A Provenance Tracking Filesystem[1]

Embodying provenance tracking in an easily attachable filesystem facilitating widespread adoption.

Gary Mawdsley CTO/CEO Lockular Limited

April 2022[2]

[1] Facilitated by Polkadot Substrate Parachains

[2] Updated August 2023 to include CoreTime as per Polkadot Decoded 2023

This document presents the Lockular Provenance Tracking filesystem, which incorporates provenance tracking directly into its architecture through the use of cryptographic methods and blockchain technology. This filesystem serves as a foundational element for ensuring complete provenance of data stored and modified on POSIX filesystems.

Maybe more detailed abstract?

Introduction

At the core of computing, filesystems play a pivotal role. The efficiency and functionality of cloud computing are largely due to the extensive virtualisation of machines and applications. These are essentially defined by metadata files that conjure a virtual environment on larger hardware infrastructures. In such a setup, every component of a cloud-based system, including the filesystems, is virtualised.

This virtualisation leads to significant gains in efficiency and security, enabling organisations to swiftly activate their computing capabilities using metadata that defines their computing environments, coupled with their application data. In today's context, an organisation's computing resources is entirely conceptualised as data: a combination of application data and metadata describing the computing environment.

Viewing filesystems not as tangible drives but as layers of metadata that facilitate access to file blocks and the filesystems' metadata enhances the interpretation expected by applications.

The proposition here leverages this understanding to extend filesystem protocols, integrating them with a standard blockchain validation process. This integration aims to provide an immutable audit trail of file activity.

While there are existing solutions from hardware and software vendors for tracking file changes, the critical distinction here is the elimination of any superuser capability to modify historical records yielding a regime of immutable provenance. This underscores the significance of incorporating blockchain technology.

What is proposed is an augmentation of the current virtual filesystem paradigm by introducing unavoidable trigger steps for the immutable tracking of file modifications. Here are the significant points:

- Provenance Tracking through Blockchain: The filesystem uses

blockchain technology to create an immutable audit trail for file operations, ensuring that no superuser can alter historical records. This is a critical feature for enhancing security and trust in the system.

- Integration of Shamir's Secret Sharing (SSS): The filesystem incorporates SSS to split the filesystem's sensitive data (both metadata and file blocks) into shares that can only be reconstructed with a predefined number of parts. This splitting and reassembly process is crucial as it provides a protocol hook on which to anchor the state transitions that serve as provenance triggers, thereby enhancing both security and the functionality of provenance tracking.
- Use of Polkadot Parachain: The system utilises a Polkadot parachain for enhanced security and interoperability within the blockchain network. This allows the filesystem to benefit from the shared security and interoperability features of the Polkadot network.
- Multi-Signature Protocol: It employs a multi-signature protocol to require consensus among multiple stakeholders (users, organisation, platform) for critical operations, adding another layer of security and collaborative decision-making.
- FUSE and NFS Integration: The filesystem is adaptable to different applications through its support for FUSE and NFS APIs, allowing it to operate both within and outside of application process spaces.
- Redis-based Shares Repository: For robust and scalable storage of the shares, the system uses a Redis-based repository, ensuring performance and reliability.
- Privacy and Security Measures: The document discusses various mechanisms to ensure the privacy and security of the data, including cryptographic commitments and zero-knowledge proofs.

These significant points outline a sophisticated system designed to enhance the security, reliability, and efficiency of data management in cloud-based and distributed computing environments. The document emphasises the innovative use of blockchain and cryptographic techniques to address the challenges of provenance tracking in filesystems.

Moreover, this approach is built on standards and integrates the following industry standard technologies and components:

- Shamir's Secret Sharing (S^3): This cryptographic technique divides a secret into several parts, requiring only a specific subset (the threshold) of these parts to reassemble the original secret. *Talk about the polynomial nature of Shamir.*

- State Transition System: A framework in which changes in state are triggered by specific inputs marshalled by a set rules. Within the context of blockchain, these state transitions are initiated by transactions that are triggered by filesystem operations that in turn trigger the storage protocol that invoke the Shamir operations.
- Polkadot Parachain: A specialised, virtual filesystem focused blockchain that links to the Polkadot network, gaining from its shared security and the ability to interoperate.
- Multi-Sig Protocol: A cryptographic scheme enabling multiple users to collaboratively engage in a transaction within the filesystem, maintaining the privacy of their inputs while collectively determining the transaction's outcome.
- FUSE and NFS Filesystem APIs: Standard virtual interfaces for filesystem integration, facilitating external app integration with the filesystem.
- Shares Repository: A platform-independent, replicated, and persistent storage solution for the shares, ensuring robustness and scalability. This is switched out to any preffered high performant cache and database. For large volumes then Cassandra is a very good choice, Redis for nimble high speed.
- IPFS Protocol: Used to handle large binary object file blocks leveraging content addressable storage. (explain about the effect of different keys on the content)

This approach is based on dividing the filesystem file blocks and filesystem metadata (referred to as the secrets) into multiple parts. The division and merging of hese parts are managed by multiple participants within a multi-signature blockchain framework. Each step of division and merging follows a defined sequence of state changes, which are enacted and recorded on a blockchain, serving as an immutable log.

Detail

Designing the Scheme

Shamir's Secret Sharing Integration

This filesystem architecture stores both its metadata and filesystem blocks as encrypted secrets, utilising a secret sharing scheme. The segmentation of these secrets into shares, and their subsequent recombination, is managed via a Polkadot substrate parachain. Specifically, the encoding of the shares aligns directly with a state transition

model. The designed state transitions are represented on a parachain and serve as checkpoints for initiating the recording of audit information onto the blockchain. The audit information captured corresponds with changes to the files. While the division of secrets into shares enhances security, the critical aspect is that the disassembly and reassembly protocol must be implemented, thus serving as a mandatory gateway through which data must pass, providing an unavoidable checkpoint for laying down immutable audit records.

Shamir's Secret Sharing (SSS) is a cryptographic method that divides a secret into multiple parts, known as shares. Note here the secret mentioned is filesystem info. To reconstruct the original secret, a predefined number of shares (threshold) are required. This ensures that no single party can access the secret without collaboration. Shamir's Secret Sharing (SSS) is considered to be information-theoretically secure, meaning that it provides a level of security that does not depend on computational assumptions. In SSS, a secret is divided into parts, and only when a sufficient number of these parts (threshold) are combined can the original secret be reconstructed. The security of SSS lies in the fact that any number of shares less than the threshold reveals no information about the secret, making it secure against any adversary with unlimited computational power, as long as the threshold condition is not met.

Shamir's Secret Sharing (SSS) is based on polynomial interpolation, specifically using Lagrange interpolation. The process is summarised in two main phases: **Share Distribution** (split) and **Secret Reconstruction** (reassembly).

Share Distribution

1. **Choose a Prime Number:** Select a large prime number p, which will define the finite field $\mathbb{F}_p$.

2. **Define the Secret:** Let S be the secret, where S is an element of $\mathbb{F}_p$.

3. **Generate a Polynomial:** Construct a polynomial $f(x)$ of degree $t-1$ (where t is the threshold number of shares needed to reconstruct the secret):

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1} \mod p$$

Here, $a_0 = S$ (the secret), and $a_1, a_2, \ldots, a_{t-1}$ are randomly chosen coefficients in $\mathbb{F}_p$.

4. **Create Shares:** For each participant i, compute a share (x_i, y_i) where $y_i = f(x_i) \mod p$ and x_i is a unique non-zero element of $\mathbb{F}_p$.

Secret Reconstruction

1. **Gather Threshold Shares:** Collect at least t shares (x_i, y_i).

2. **Use Lagrange Interpolation:** Compute the Lagrange basis polynomials $l_j(x)$ for $j = 1, 2, \ldots, t$:

$$l_j(x) = \prod_{\substack{1 \le m \le t \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \mod p$$

3. **Reconstruct the Secret:** Evaluate the polynomial at $x = 0$ to find S:

$$S = \sum_{j=1}^{t} y_j l_j(0) \mod p$$

This method ensures that the secret S is reconstructed only when at least t shares are combined, and any fewer than t shares reveal no information about the secret, maintaining its confidentiality.

Multi-signature (multi-sig) protocols are integrated with Shamir's Secret Sharing (SSS) to enhance the security and robustness of the secret sharing scheme by requiring multiple parties to agree before any critical actions, such as reconstructing the secret, is executed. Instead of allowing any individual who gathers the threshold number of shares to reconstruct the secret, the multi-sig setup requires signatures from multiple authorised parties to initiate the reconstruction process. This means that even if someone collects enough shares, they would still need the collaboration of other stakeholders to access the secret.

Signatures of the parties provided via their wallets

By distributing the control among multiple parties, the risk of a single point of failure or abuse is significantly reduced. Each participant holds only a part of the necessary credentials (signatures), and the secret can only be accessed when a predefined subset of these participants agrees and provides their signatures. This operates in a blockchain environment and so the transaction to reconstruct the secret is treated like any other transaction that requires validation. The multi-sig setup ensures that the transaction is only validated when the required number of signatures from the designated parties is present.

Different roles could be assigned to different parties involved in the multi-sig arrangement. For example, some might only be able to initiate a request for secret reconstruction, while others might be responsible for approving these requests. This role-based access control can further enhance the security and integrity of the secret management process. This is beyond the current feature set.

The PinkScorpion Protocol - State Transitions

Shamir's Distribution (splitting) and Reconstruction (reassembly) operations are modelled as state transitions and implemented on a blockchain. This is the means by which transitions between states occur, albeit with the intense processing required taking place off chain (see below). The filesystem's inherent blockchain transactions

represent the processes of data splitting and data reassembly. Each transaction is validated by public blockchain validators.

State Transition System - Splitting the Secret (Distribution)

- Initial State: The initial parachain state where the secret has not yet been split or distributed.
- Secret Splitting Transaction: A transaction type representing the splitting of the secret. This transaction takes the secret as input (in fact a reference to it as the secret itself shouldn't be on-chain) and generates N shares using (S^3), where a minimum of M shares (the threshold) is required to reconstruct the secret.
- State Transition for Splitting: The execution of the secret splitting transaction results in a state transition where the new state records the distribution of shares (commitments to these shares) to the respective storage locations.

State Transition System - Reassembling the Secret (Reconstruction)

- Reassembly Request Transaction: A transaction type for initiating the reassembly of the secret. This requires the host locations to submit their commitment to their shares.
- Verification and Reassembly: Upon receiving the required threshold number of shares, a state transition occurs that verifies the shares and reconstructs the secret. This involves off-chain workers for privacy and computational complexity concerns.
- Final State: The final state reflects the successful reassembly of the secret. In the future this may be used to trigger further actions on the parachain, such as unlocking assets, changing permissions, or initiating other state transitions.

Transaction Validation Process in Polkadot

What follows is a precise defintion of how the validation of the given state transitions occurs in respect of the actors involved.

- Let S represent the state of the blockchain.
- T represents a transaction that modifies the state.
- S' is the new state after applying the transaction T to state S.
- $V = \{v_1, v_2, \ldots, v_n\}$ is the set of validators responsible for validating the transaction.

- Each validator v_i has a corresponding weight w_i, which could represent their stake or voting power in the network.

Transaction Validation Process

1. **Transaction Proposal**:
   - A transaction T is proposed to change the state from S to S'.
   - T is broadcast to all validators in V.
2. **Validation by Each Validator**:
   - Each validator v_i checks whether the transaction T is valid under the current state S.
   - This involves executing the transaction logic and verifying that it adheres to the network's consensus rules enshrinned in the filesystem blockchain defintion.
   - This is represented as a validation function f:

$$f(S, T) = \begin{cases} \text{true} & \text{if } T \text{ is valid under } S \\ \text{false} & \text{otherwise} \end{cases}$$

The document is written as though the implementation follows the construction of a blockchain definition (parachain) but it may become apparent that a smart contract is the most expedient route.

3. **Collection of Validator Signatures**:
   - If $f(S, T) = \text{true}$, validator v_i signs the transaction.
   - The signature is represented as $\sigma_i(T)$, which is the signature of validator v_i on transaction T.
4. **Aggregation of Signatures**:
   - The signatures are collected and aggregated.
   - Let $\Sigma(T)$ represent the set of all valid signatures from validators who approved the transaction.
5. **Threshold Check**:
   - For the transaction T to be considered valid, the sum of the weights of the approving validators must meet a predefined threshold θ (defined for the blockchain).
   - The transaction is valid if:

$$\sum_{v_i \in \Sigma(T)} w_i \geq \theta$$

Parachain speeds are increasing all the time and as the document update is being made it should be noted that Aleph Zero a Layer1 blockchain with a polkadot bridge via a parachain slot has reached 1000 tps with 10000 tps in their sights

6. **State Transition**:
   - If the threshold is met, the state transition is applied, moving from S to S'.
   - This is represented as:

$$S \xrightarrow{T} S'$$

Multi-Sig Protocol for Collaborative Transactions

Here the focus is on a group different from the set of blockchain validators. This group comprises the parties interested in the integrity of the filesystem: the user, the design authority (org) and the platform.

Multi-Signature (Multi-Sig) protocols require multiple parties (signatories) to sign off on a file operation before it is executed. This adds a layer of security and consensus, ensuring that actions (like reconstructing a secret or authorising a state transition) receive approval from multiple stakeholders.

Parties Involved

- **fsUser**: The user of the filesystem who initiates the operation.
- **designAuthority**: The authority responsible for overseeing the design and integrity of the filesystem.
- **platform**: The underlying platform or infrastructure on which the filesystem operates.

Multi-Sig Protocol Description

The multi-sig protocol ensures that a transaction representing a state transition in the filesystem is only formed when all required parties agree to the operation. This is crucial for maintaining the security and integrity of the filesystem operations.

Precisely this is the multi-sig process:

Let T represent the transaction proposed to change the state of the filesystem. The transaction T must be signed by all required parties to be considered valid. Let $\sigma_{\text{fsUser}}(T)$, $\sigma_{\text{designAuthority}}(T)$, and $\sigma_{\text{platform}}(T)$ represent the signatures of the filesystem user, the design authority, and the platform, respectively.

$$\text{Valid}(T) = \begin{cases} \text{true} & \text{if } \sigma_{\text{fsUser}}(T) \land \sigma_{\text{designAuthority}}(T) \land \sigma_{\text{platform}}(T) \text{ are all present} \\ \text{false} & \text{otherwise} \end{cases}$$

Transaction Formation and Validation

For the transaction T to proceed:

1. Each party must independently sign the transaction, indicating their approval.

2. The signatures are collected and verified.
3. If all required signatures are verified as present and valid, the transaction T is approved to trigger the state transition.

$$T \text{ is executed if } \text{Valid}(T) = \text{true}$$

Validity is true allows the reconstruction to take place.

FUSE and NFS Filesystem APIs

This section outlines two conventional methods for designing a virtual filesystem that conforms to POSIX standards for the operating system. By supporting both APIs, the filesystem is integrated in two distinct and essential manners, each tailored to specific use cases.

Both FUSE (Filesystem in Userspace) and NFS (Network File System) protocols provide mechanisms that allow for the creation of custom file systems, though they serve different specific purposes and operate in distinct contexts.

FUSE (Filesystem in Userspace) Purpose: FUSE is designed to allow non-privileged users to create their own file systems without altering the OS kernel code. This is achieved by running the file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces.

Functionality: FUSE provides a framework where developers can implement a filesystem in user space, handling all the standard operations defined by the operating system's VFS (Virtual File System). This allows for the creation of highly customized file systems over various types of resources and backends like databases, cloud services, or custom storage mechanisms, without needing deep integration into the operating system kernel.

Customisation: It enables bespoke filesystem development that is tailored to specific needs or experimental filesystem features, which is developed and tested without risking system stability.

NFS (Network File System) Purpose: NFS, on the other hand, is used primarily for distributed file systems to allow a computer to access files over a network as easily as if they were on its local disks. This protocol defines a few operations for file handling and directory browsing which are implemented by a server allowing clients to access files remotely.

Functionality: NFS allows the user to mount part of a remote file

system on a local system, making it appear as if the files are local. This is crucial for environments where multiple systems need to access and share the same files.

Customisation: While NFS itself is less about creating entirely new types of file systems and more about extending the accessibility of existing file systems across networks, it is integrated into custom solutions that require remote file access. However similar to FUSE it is entirely possible to reconstruct the filesystem behind the NFS API. This IS the Lockular filesystem!

Both FUSE and NFS address the need to extend the functionality of traditional file systems, albeit in different ways. FUSE is more about creating entirely new file systems with unique behaviours or backing stores, while NFS is about extending the reach of an existing file system across a network with potential new backend implemetationsn.

Both is part of a broader strategy to handle files in a way that traditional local file systems might not support, such as in distributed, decentralized, or cloud-based architectures. FUSE allows the use case where the Lockular filesystem is more tightly integrated into the process offering an elevated security profile, but at a cost of less flexibility and a larger memory footprint.

Privacy and Security Considerations

Privacy: That the secret or individual shares are not exposed on the blockchain is ensured. Use of cryptographic commitments and zero-knowledge proofs are made as necessary. In the context of our use of Shamir's Secret Sharing (SSS) to create shares of filesystem data, where operations are controlled via a multi-signature (multi-sig) protocol, zero-knowledge proofs (ZKPs) are particularly useful:

- Identity Verification: ZKPs are used to prove that a party submitting a share is authorised to do so without revealing their actual identity or other sensitive credentials. This is crucial for the filesystem in maintaining privacy and security, ensuring that only eligible parties participate in the process.
- Share Integrity: When a share is submitted by an authorised party, ZKPs are used to prove that the share has not been tampered with and is indeed a valid part of the secret. This proof is therefore achieved without revealing the share itself, thus maintaining confidentiality.
- In the future ZKPs could be used to prove that the submission

adheres to the defined rules of the system (e.g., correct format, timely submission) without needing to expose the details of the share or the identity of the submitter.

In practical implementation terms means each party authorised to submit shares is registered in the system with specific permissions and roles defined via a multi-sig protocol:

- When a party attempts to submit a share, they must first prove their authorisation using a ZKP. This proof confirms their role and permission without revealing their identity or credentials. The party then submits the share, accompanied by a ZKP that validates the integrity and validity of the share itself. This ensures the share is correct and fits the expected parameters set by the secret sharing scheme.
- The filesystem, upon receiving a share submission, checks the ZKP for authorization and then verifies the ZKP for share integrity. If both proofs are valid, the share is accepted into the system for eventual secret reconstruction.

This approach ensures that only authorised parties can submit shares, protecting against unauthorised access and fraudulent submissions. The use of ZKPs maintains the privacy of the share and the anonymity of the submitter, crucial for security and compliance.

By using zero-knowledge proofs in this manner, we enhance both the security and privacy of the share submission process in the filesystem. ZKPs facilitate a robust verification mechanism that ensures compliance with system rules and integrity of the data, all while maintaining the necessary confidentiality and anonymity required in sensitive operations like secret sharing as used here.

To counter replay attacks several strategies are employed that ensure that even if a valid transaction or share submission is captured by an attacker, it cannot be reused maliciously. Here are key methods we use to prevent replay attacks:

- Unique Identifiers: Each transaction or share submission includes a nonce (a number used once) that is unique to that specific action. The system tracks all nonces used and rejects any transaction with a nonce that has been used previously.
- Timestamps: Additionally, timestamps are used to ensure the freshness of a transaction. The system can reject transactions that carry a timestamp outside of an acceptable time window.
- Track State Transitions: In the filesystem scheme the transaction that leads to a Shamir operation is a consequence of a control state

Timewindow is specifically mentioned in the PinkScorpion patent.

egine. Transactions that do not logically follow the current state (e.g., attempting to use a share that has already been used for reconstruction) are automatically rejected.

Shares Repositories

Redis is an excellent choice for storing shares resulting from the Shamir's Secret Sharing process due to several reasons:

- High Performance: Redis is an in-memory data structure store, which allows it to perform read and write operations at high speeds. This is crucial for operations that require quick access to data shares, such as those involved in cryptographic processes.
- Reliability: Redis provides persistence mechanisms that help ensure data is not lost even in the event of a system failure. This is achieved through point-in-time snapshots or append-only files (AOF).
- Replication: Redis supports master-slave replication, allowing data from the master server to be replicated to one or more slave servers. This helps in providing redundancy and increasing data availability, which is vital for maintaining the integrity and availability of shares in a distributed system. This is crucial in defending against a DDOS attack.
- Data Structure Variety: Redis supports various data structures such as strings, hashes, lists, sets, and sorted sets with range queries. This versatility is beneficial when implementing a filesystem.
- Atomic Operations: Redis supports atomic operations on complex data types, which is essential for ensuring data integrity during concurrent accesses and modifications.

Using Redis as a key-value store for the shares in a Shamir's Secret Sharing implementation provides a robust, scalable, and efficient solution for managing sensitive data in a secure and distributed environment.

Cassandra is also a good choice for managing high volumes of key-value data where rapid access and redundancy based on replication are required. Here are several reasons why Cassandra is well-suited to the provenance tracking fileystem regime:

- High Performance for Writes: Cassandra offers excellent write performance, which is a critical factor when dealing with high volumes of data. It achieves this through its log-structured merge-tree (LSM tree) storage mechanism, which is optimised for high write throughput.

- Scalability: Cassandra is designed to scale horizontally; you can increase capacity by adding more nodes to the cluster without downtime. This scalability is crucial for handling large volumes of data and user requests.
- Distributed Architecture: Data in Cassandra is distributed across multiple nodes in the cluster, which not only helps with load balancing but also ensures that there is no single point of failure. This distribution is key to achieving rapid access from different points in a network.
- Tunable Consistency: Cassandra allows you to choose the level of consistency you need for read and write operations. For scenarios requiring rapid access, you can opt for lower consistency levels to achieve faster response times. For operations requiring higher accuracy, you can choose stronger consistency levels.
- Data Redundancy: Cassandra automatically replicates data across multiple nodes. The replication factor is configured based on your redundancy needs. This replication is crucial for ensuring data availability and durability, even in the event of node failures.
- Decentralised Operation: There are no master nodes in Cassandra; all nodes are the same, which removes bottlenecks and single points of failure. This decentralised nature enhances the system's ability to handle large volumes of requests and data.
- Flexible Data Storage: While Cassandra excels at handling wide column storage, it can also effectively manage simple key-value pairs. This flexibility allows it to be used in various applications where rapid access to key-value data is necessary.

Robust Ecosystem and Community Support: Being an open-source project with strong backing and widespread use, Cassandra has a robust ecosystem and community. This support is invaluable for troubleshooting, optimisations, and learning best practices for deployment. Given these characteristics, Cassandra is well-equipped to handle applications that require managing large volumes of key-value data with requirements for rapid access and high availability through data replication.

Cassandra, unlike Redis, does not have the same type of size restrictions related to the volume of data stored, primarily because it does not rely on keeping all data in memory. For large data volume Cassandra is used by the Lockular filesystem as the storage engine for the shares.

Here are some key points regarding Cassandra's capacity and scalability:

- **Storage Capacity**
  Disk-Based Storage: Cassandra is designed to handle large amounts of data stored on disk. This allows it to manage data volumes that far exceed the size of the available RAM. The primary limitation in terms of data size is the disk space available across the nodes in the cluster.

- **Scalability**
  Horizontal Scaling: Cassandra scales horizontally by adding more nodes to the cluster. This means that as your data grows, you can expand your cluster to handle this increase. Each node in the cluster handles a portion of the data, distributing the load and storage requirements.

- **Performance Considerations**
  Read/Write Throughput: While Cassandra can handle large volumes of data, the configuration of the cluster (number of nodes, network setup, disk speed, etc.) will affect read and write performance. Proper tuning and hardware selection are crucial for maintaining high performance. Lockular's filesystem allows for tuning via a set of profiles.

- **Data Distribution**
  Partitioning: Similar to Redis, Cassandra automatically partitions data across the cluster using a partition key defined in the data schema. This distribution helps in managing large datasets efficiently but requires careful design to avoid hotspots where too much data or too many requests are directed at a single node. The filesystem design reflects a partioning schema that optimises the storage of file blocks.

- **Replication and Redundancy**
  Replication Factor: Cassandra replicates data across multiple nodes to ensure availability and fault tolerance. The replication factor is set according to the application's needs for redundancy. Higher replication factors increase data availability and durability but require more storage space and can impact write performance.

- **Maintenance**
  Compaction and Repair: As Cassandra is a disk-based system, it requires regular maintenance operations such as compaction (reorganizing data on disk to maintain performance) and repair (synchronizing data across replicas). These operations are essential for long-term performance and consistency but is resource-intensive.

- **Node Limits**
  Practical Limits: While theoretically, you can continue to add nodes to a Cassandra cluster, practical limits are imposed by management complexity, network bandwidth, and latency considerations. Very large clusters (hundreds of nodes) require careful planning and management.

In summary, Cassandra does not have inherent limitations on the volume of data it can handle, like Redis does with its memory-based storage. However, managing very large datasets in Cassandra requires careful planning regarding cluster configuration, node resources, and maintenance practices to ensure that the system remains performant and manageable. Lockular is able to advise on deployment configurations.

Technical Challenges and Solutions

Consider challenges across the architecture....

The biggest challenge to launch on to mainnet is obtaining a lease for a parachain slot. Work during 2023-2024 is seeing the introduction of a more granular approach labelled *CoreTime*. But first we will discuss leasing a Polkadot parachain slot. This involves participating in a parachain slot auction, a competitive and permissionless process designed to allocate slots on the Polkadot Relay Chain to various blockchain projects. Here's a brief overview of the process:

- Parachain Slots: Parachain slots are limited time slots available on the Polkadot Relay Chain where individual blockchain projects (parachains) can operate and benefit from the shared security and interoperability provided by Polkadot.
- Parachain Slot Auctions: To secure a parachain slot, projects must participate in an auction. These auctions use a candle auction format, which is a historically proven method where the exact ending time of the auction is not known to participants to prevent last-minute bidding wars.
- Crowdloans: Many projects fund their bids through crowdloans, gathering DOT (the native token of Polkadot) from supporters. In return, supporters are usually rewarded with the project's tokens. This allows projects to raise the necessary funds to bid in the auction without selling tokens directly at an early stage.
- Bidding Process: During the auction, projects place bids on how much DOT they are willing to lock up for the duration of the slot lease. The project that bids the highest amount of DOT will generally win the slot.

Parachain slot auctions are not that practical particularly when the application of parachain will expload

- Lease Duration: Parachain slots are leased for a specific period, not owned permanently. This period can vary but is divided into lease periods (each several months long). Projects can bid for multiple consecutive lease periods.
- Slot Renewal: As the lease period approaches its end, projects can participate in subsequent auctions to extend their lease on the parachain slot.
- Return of Locked DOT Once the lease period is over, the locked DOT is returned to the project or its crowdloan contributors, depending on the initial arrangement.
- Integration and Launch

After winning a slot, the project can integrate with the Polkadot network and begin operation as a parachain, leveraging the network's shared security and interoperability features. This process is crucial for projects seeking to become parachains on Polkadot, as securing a slot is necessary to enjoy the benefits of the Polkadot ecosystem. The competitive nature of the auctions ensures that slots are allocated to projects with substantial community support and readiness for deployment. This model can stiffle ubiquitous adaption of parachains.

A Remedy

Coretime is an innovative new feature (2023) designed to optimise the allocation and management of blockspace across the network, addressing some of the challenges associated with parachain slot auctions. Here's how Agile Coretime helps solve the bottleneck problem of Polkadot slot auctions:

- Flexible Blockspace Allocation: Agile Coretime allows projects to access the right amount of blockspace for every stage of their growth, which means they never overpay for blockspace. This flexibility helps new and smaller projects enter the ecosystem without the need for large upfront investments.
- Bulk and On-Demand Coretime: Projects can purchase coretime in bulk in advance, which is automatically renewed to prevent spiking fees during periods of high demand. Additionally, on-demand coretime is purchased in smaller amounts, removing entry barriers for newer projects or those with variable blockspace needs. Coretime slots may be minted as NFTs!
- Secondary Markets for Coretime: Coretime is traded on secondary markets, allowing projects to sell excess coretime or purchase additional coretime as needed. This market-driven approach helps

ensure that blockspace resources are used efficiently and are accessible to a wider range of projects.

- Predictable Pricing: By securing bulk coretime at a fixed price, projects can plan for long-term growth without worrying about fluctuating costs due to changes in demand for blockspace. This predictability is crucial for sustainable development and financial planning.
- Coretime Chain: The management of coretime will be centralised on the Coretime Chain, a dedicated marketplace for all things related to coretime. This specialised chain facilitates the buying, selling, and management of coretime, streamlining the process and making it more transparent.
- Integration and Accessibility: Coretime functions is accessed programmatically (e.g., over XCM or interacting with the Coretime Chain via Polkadot JS libraries), manually, or via user interfaces designed for this purpose. This accessibility ensures that projects of all sises and technical capabilities can effectively manage their coretime needs.

By introducing Agile Coretime, Polkadot is aiming to alleviate the competitive pressure of parachain slot auctions, making the ecosystem more accessible and efficient. This approach will not only supports the growth of individual projects like Lockular's filesystem but also it enhances the overall scalability and flexibility of the Polkadot network. Lockular's present filesystem implementation is focused on the use of the coretime approach to gain access to the polkadot validator capability.

Implementation

Implementing a filesystem protocol with state transitions modeled in a Polkadot parachain involves several key steps, leveraging the unique capabilities of the Polkadot network, such as shared security, interoperability, and the modular framework provided by Substrate (the development framework for building parachains). The steps involved are:

- Design the Filesystem Architecture:

  Define State and Transactions: Determine what constitutes the state of the filesystem (e.g., file metadata, access permissions) and the transactions that can alter this state (e.g., create, delete, modify files).

Allow for Identity Roles and Permissions: Define different roles within the system, such as users, administrators, and validators, and specify their permissions regarding file operations. (a future ambition).

- Develop the Substrate Runtime

  Set Up a Substrate Node: Set up a basic Substrate node containing the rich set of tools and libraries for developing custom blockchain logic.

  Implement Custom Pallets: Custom pallets (modules) for handling filesystem operations (FUSE and NFS). These pallets will contain the logic for state transitions, handling transactions, and enforcing rules.

  State Transition Functions: Implement functions to handle the state transitions based on the transactions received. This includes validation, execution, and committing changes to the state.

- Integrate Cryptographic Techniques

  Use of Cryptography: Apply zero-knowledge proofs to ensure only valid actors take part in the Shamir protocol.

  Shamir's Secret Sharing: Implement features like secure data recovery and multi-party control, integrated with Shamir's Secret Sharing.

- Ensure Consensus and Security

  Validator Setup: Define how validators are chosen, their roles in the network, and how they participate in the consensus process. Hopefully this is intrinsic based on the parachain chosen.

  Consensus Mechanism: Adapt the Substrate's inherent consensus mechanisms (such as GRANDPA and BABE) to fit the needs of your filesystem, ensuring that state transitions are agreed upon and finalised securely. Again careful choice of parachain and implementing as smart contract will mean this is inherited.

- Deployment on Polkadot

  Parachain Development: Prepare the Substrate-based blockchain to become a parachain or a parathread in the Polkadot network.

  Parachain Slot Auction: Participate in a parachain slot auction to secure a slot on the Polkadot Relay Chain, or opt for a parathread if intermittent connectivity suffices. or use Aleph Zero.

- Launch and Iteration

Mainnet Launch: After successful testing and securing a parachain slot, launch the filesystem on the Polkadot mainnet.

Continuous Improvement: Monitor the performance and user feedback to continuously improve the filesystem, add new features, and address any emerging security concerns.

Off-chain workers in Substrate are a powerful feature that allows blockchain nodes to perform intensive or complex computations, or interact with external data sources, without burdening the blockchain's main transaction processing capabilities. This significantly enhances the efficiency and capabilities of a blockchain system, where the Lockular filesystem is modeled as a Polkadot parachain. Here's how off-chain workers are incorporated into the implementation steps of such the filesystem:

Integration with Off-chain Workers

- Step 2: Develop the Substrate Runtime

  Implement Off-chain Workers: As part of developing custom pallets to support the Shamir Secret Sharing approach in a multi-sig context, off-chain workers are used to handle operations that are computationally intensive.

- Step 3: Integrate Cryptographic Techniques

- Data Processing: Use off-chain workers are to perform cryptographic operations that are too heavy for on-chain execution, such as generating cryptographic proofs (e.g., zero-knowledge proofs) required to authenticate Shamir shares.

- Step 4: Ensure Consensus and Security

- Data Validation: Off-chain workers can fetch and validate data from external sources before it is used in on-chain logic. This is particularly useful for ensuring the integrity and authenticity of data being incorporated into the filesystem.

- Step 6: Deployment on Polkadot

- Data Handling at Scale: When deploying the filesystem, use off-chain workers to handle scale-related tasks such as batch processing of file operations or managing large datasets, which are impractical to handle directly on-chain due to gas costs and performance constraints.

  Benefits of Off-chain Workers in Filesystem Implementation

- Efficiency: Off-chain workers can process transactions and perform computations without consuming the blockchain's resources, leading to more efficient use of computational power and faster transaction processing.
- Scalability: By offloading heavy computations and data handling to off-chain workers, the filesystem can scale more effectively, handling more complex operations or larger datasets than would be feasible if all processing were done on-chain.
- Flexibility: Off-chain workers can interact with external APIs and data sources, bringing a greater range of capabilities to the filesystem, such as integrating real-time data or interfacing with other systems and protocols.
- Cost-Effectiveness: Performing operations off-chain can reduce the cost associated with blockchain transactions, which is particularly important for operations that require high computational bandwidth or frequent data updates.

Incorporating off-chain workers into the development and operation of a filesystem on a Polkadot parachain significantly enhances its performance, capabilities, and user experience, making it a more robust and versatile solution.

Cost analysis of running the Provenance Tracking Filesystem

TODO: Translate the cost in USD based on storage and validator fees.

Case Studies and Examples

TODO: Explain the Secure Design Services cloud environemnt.

Conclusion and Roadmap

The paper discussed the implementation and technical details of a Provenance Tracking Filesystem designed to operate on a blockchain network, specifically using Polkadot's parachain technology. It emphasises the importance of maintaining immutable provenance in respect of data stored on a filesystem where the filesystem is core in critical infrastructure design.