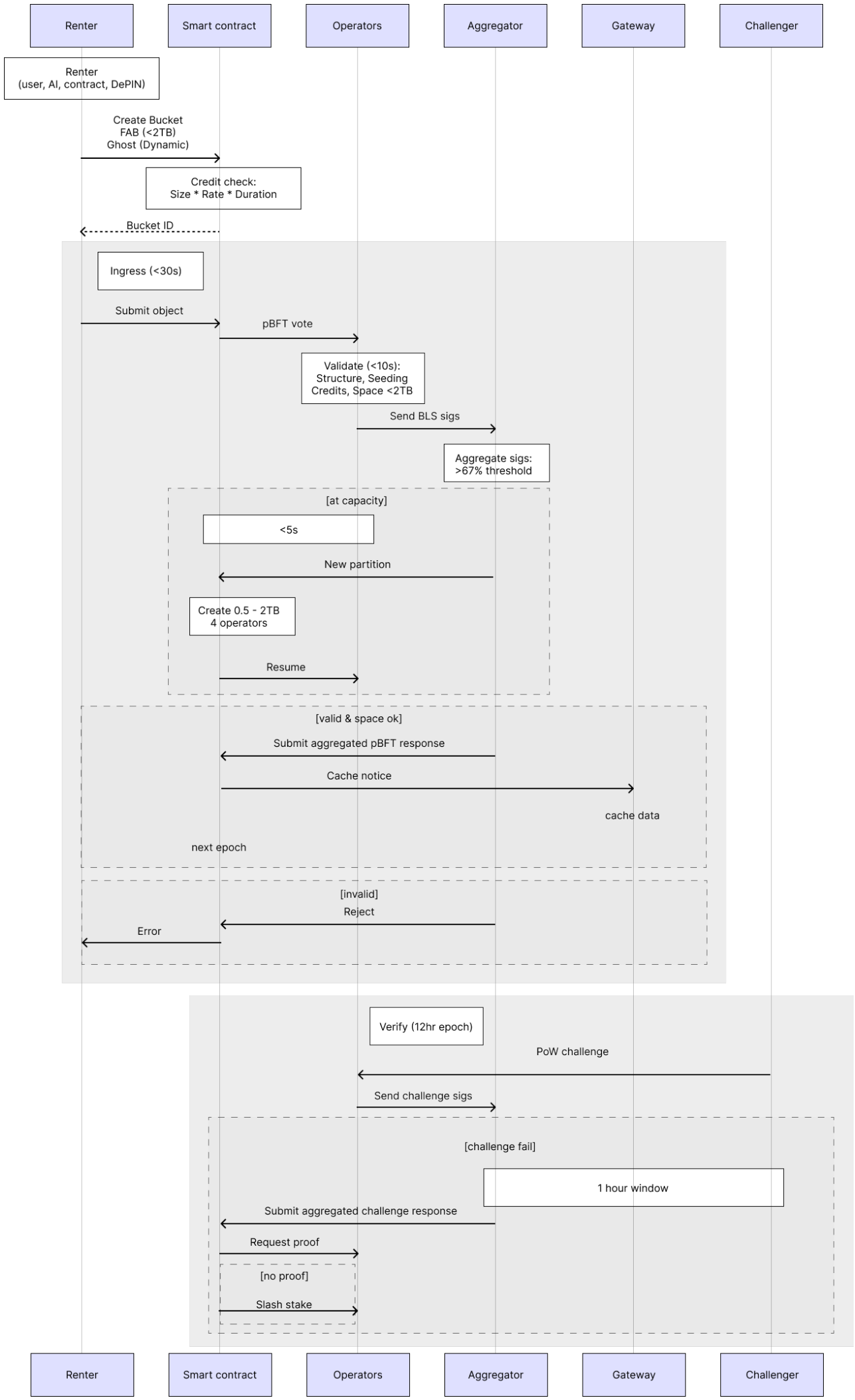
Upon successful pBFT consensus:

- LOAD AVS Gateway optimistically caches the object data
- Object becomes immediately available for its specified duration
- Object is added to the next epoch queue

9.6 Verification and Challenges

After one epoch:

- The Challenger initiates off-chain Proof of Storage challenges (PoPCSH type)
- Operators of the bucket's partition must respond
- Failed challenges trigger onchain dispute resolution
- Operators face penalties if unable to provide valid proof


Renter
Smart contract
Operators
Aggregator
Gateway
Challenger
Renter
(user, AI, contract, DePIN)
Create Bucket
FAB (<2TB)
Ghost (Dynamic)
Credit check:
Size * Rate * Duration
Bucket ID
Ingress (<30s)
Submit object
pBFT vote
Validate (<10s):
Structure, Seeding
Credits, Space <2TB
Send BLS sigs
Aggregate sigs:
>67% threshold
[at capacity]
<5s
New partition
Create 0.5 - 2TB
4 operators
Resume
[valid & space ok]
Submit aggregated pBFT response
Cache notice
cache data
next epoch
[invalid]
Reject
Error
Verify (12hr epoch)
PoW challenge
Send challenge sigs
[challenge fail]
1 hour window
Submit aggregated challenge response
Request proof
[no proof]
Slash stake
Renter
Smart contract
Operators
Aggregator
Gateway
Challenger

10. Use Cases

The LOAD AVS Network's architecture enables diverse applications across multiple domains. Here are the key use cases that demonstrate its utility:

10.1 Data Lake Infrastructure

LOAD AVS Network serves as an open, decentralized data lake [26] architecture facilitating permissionless data access, contribution, and sharing. Organizations can build collaborative data ecosystems with support for both structured and unstructured data storage, powering data-heavy cloud computing protocols (e.g. onchain serverless functions, onchain AWS Lambda)

10.2 Smart Contract Interoperability

Entry and exit points managed through AVS Ethereum mainnet contracts enable permissionless cross-contract communication. This architecture supports automated data storage and retrieval while facilitating trustless contract-to-contract interactions.

10.3 AI Agent Infrastructure

Serving web3-aligned AI agents [27], both onchain and hybrid, LOAD AVS provides reliable data storage for AI training and inference. The network's architecture ensures high-performance data retrieval with permissionless access for autonomous systems.

10.4 DePIN Support

LOAD AVS functions as a decentralized storage backbone for DePIN [28] networks, enabling efficient data management for IoT devices. The network's scalable architecture supports real-time data ingestion and retrieval, making it ideal for sensor data storage and management.

10.5 Web Content Hosting

The network supports static website hosting with decentralized content delivery. This provides a reliable and cost-effective alternative to traditional hosting solutions while maintaining high availability through operator replication.

10.6 Decentralized File Sharing

As a decentralized alternative to centralized storage systems, LOAD AVS enables secure peer-to-peer file sharing with support for various file types and sizes[1][2][3]. Data availability is maintained through systematic replication across operators.

10.7 EIP-4844 data longevity

LOAD AVS can be used as a way to extend the lifetime of EIP-4844 blobs [29], powering archives and applications that depend on historical data. Both EIP-4844 on EVM chains and blob implementations in alternative data availability [31] layers prune data after a short period of time, pushing users to deploy their own storage solutions.

10.8 EIP-4444 storage

EIP-4444 [30] proposes that historical chain data will be pruned to keep node hardware requirements low. LOAD AVS can serve as decentralized data storage for chains that implement this, ensuring historical data is retrievable and hardware requirements can be kept low.

Also, high-throughput L2 chains that implement pruned nodes [ref] need to maintain time-flexible DA [31] and historical data storage after pruning their own on-node storage.

11. References

[1] Cohen, B. (2003). "Incentives build robustness in BitTorrent." Workshop on Economics of Peer-to-Peer systems, 6, 68-72. https://bittorrent.org/bittorrentecon.pdf

[2] Benet, J. (2014). "IPFS - Content Addressed, Versioned, P2P File System." arXiv preprint arXiv:1407.3561. https://arxiv.org/abs/1407.3561

[3] Protocol Labs. (2017). "Filecoin: A Decentralized Storage Network." Technical Whitepaper. https://filecoin.io/filecoin.pdf

[4] Williams, S., et al. (2019). "Arweave: A Protocol for Economically Sustainable Information Permanence." Technical Whitepaper. https://www.arweave.org/whitepaper.pdf

[5] Miguel C., Barbara L. (1999). “Practical Byzantine Fault Tolerance” https://www.pmg.csail.mit.edu/papers/osdi99.pdf

[6] Wang, G., et al. (2024). "EigenLayer: Restaked Security for Smart Contracts." Technical Whitepaper. https://docs.eigenlayer.xyz/eigenlayer/overview/whitepaper

[7] Buterin, V. (2013). "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform." https://ethereum.org/en/whitepaper

[8] EigenLabs. (2023). "EigenLayer: The Restaking Primitive for Ethereum." EigenLayer Documentation. https://docs.eigenlayer.xyz/eigenlayer/overview/

[9] "Disk Partitioning." Wikipedia. https://en.wikipedia.org/wiki/Disk_partitioning

[10] "Object Storage." Wikipedia. https://en.wikipedia.org/wiki/Object_storage

[11] Oracle Cloud Infrastructure. (2023). "Managing Buckets." Oracle Cloud Documentation. https://docs.oracle.com/iaas/Content/Object/Tasks/managingbuckets.htm

[12] "Epoch (computing)." Wikipedia. https://en.wikipedia.org/wiki/Epoch_(computing)

[13] MDN Web Docs. (2023). "HTTP Overview." Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[14] "RESTful API Design - Object Storage." https://cloud.google.com/storage/docs/json_api

[15] "Object Storage API Reference." https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html

[16] Circle Internet Financial. (2018). "USDC: A Price-Stable Cryptocurrency for Global Payments." Circle Documentation. https://www.circle.com/en/usdc

[17] Tether Operations Limited. (2014). "Tether: Digital Money for a Digital Age." Tether Documentation. https://tether.to/en/transparency

[18] EigenLabs. (2023). "EigenLayer Rewards Claiming Overview." EigenLayer Documentation. https://docs.eigenlayer.xyz/eigenlayer/rewards-claiming/rewards-claiming-overview

[19] Coinbase. (2023). "What is a stablecoin?" Coinbase Learn. https://www.coinbase.com/learn/crypto-basics/what-is-a-stablecoin

[20] Chainlink. (2023). "Price Feed Oracle Networks: Hardware Pricing Integration." https://docs.chain.link/data-feeds

[21] LOAD AVS Storage Replication in Distributed Systems. https://en.wikipedia.org/wiki/Replication_(computing)#Storage_replication

[22] Microsoft Tech Community. (2023). "Understanding SSD Endurance: Drive Writes Per Day (DWPD), Terabytes Written (TBW)." Microsoft Documentation. https://techcommunity.microsoft.com/blog/filecab/understanding-ssd-endurance-drive-writes-per-day-dwpd-terabytes-written-tbw-and-/426024

[23] Chainlink. (2023). "Market Manipulation vs. Oracle Exploits: Understanding DeFi Security Risks." Chainlink Education Hub.

https://chain.link/education-hub/market-manipulation-vs-oracle-exploits

[24] Dwork, C., & Naor, M. (1992). "Pricing via Processing or Combatting Junk Mail." Annual International Cryptology Conference, CRYPTO '92. (Proof of Work) https://link.springer.com/chapter/10.1007/3-540-48071-4_10

[25] "Application-Specific Integrated Circuit (ASIC)." Wikipedia. https://en.wikipedia.org/wiki/Application-specific_integrated_circuit

[26] Google Cloud. (2023). "What is a data lake?" Google Cloud Documentation. https://cloud.google.com/learn/what-is-a-data-lake

[27] Walters, S., Gao, S., Nerd, S., Da, F., Williams, W., Meng, T.C., Han, H., He, F., Zhang, A., Wu, M., Shen, T., Hu, M., & Yan, J. (2025). "Eliza: A Web3 friendly AI Agent Operating System." arXiv:2501.06781. https://arxiv.org/abs/2501.06781

[28] Lin, Z., Wang, T., Shi, L., Zhang, S., & Cao, B. (2024). "Decentralized Physical Infrastructure Network (DePIN): Challenges and Opportunities." arXiv:2406.02239. https://arxiv.org/html/2406.02239v1

[29] Buterin, V., Feist, D., Loerakker, D., Kadianakis, G., Garnett, M., Taiwo, M., & Dietrichs, A. (2022). "EIP-4844: Shard Blob Transactions." Ethereum Improvement Proposals. https://eips.ethereum.org/EIPS/eip-4844

[30] Kadianakis, G., Garnett, M. (lightclient), & Stokes, A. (2021). "EIP-4444: Bound Historical Data in Execution Clients." Ethereum Improvement Proposals. https://eips.ethereum.org/EIPS/eip-4444

[31] Chaudhuri, A., Basak, S., Kiraly, C., Ryajov, D., & Bautista-Gomez, L. (2024). "On the Design of Ethereum Data Availability Sampling: A Comprehensive Simulation Study." arXiv:2407.18085. https://arxiv.org/abs/2407.18085