Endless Technical White Paper

March 2025

Table of Contents

1 Overview **1**

2 Accounts **3**

2.1 Enhanced Account Model 3

2.1.1 Limitations of Traditional Multisignature 3

2.1.2 Limitations of On-Chain Multisignature 4

2.1.3 Endless Off-Chain Multisignature 4

2.2 Enhanced Account Address 5

2.2.1 Address Format Verification 6

2.2.2 Storage Structure and Special Addresses 6

2.3 Support for Keyless Login 7

2.3.1 Key Features of Keyless Accounts 7

2.3.2 Working Principle 7

3 Consensus Model **13**

3.1 Traffic Resource-Based Consensus Model 13

3.1.1 Core Technology Analysis 13

3.1.2 Comprehensive System Architecture 17

3.1.3 Key Advantages and Features 20

3.2 Storage Resource-Based Consensus Model 22

3.2.1 Technical Solution 22

3.2.2 Endless KZG Polynomial Proof 24
3.2.3 File Storage Commitment 26
3.2.4 On-Chain Random Challenge Generation 27
3.2.5 Storage Provider's KZG Proof Generation 27
3.2.6 On-Chain Verification 27
3.3 Fee and Penalty Mechanism 27
3.3.1 Treasury and Fee Management 27
3.3.2 Challenge and Penalty Mechanism 28
3.3.3 Gas Fee Subsidy Mechanism 29
3.3.4 Challenge Limitations 29
3.3.5 User Payment Default Handling Mechanism 29
3.4 Consensus Model of the Endless Blockchain 30

4 Asset Standards 33
4.1 Unified Conversion of FT Asset Standards 33
4.2 Unified Conversion of DA Asset Standards 33

5 Transactions 35
5.1 Sponsored Transactions 35
5.2 Secure Transactions 36
5.3 Transaction Processing 38
5.3.1 Transaction Processing Optimization Mechanisms 39
5.3.2 Parallel Transaction Execution 41
5.3.3 Experimental Results and Performance Optimization 42

6 Indexer 45
6.1 Performance Comparison 46
6.1.1 Synchronization Speed 46

6.1.2 Write Speed 48
6.1.3 Query Speed 49
6.1.4 Disk Usage 49
6.2 Advantages and Limitations 50

7 Token Standards **51**
7.1 Introduction to Token Locking Standards 51
7.2 Core Features of the Contract 51
7.3 Design Objectives and Functions 52
7.4 Design Goals 52

8 Move Smart Contracts **54**
8.1 Move Contract Language 54
8.2 Move Contract Modules 56
8.2.1 Module Overview 56
8.2.2 Resources 57
8.3 Execution and Security of Move Smart Contracts 58

9 On-chain Trusted Randomness **60**

1 Overview

Endless Genesis Cloud is a distributed network infrastructure composed of decentralized blockchain, decentralized storage, and decentralized traffic services. By integrating various technological components through a consensus model, the system aims to lower the technical barriers for migrating from Web2 to Web3, simplify user onboarding, and provide a comprehensive Web3 aggregation solution.

Endless Genesis Cloud leverages advanced cryptographic and distributed computing technologies, including ZK proofs, ED25519 signatures, BLS12-381 signatures, KZG proofs, the Move programming language, the Block-STM parallel execution engine, and the Diem BFT consensus mechanism, to offer core functionalities such as decentralized storage services, application hosting, and traffic distribution.

As the underlying consensus layer of Endless Genesis Cloud, the Endless Blockchain is an optimized and improved version of the Aptos chain, with enhancements in account systems, security mechanisms, user experience, storage architecture, performance optimization, and asset standards. Additionally, the Endless Blockchain introduces key functionalities such as a storage consensus protocol, a traffic consensus protocol, an asset locking standard, and gas fee sponsorship, further enhancing the overall ecosystem's scalability and usability.

The following chapters will introduce the core technological components of

Endless Genesis Cloud.

2 Accounts

2.1 Enhanced Account Model

2.1.1 Limitations of Traditional Multisignature

On the `Aptos` or `Sui` blockchain, every account contains a 32-byte authentication key (`auth_key`) upon creation. This `auth_key` is derived from the public key and the authentication scheme. For example, for a ”single-signature” account using the `Ed25519` signature scheme, its authentication key is computed as follows:

$$\texttt{auth\_key} = \texttt{sha3\_256}(\texttt{pub\_key} \mid 0\text{x}00)$$

where `0x00` indicates that the account adopts the `Ed25519` authentication scheme.

Aptos/Sui also supports ”multisignature” accounts. For example, on the Aptos platform, a ”multisignature” account using the 1-of-2 signature scheme calculates its authentication key as follows:

$$\texttt{auth\_key} = \texttt{sha3\_256}(0\text{x}2 \mid 0\text{x}01 \mid \texttt{pub\_key}_0 \mid 0\text{x}01 \mid \texttt{pub\_key}_1 \mid 0\text{x}00)$$

where:

- `0x2` indicates that the account is associated with 2 keys.
- `0x01` represents the authentication scheme of the keys, i.e., `Ed25519`.
- The final `0x00` designates the account as a ”multisignature account.”

Multisignature accounts are typically used in conjunction with on-chain multisignature `Move` modules (such as Aptos' `0x1::multisig_account`) for on-chain multi-party authentication.

2.1.2 Limitations of On-Chain Multisignature

Compared to off-chain multisignature solutions, on-chain multisignature mechanisms exhibit the following limitations:

- Higher gas fees are required.
- Multiple transaction interactions increase execution complexity.
- The account model is fixed, making it impossible to flexibly convert between ”single-signature accounts” and ”multisignature accounts.”

2.1.3 Endless Off-Chain Multisignature

On the Endless platform, an account's authentication key (`auth_key`) is composed of a set of addresses, which may include one or multiple account addresses:

- When `auth_key` contains only a single account address, the account is a ”single-signature account.”
- When `auth_key` contains multiple account addresses, the account is a ”multisignature account.”
- Endless accounts natively support K-of-N multisignature configurations.

Endless Account Structure

The basic structure of an Endless account is as follows:

```
/// Simplified Endless account structure
pub struct AccountData {
    pub sequence_number: u64,
    pub authentication_key: Vec<AccountAddress>,
    pub num_signatures_required: u64,
}
```

Endless platform natively supports an ”off-chain multisignature” mechanism. Users can manage multisignature configurations via CLI commands or DApps (such as the Endless Multisig DApp), allowing them to add or remove account addresses from the `auth_key` set.

Advantages of Endless Off-Chain Multisignature

- **Lower gas consumption**: More efficient compared to traditional on-chain multisignature schemes.
- **Smoother user experience**: With well-designed DApps such as Endless Multisig DApp, account management becomes more convenient.

For more examples on conducting transactions and managing authentication keys using multisignature accounts, please refer to: Your First Multisig.

2.2 Enhanced Account Address

The Endless Blockchain utilizes Base58 encoding for account addresses, with the following characteristics:

- Most account addresses range from 43 to 44 characters in length.
- Account addresses can be quickly distinguished by checking their first and last characters.
- Supports vanity addresses (customized addresses containing specific characters).

2.2.1 Address Format Verification

An account address consists of multiple characters. To ensure that an address aligns with user expectations, verifying only the first and last characters is insufficient. It is recommended to validate the entire address.

For example, the following is an Endless account address:

`5SHvmLEaSr76dsKy4XLR5vMht14PRuLzJFx6svJzqorP`

Currently, the `Base58` encoded address format is fully supported by the `Endless CLI` and the Endless Blockchain Explorer The following link showcases an example of an account transaction within the blockchain explorer.

2.2.2 Storage Structure and Special Addresses

At the underlying implementation level, an Endless account address is actually stored as a 32-byte array. In certain contexts (such as the CLI or blockchain explorer), the account address may be displayed in hexadecimal format. For example, two core system account addresses in the Endless Blockchain are:

- `0x000...001` (abbreviated as `0x1`): The system account, responsible for executing system contracts.
- `0x000...004` (abbreviated as `0x4`): The system token account, responsible for managing `Tokens` and `NFTs`.

2.3 Support for Keyless Login

Keyless accounts represent a new Web3 account model that allows users to access Web3 applications across devices without the need for a traditional crypto wallet, delivering a seamless decentralized experience.

Users can log in to Web3 applications directly using familiar social accounts (such as Google or Apple accounts), simplifying the complexity of traditional blockchain accounts.

2.3.1 Key Features of Keyless Accounts

- **Convenient Login**: Users can access Web3 applications directly with their existing social accounts.
- **Easy Recovery**: No need to worry about losing private keys or mnemonic phrases.
- **Multi-Device Access**: Users can effortlessly manage their accounts across different devices.

Users can visit: https://scan.endless.link/ to explore the Keyless account functionality.

2.3.2 Working Principle

Keyless Accounts

Before the introduction of Keyless accounts, the only way to secure an Endless account was by protecting the associated private key. However, in practice, private keys may be lost (e.g., if a user forgets to back up their mnemonic phrase)

or stolen (e.g., if a user is deceived into revealing their private key), making private key management a burden for users.

Endless introduces Keyless accounts, which are generated from existing OIDC accounts (such as Google/Apple Web2 accounts), meaning ”Blockchain Account = OIDC Account.” Note:

- A Keyless account is determined jointly by the OIDC provider and the service provider.
- Different OIDC providers or service providers will generate different Keyless accounts.

Public Key

A Keyless account ’s public key consists of:

- `iss_val`: OIDC provider identifier (derived from the JWT `iss` field, e.g., `https://accounts.google.com`).
- `addr_idc`: Identity Commitment (IDC), which includes:
  - `uid_val`: Unique identifier of the user within the OIDC provider.
  - `uid_key`: JWT identity field (typically `sub` or `email`).
  - `aud_val`: Service provider ID (derived from the JWT `aud` field).

The ‘IDC‘ is computed via a SNARK-friendly hash function:

$$\text{addr}_{\text{idc}} = H'(\text{uid\_\_key}, \text{uid\_\_val}, \text{aud\_\_val}; r)$$

$$\text{where} \quad r \xleftarrow{\$} \{0,1\}^{256}$$

Pepper

The blinding factor r is referred to as Pepper, which has the following properties:

- Losing Pepper results in the loss of the Keyless account.
- Exposure of Pepper only reveals the Web2-Web3 identity mapping.

A ”Pepper Service” assists in generating and storing Pepper.

Authentication Key

Definition:

$$\text{auth}_{\text{key}} = \text{H}(\text{iss}_{\text{val}}, \text{addr}_{\text{idc}})$$

where H is a cryptographic hash function.

Private Key

Users are not required to store a private key; the private key is embodied by the ability to log in using an OIDC account via OAuth.

Signature

The signature for Keyless accounts is based on ”Zero-Knowledge Proofs”. Transaction signature definition:

$$\sigma_{\text{txn}} = (\text{header}, \text{epk}, \sigma_{\text{eph}}, \text{exp\_\_date}, \text{exp\_\_horizon}, \text{extra\_\_field}, \text{override\_\_aud\_\_val}, \sigma_{\text{tw}}, \pi)$$

Where:

1. header is the `Header` of `JWT`, containing the signature scheme of the OIDC provider and the key ID of JWK.

2. epk is an ephemeral public key generated by the application (DApp or wallet), paired with the ephemeral private key esk, typically an `ED25519` key pair.

3. σ_{eph} is the signature on the transaction txn using the ephemeral private key esk.

4. exp__date is the expiration time for the ephemeral key pair esk and epk.

5. exp__horizon is a public parameter that limits the maximum validity period of ephemeral key pairs, preventing DApps from incorrectly setting excessively long expiration periods.

6. extra__field is optional and used to expose specific fields from the `JWT`. For example, if a user wishes to share their email, extra__field could be set to `ëmail:¨älice@gmail.com¨`.

7. override__aud__val is an optional parameter for ”account recovery”. If set, it overrides the `aud` in the `JWT` and uses override__aud__val to generate the `IDC`.

8. σ_{tw} is the signature on π by the zero-knowledge proof service, which is optional and determined by on-chain configuration.

9. π is the zero-knowledge proof of knowledge (ZKPoK).

Verification process:

1. Verify that $\text{auth}_{\text{key}} = H(\text{iss}_{\text{val}}, \text{addr}_{\text{idc}})$.

2. Ensure that exp__horizon $\in (0, \text{max\_\_exp\_\_horizon})$.

3. Verify the validity period of the EPK: current__block__time() < exp__date.

4. Verify the signature σ_{eph}.

5. Retrieve the JWT public key jwk.

6. Generate the public input hash:

$$pih = H_{zk}(epk, addr\_\_idc, exp\_\_date, exp\_\_horizon, iss\_\_val, extra\_\_field, header, jwk, override\_\_aud\_\_val)$$

7. If a training round key is configured, verify σ_{tw}.

8. Verify the ZKPoK proof π, ensuring the existence of a secret input w that satisfies the relation:

$$R\left(w = \begin{bmatrix} w_{pub} = \begin{pmatrix} pih; \\ epk, addr\_\_idc, exp\_\_date, exp\_\_horizon, iss\_\_val, \\ extra\_\_field, header, jwk, override\_\_aud\_\_val \end{pmatrix}, \\ w_{priv} = \begin{pmatrix} aud\_\_val, uid\_\_key, uid\_\_val, \\ r, \sigma_{old}, jwt, \rho \end{pmatrix} \end{bmatrix}\right)$$

The reason for introducing zero-knowledge proofs in the signing process is that transaction verification for Keyless accounts depends on JWT information. However, directly recording JWT data on-chain could potentially compromise user privacy. Therefore, Endless employs zero-knowledge proof techniques to ensure the legality of the transaction signature while avoiding the need to store JWT information on-chain. Specifically, the zero-knowledge proof of knowledge (ZKPoK) mechanism enables verifiers to validate the transaction signature σ_{txn} without revealing the sensitive JWT content.

More Robust Keyless Accounts

By integrating Endless 'multisignature feature, Keyless accounts can remain functional even when the OAuth server is unavailable. See the Endless Off-Chain

Multisignature documentation for details.

3 Consensus Model

3.1 Traffic Resource-Based Consensus Model

The Endless Blockchain project was founded to address the challenges faced by the Web3 ecosystem in integrating cloud services. Our goal is to break through the limitations of traditional cloud services and build an open, efficient, and secure blockchain ecosystem. This ecosystem enables Web3 developers to seamlessly access cloud services, allowing them to focus on the innovative development of decentralized applications (DApps).

Endless achieves efficiency, security, and fairness in cloud service utilization through precise client traffic statistics combined with automated smart contract settlements on the blockchain. This innovative model not only simplifies the development process for Web3 developers but also establishes a fair and transparent cooperation mechanism for cloud service providers and end users, promoting the adoption and optimization of decentralized cloud computing services.

3.1.1 Core Technology Analysis

Exploring the BLS12-381 Algorithm

The BLS12-381 algorithm is one of the core technologies behind the Endless Blockchain's traffic statistics functionality. It is based on the Barreto-Naehrig

(BN) curve and possesses outstanding signature and aggregation capabilities, making it a crucial component in cryptographic applications. In the mathematical framework of BLS12-381, the definition of the elliptic curve is foundational. Over a finite field p, the equation of the BLS12-381 curve is elegantly simple:

$$y^2 = x^3 + b$$

where p is a carefully selected large prime number that ensures the cryptographic security of the system, and b is a key parameter that determines the shape and properties of the curve.

On this elliptic curve, point addition and multiplication operations follow strict and intricate rules. Given two distinct points on the curve,

$$P\,(x_1, y_1)$$

and

$$Q(x_2, y_2)\,,$$

the addition operation involves computing a critical slope parameter:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}\,,$$

which then leads to the sum

$$R = P + Q$$

with coordinates:

$$x_3 = \lambda^2 - x_1 - x_2\,,$$

$$y_3 = \lambda(x_1 - x_3) - y_1 .$$

If the two points are identical, a slightly different formula is used:

$$\lambda = \frac{3x_1^2}{2y_1}, \quad x_3 = \lambda^2 - 2x_1 ,$$

$$y_3 = \lambda(x_1 - x_3) - y_1 .$$

Point multiplication is achieved through repeated point addition. For a point multiplied by an integer, the computation is defined as:

$$nP = P + P + \cdots + P \quad (\text{n additions of P}).$$

This addition-based multiplication method ensures mathematical rigor and serves as a foundation for subsequent signature and aggregation operations.

BLS12-381 Signature Mechanism Analysis

The BLS12-381 signing process is secure and well-structured, ensuring the authenticity and integrity of traffic data. First, a user randomly selects a private key from the integer ring modulo p, Z_p. This private key acts as the core secret of the signing process, similar to a user's digital identity password. Based on the private key, a public key is generated as follows:

$$pk = sk \cdot G,$$

where G is a fixed base point on the elliptic curve. The public key functions like a digital identity card that can be shared publicly to verify the authenticity of signatures.

When a user signs a message m, they first compute the hash of the message using a cryptographic hash function H:

$$h = H(m).$$

A hash function acts like a unique fingerprint generator, always producing a fixed-length hash value regardless of the message's content. The user then signs the hash with their private key to generate a signature:

$$\sigma = sk \cdot h.$$

To verify the signature, the recipient computes the message hash $h = H(m)$ and checks the validity of the equation:

$$pk \cdot \sigma = h \cdot G.$$

If the equation holds—analogous to a key precisely fitting its corresponding lock—it confirms the signature's validity and ensures that the message has not been tampered with in transit. If the equation does not hold, the signature is deemed invalid, indicating potential risks.

Unveiling the Efficient Aggregation of BLS12-381

One of the standout features of BLS12-381 in Endless 'traffic statistics function is its efficient aggregation capability, which significantly enhances the system's performance and data processing efficiency. Suppose there are n different messages $m_1, m_2, \cdots, m_n$, each with a corresponding signature $\sigma_1, \sigma_2, \cdots, \sigma_n$ and public key $pk_1, pk_2, \cdots, pk_n$. To generate an aggregate signature, we simply sum up all individual signatures:

$$\sigma_{agg} = \sum_{i=1}^{n} \sigma_i.$$

During verification, the recipient computes the hash of each message:

$$h_i = H(m_i), \quad i = 1, 2, \cdots, n,$$

and checks whether the equation:

$$\sum_{i=1}^{n} pk_i \cdot \sigma_{agg} = \sum_{i=1}^{n} h_i \cdot G$$

holds.

If it does, all signatures are verified as valid, ensuring that messages remain intact during transmission and aggregation. This efficient aggregation method allows multiple signatures to be condensed into a single compact signature, significantly reducing the computational cost of signature verification and minimizing network bandwidth consumption. As a result, Endless 'traffic statistics function can operate stably and efficiently even in large-scale application scenarios.

3.1.2 Comprehensive System Architecture

Client SDK: The Frontline Sentinel for Data Collection

The Client SDK is a key front-end component of the Endless traffic statistics functionality, acting as a vigilant scout that continuously monitors and collects network traffic data from client applications.

One of its core capabilities is precise traffic measurement. Whether it is data transmission during uploads or resource retrieval during downloads, the Client SDK captures and accurately accounts for all network traffic in real time, providing reliable data for subsequent billing and analysis.

Regarding private key management, the Client SDK integrates a custom-developed Rust library alongside advanced obfuscation compilation techniques. Rust is renowned for its exceptional memory safety and concurrency management, which offer a strong foundation for securely storing and utilizing private keys. Meanwhile, obfuscation compilation acts as an invisible shield, making it significantly harder for attackers to decipher and extract private keys, thereby enhancing data security.

Once traffic data is collected, the Client SDK signs the data using the BLS12-381 algorithm. This process applies an unforgeable digital signature to the data, ensuring its authenticity and integrity throughout transmission. Finally, at predetermined time intervals, the Client SDK submits the signed traffic data to the signature network, preparing it for further processing and verification.

Signature Network: The Central Hub for Data Validation and Aggregation

The Signature Network is a decentralized infrastructure composed of multiple distributed nodes, responsible for data validation, storage, and aggregation within Endless' traffic measurement system.

- When the Signature Network receives traffic data submitted by the Client SDK, it first undergoes rigorous verification using the BLS12-381 signature scheme. Only validated data is considered trustworthy, subsequently stored within the network and synchronized across other nodes.

- This decentralized storage and synchronization mechanism not only guarantee data security and reliability but also facilitate data traceability and auditing.

- Another key function of the Signature Network is traffic data aggregation.

It adheres to predefined aggregation rules to merge multiple BLS signatures into a condensed aggregate signature, thereby significantly reducing the amount of data submitted to the Endless Blockchain.

This optimization technique reduces the blockchain's computational load, improves operational efficiency, and conserves network bandwidth resources. Additionally, the Signature Network provides cloud service providers with user-friendly API interfaces, enabling them to retrieve historical traffic data for settlement and business analytics.

Blockchain Smart Contracts: The Impartial Arbiter for Automated Settlement

Smart contracts serve as a fundamental component enabling automated settlements within the Endless Blockchain 's traffic statistics system. Acting as an impartial and self-executing judge, smart contracts ensure the accuracy, fairness, and transparency of traffic-based fee settlements.

- Cloud service providers submit aggregated traffic data retrieved from the Signature Network to the blockchain's smart contract, which first performs strict BLS signature verification.

- Only validated traffic data is deemed authentic and progresses to the settlement process.

- Settlement is executed automatically based on the logic encoded within the smart contract, eliminating the need for manual intervention.

This approach enhances the efficiency and accuracy of settlements while mitigating errors or fraud that could result from human involvement. It ensures data trustworthiness and pricing transparency between cloud service providers and Web3 developers.

3.1.3 Key Advantages and Features

Low Barrier to Entry: A Convenient Gateway for Web3 Developers

Endless provides a seamless SDK integration, allowing developers to easily access the Endless ecosystem and utilize extensive decentralized cloud service resources simply by incorporating the appropriate software development kit (SDK).

By reducing complexity, developers no longer need to spend excessive time learning cloud service configurations or deployment processes, nor do they need to engage in intricate agreements with cloud service providers. This enables them to focus entirely on DApp innovation and business expansion, accelerating the development and launch of Web3 applications.

Ultimate Security Assurance: A Secure Fortress for Private Keys

Endless leverages a custom-developed Rust library combined with obfuscation compilation technology to provide Web3 developers with a highly secure private key management solution:

- Rust ’s memory safety and concurrency control effectively mitigate the risk of private key leaks.

- Obfuscation compilation technology transforms and encrypts the SDK, making it extremely difficult for attackers to analyze and reverse-engineer the logic behind key storage and usage.

- Developers can generate their dedicated SDK using the Rust Lib provided by Endless, achieving dual protection for both key security and SDK abuse prevention.

High-Performance Optimization: A High-Speed Data Processing Engine

The extensive optimization of the BLS12-381 algorithm empowers Endless ' traffic statistics capabilities with exceptional data processing efficiency:

- Implements signature aggregation technology to merge multiple signatures into a single signature, reducing computational and storage overhead.
- Enhances network throughput and lowers blockchain load through an efficient traffic data signing and distributed storage strategy.
- Ensures efficient and stable data statistics and validation even under high-concurrency conditions.

Automated Smart Contract Settlements: A Fair and Transparent Billing Mechanism

Endless utilizes a smart contract-based automated billing system that allows cloud service providers to complete on-chain settlements driven by real-time data without incurring additional operational costs:

- All settlement operations are fully traceable on-chain, ensuring transparency.
- Prevents opaque pricing issues commonly found in centralized platforms, guaranteeing fair billing.
- Settlement results can be confirmed in real time, aligning perfectly with Web3 's trustless and decentralized economic model.

Data Traceability: A Reliable Foundation for Auditing and Queries

Through the Endless Signature Network 's distributed storage and synchronization mechanism, all traffic data is permanently archived and traceable on the blockchain:

- Ensures critical data remains immutable, supporting intelligent auditing and regulatory compliance.
- Allows developers and cloud service providers to query historical data anytime for business analysis and optimization.
- Provides a verifiable traffic data notarization mechanism to prevent data fraud or tampering.

3.2 Storage Resource-Based Consensus Model

3.2.1 Technical Solution

Endless adopts a verifiable proof-of-storage mechanism based on cryptographic commitments and the Challenge-Response protocol to construct an efficient, secure, and decentralized storage consensus system. Leveraging the blockchain's immutability, this model ensures data integrity and auditability. The core process is as follows:

1. **File Upload (Files to Provider)**: Users encrypt and fragment files before sending them to storage providers (such as decentralized storage nodes or cloud storage service providers).
2. **Metadata Generation (Metadatas to User)**: Storage providers generate metadata for the files (including hash values, storage location, timestamps, etc.), returning them to users to ensure verifiability.

3. **Digital Signature Binding (Signatures)**: Storage providers use digital signature algorithms (e.g., ECDSA) to sign the metadata, proving data ownership, integrity, and storage responsibility.

4. **Cryptographic Commitment Generation (Commitment)**: Storage providers compute a storage proof using advanced cryptographic techniques (e.g., KZG polynomial commitments) and record it on-chain for non-repudiable data storage.

5. **Challenge Mechanism (Challenge)**: The blockchain network or users can periodically issue random challenges requiring storage providers to prove they still possess the full data, preventing data loss or malicious deletions.

6. **Proof Generation (Proof)**: Storage providers compute a proof of storage based on the challenge request and submit it on-chain for verification, ensuring data availability.

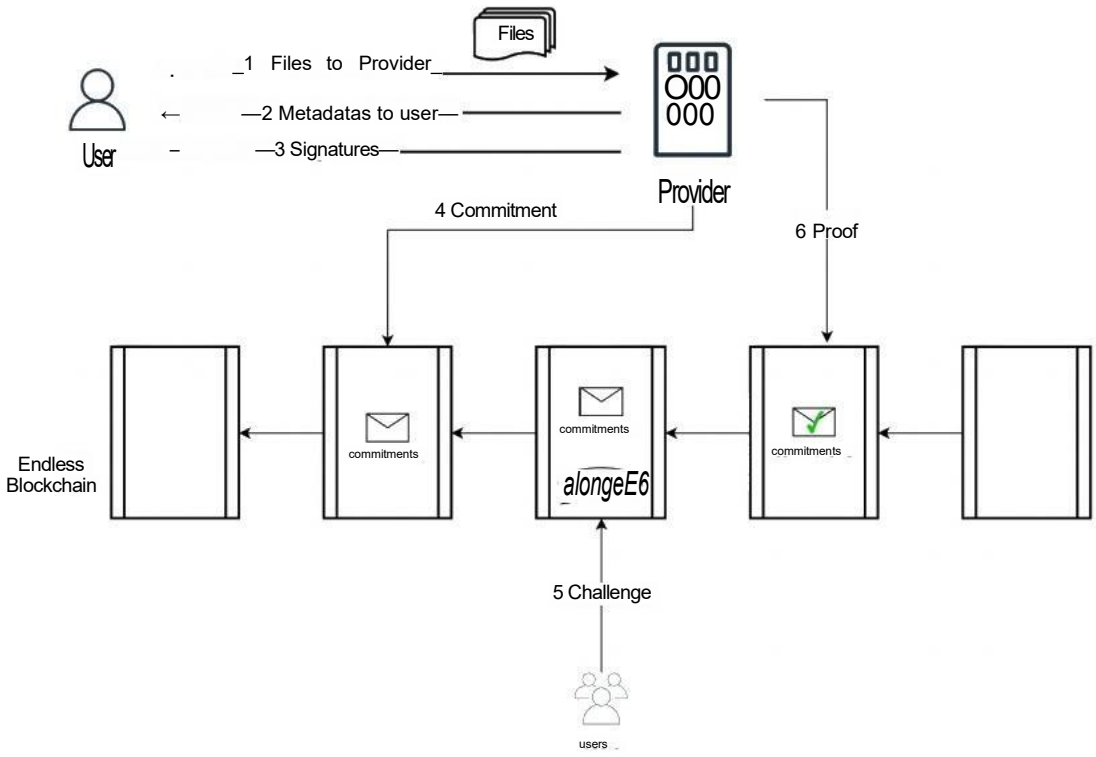

Figure 3.1:Storage Resource-Based Consensus Model

3.2.2 Endless KZG Polynomial Proof

Let F be a field and G be a group with generator g.Define the notation [a]=a·g.Let the highest degree of a polynomial be m,and let s be a security key.The values of d elements in group G are represented as:

$$[s],[s^2],[s^3],\ldots,[s^n].$$

For a polynomial of degree m,defined as.$f(X) = \sum_{i=0}^{m} f_i X^i$,its commitment Cf∈G is given by:

$$C_f = \sum_{i=0}^{m} f_i \left[s^i\right].$$

The KZG proof for f(y)=z is defined as:

$$\pi[f(y)=z]=CT,$$

where

$$T_y(X) = \frac{f(X) - z}{X - y}$$

is a polynomial of degree (m − 1).

To construct the proof, T's coefficients are computed recursively:

$$T_y(X) = \sum_{i=0}^{m-1} t_i X^i,$$

$$t_{m-1} = f_m,$$

$$t_j = f_{j+1} + y \cdot t_{j+1} \quad \text{(computed recursively from high to low degree).}$$

Expanding the polynomial $T_y(X)$ results in:

$$T_y(X) = f_m X^{m-1} + (f_{m-1} + y f_m) X^{m-2} + (f_{m-2} + y f_{m-1} + y^2 f_m) X^{m-3}$$
$$+ (f_{m-3} + y f_{m-2} + y^2 f_{m-1} + y^3 f_m) X^{m-4} + \cdots + (f_0 + y f_1 + \cdots + y^{m-1} f_m).$$

Let $\psi \in F_p$ be an ℓ-th root of unity such that $\psi^\ell = 1$. To verify a polynomial's values at ℓ points:

$$f(y) = z_0, \quad f(\psi y) = z_1, \quad \ldots, \quad f(\psi^{\ell-1} y) = z_{\ell-1},$$

we observe that:

$$(x - y)(x - \psi y) \cdots (x - \psi^{\ell-1} y) = x^\ell - y^\ell.$$

This allows us to compute:

$$g(x) = f(x) \;//\; (x^\ell - y^\ell),$$

where // denotes truncated polynomial division, producing the proof:

$$\pi\,[f(y) = z_0, \ldots, f(\psi^{\ell-1} y) = z_{\ell-1}] = [g(s)].$$

During verification, the residual polynomial is computed as:

$$h(x) = f(x) \mod (x^\ell - y^\ell),$$

and a bilinear pairing equation is checked:

$$e(C_f, \cdot) = e(\pi\,[f(y) = z_0, \ldots, f(\psi^{i-1} y) = z_{i-1}], [s^\ell - y^i])\, e(h(s), \cdot)$$

3.2.3 File Storage Commitment

When user U uploads files to storage provider P, P must generate daily commitments for newly uploaded files and submit them on-chain. Given K files uploaded in a day, each file f_i ($i \in [0, K)$) is divided into n_i segments:

$$\text{Segments}_i = [\text{seg}_0, \text{seg}_1, \ldots, \text{seg}_{n_i-1}].$$

The Merkle root for segments is computed as:

$$r_i = \text{MerkleRoot}(\text{Segments}_i).$$

File F_i's metadata $[\text{Metadata}]_i$ contains:

- Storage provider's account address $\text{addr}_{\text{server}}$.
- Merkle root r_i.
- File sequence number i.
- Accumulated uploaded bytes AccumulatedByteSize.
- Number of segments n_i.

Storage providers sign the metadata:

$$\text{sig}_i = \text{Signature}_{\text{ed25519}}([\text{Metadata}]_i).$$

For the K files, the commitment is generated as:

$$\text{Commitment} = \text{Fk20}([\text{sig}_0, \text{sig}_1, \ldots, \text{sig}_{K-1}]).$$

This commitment is submitted daily on-chain.

3.2.4 On-Chain Random Challenge Generation

On-chain challenges Challenge are generated from Commitment, producing a random pair $[r_f, r_s]$:

- $r_f \in [0, K)$ selects a file.
- $pos = r_s \% N$ selects a segment from the file.

3.2.5 Storage Provider's KZG Proof Generation

Storage provider P generates:

1. **Merkle Proof**:

$$Proof_{Merkel} = [RawData_{pos}, MerkelPath] .$$

2. **Fk20 Proof**:

$$Proof_{Fk20} = Prove\ (Commitment, r_f) .$$

3.2.6 On-Chain Verification

Smart contracts verify:

$$isPass = Verify_{merkel}\ (Proof_{merkel}) \wedge Verify_{Fk20}\ (Proof_{Fk20}) .$$

If isPass = false, the storage provider is penalized.

3.3 Fee and Penalty Mechanism

3.3.1 Treasury and Fee Management

- **Treasury**: On-chain allocated funds primarily used to subsidize the storage provider 's Gas fees.

· **Price Configuration**:

– Pricing can be set based on **Byte/KB/MB/GB**, with billing calculated on a **daily** basis.

– Prices can be updated at a minimum interval of **one month**.

· **Storage Provider Charges**:

– Storage fees are collected once per day.

– Fees are credited to the storage provider's account but remain frozen and can only be withdrawn after a 7-day challenge period.

· **Withdrawal Conditions**:

– The storage provider's account balance must be MIN__BALANCE (10,000 EDS) to withdraw funds.

3.3.2 Challenge and Penalty Mechanism

During storage services, if a storage provider fails to meet contractual storage obligations, users may initiate challenges. The challenge mechanism includes the following rules:

Definition: Let **package** represent the set of files uploaded by a user in a single day (stored within the same **commitment**). Assume the package contains N valid files.

· **Package Challenge Failure**:

– The storage fee paid for the past 7 days is fully refunded to the user.

– The storage provider incurs a triple penalty:

* 1x fee + 1 EDS refunded to the user.

* 2x fee + 1 EDS awarded to the challenger.

- **User Challenges Storage Provider for Missing Package Deletion Records**:
  - The storage fee paid for the past 7 days is fully refunded to the user.
  - The storage provider incurs a tenfold penalty:
    * 5x fee + 2 EDS refunded to the user.
    * 5x fee + 2 EDS awarded to the challenger.

3.3.3 Gas Fee Subsidy Mechanism

- If no storage challenge failure occurs within 14 days, the Gas fees for challenges completed 7 days earlier will be reimbursed.

3.3.4 Challenge Limitations

- **General Public Challenge Limit**:

$$10 \times \left(\frac{1}{\text{GB} \cdot \text{day}}\right) + 2 \times \log_2(\text{Number of Files})/\text{day}$$

- **User-Specific Challenge Limit per File**:

$$1 \times \left(\frac{1}{\text{File} \cdot \text{day}}\right)$$

3.3.5 User Payment Default Handling Mechanism

- When a user fails to pay for storage:
  - The storage provider has the option to delete all user files.
  - The smart contract will simultaneously delete the on-chain storage records.

3.4 Consensus Model of the Endless Blockchain

Endless adopts a hybrid consensus model that combines Byzantine Fault Tolerance (BFT) and Proof-of-Stake (PoS) mechanisms to ensure the security, decentralization, and high performance of the blockchain.

BFT Consensus Mechanism

The Endless blockchain is maintained by a set of validators who participate in a Byzantine Fault Tolerant (BFT) consensus protocol to validate and process user transactions. Specifically:

- Token holders can lock/stake their EDS tokens to support validators of their choice.
- The voting power of each validator in consensus is proportional to the amount of EDS tokens staked on them.
- A validator can exist in one of the following states:
  - **Active State**: Participating in consensus, executing transaction validation, and block proposal.
  - **Inactive State**: Occurs under the following conditions:
    - Fails to meet the minimum staking requirement due to insufficient staked tokens.
    - Removed from the validator set due to the validator rotation mechanism.
    - Chooses to go offline to sync blockchain state.
    - Identified by the consensus protocol as a non-participant due to poor historical performance.

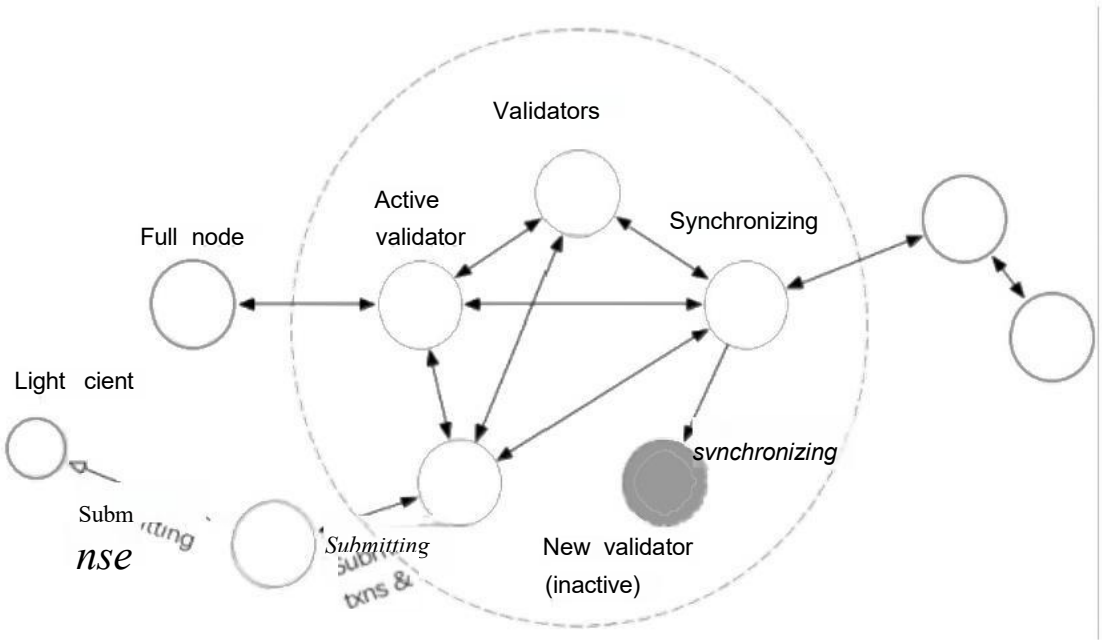

Figure 3.2:Endless Blockchain Consensus Model

PoS (Proof-of-Stake)Consensus Mechanism

To become a validator on the Endless blockchain and participate in transaction verification,candidate validators must stake a certain amount of EDS tokens. The specific rules of the PoS mechanism are as follows:

· **Staking affects multiple consensus parameters**:

-Transaction propagation phase:Determines the 2f+1 weight of PoAv (Proof of Availability).

-Block metadata ordering: Influences voting power and leader selection.

·**Validator reward distribution:**

-Validators can independently decide how to distribute PoS rewards between themselves and their delegators.

- **Delegator staking rights**:
  - Any token holder can stake their funds with one or more validators, earning PoS reward shares based on pre-agreed distribution terms.
  - At the end of each epoch, rewards are automatically settled via on-chain smart contracts.
- **Validator participation rules**:
  - Any entity with sufficient minimum staking can freely join the Endless blockchain as a validator.
  - All parameters, including the minimum staking amount, can be adjusted through decentralized governance processes and executed on-chain.

4 Asset Standards

4.1 Unified Conversion of FT Asset Standards

On the Endless blockchain, all fungible tokens (FT) adhere to the **FungibleAsset** standard, ensuring a consistent token interface and reducing integration complexity for developers.

The **FungibleAsset** standard defines the attributes and API interfaces for all FT assets:

- This standard ensures that all fungible tokens within the Endless ecosystem conform to a universal interface.
- Developers can conveniently create user-specific FT assets using development tools such as the Endless CLI and TypeScript SDK.

For detailed information on the **Fungible Asset** standard, please refer to: Endless Fungible Asset Standard

4.2 Unified Conversion of DA Asset Standards

Similar to the FT asset standard, the Endless blockchain employs the **DigitalAsset** (DA) standard to standardize the management of non-fungible tokens (NFTs).

The **DigitalAsset** standard offers the following features:

- Supports NFT minting via the Endless CLI and TypeScript SDK.
- The Endless CLI provides the `nft` subcommand, making NFT creation more convenient.
- The `nft` subcommand supports the **Soul Bound** feature, allowing users to mint non-transferable NFTs for purposes such as on-chain identity verification or non-tradable assets.

For more details and examples related to the **Digital Asset** standard, please refer to: Endless Digital Asset Standard

5 Transactions

5.1 Sponsored Transactions

In blockchain systems, users typically need to pay transaction fees (Gas Fees) to execute transactions. However, for new users, developers, or specific decentralized applications (dApps), these fees may become a barrier to adoption and participation.

Sponsored transactions allow third parties to cover transaction fees on behalf of users, simplifying the onboarding process and enhancing the overall user experience. For example, the mainstream blockchain Aptos enables third-party gas station services to sponsor transaction fees. However, the centralized nature of such services may result in the following issues:

- **Single point of failure**: If the service provider's server goes down, the entire sponsorship service becomes unavailable.

- **Transaction privacy risks**: Since fee payments are controlled by a centralized service, user privacy may be compromised.

To address these issues, **Endless offers a fully on-chain sponsored transaction solution**, enabling a decentralized payment mechanism. When a Move module implements the sponsorship functionality, any transaction invoking this function will have its gas fees deducted directly from the associated Move

module account.

Key considerations for developers implementing the sponsorship function:

- **Access control rules**: Support mechanisms such as whitelists/blacklists to regulate which accounts can invoke the sponsored transaction function.

- **Management of the Move module account fund pool**: Ensure sufficient sponsorship funds are available to continuously support transaction fee payments.

Since the entire sponsored transaction process is executed fully on-chain and remains decentralized, it provides a more robust and fault-tolerant service architecture compared to traditional centralized gas station models.

For technical details, refer to: Sponsored Transaction.

5.2 Secure Transactions

While blockchain technology is favored for its decentralization, transparency, and security, smart contracts may still contain vulnerabilities that pose serious threats to the ecosystem. Common vulnerabilities include:

- Arithmetic overflow/underflow;
- Reentrancy attacks;
- Access control issues;
- Oracle manipulation;
- Logic bugs.

Attackers often exploit smart contract vulnerabilities to conduct fraud, such as:

- Creating fake investment projects or games to lure users into transactions, then using reentrancy attacks to steal assets;
- Coordinating multiple malicious contracts and external data sources to manipulate markets and evade tracking or forensic analysis.

Analysis of Transaction Preview Vulnerabilities

Currently, mainstream blockchain wallets provide transaction preview functionality. When a user submits a transaction (such as a transfer or DApp interaction), the wallet performs the following steps:

- Calls a specialized RPC method via on-chain nodes to simulate transaction execution;
- Estimates gas fees and provides a transaction preview result;
- Displays the expected balance changes for the user's account and related accounts.

However, the transaction preview result *does not necessarily* match the actual transaction execution result, creating an attack vector that adversaries can exploit to deceive users into submitting malicious transactions:

- **Path A**: During the preview phase, the balance changes appear as expected to the user.
- **Path B**: During actual execution, the transaction logic diverges, leading to asset theft.

Endless' Strong Security Mode Validation

To mitigate such attacks, **Endless enhances transaction security validation** by introducing a **Strong Security Mode (Safety Switch)**:

- During transaction execution, the system verifies whether the previewed results match the final execution results.
- If the account balance changes unexpectedly, the system automatically intercepts the transaction to prevent asset loss.

For example, if a malicious contract attempts to exploit **Path B** to steal user funds, the activated Safety Switch ensures:

- The transaction is intercepted during execution, preventing financial loss.
- The security of user assets remains intact.

For technical details, refer to: Secure Transaction Specification.

5.3 Transaction Processing

The Endless blockchain maximizes system throughput, enhances concurrency, and reduces system complexity by leveraging **Pipeline Processing**, **Batch Processing**, and **Parallel Transaction Execution**. This optimized design not only improves overall performance but also unlocks new interaction models between validators and clients, such as:

1. Clients can receive notifications when a specific transaction is included in a **Persisted Batch**.
2. Clients can choose to execute transactions locally, reducing network latency and improving transaction confirmation speed.

3. Clients can wait for validators to certify the transaction execution results to ensure finality and consistency.

5.3.1 Transaction Processing Optimization Mechanisms

The Endless blockchain adopts various optimization techniques to enhance transaction throughput, reduce latency, and strengthen network security. Key optimization mechanisms include **Batch Processing**, **Streaming Transaction Propagation**, **Block Metadata Ordering**, and **Blockchain Time Synchronization**.

Batch Processing

Batch processing is a critical efficiency optimization strategy in multiple operational phases of the Endless blockchain. During the propagation phase, transactions are grouped into multiple batches by each validator, and in the consensus phase, these batches are merged into complete blocks.

- **Core Logic**: Transactions are grouped into batches and processed simultaneously in the consensus flow.

- **Latency Optimization**: While batch processing may introduce minor latency, the system supports **flexible configuration**, enabling automatic optimization between low latency and high throughput.

Streaming Transaction Propagation

To improve network utilization, Endless adopts a decoupled architecture between transaction propagation and consensus, enabling continuous transaction streaming instead of traditional batch broadcasting.

- All validators continuously stream batch transactions, maximizing the usage of available network resources.

- Each batch carries a timestamp, which serves two key purposes:

  – **Garbage Collection (GC)**: Efficiently cleans up expired transactions, reducing storage overhead.

  – **Defense Against Storage Attacks**: Ensures transactions do not accumulate indefinitely, mitigating the risk of malicious DoS attacks.

Block Metadata Ordering

Endless utilizes the **DiemBFTv4** consensus protocol and enhances performance by decoupling transaction propagation from execution, which significantly improves blockchain throughput and reduces consensus latency:

- **Decentralization Optimization**: Non-consensus tasks such as transaction propagation and execution are completely independent of the consensus process.

- **Protocol Security**: **DiemBFTv4** ensures **high availability and security** even in **partially synchronous** environments, maintaining an efficient consensus mechanism under real-world network conditions.

Blockchain Time Synchronization

To enhance time consistency across the network, each block in the Endless blockchain contains an **Approximate Consensus on Physical Timestamp**. This design supports multiple on-chain functionalities, including:

- **Time-Dependent Logic**: Smart contracts can leverage global time for on-chain computations.

- **Accurate On-Chain Data Association:** Ensures transactions are executed in chronological order,enabling up-to-date data writes.
- **Transaction Expiry Mechanism:** Implements a **TTL(Time-to-Live)** mechanism to prevent outdated transactions from lingering,thereby improving blockchain efficiency and throughput.

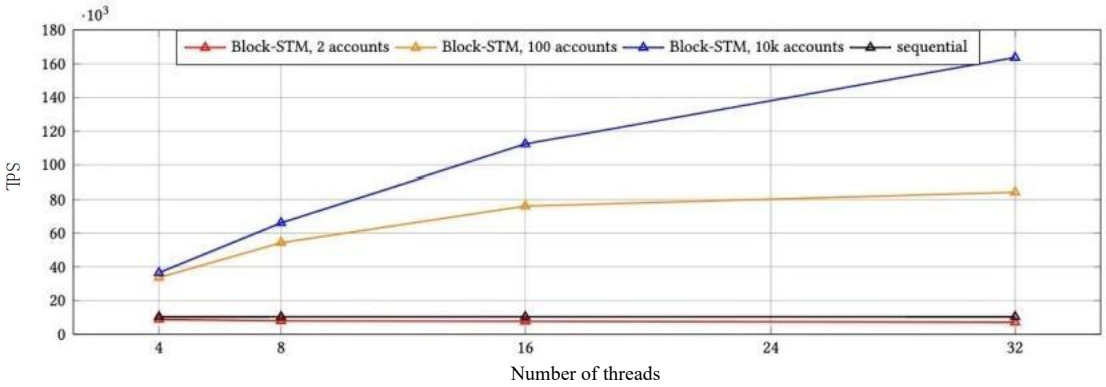

Figure 5.1:TPS Growth with Batch Processing for Different Account Numbers in Block-STM

5.3.2 Parallel Transaction Execution

In the Endless blockchain,once the metadata ordering of a consensus block is completed,any validator,full node,or client can execute transactions.At least 2f+1 weighted validators must persistently commit the transaction batches Since transaction propagation occurs continuously,additional honest validators will sequentially receive transaction batches over time.

Parallel Data Model

The Move data model supports global data and module addressing.Transactions that do not conflict with each other can be executed in parallel.Endless

employs a pipeline design that allows transaction groups to be reordered to minimize conflicts, thereby enhancing concurrent execution capabilities. This model introduces the concept of **"incremental writes"**, enabling all transactions to be processed in parallel, while conflict transactions apply their incremental writes sequentially in the final phase to ensure deterministic transaction execution.

Parallel Execution Engine

The **Block-STM** parallel execution engine is responsible for managing conflicts within ordered transaction sets and adopts an optimistic concurrency control mechanism to maximize parallel execution capabilities. Transaction batches are initially executed in parallel using optimistic execution, and in the event of a validation failure, they are re-executed. Block-STM leverages a multi-version data structure to mitigate write-write conflicts, thereby supporting more complex concurrent transaction processing.

5.3.3 Experimental Results and Performance Optimization

Experimental results indicate that under low-conflict scenarios, Block-STM achieves up to a **16× speedup**, while in high-conflict scenarios, it still provides a performance improvement of more than **8×**. Block-STM dynamically extracts parallelism from workloads, reducing the number of transactions, improving execution efficiency, and significantly lowering user latency.

1. Transaction Ordering and Parallelism Optimization

The block metadata ordering step does not restrict transaction reordering during the parallel execution phase. Transactions can be dynamically reordered

across multiple blocks to optimize concurrent execution while ensuring that all honest validators maintain determinism in transaction execution results.By integrating Block-STM and utilizing transaction reordering techniques,the parallel execution efficiency can be further enhanced.

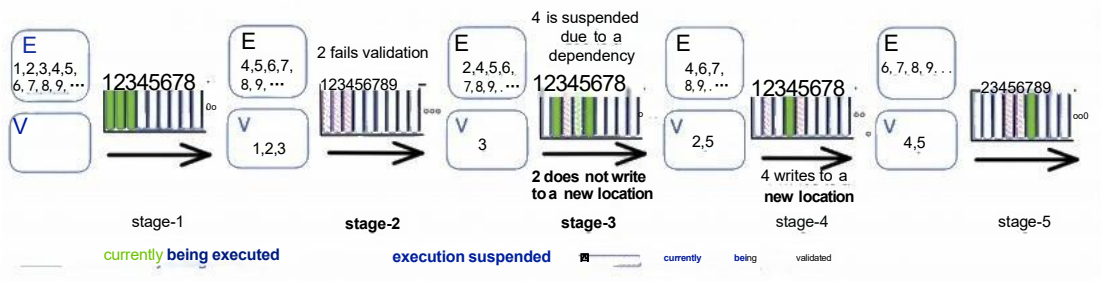

Figure 5.2:Block-STM Collaboration Model

2.Performance Optimization

The Endless blockchain significantly enhances system performance in transaction processing,throughput,latency,scalability,and hardware efficiency through various optimization techniques.These optimizations not only improve the overall system performance but also ensure efficient and stable operation in large-scale application scenarios.

Parallelization and Modular Transaction Processing Parallelization and modular transaction processing are core strategies for performance optimization in the Endless blockchain.By dividing the transaction processing workflow into independent parallel execution stages,the system maximizes computational resource utilization,significantly enhancing overall throughput.

Throughput Enhancement Thanks to the batch processing mechanism and pipeline processing architecture,the Endless blockchain can process large groups

of transactions at each stage, significantly boosting system throughput. Even under high-load conditions, the system effectively handles massive transaction requests, avoiding the network congestion issues commonly seen in traditional blockchain systems.

Latency Optimization The system leverages the Block-STM parallel execution engine, which minimizes transaction processing latency through a multi-version data structure and optimistic concurrency control. In low-conflict environments, transaction processing speeds can improve by over **16×**, while in high-conflict scenarios, the system maintains at least an **8× performance improvement**.

Optimization of Transaction Latency and Final Confirmation Time The Endless blockchain is deeply optimized to reduce transaction latency and final confirmation time, ensuring that users receive rapid transaction confirmations to enhance their transaction experience.

Block Metadata Ordering By decoupling non-consensus tasks from the consensus phase, the system significantly reduces transaction processing latency. Coupled with the **DiemBFTv4** consensus protocol, the network achieves fast and secure transaction confirmation even in **partially synchronous** environments.

Physical Timestamps Each block proposal carries a physical timestamp to support time-dependent logic operations and transaction expiration management. This mechanism ensures the timely processing of transactions and prevents execution delays caused by time lag.

6 Indexer

The Endless Indexer serves as a crucial supporting service for the Endless blockchain, providing rich query interfaces that support retrieval of key data such as address transaction history, Coin details, and NFT details.

As a high-throughput blockchain, Endless generates a large volume of data at a rapid growth rate, posing significant challenges for the implementation of the Indexer. To meet the demands of efficient indexing in a big data environment, the Endless Indexer adopts RocksDB as its underlying data storage and leverages metadata processing and storage, combined with a chain-based Hook mechanism, to achieve efficient indexing of the entire transaction history on Endless.

The Endless Indexer offers two modes of data acquisition to accommodate different deployment environments:

- **Local Environment**: When the Indexer and the Endless full node are in the same local environment, Unix Domain Socket is used for synchronization, enabling efficient and low-latency data interaction.
- **Remote Environment**: In distributed or cross-regional deployment scenarios, the Indexer employs GRPC as the transaction data transmission protocol to ensure stable and efficient data transfer.

6.1 Performance Comparison

Compared to the Aptos Indexer, the Endless Indexer demonstrates significant advantages in synchronization speed, query speed, and database storage efficiency.

6.1.1 Synchronization Speed

Synchronization speed is primarily influenced by transaction data retrieval speed, transaction processing efficiency, and database write throughput. In an environment with rapidly growing data, database write performance becomes a critical factor in overall system performance. The Aptos Indexer uses Postgres, whereas the Endless Indexer utilizes RocksDB. The fundamental differences between their storage engines in terms of write mechanisms and performance directly impact their efficiency in handling large-scale data scenarios.

Write Performance of Postgres

As a relational database, Postgres' write performance is affected by multiple factors as data volume grows:

- **Transaction Log and Index Updates**: Every write operation requires updating the Write-Ahead Log (WAL) and modifying associated indexes. As data volume increases, index structures become larger, leading to higher update costs and reduced overall write efficiency.

- **Table Bloat**: Frequent write and update operations cause table bloat, requiring periodic VACUUM and ANALYZE operations to reclaim disk space and optimize query performance.

- **Lock Contention**: In high-concurrency write scenarios, Postgres may experience performance bottlenecks due to contention for table or row locks, leading to a drop in write speed.

Overall, in high-data-volume and high-concurrency write environments, Postgres' write speed tends to degrade as data scales up.

Write Performance of RocksDB

RocksDB is a key-value storage database based on the Log-Structured Merge Tree (LSM Tree) structure, optimized for large-scale data storage and high-concurrency write scenarios. Its main advantages include:

- **Optimized Sequential Writes**: RocksDB employs an LSM tree structure, temporarily storing write operations in an in-memory MemTable and flushing them to disk in batches, thereby improving write throughput.

- **Tiered Storage**: By leveraging the multi-level compaction mechanism of the LSM tree, RocksDB efficiently organizes data, avoiding frequent random writes and significantly reducing disk I/O overhead.

- **Write Amplification Control**: Through intelligent compaction and compression strategies, RocksDB effectively mitigates write amplification effects, allowing write performance to maintain excellent linear scalability as data volume grows.

Compared to Postgres, which uses a relational storage structure, RocksDB performs more stably in high-frequency transaction write scenarios, better supporting the Endless Indexer's need for efficient indexing in large-scale data environments.

6.1.2 Write Speed

To provide an intuitive comparison of the write performance of Postgres and RocksDB as data volume increases, we present the following table:

Table 6.1: Comparison of Write Speed Decline Trends

Data Growth	Postgres Write Speed Decline	RocksDB Write Sp
Small-scale Data	Fast	Fast
Medium-scale Data	Noticeable Decline	Slight Decl
Large-scale Data	Significant Decline	Stable
Ultra-large Data	Severe Decline, Optimization Needed or Sharding	Maintains High E

Assuming the write speed S changes with data volume D, we can express the relationship with the following functions:

- Postgres: The write speed S_P decreases as data volume D increases, which can be approximated as:

$$S_P(D) = \frac{C}{D^{\alpha}}, \quad \alpha > 1$$

  where C is a constant, and α represents the rate of write performance degradation.

- RocksDB: The write speed S_R changes only slightly with increasing data volume D, which can be approximated as:

$$S_R(D) = \frac{C}{\log(D+1)}, \quad \text{logarithmic decline}$$

In summary, RocksDB demonstrates significantly better write performance than Postgres in rapidly growing data scenarios, especially in large-scale data processing and high-concurrency write environments. The Endless Indexer utilizes

RocksDB, enabling it to handle massive data from high-speed blockchains more efficiently, whereas the Aptos Indexer, using Postgres, faces greater performance constraints.

During periods of high blockchain load (i.e., high TPS), the Endless Indexer is able to maintain near real-time synchronization, whereas the Aptos Indexer lags behind the full node by several hours.

6.1.3 Query Speed

The following table compares the query time for retrieving the total transaction count and transaction list of address 0x1 in a local environment (to eliminate network latency factors):

Table 6.2: Query Time Comparison (Unit: Seconds)

Query Target	Aptos Indexer	Endless Indexer
0x1 Transaction Count	60.76759320	0.00051460

Since the Aptos Indexer uses Postgres as its database, query time increases proportionally with the number of transactions. In contrast, the Endless Indexer, using RocksDB, benefits from its LSM (Log-Structured Merge Tree) design, causing query speed to decline only slightly.

6.1.4 Disk Usage

The following table compares the disk space required when indexing the blockchain at the same height:

Compared to the Aptos Indexer, the Endless Indexer reduces disk usage by 99%.

Table 6.3: Disk Usage Comparison (Unit: MB)

Metric	Aptos Indexer	Endless Indexer
Disk Usage	61400	722

6.2 Advantages and Limitations

Although the Endless Indexer demonstrates significant advantages in terms of performance, it still has certain limitations, including:

- **Limited Flexibility of RESTful APIs**: Compared to the Aptos Indexer, which supports GraphQL queries, the Endless Indexer provides only RESTful APIs. This results in lower query flexibility, making it difficult to perform complex multi-condition queries and data aggregation operations.

- **Restricted Query Conditions**: Since the Endless Indexer adopts a key-value (KV) database as its underlying architecture, its query conditions are less flexible compared to relational databases such as Postgres. As a result, it is better suited for applications with relatively fixed query patterns.

7 Token Standards

7.1 Introduction to Token Locking Standards

The Endless system contract introduces a new smart contract, `locking_coin_ex.move`, designed for managing token locking and distribution. This contract standardizes the token distribution process through a locking and unlocking mechanism, ensuring that tokens are gradually unlocked over a specified period, thereby effectively regulating token circulation. Additionally, the contract provides a `view` API that allows users to query token locking status at any time.

By establishing a standardized token locking and release mechanism, this contract enables all DApp projects that adopt the standard to manage token assets more efficiently, transparently, and fairly.

7.2 Core Features of the Contract

- **Token Locking**: The contract allows administrators to lock tokens at a designated address and set an unlocking schedule. The locked tokens will be gradually unlocked over the specified period.

- **Token Unlocking**: According to the predefined unlocking schedule, the contract will automatically release the corresponding amount of tokens at the end of each unlocking period.

- **Query Functionality**: The contract provides a variety of query interfaces, enabling users to check the total locked amount in the system, the locking details of all participants, the staked amount, and the unlocking status of a specific user.

- **Event Logging**: During the token unlocking and claiming process, the contract records relevant events to facilitate data tracking and auditing by users.

7.3 Design Objectives and Functions

The core design principle of this contract is to regulate token circulation by implementing a ”locking” and ”gradual unlocking” mechanism. This approach prevents a large amount of tokens from being released in a short period, helping to maintain market stability.

7.4 Design Goals

- **Prevent Market Volatility**: By gradually unlocking tokens, the contract mitigates significant market fluctuations caused by the sudden release of a large number of tokens.

- **Incentivize Long-term Holding**: The token locking mechanism encourages users to hold tokens for an extended period, enhancing the stability of token value.

- **Transparent Management**: Leveraging the automated execution and event logging functionality of smart contracts, the entire token locking and unlocking process remains transparent and auditable.

For more details, please refer to the Token Locking & Distribution documentation.

8 Move Smart Contracts

8.1 Move Contract Language

Move is an innovative smart contract programming language that emphasizes security and flexibility. The Endless blockchain adopts Move 's object model to represent its ledger state and utilizes Move code (modules) to define state transition rules. User-submitted transactions can perform the following operations:

- Deploy new modules;
- Upgrade existing modules;
- Invoke entry functions defined in a module;
- Execute scripts that interact directly with the public interfaces of modules.

The Move ecosystem consists of a compiler, a virtual machine, and various development tools. Inspired by the Rust programming language, Move introduces concepts such as linear types to make data ownership more explicit at the language level. Move emphasizes resource scarcity, protection mechanisms, and access control. Move modules define the lifecycle, storage scheme, and access patterns of each resource, ensuring the security of assets such as tokens. Specifically:

- The creation of resources must be strictly verified, preventing unauthorized generation;

- Resources cannot be double-spent, ensuring the consistency of assets across the network;

- Resources cannot be lost or destroyed arbitrarily, maintaining data integrity.

Move employs a Bytecode Verifier to ensure type and memory safety, even when running untrusted code. Furthermore, to enhance code reliability, Move provides a formal verification tool called Move Prover, which verifies the functional correctness of Move programs based on specified specifications, ensuring that contract logic behaves as expected.

To support a broader range of Web3 application scenarios, the Endless blockchain incorporates several optimizations tailored for Move:

- **Fine-grained Resource Control**: Enables more precise resource management, facilitating efficient parallel execution while ensuring that the cost of accessing and modifying on-chain data remains nearly constant.

- **Fine-grained Storage Optimization**: Provides table-based storage structures that allow a single account to store large-scale datasets, such as massive collections of NFT assets.

- **On-chain Native Shared Accounts**: Endless supports fully on-chain representations of shared or autonomous accounts, enabling decentralized autonomous organizations (DAOs) to collaborate using shared accounts while allowing these accounts to serve as containers for heterogeneous resources.

8.2 Move Contract Modules

8.2.1 Module Overview

Move modules are composed of Move bytecode and contain declarations of data types (structures) and procedures. Each module is uniquely identified by the account address that declares it and the module name. For example, in Figure 2, the first currency module is identified as `0x1::coin`. Modules can depend on other on-chain modules to facilitate code reuse. For instance, in Figure 2, the wallet module depends on the coin module to provide extended functionality.

Each module name must be unique within the same account address, meaning that an account cannot declare multiple modules with the same name. For example, in Figure 2, the account at address `0x1` cannot declare another module named `coin`. However, an account at address `0x3` can declare a separate `coin` module, uniquely identified as `0x3::coin`. It is important to note that `0x1::coin::Coin` and `0x3::coin::Coin` are considered distinct types—they cannot be used interchangeably and do not share common module code.

On the other hand, `0x1::coin::Coin<0x2::wallet::USD>` and `0x1::coin::Coin<0x2::w` are different instantiations of the same generic type. While they cannot be used interchangeably, they share the underlying module code.

Modules are organized into packages located at the same address. The package owner can publish these modules on-chain as a unit, including both bytecode and package metadata. Package metadata determines its upgradeability:

- **Immutable Packages**: Once deployed, the package contents cannot be modified.
- **Upgradeable Packages**: The package supports upgrades but must pass compatibility checks before upgrading:

-Existing entry functions cannot be modified;

-Existing in-memory stored resources cannot be altered;

-New functions and resources can be added.

The Endless framework consists of the core library and configurations of the Endless blockchain and is defined as a standard upgradeable module package.

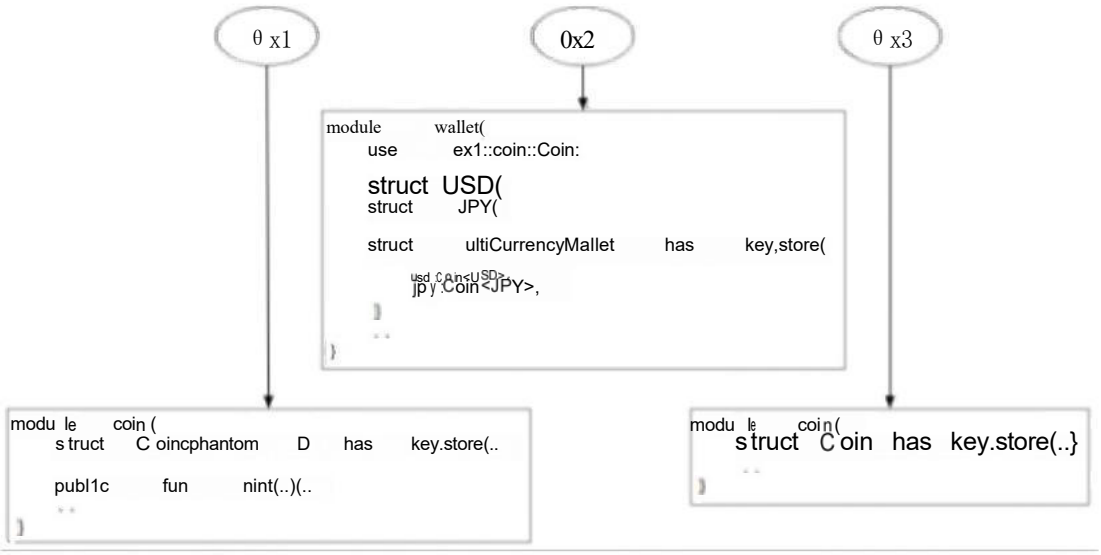

Figure 8.1:Example of On-chain Move Modules

8.2.2 Resources

In the Move language,each account address can be associated with specific data values,and only one value of a given type can be stored under that address. For example,in Figure 3,address Ox50 holds a value of type 0x3::coin::Coin. Different instantiations of the same generic type are treated as distinct data types,providing a foundation for system scalability.The rules governing data changes,deletions,and declarations are encoded in the module that defines the data,and Move's security and verification mechanisms prevent unauthorized code from manipulating these data types directly.

While each address can store only one top-level value per type,this limitation can be circumvented using wrapper types.For instance,developers can create container data structures to manage multiple values without violating Move's storage constraints.

Not all data types can be stored on-chain—only those with both the key ability(Key Ability)and storage ability (Store Ability)can be stored as top-level or nested values.In the Move ecosystem,data types possessing both abilities are called resources and ensure that such data can be securely and efficiently stored and managed on the blockchain.

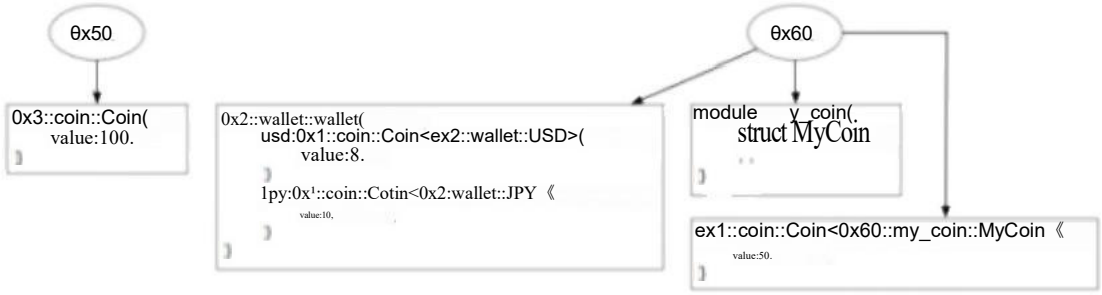

Figure 8.2:Example of On-chain Data

8.3 Execution and Security of Move Smart Contracts

The Endless blockchain utilizes the Move smart contract language in combination with its parallel execution architecture to achieve efficient and secure contract execution.During execution,the Move Virtual Machine(Move VM) manages ledger state updates and resource handling,ensuring that no state conflicts or resource leaks occur while contracts are running.

Each step of Move contract execution undergoes formal verification and static analysis to ensure compliance with established security standards,thereby reduc-

ing the risk of smart contract vulnerabilities. Specifically:

- **Bytecode Verification**: Before execution, the Move VM performs checks using the Bytecode Verifier to ensure that the code adheres to type safety and memory safety standards.

- **Resource Security**: Move's linear type system enforces unique resource ownership, ensuring that resources cannot be illegally duplicated, lost, or consumed multiple times.

- **Access Control**: The smart contract employs a modular access control mechanism, allowing developers to define custom permission policies, restricting access to critical state variables and functions to enhance security.

The Endless blockchain supports a modular smart contract development architecture, enabling developers to flexibly extend and upgrade existing contracts without modifying underlying logic. This design:

- Facilitates composability of smart contracts, simplifying code reuse and improving development efficiency;

- Prevents large-scale contract migrations by allowing incremental upgrades, optimizing the sustainability of Web3 applications;

- Maintains system security during upgrades, mitigating potential regression vulnerabilities.

9 On-chain Trusted Randomness

In blockchain systems, the trustworthiness and security of randomness are crucial, especially for applications that rely on randomness, such as elections, lotteries, and gaming. The Endless chain implements an on-chain trusted randomness generation mechanism to achieve enhanced security and efficiency.

- **Weighted Publicly Verifiable Secret Sharing (wPVSS) Algorithm**: Endless employs the wPVSS algorithm to ensure that each validator node can efficiently perform randomness generation while minimizing communication overhead.

- **Weighted Distributed Key Generation (wDKG) Protocol**: Endless utilizes the wDKG protocol, which offers higher communication efficiency, further enhancing the reliability of random number generation.

- **Weighted Verifiable Random Function (wVRF)**: In each round, validator nodes evaluate the wVRF to ensure the trustworthiness and security of the randomness while avoiding a linear increase in communication overhead relative to validators' staked amounts.

- **Application Scenarios**: The Endless smart contract provides a randomness API, such as `randomness::u64_integer()`, which generates unbiased 64-bit unsigned integers. By leveraging these APIs, DApp developers can

implement secure and verifiable random number generation, enhancing the security and fairness of their applications.

For more details on using the Endless randomness API, including examples and security considerations, please refer to:

- API Usage Guide
- Example Programs
- Security Considerations