CFS: A Multi-Credential POSIX-Compliant File System for Secure and Verifiable Data Storage

CS 262a Project #1, by: Qingyang Hu, Yucheng Huang, and Manshi Yang

Problem Statement

- DataCapsule's **single-writer** semantics lack flexibility and scalability, which limits the scalability of the Global Data Plane.
- There is a lack of a **Common Access API** (**CAAPI**) on the client side for file system, which requires developers to have a comprehensive understanding of DataCapsule.
- In need of a **multi-credential** file system that supports multiple writers and implements read/write provenance, while still guaranteeing data security through a cryptographic approach.

DataCapsule Background

- Standardized metadata wrapped around opaque data transactions.
- Every transaction explicitly sequenced in a hash chain history and is append-only.
- Each is uniquely named and globally findable.
- Resembles a "blockchain in a box" structure.

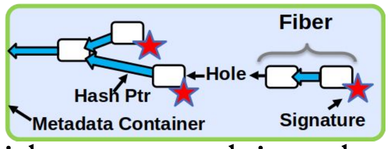

Figure 1: Data capsule structure

Performance

- Our performance is about 10x slower than NFS without network latency.
- Write is comparably slow due to synchronous and sequential requests.
- Expected to get better performance with caching, journaling, and QUIC.
- Latency breakdown and application benchmark will be added later on.

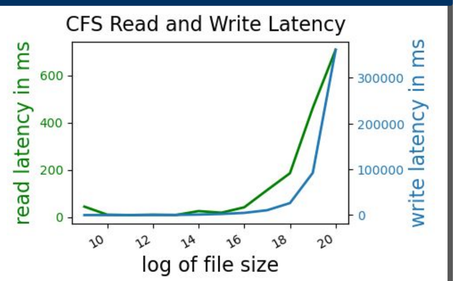

Figure 4: read/write latency w/ different file size

	Read	Write
Mean	1.52ns	151.21ms
Median	1.44ns	104.17ms

Table 1: read/write per-block latency

Architecture

DataCapsule Server

- Store and serve DataCapsule blocks.
- Handle read requests from clients and write requests from middlewares.

Middleware

- Enforce client **write permission** through ACL and signatures.
- Verify, sign and forward writes request from **multiple clients**.
- Use Trusted Execution Environment.

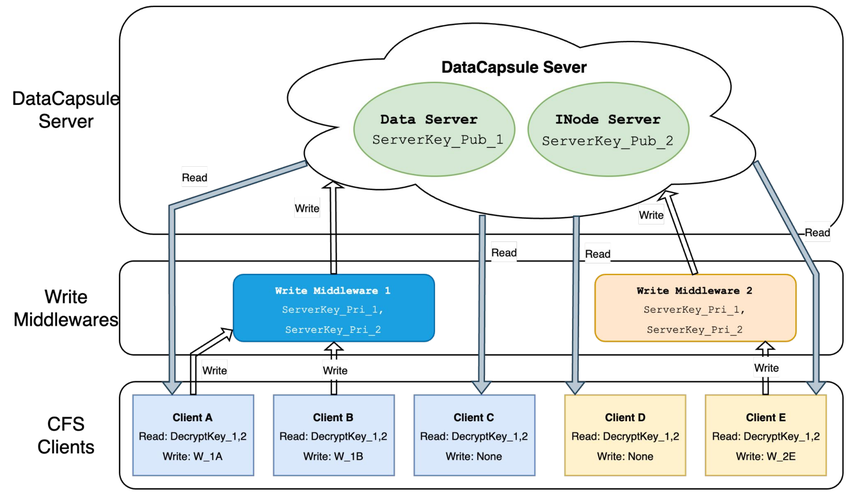

Figure 2: CFS high-level architecture

CFS Client

- Use **FUSE** (Filesystem in Userspace) to provide a **POSIX-compliant** interface.
- Reconstruct local INode information and serve data from two separate DataCapsules.
- Implement configurable **LRU cache** of blocks to improve read performance.
- Support **journaling** for **batched requests** to improve write performance and **crash recovery**.
- Represent **user identities** using a combination of the client's public key and their local UID.
- **IAM** provided using permission bits and enforced by operating system and middleware.

Capsule Block Design

INode Block Capsule

- ✓ ACL with crypto-graphic signatures.
- ✓ Timestamp for snapshots and conflicts resolution
- ✓ Structure to represent filesystem hierarchy
- ✓ File/Folder metadata
- ✓ Hashes of data blocks

Data Block Capsule

- ✓ Configurable size

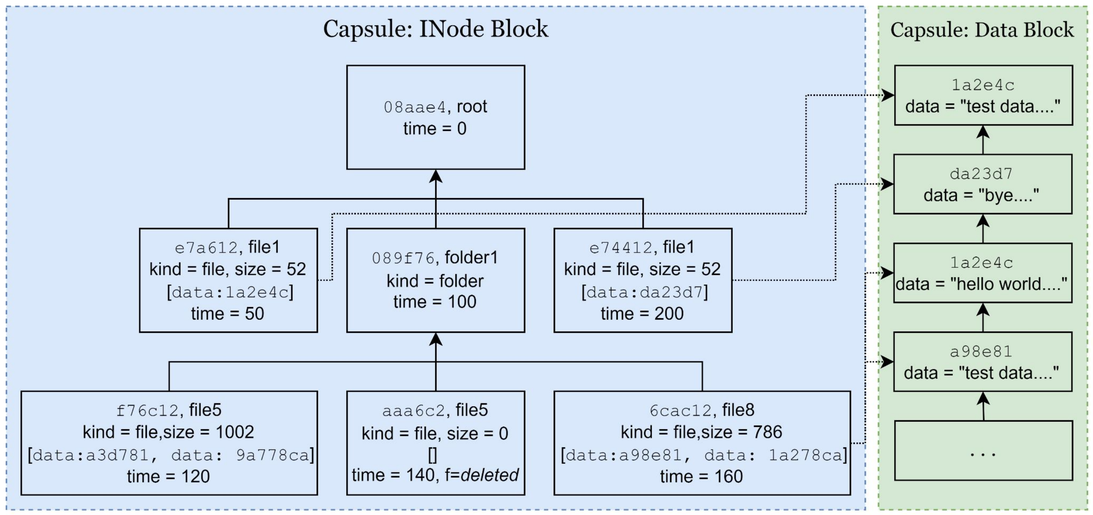

Figure 3: CFS detailed block structure