0G: Towards Fully Decentralized AI Operating System

Zero Gravity Labs
0g@0g.ai

September 4, 2025

Abstract

The advent of large language models, coupled with the rapid development of parallel computing hardware (GPUs, TPUs, and NPUs), has significantly accelerated the progress toward AGI. Recent advanced models, trained on vast amounts of data from across the internet, have demonstrated significant superiority over humans in many aspects and real-world scenarios. However, the current centralized and monopolistic model training process has raised many concerns regarding the security, privacy, and fairness of applying these powerful models. Therefore, there recently appears a strong demand for transparency and democratization in the production and deployment process of AI products. Decentralization powered by blockchain technologies has been recognized as a promising path to achieve this. Unfortunately, the infrastructures in current blockchain industry still face tremendous scalability challenges in achieving the internet-level scale required for real-world AI systems and application scenarios. 0G makes a first step towards the ultimate solution by providing a decentralized AI operating system through a modular and layered architecture design. It consists of a storage network, a data availability network, and a data serving network, managing the computing and storage resources in all these networks in a permissionless way, and orchestrating them organically with the separate 0G consensus network. All of these network components utilize carefully designed and innovative sharding mechanisms to achieve infinite horizontal scalability, thereby removing obstacles to the true democratization of AI.

1 Introduction

Artificial Intelligence (AI) has rapidly evolved into a powerful tool that is transforming industries and reshaping our everyday lives. However, the journey of AI development and deployment is often confined within centralized platforms, leading to significant concerns around privacy, fairness, and alignment. As we continue to witness the proliferation of AI, a critical trend is emerging: the democratization of AI through decentralization.

Centralized AI platforms, while convenient and powerful, introduce several critical issues. When a handful of entities control the AI infrastructure, they also control the data and the algorithms. This centralization creates a scenario where users' privacy can be compromised, either through data breaches or through misuse of personal information. Moreover, the decision-making processes of AI systems can become opaque, making it difficult to ensure fairness and accountability. Alignment, or ensuring that AI systems act in accordance with human values and intentions, becomes particularly challenging in centralized environments. If AI development is monopolized by a few organizations, the risk of misalignment with the broader population's values increases, potentially leading to outcomes that are beneficial to some but detrimental to others.

Decentralization offers a promising alternative. By moving AI workflows into a decentralized environment, the entire process becomes more transparent. Every step of the AI lifecycle—from data collection and processing to model training and deployment—can be tracked, audited, and verified. This transparency helps in addressing concerns about provenance (where the data comes from) and attribution (who owns or has contributed to the data and models). Furthermore, a decentralized approach makes it easier to ensure that AI systems are aligned with a broader and more diverse set of values, as it is less likely to be controlled by a single entity with a narrow agenda.

However, while the benefits of decentralizing AI are clear, the challenges are equally daunting. The AI workflow is inherently complex, involving massive amounts of data processing and neural network computing. Replicating this workflow in a decentralized environment is non-trivial. It requires well-designed abstraction and super scalable infrastructure that can handle the demands of AI workloads while maintaining the benefits of decentralization.

0G makes a first step towards the ultimate solution by providing a de-

centralized AI operating system through a modular and layered architecture design. It consists of a storage network, a data availability (DA) network, and a data serving network, managing the computing and storage resources in all these networks in a permissionless way, and orchestrating them organically with the separate 0G consensus network. All of these network components utilize carefully designed and innovative sharding mechanisms to achieve infinite horizontal scalability, thereby removing obstacles to the true democratization of AI.

The 0G storage network consists of a bunch of storage nodes connected through a peer-to-peer gossip network. Each data block written into the storage network accompanies a transaction on the consensus network to record the commitment and the ordering of the data. To scale the data throughput infinitely, the storage network is organized in a partitioned way and connects to an arbitrary number of consensus networks that run in parallel and independently. The data requests from different independent applications may be written into different storage partitions and their data commitments can be recorded into different consensus networks simultaneously. All the consensus networks share the same set of validators with the same staking status so that they keep the same level of security. Each storage node actively participates in a mining process by submitting proof of accessibility for a specific piece of data to a smart contract deployed on a consensus network. Once the proof is validated by the smart contract, the storage node gets rewarded accordingly. This incentive-based mechanism rewards the nodes for contributions rather than punishing them for misbehaviors, so it can better encourage nodes to participate in the maintenance of the network, and hence can promote the network to achieve better scalability in practice. 0G storage is also designed as a general storage system with multiple stacks of abstractions and structures including an append-only log layer for archiving unstructured data and a key-value layer for managing mutable and structured data. This allows 0G to support reliable data indexing and a greater variety of data types for AI processing.

The 0G DA network comes from the demand for off-chain verification of executed states in Layer2s or decentralized AI networks with optimistic mechanism. Specifically, the off-chain verifier, usually the light client, needs to be able to access the entire transaction history data to verify the execution of the transactions. In Layer2 scenarios, the blocks containing executed

transactions in Layer 2 networks need to be published and stored somewhere for light client to conduct further verification. Similar requirements also exist in decentralized AI infrastructures where the results of training or inference tasks on devices in the networks need to be further verified due to the demands of users or system incentives. For example, in ORA/OpML [3] scenarios, it requires participants to provide fraud proofs for specific AI tasks during a challenge window and the proofs may need to contain the data and models used in those tasks. Some incentive mechanisms may also require randomly choosing and verifying old historical tasks to give rewards accordingly, and hence to achieve a good trade-off between verification cost and effectiveness.

In contrast to other alternative DA solutions, e.g., Celestia [5] and EigenDA [6], 0G DA targets to the ultimate scalability and security. It embraces the idea of separating the workflow of data availability into both the Data Publishing Lane and the Data Storage Lane. The large volume of data transfers happen on the Data Storage Lane that is supported by the DA nodes with horizontal scalability through erasure-coding based data slicing, while the Data Publishing Lane guarantees the data availability property by checking the aggregated signatures of the corresponding DA nodes on the consensus network, which only requires tiny data flowing through the consensus protocol to avoid the broadcasting bottleneck.

The DA nodes in the network also need to be stakers on 0G consensus and form a series of quorums with majority honesty assumption. This is the security foundation of the data availability. The quorums of 0G are constructed randomly by the consensus system through verifiable random function (VRF) which theoretically guarantees the same distribution of the honest participants as in the validator set of the entire consensus network, so that the data availability client cannot collude with them. In other words, as subsets of all the validators, the quorums share the same security property as the entire consensus network. The aggregated signatures of the quorum will be submitted to the consensus of 0G for data availability confirmation, which can be orders of magnitude faster and more efficient than Ethereum. Combined with the multi-consensus design of 0G with infinite scalability, the consensus protocol will not become the bottleneck of the data throughput. In addition, as with EigenDA, the data availability client needs to conduct erasure coding for the data to have it split into chunks. In practice, this

process is very costly and can be the major bottleneck for the throughput of a single client. 0G provides a GPU-accelerated solution for the erasure coding process to significantly speed it up.

Further, in order to support efficient and scalable AI computation and data access, 0G proposes a decentralized serving network. It is a general framework that can support all kinds of serving workloads including data retrieval, AI inference, and even AI training tasks. It consists of a set of smart contracts and distributed software to allow any type of services to integrate into the decentralized network. Attached with 0G serving framework SDK, a service provider can register and publish the service type, pricing, and verification method into the smart contract. A user can discover a service that she is willing to use through our platform and pre-pay certain amount of 0G tokens to the smart contract. She then can directly interact with the service, sending request, receiving and verifying the response, and acknowledging the response with signature. The service provider maintains the request/response sequence with user acknowledgment and send it to the smart contract for settlement at any time to get the corresponding reward from the user paid fees. Through this way, the service workload can be perfectly partitioned for horizontal scalability and the blockchain overhead is minimized. Since all the service process traces are sent to the smart contract for settlement, these information can also be used to evaluate the contribution and quality of the services in the entire network, which enables the appropriate and fair incentives for better service providers.

2 Overview of the System Design

0G system consists of a data availability layer (0G DA) on top of a decentralized storage system (0G Storage). There are one or multiple separate consensus networks that are part of both the 0G DA and the 0G Storage. For 0G Storage, each consensus network is responsible for determining the ordering of the uploaded data blocks, realizing the storage mining verification and the corresponding incentive mechanism through smart contract. For 0G DA, the consensus is in charge of guaranteeing the data availability property of each data block via verifying the aggregated signatures from the corresponding data availability quorum.

Figure 1 illustrates the architecture of the 0G system when deploying a

single consensus network. When a data block enters the 0G DA, it is first erasure coded and organized into multiple consecutive chunks through erasure coding. The merkle root as a commitment of the encoded data block is then submitted to the consensus layer to keep the order of the data entering the system. The chunks are then dispersed to different storage nodes in a quorum in 0G Storage. The storage nodes periodically participate the mining process by interacting with the consensus network to accrue rewards from the system. The DA client then collects the aggregated signatures from the quorum and submits the signatures to the consensus for verification.

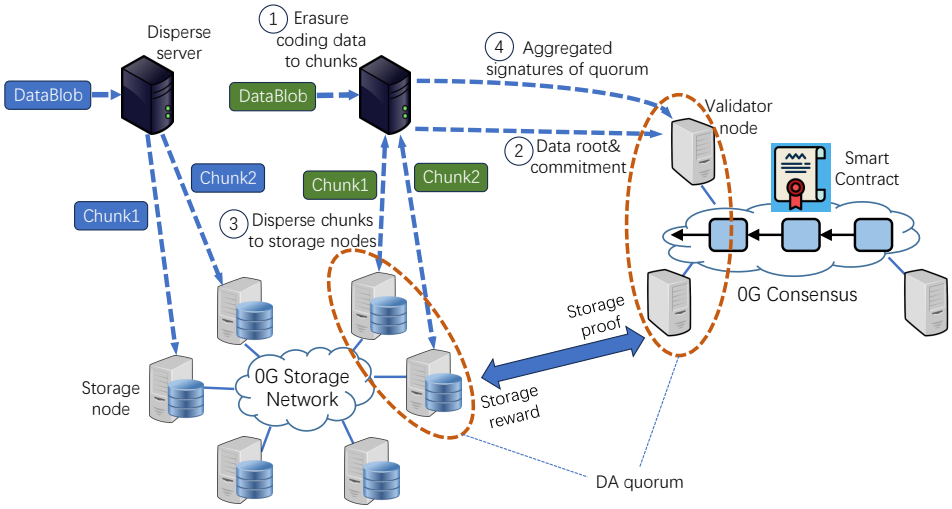

Figure 1: The architecture of the 0G system.

When deployed with multiple POS-based consensus networks, each validator participates in the maintenance of all the consensus networks by using a shared staking status. The shared staking status can be recorded in a smart contract on one of the multiple consensus networks or a blockchain network outside of the 0G consensus network set, e.g. Ethereum. Assuming C_0 is the consensus network that maintains the shared staking status, the token T_0 on C_0 would be staked for POS voting. When token T_i is produced on network C_i ($i \neq 0$) via incentives, it can be mapped to T_0 on C_0 by burning T_i on C_i through a secure cross-chain bridge channel. When a validator maintains the POS protocol in any network C_i, it always uses the amount of staked T_0 on C_0 as its voting power in C_i. Therefore, all the network

C_i share the same level of POS security. By using Ethereum as network C_0, it brings the benefit of incarnating the token T_0 as a standard ERC20 token and also enables the easy integration with existing popular restaking framework like EigenLayer [7]. Note that, the sources of the shared staking status may also come from the high value tokens (e.g., BTC and ETH) in restaking frameworks like EigenLayer and Babylon [8], so that to make 0G consensus reach the Bitcoin or Ethereum level of security. As illustrated in Figure 2, this shared staking design introduces the unlimited scalability to 0G system.

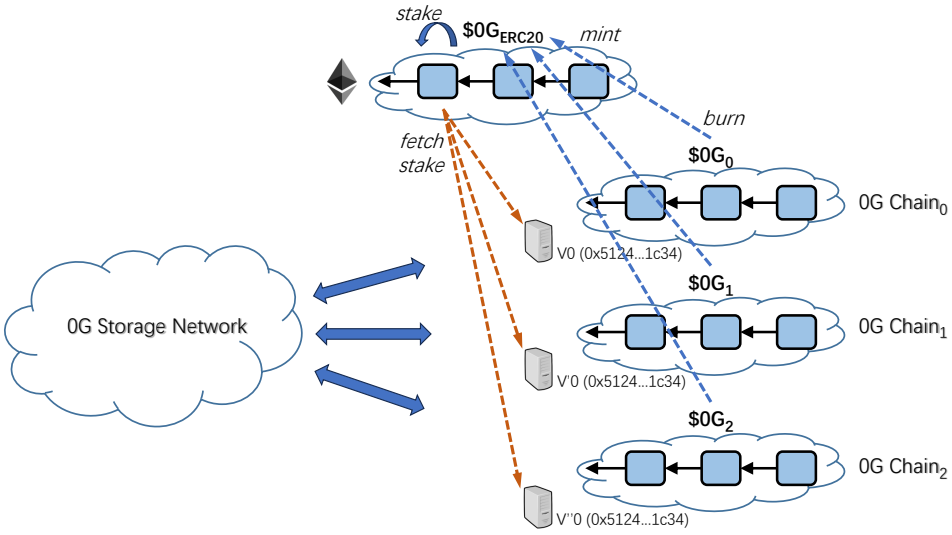

Figure 2: The infinite scalability of the 0G system.

3 Design of the 0G Storage

0G Storage employs layered design targeting to support different types of decentralized applications. Figure 3 shows the overview of the full stack layers of 0G Storage.

The lowest is a log layer that is a decentralized system consisting of multiple storage nodes that form a storage network with a built-in incentive mechanism that rewards data storage. The ordering of the uploaded data is guaranteed by a sequencing mechanism that provides log-based semantics and abstraction. This layer is used to store unstructured raw data for

permanent persistency.

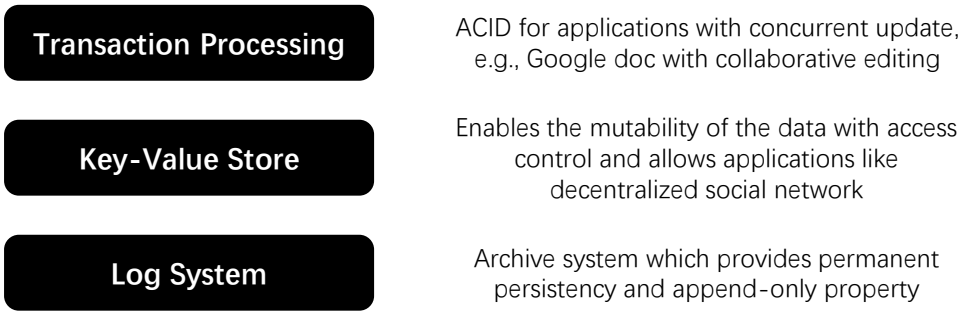

Figure 3: Full stack solution of 0G Storage.

On top of the log layer, 0G Storage provides a Key-Value store runtime to manage structured data with mutability. Multiple key-value store nodes share the underlying log system, putting the structured key-value data into the log entry and appending it to the log system. They play the log entries in the shared log to construct the consistent state snapshot of the key-value store. The throughput and latency of the key-value store are bounded by the log system, so that the efficiency of the log layer is critical to the performance of the entire system. The key-value store can associate access control information with the keys to manage the update permission for the data. This enables the applications like social network, e.g., decentralized Twitter, which requires the maintenance for the ownership of the messages created by the users.

0G Storage further provides transactional semantics on the key-value store runtime to enable concurrent updates for the keys from multiple users who have the write access permission. The total order of the log entries guaranteed by the underlying log system provides the foundation for the concurrency control of the transactional executions on top of the key-value store. With this capability, 0G Storage can support decentralized applications like collaborative editing and even database workloads.

4 Log System Protocol

The log layer of 0G Storage provides decentralized storage service via a permissionless network of storage nodes. These storage nodes collaboratively

serve archived data, where each node optionally specifies which portion of data it keeps in local storage.

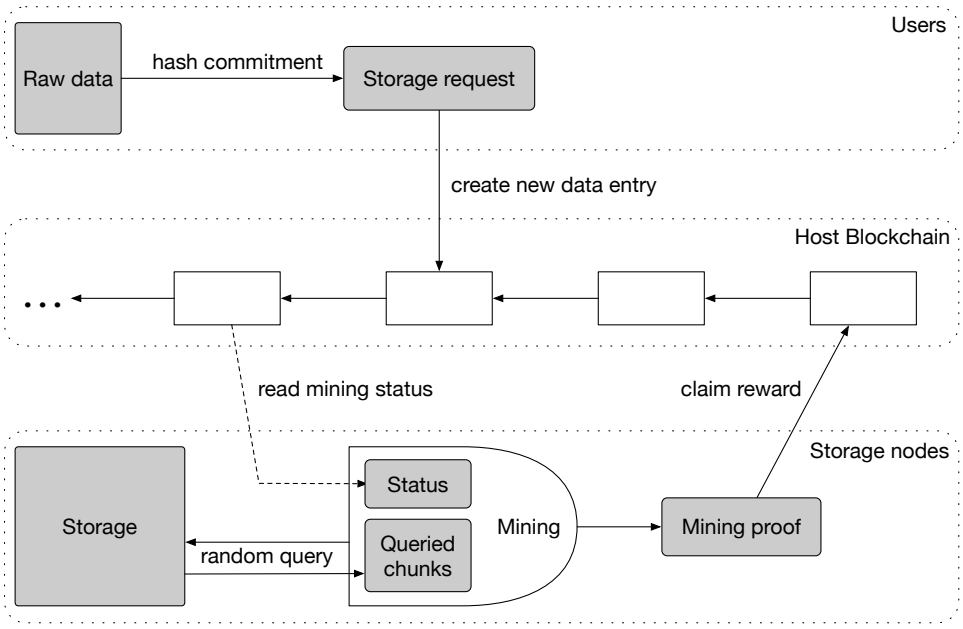

Figure 4: Overview of 0G Storage protocol. Data creation and reward distribution are fully decoupled: users directly submit storage requests to the 0G Storage contract deployed on host blockchain; storage nodes claim rewards by proving to the contract that they have the ability to answer random queries to archived data. 0G tokens are handled in another isolated contract, which enables non-blocking token transfer.

4.1 Protocol Overview

The storage state of 0G Storage network is maintained in a smart contract deployed on 0G blockchain. This contract is called the 0G Storage *contract* and the underlying blockchain is called the *host blockchain* or *host chain* for short.

The design of 0G Storage network fully decouples data creation, reward distribution, and token circulation.

The 0G Storage contract is responsible for data storage request processing, data entry creation, and reward distribution. Data storage requests are submitted by users who wish to store data in the 0G Storage network, where each request includes necessary metadata such as data size and com-

mitments, and it comes along with the payment for storage service. Data entries are created for accepted data requests, keeping record of stored data while reward distribution is handled independently through a mining process. storage nodes submit mining proofs to the 0G Storage contract to claim rewards for maintaining the 0G Storage network.

4.2 Storage Granularity

The log layer of 0G Storage is updated (append-only) at the granularity of *log entries*, where every entry is created by a storage-request transaction sent to the 0G Storage contract. The log layer is akin to a filesystem, with every log entry corresponding to a file.

The log system operates at the level of fixed-size *sectors*, with each sector storing 256 B of data. To avoid one sector being shared by distinct log entries, every log entry must be padded to multiple sectors.

The mining process of 0G Storage requires proving data accessibility to random challenge queries. To maximize the competitive edge of SSD storage, the challenge queries are set to the level of 256 KB *chunks*, *i.e.* 1024 sectors. As every challenge query requires the miner to prove accessibility to a whole chunk of data, storage nodes would maintain data at the granularity of chunks.

4.3 Data Flow

In 0G Storage, committed data are organized sequentially. Such a sequence of data is called a *data flow*, which can be interpreted as a list of data entries or equivalently a sequence of fixed-size data sectors. Thus, every piece of data in 0G can be indexed conveniently with a universal offset. This offset will be used to sample challenges in the mining process of PoRA.

The default data flow is called the "main flow" of 0G, and it incorporates all new log entries (unless otherwise specified) in an append-only manner.

There are also specialized flows that only accept some category of log entries, *e.g.* data related to a specific application. The most significant advantage of specialized flows is a consecutive addressing space, which may be crucial in some use cases. Furthermore, a specialized flow can apply customized storage price, which is typically significantly higher than the

floor price of the default flow, and hence achieves better data availability and reliability.

5 Incentive Mechanism

This section describes the incentive mechanism design of the 0G Storage, which consists of two types of actors: users and miners (a.k.a. "storage nodes"). Users pay 0G tokens to create data entries in the log and add data to the network, while miners provide data services and receive 0G tokens as a reward from the network. The payment from users to miners is mediated by the 0G consensus network, as the service is sustained by the whole network rather than any specific miner.

0G Storage implements storage service in a "pay once for a fixed period of time" manner. Users pay a storage endowment for each created data entry for a certain period of time, e.g., 3 years, and thereafter the endowment is used to incentivize miners who maintain that data entry by linear distribution. Users can recharge to extend the storage period.

The storage endowment is maintained per data entry, and a miner is only eligible for storage rewards from data entries that he has access to. The total storage reward paid for a data entry is independent of the popularity of that data entry. For instance, a popular data entry stored by many miners will be frequently mined, but the reward is amortized amongst those miners. On the other hand, a less popular data entry that is rarely mined would have its storage reward accumulate and hence provide a higher payoff to miners who store this rare data entry.

5.1 Storage Request Pricing

The cost of each 0G Storage request is composed of two parts: a) fee and b) storage endowment. The fee part is paid to host chain miners/validators for invoking the 0G contract to process storage requests and add new data entries into the log, which is priced like any other smart contract invocation transaction. In what follows we focus on the storage endowment part, which supports the perpetual reward to 0G Storage miners who serve the corresponding data.

Given a data storage request SR with specific amount of endowment

$\mathsf{SR}_{endowment}$ and size of committed data SR_{data_size} (measured in number of 256 B sectors), the unit price of SR is calculated as follows:

$$\mathsf{SR}_{unit\_price} = \mathsf{SR}_{endowment} / \mathsf{SR}_{data\_size} \tag{1}$$

This unit price SR_{unit_price} must exceed a globally specified lower bound to be added to the log, otherwise the request will be pending until when the lower bound decreases below SR_{unit_price} (in the meantime miners will most likely not store this unpaid data). Users are free to set a higher unit price SR_{unit_price}, which would motivate more storage nodes mining on that data entry and hence lead to better data availability.

5.2 Proof of Random Access

The 0G network adopts a Proof of Random Access (PoRA) mechanism to incentivize miners to store data. By requiring miners to answer randomly produced queries to archived data chunks, the PoRA mechanism establishes the relation between mining proof generation power and data storage. Miners answer the queries repeatly and computes an output digest for each loaded chunk util find a digest satisfies the mining difficulty (i.e., has enough leading zeros). PoRA will stress the miners' disk I/O and reduce their capability in responding user queries. So 0G Storage adopts intermittent mining, in which a mining epoch starts with a block generation at a specific block height on the host chain and stops when a valid PoRA is submitted to the 0G Storage contract.

In a strawman design, a PoRA iteration consists of a computing stage and a loading stage. In the computing stage, a miner computes a random recall position (the universal offset in the flow) based on an arbitrary picked random nonce and a mining status read from the host chain. In the loading stage, a miner loads the archived data chunks at the given recall position, and computes output digest by hashing the tuple of mining status and the data chunks. If the output digest satisfies the target difficulty, the miner can construct a legitimate PoRA consists of the chosen random nonce, the loaded data chunk and the proof for the correctness of data chunk to the mining contract.

In the following, we introduce some major considerations in improving the fairness in PoRA mining and formalize the PoRA mining mechanism.

5.2.1 Fairness for Small Miners

When the storage size of the 0G Storage network significantly exceeds the storage capacity of a single machine, a single machine will take almost all the time in finding an available recall position. This makes the PoRA becomes proof of work. To make the PoRA mining process be friendly to small miners with a single machine, the mining range is limited to a threshold 8 TB. When the size of archived data chunks exceeds 8 TB, a miner must specify a mining range over the data flow sequence in size of 8 TB. For large miners have enough machines to store all the data, it can mine on different data ranges concurrently.

5.2.2 Disincentivize Storage Outsourcing

To promote the whole network storing enough replicas of data chunks, 0G Storage limits the reward share of a single miner with a single storage replica. PoRA mechanism seals the data for each miner in different ways, challenges the accessing of sealed data, and prevents mining with data chunks from others. If a miner outsources the data storage and queries the archived data chunks in PoRA mining, it must seal the answering data to compute the output digest, which is a large cost. A miner can seal and unseal tens of MB data chunks by a single thread, so it can still serving the user queries efficiently. As reading data from SSDs can reach up to 7 GB/s, the data sealing is a heavy task for PoRA mining. As long as the cost imposed by data sealing exceeds the cost of purchasing disks and synchronizing data, the miners intend to store a replica of data chunks instead of sealing data on PoRA mining.

More specifically, every 4KB data chunk will be sealed by the miner ID and other context data. Suppose seal_seed is a 32-byte digest of the miner ID and the context data, figure 5 defines the seal process.

5.2.3 Disincentivize Distributed Mining

As a mining mechanism for incentivizing data storage, a single storage replica should produce a similar hashrate for fairness. In PoRA mining process, the hashrate is bottlenecked by the storage Input/Output (I/O). So we expected all the miners to have similar storage I/O per unit of storage. However, a mining farm can significantly increase its I/O compared

Input : seal_seed, unsealed_data
Output: sealed_data

Regard unsealed_data in length of 4 KB as an array of 32-byte elements $\vec{d}$
$h \leftarrow$ seal_seed
for i **from** 0 **to** 127 **do**
    $\vec{d}[i] \leftarrow \vec{d}[i]$ `XOR` h
    $h \leftarrow \text{Keccak256}(\vec{d}[i])$
Regard $\vec{d}$ as a 4-KB data chunk sealed_data

Figure 5: Seal 4 KB data chunks

to normal miners by storing data chunks in a distributed system with in-memory filesystem. As the DDR memory has a significantly larger bandwidth compared to SSD, the mining farm can produce much more hashrate while contributing negligible to the data storage.

0G Storage prevents this behavior by imposing a large amount of data transfer from the computing stage to the loading stage to encourage miners to complete two stages on the same machine. So if the computing stage and loading stage are separated by different machines, the hashrate is bounded by the network bandwidth, which is usually smaller than the storage I/O bandwidth. In the computing stage, a random scratchpad will be generated coupled with the recall position. It has the same length as the data chunk to be loaded. Before computing the hash of the loaded data chunk, the data chunk should be mixed with the scratchpad by an XOR operation.

However, there is a dilemma between the read bandwidth and the transaction fee for submitting PoRA to the host chain. Since loading data chunks from local disks should be more efficient than downloading them via the network, the loading stage should leverage the fast SSD read speeds in sequential access to maximize the data chunk loading bandwidth. According to our benchmarks, randomly loading 256-KB data chunks reaches 80% read speed of sequential access. However, when submitting the data chunk to the host chain, a 256-KB transaction is too large and will consume a large amount of transaction fee. Reducing the size of the data chunk could mitigate this issue, but would impair the read speed.

PoRA mechanism adopts "batched data loading" to resolve this dilemma. In each iteration, a miner loads a 256-KB data chunk from storage, divides it into 4-KB data chunks, and computes the hash for each

4-KB data chunk individually. If one of the 64 outputs matches the target quality, the miner finds a valid answer. So only the 4-KB data chunk contributing to this answer is required to be submitted to the host chain.

5.2.4 Formal Definition for PoRA Mechanism

Precisely, the mining process has the following steps:

1. Register the miner_id on the mining contract;
2. For each mining epoch, repeat the following steps:
   (a) Wait for the layer-1 blockchain to release a block at a given epoch height;
   (b) Get the block hash block_hash of this block and the relevant context (including merkle_root, data_length, context_digest) at this time;
   (c) Compute the number of minable entries $n = \lfloor \mathsf{data\_length}/256\text{KB} \rfloor$.
   (d) For each iteration, repeat the following steps:
      i. Pick a random 32-byte nonce;
      ii. Decide the mining range parameters start_position and mine_length; mine_length should be equal to $\min(8\text{TB}, n \cdot 256\text{KB})$;
      iii. Compute the recall position τ and the scratchpad $\vec{s}$ by the algorithm in figure 6;
      iv. Load the 256-kilobyte sealed data chunk $\vec{d}$ started from the position of $h \cdot 256\text{KB}$;
      v. Compute $\vec{w} = \vec{d}$ XOR $\vec{s}$ and divide $\vec{w}$ into 64 4-kilobyte pieces;
      vi. For each piece $\vec{v}$, compute the Blake2b hash of the tuple (miner_id, nonce, context_digest, start_position, mine_length, $\vec{v}$);
      vii. If one of the Blake2b hash outputs is smaller than a target value, the miner found a legitimate PoRA solution.

Input : miner_id, nonce, context_digest, start_position, mine_length
Output: Recall position τ and the scratchpad $\vec{s}$

$h \leftarrow$ Blake2b(miner_id, nonce, context_digest, start_position, mine_length)
Initialize an empty array $\vec{s}$ with 4096 64-byte elements
for $i \in \{x \in \mathbb{Z} \mid 0 \leq x < 4095\}$ **do**
    $h \leftarrow$ Blake2b(h)
    $\vec{s}[i] \leftarrow h$
$r \leftarrow$ Keccak256(h)
$n \leftarrow$ mine_length/256KB
$\tau \leftarrow$ start_position $+ r \bmod n$

Figure 6: Computing the recall position and the scratchpad

5.3 How to Balance the Number of Replicas?

Every miner in 0G Storage is required to explicitly claim the range of data entries that he maintains, and then he may only use the claimed data in his mining process. That is, a miner cannot receive mining rewards from unclaimed data, and on the other hand, the mining output is decreased if he fails to answer a challenge within his claimed range. As a result, a miner has no incentive to lie about the range of data used for mining.

Since miners truthfully expose the list of data entries that they maintain, it is easy to observe and estimate the number of replicas for every specific piece of data. Furthermore, recalling that the storage endowment is paid to miners who maintain corresponding pieces of data, miners are incentivized to store rare data and hence balance the number of replicas.

6 Key-Value Runtime

0G Storage provides a Key-Value runtime upon the log layer. Each key-value node can access the key-value store state through the runtime interface. The key-value runtime provides the standard interface like `Put()` and `Get()`, and accepts serialized key-value pair from any application-specific structure.

During the normal execution of the key-value store node, it maintains the latest key-value state locally. It updates the value of a key through `Put()` API which composes a log entry containing the updated key-value pair and appends it to the log. The runtime constantly monitors the new log entries in the log and fetches them back to the key-value node, updating the local key-

value state according to the log entry contents. In this sense, multiple key-value store nodes essentially synchronize with each other through the shared decentralized log. A user-defined function will be used to deserialize the raw content in the log entry to the application-specific key-value structure. Applications can use `Get()` API to access the latest value of any given key. To improve the efficiency of the updates for small key-value pairs, the `Put()` allows batched updates with multiple key-value pairs at once. Figure 7 illustrates the architecture of the decentralized key-value store. To manage the access control, the ownership information of each key can also be maintained in the log entries. All the honest key-value nodes follow the same update rule for the keys based on the ownership to achieve state consistency.

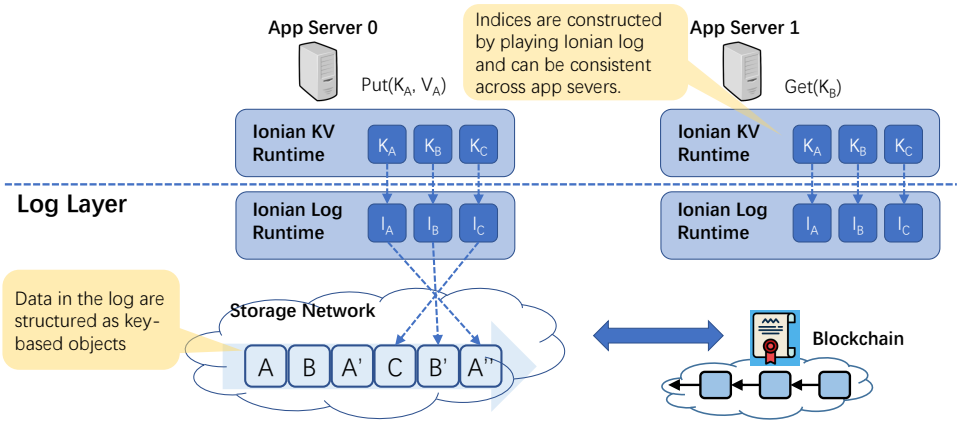

Figure 7: Decentralized Key-Value Store

When a new key-value node just joins the network, it connects to the log layer and plays the log entries from head to tail to construct the latest state of the key-value store. During the log entry playing, an application-specific key-value node can skip irrelevant log entries that do not contain the stream IDs that it cares.

7 Transactional Processing on Key-Value Store

0G Storage employs concurrency control in the key-value runtime to support transactional processing for concurrent operations on multiple keys. The total order of the log entries guaranteed by the underlying log system provides

the foundation for this concurrency control of the transactional executions on top of the key-value store. This mechanism is optimistic and illustrated in Figure 8.

When an application server linking with the 0G Storage key-value runtime starts a transaction using `BeginTx()` interface, it notifies the runtime that the transaction will work on the current state snapshot constructed by playing the log to the current tail. The further key-value operations before the invocation of `EndTx()` update the key-values locally in the server without exposing the updates to the log. When `EndTx()` is invoked, the runtime composes a *commit record* containing the log position the transaction starts from and the read-write set of the transaction. This commit record is then appended to the log.

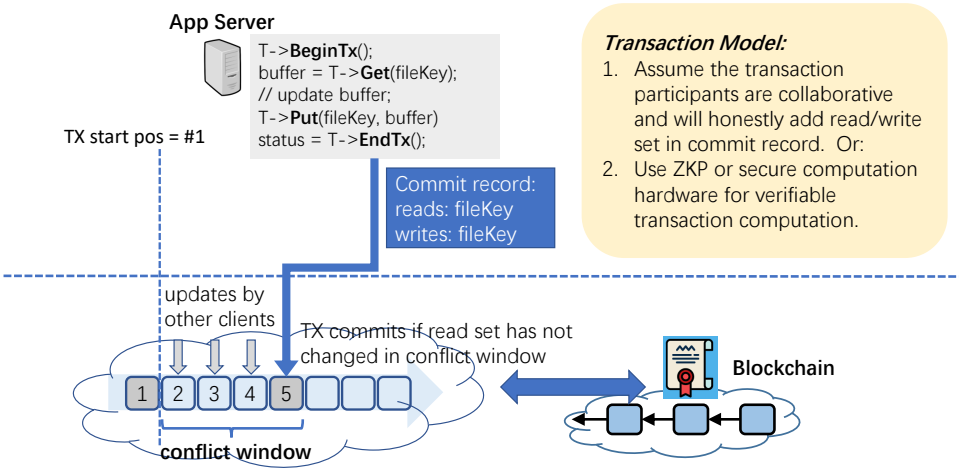

Figure 8: Transactional Processing on 0G Storage KV Store

When an application server with the key-value runtime encounters the commit record while playing the log, it identifies a *conflict window* consisting of all the log entries between the start log position of the transaction and the position of the commit record. The log entries in the conflict window therefore contain the key-value operations concurrent with the transaction submitting the commit record. The runtime further detects whether these concurrent operations contain the updates on the keys that belong to the read set of the transaction. If yes, the transaction is aborted. Otherwise, it is committed successfully.

This transaction model assumes that the transaction participants are collaborative and will honestly compose the commit record with the correct content. Although this assumption in a decentralized environment is too strong, it is still achievable for specific applications. For example, for an application like Google Docs, a user normally shares access with others who are trusted. In case this assumption cannot hold, the code of the transaction can be stored in 0G log and some mechanism of verifiable computation like zero-knowledge proof or hardware with trust execution environment (TEE) can be employed by the transaction executors to detect the validity of the commit record.

8 Service Marketplace over 0G Compute Network

As a decentralized AI operating system, 0G also provides a separate platform managing the decentralized computing resources as services. It organizes the computation resources in a compute network and allows services deployed and registered with verifiability so that users can choose trusted services in the decentralized environment.

0G compute network is designed as a free marketplace where users and service providers decide the prices of the services through a peer-to-peer fashion. The service provider is free to quote their services, and then users are free to choose the services that are priced appropriately for them to use. The system assumes that any service result can be verified by the user who submits the request. In the trust model, it treats the service providers as more trustworthy than users and less powerful to apply sybil attack. Therefore, a user may face the risk of losing the cost of a single request. In other words, when a user submits a request and sends fee to smart contract, she may spend the fee but fail to get the service result if the service provider is malicious. In this case, the user can choose other service providers in the network to retry the request.

The service provider should first register into the smart contract with information about the service types that it provides, the price of each type, and the specific verification mechanism. When a user wants to access the service, she pre-charges some amount of fee to the smart contract for this service provider. The user then can issue requests to the service provider and the service provider decides to respond with the serving results based

on whether the remaining fee is enough. Any request and response have the signatures of the user and the service provider, respectively. The user can verify each response and decide to stop sending further request if the verification is failed. The service provider can decide at any point to send the request traces with user's signature to the smart contract for settlement. Once settlement is done, the corresponding portion of the pre-charged fee can be sent to service provider's account.

The performance trade-off is how often the service provider submits the settlement request to the smart contract. For the high-latency services, the provider can submit settlement per user request. But for the low-latency services, the provider may need to batch more user requests for one settlement. Another consideration is how the user request trace should be organized as the smart contract input for settlement. Plainly submitting all the requests in the trace may involve too much on-chain gas consumption. 0G therefore employs zero-knowledge proof mechanism to optimize the on-chain settlement cost.

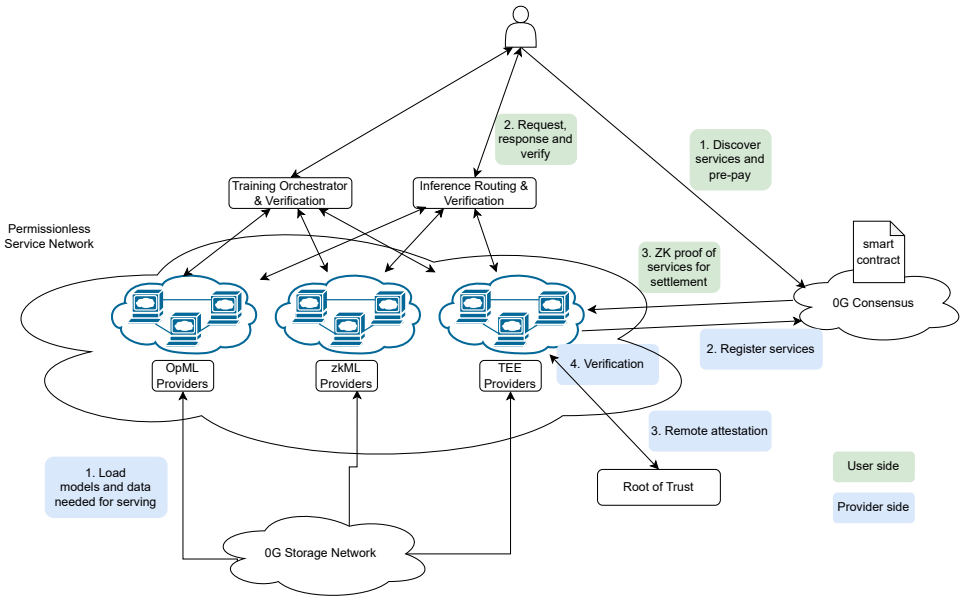

Figure 9: The Architecture of 0G Compute Network and Service Marketplace.

0G compute network provides a bunch of useful SDK and APIs for the service providers to easily register their services into the marketplace smart

contract, and allows users to conveniently discover appropriate services and interact with them smoothly. The services may include AI model inference and training workers, and also the data caching and fetching services.

Together with the 0G storage and DA networks, the 0G compute network and the service marketplace complete the infrastructure foundation of the dAIOS to enable easy migration of AI applications and platforms from Web2 to Web3, and pave the way towards AGI democratization.

References

[1] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities, 2019.

[2] Ethereum White Paper. `https://ethereum.org/en/whitepaper/`, 2020.

[3] ORA-OpML. `https://docs.ora.io/doc`.

[4] Optimism. `https://www.optimism.io/`.

[5] Celestia. `https://celestia.org/`.

[6] EigenDA. `https://www.blog.eigenlayer.xyz/tag/eigenda/`.

[7] EigenLayer. `https://www.eigenlayer.xyz/`.

[8] BabylonChain. `https://babylonchain.io/`.