SNIA DEVELOPER CONFERENCE
SDC 25
By Developers FOR Developers

Hyatt Regency Santa Clara, CA
September 15-17, 2025

Scalable Metadata in Distributed File Systems

Revisiting the GoogleFS Design for Exabyte-Scale Namespaces

Leil Storage
Luis Guillermo Silva Rojas
Piotr Modrzyk

www.sniadeveloper.org

Speakers

Piotr Modrzyk

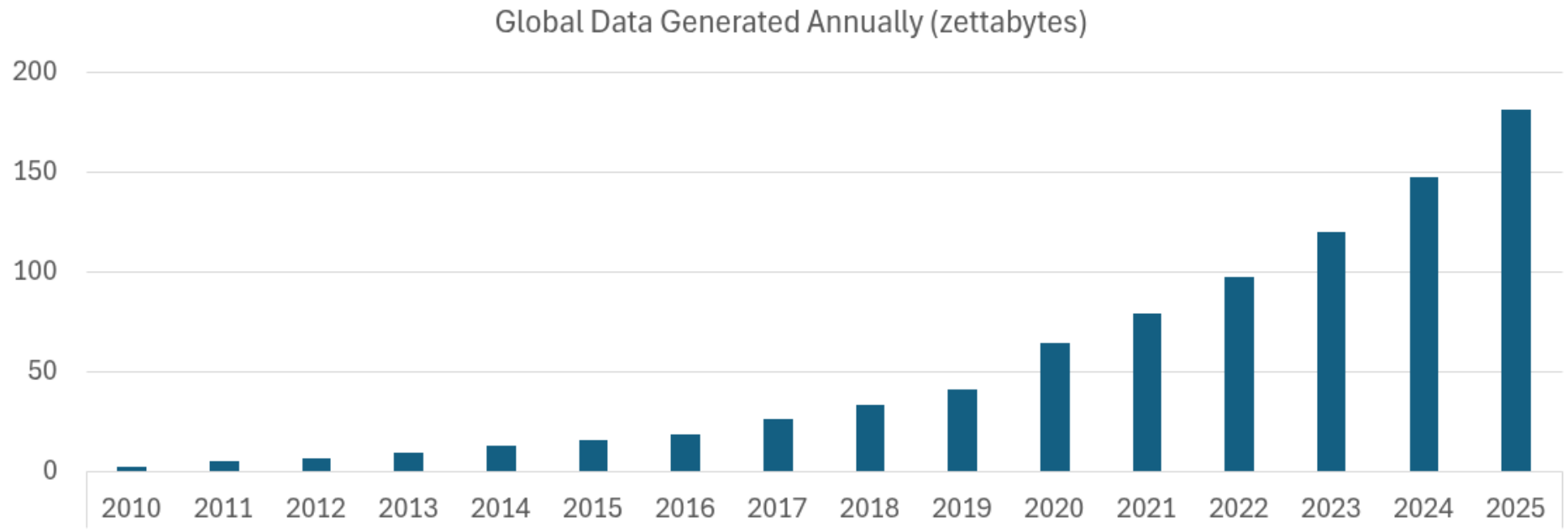
Principal Architect at Leil Storage
and SaunaFS

Outline

- Introduction
- GFS and SaunaFS architecture overview
- Limitations for big namespaces
- Related work
- Solution
- Implementation details in SaunaFS
- Results
- Conclusions

Introduction: Data Growth & Scaling Challenges

- Explosive growth: AI datasets, media, simulations, IoT.
- Clusters at thousands of nodes, exabyte capacities.
- Need for reliable, efficient data access.



Introduction: The Critical Bottleneck

- Metadata describes files, directories, access patterns.
- Every file operation depends on metadata speed.
- At massive scale, metadata often limits performance & availability.

OPERATIONS (CURRENT)

#	HOST	IP	MOUNTED PATH	STATFS	GETATTR	SETATTR	LOOKUP	MKDIR	RMDIR	SYMLINK	READLINK	MKNOD	UNLINK	RENAME	LINK	READDIR	OPEN	READ	WRITE	TOTAL
1	192.168.25.207	192.168.25.207	/	2	3293	0	603563	0	0	0	0	283128	0	0	0	2	283128	0	283128	1456244

OPERATIONS (LAST HOUR)

#	HOST	IP	MOUNTED PATH	STATFS	GETATTR	SETATTR	LOOKUP	MKDIR	RMDIR	SYMLINK	READLINK	MKNOD	UNLINK	RENAME	LINK	READDIR	OPEN	READ	WRITE	TOTAL
1	192.168.25.207	192.168.25.207	/	1	2721	0	488758	1004	0	0	0	176051	0	0	0	0	176051	0	176051	1020637

Introduction: Building Future-Proof File Systems

- Low-latency, scalable metadata services.
- Massive namespace support, fault tolerance.
- Key to exabyte-scale storage & collaborative, always-on workloads.

Google File System (GFS): Basics

- Originally published in 2003.
- Built for Scale on Commodity Hardware.
- Large, Fixed-Size Chunks.
- Optimized for Appends and Streaming Reads.
- Replication for Fault Tolerance.
- Relaxed Consistency Model.
- Single active Master for Simplicity.

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance,

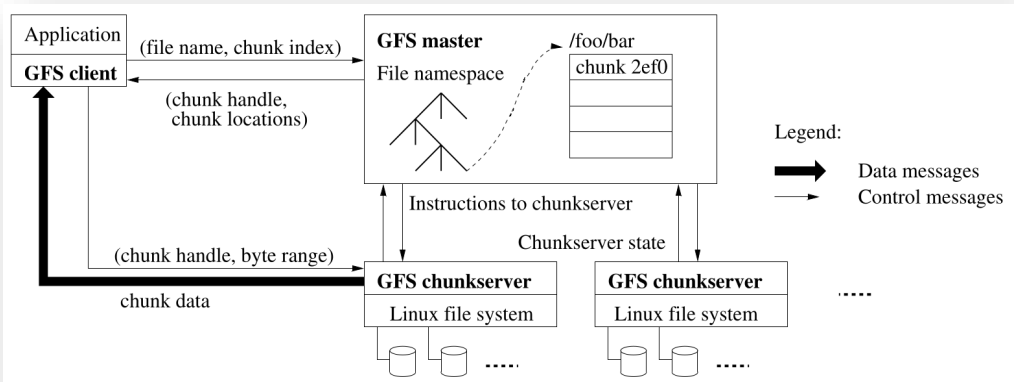


Figure 1: GFS Architecture

Google File System (GFS): Influence

Open source

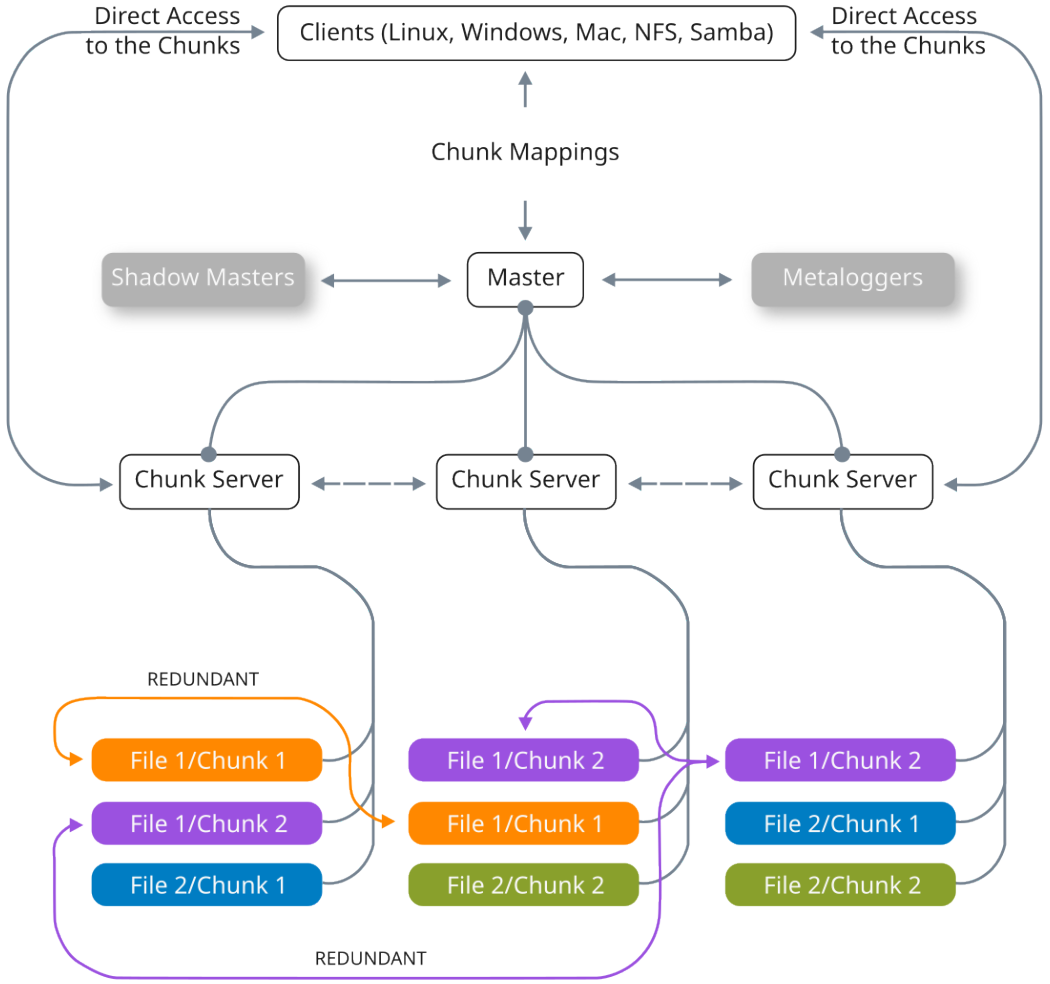
- Hadoop Distributed File System (HDFS)
- BeeGFS (FraunhoferFS) (2007)
- Lustre
- MooseFS (2008)
- LizardFS (2013)
- **SaunaFS (2023)**
- ...and more

Proprietary

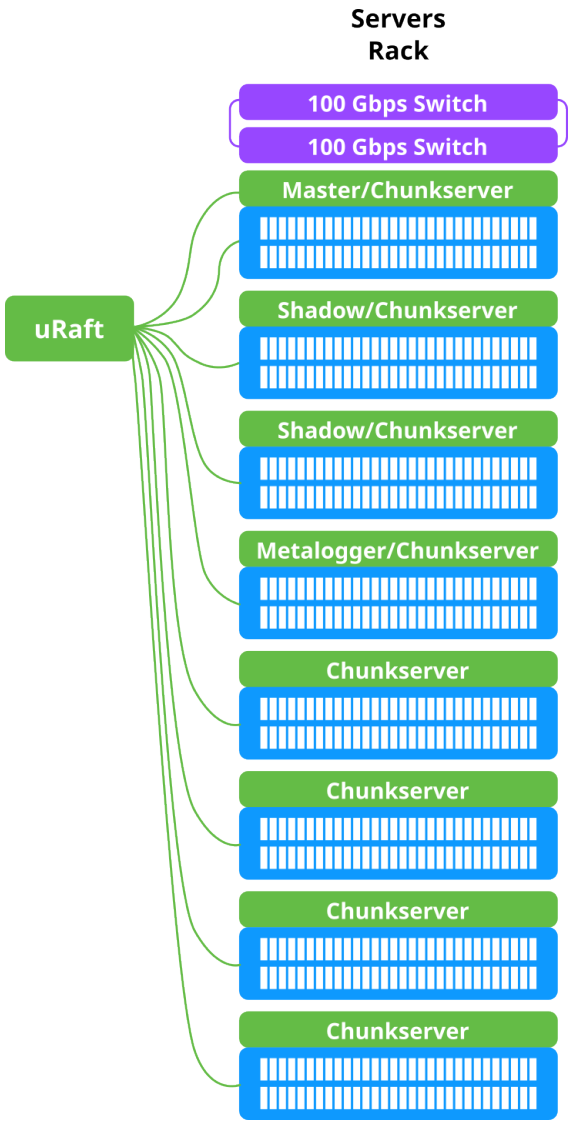
- Google Colossus
- Microsoft Cosmos
- Facebook Tectonic
- Alibaba Pangu

SaunaFS - Great-grandson of GoogleFS

- Mostly open source.
- Maintained by Leil Storage.
- Shares most of the concepts introduced by GFS.
- Presents unique capabilities like native support for Host-Managed Shingled Magnetic Recording drives (SMRs are mixable with CMR).
- Pluggable architecture for Chunkservers.
- Infinite Cold Engine (ICE) for energy efficiency.
- HA module based on RAFT (uRaft).



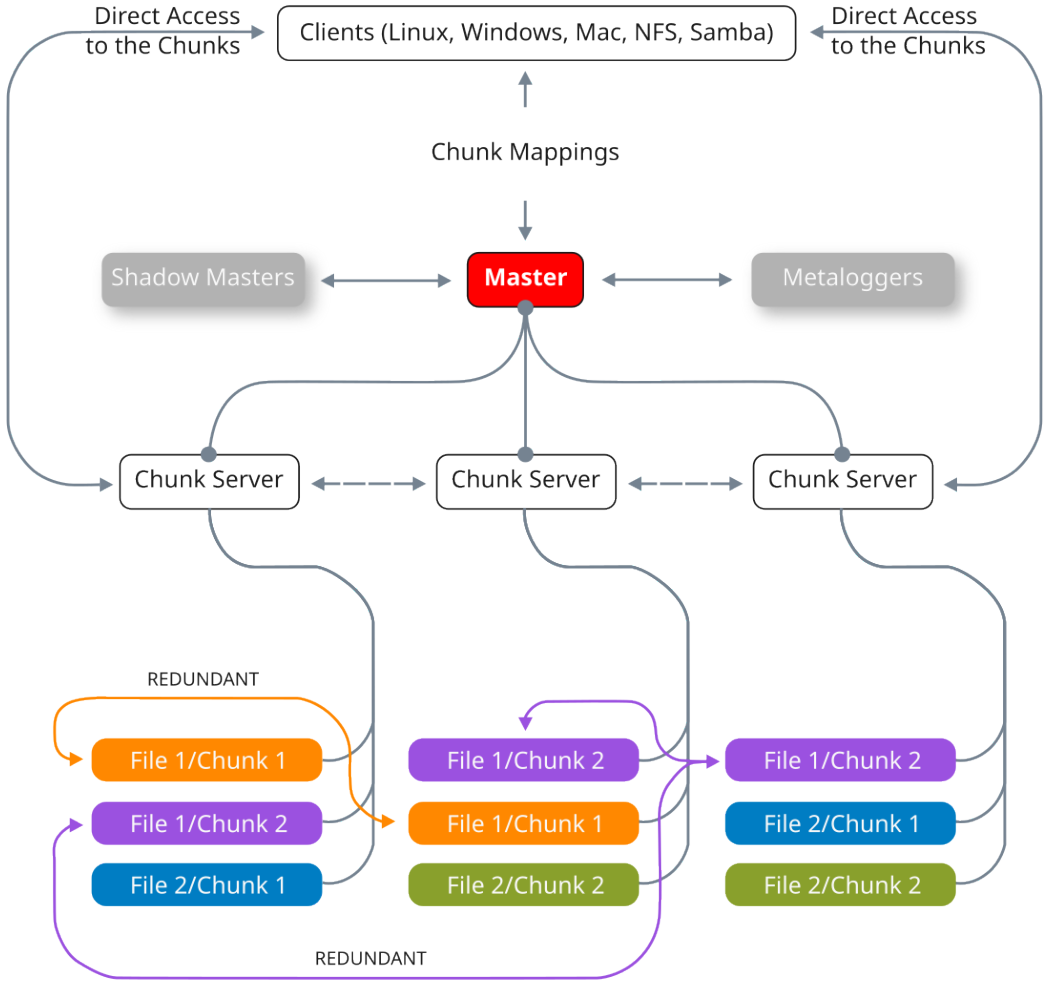
SaunaFS – Example Installation



- 8 server nodes, 60 or 102 high-capacity drives per node.
- Erasure Coding with 6 data parts and 2 parities, internally known as EC(6,2) or simply EC62.
- Using 36 TB drives, a **single rack** could have about **28 PB** of space.
- Depending on the average file size, the amount of metadata could represent a bottleneck for a single active Master.

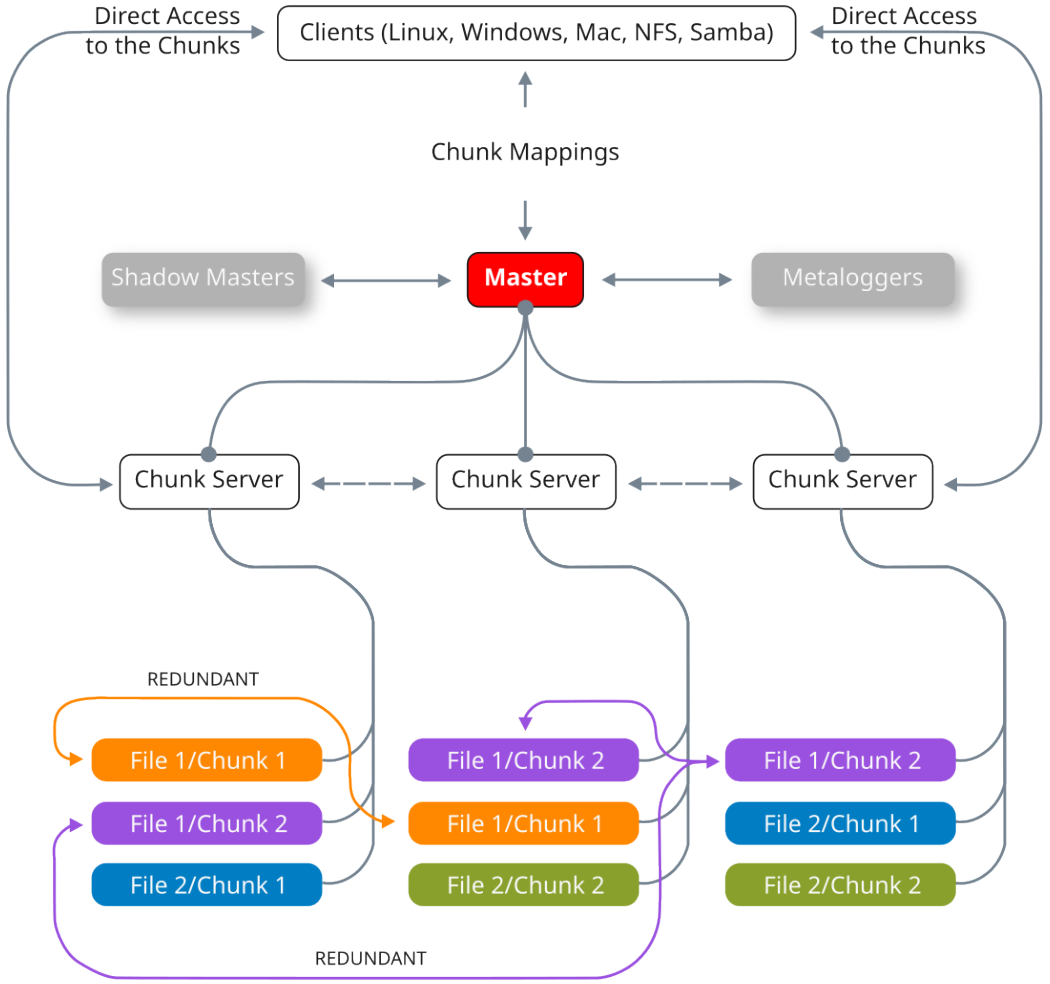
SaunaFS Metadata Limitations

- The full metadata structures are in-memory for fast access.
- If the active Master fails, uRaft promotes a Shadow immediately, but:
  - The Chunkservers need to **register the Chunks** (billions).
  - The clients need to **reconnect** the sessions.
- Metadata dumping process uses **fork**, which also limits multithreading.



SaunaFS Metadata Limitations

- In-memory metadata causes long loading times for large namespaces (minutes)
- Failover events introduce downtime during master recovery
- Adding more nodes does **not** improve metadata performance (limited horizontal scalability)
- Master overload risk increases with more clients
- Single namespace size constrained by server RAM capacity



Problem Statement

- Single active master = simpler design.
- But it limits:
  - Availability – risk of downtime during failures.
  - Scalability – cannot handle growing workloads efficiently.
  - Metadata capacity – constrained by one server's resources.

- Problem: Not suitable for very large, highly available distributed file systems.

Objectives

- **Eliminate the single-MDS bottleneck** with a distributed metadata service.
- **Ensure continuous availability** even during node failures.
- **Achieve horizontal scalability** in metadata throughput (IOPS) & capacity.
- **Support massive namespaces** for very large clusters.

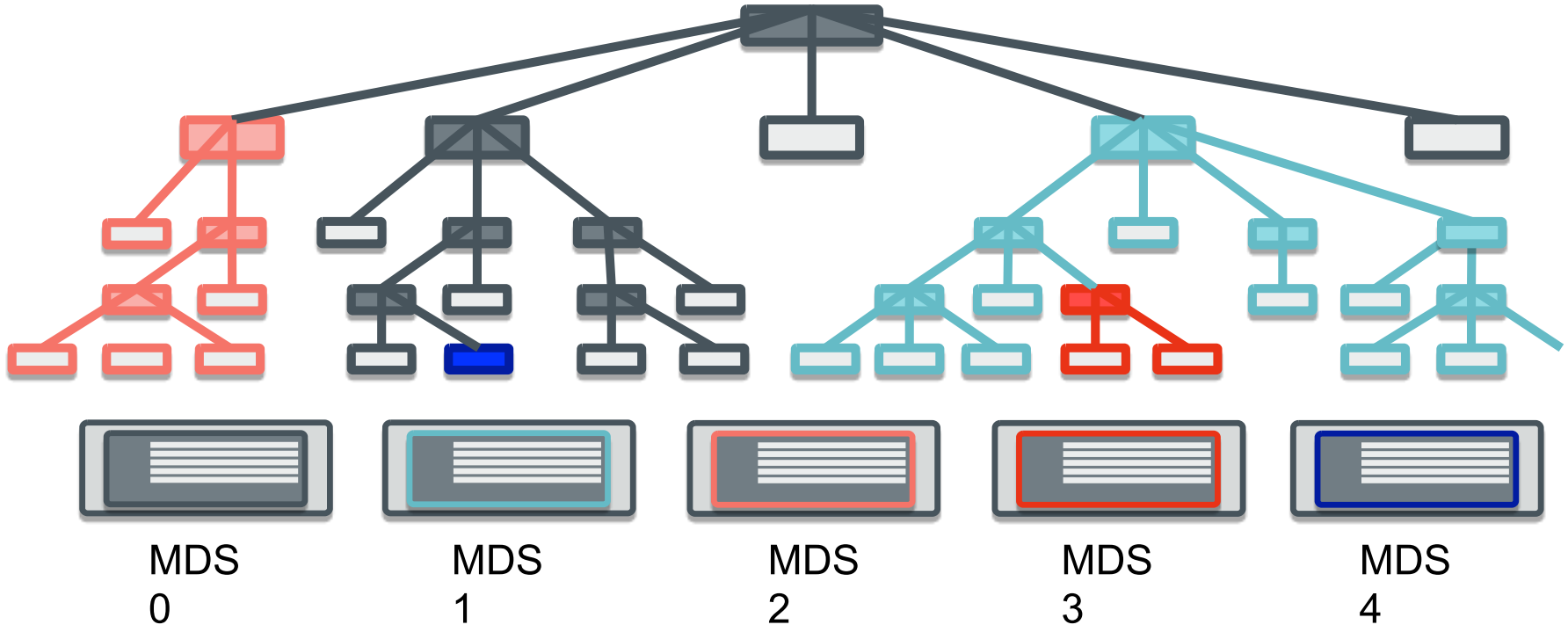
Related Work

- Shard/Partition/Hash
  - CephFS
  - Lustre
  - HDFS Federated
- Local filesystem
  - BeeGFS
  - Lustre + ZFS/DMU
- Databases
  - Google Colossus (Spanner)
  - Microsoft Cosmos (SQL Server Hekaton)
  - Facebook Tectonic (ZippyDB)
  - Deepseek 3FS (FoundationDB)
  - JuiceFS (TiKV, FoundationDB, PostgreSQL, MySQL, etc., ...)
  - CERN EOS (QuarkDB)

*Categories are not exclusive

Related Work: CephFS

Dynamic Subtree Partitioning

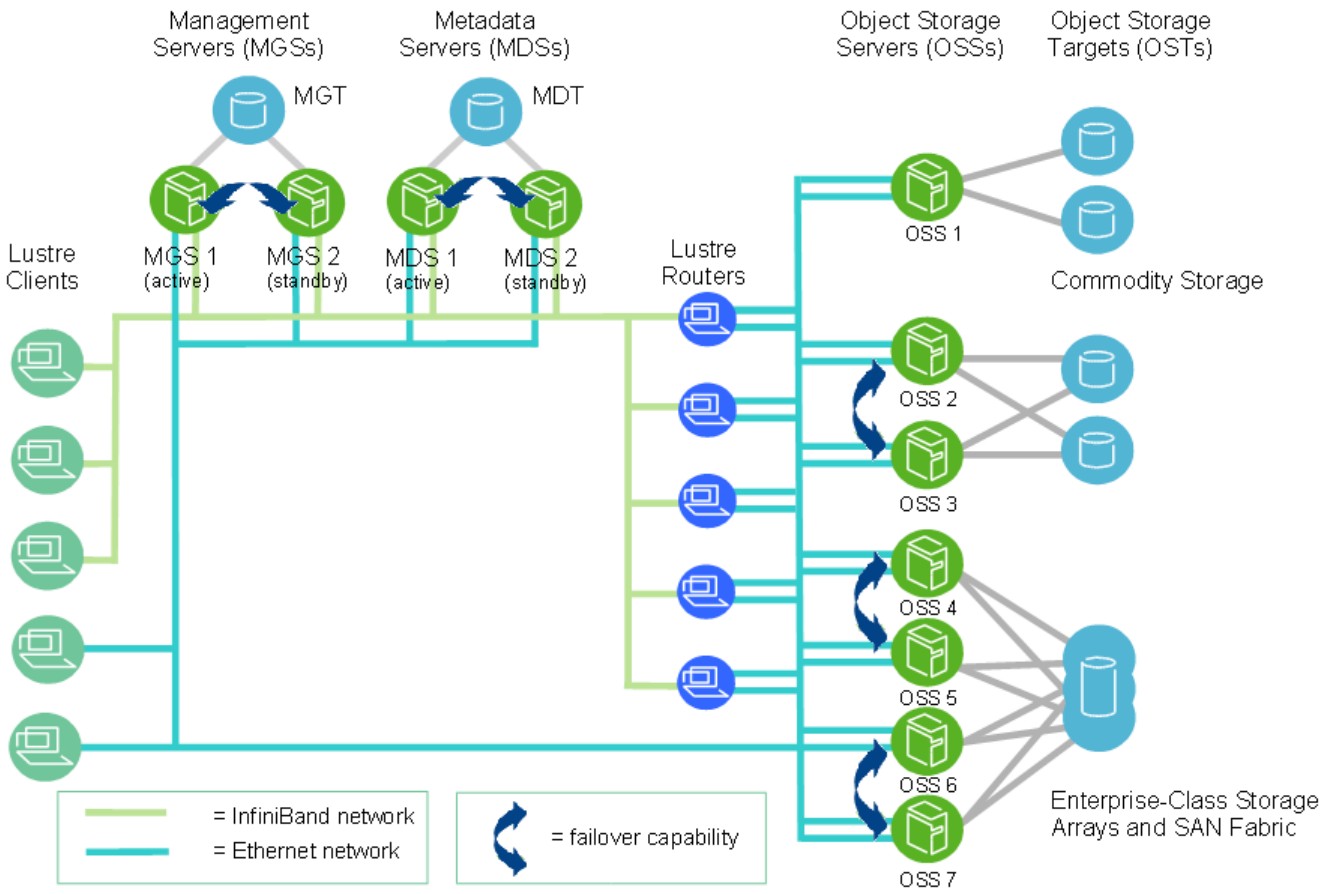


In this file system hierarchy tree, the shaded nodes represent directories and unshaded nodes files. The subtrees are partitioned in such a way that the metadata for the grey subtree is handled by MDS 0, the light blue subtree by MDS 1, the orange subtree by MDS 2, the red subtree by MDS 3 and the dark blue subtree(which is just a single directory) by MDS 4.

https://docs.ceph.com/en/reef/cephfs/dynamic-metadata-management/

Related Work: Lustre

Distributed Namespace Environment (DNE)

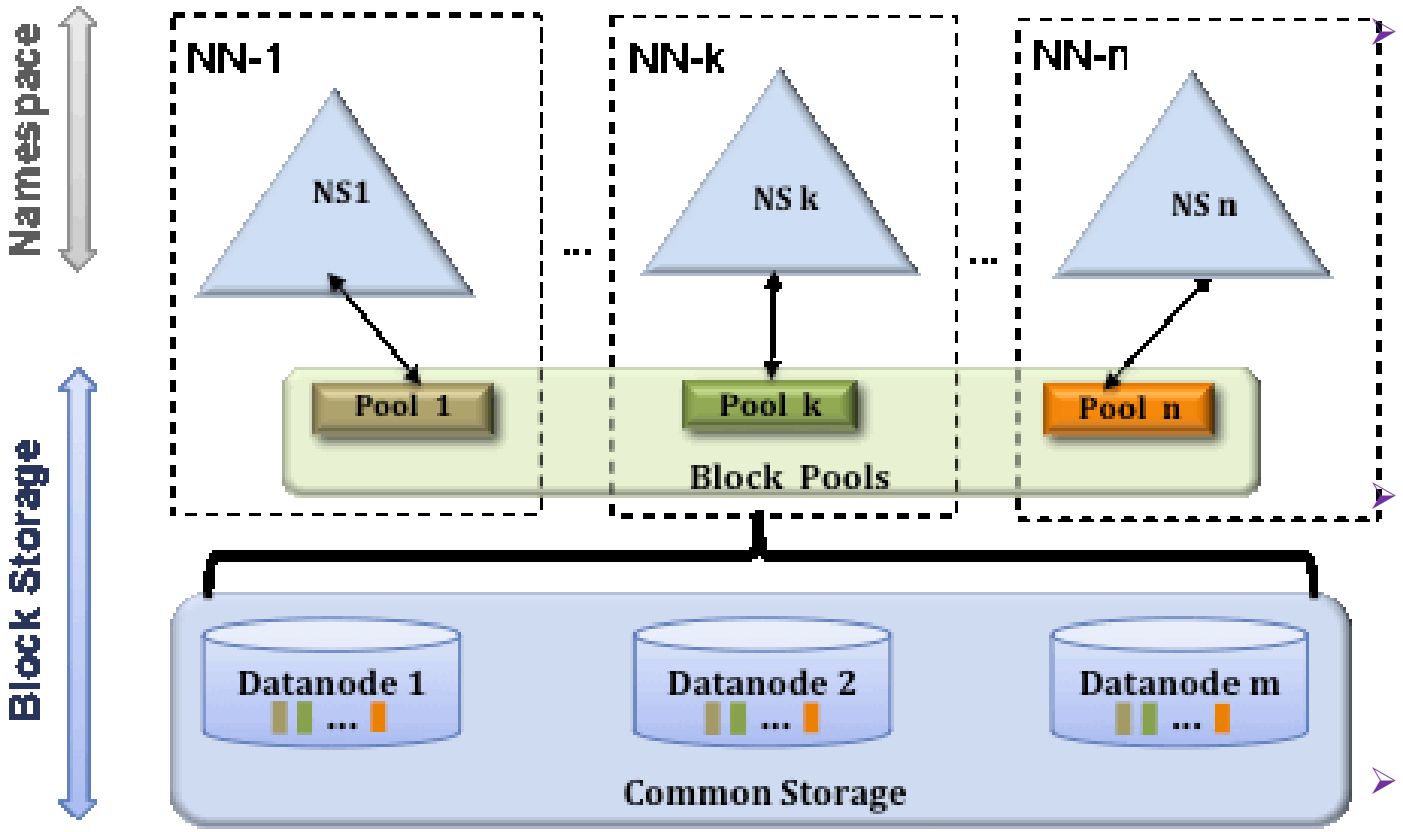


- Metadata is distributed across the metadata nodes (MDTs) on a per directory basis.
- The content of a directory is always stored on one metadata node.
- A metadata node is chosen manually or can be balanced by automatic/manual striping.
- Uses a unique approach, inode on top of ZFS via DMU (Data Management Unit).

https://doc.lustre.org/lustre_manual.xhtml , https://wiki.lustre.org/ZFS_OSD

Related Work: HDFS Federated

Manually Partitioning the Namespace for Horizontal Scalability

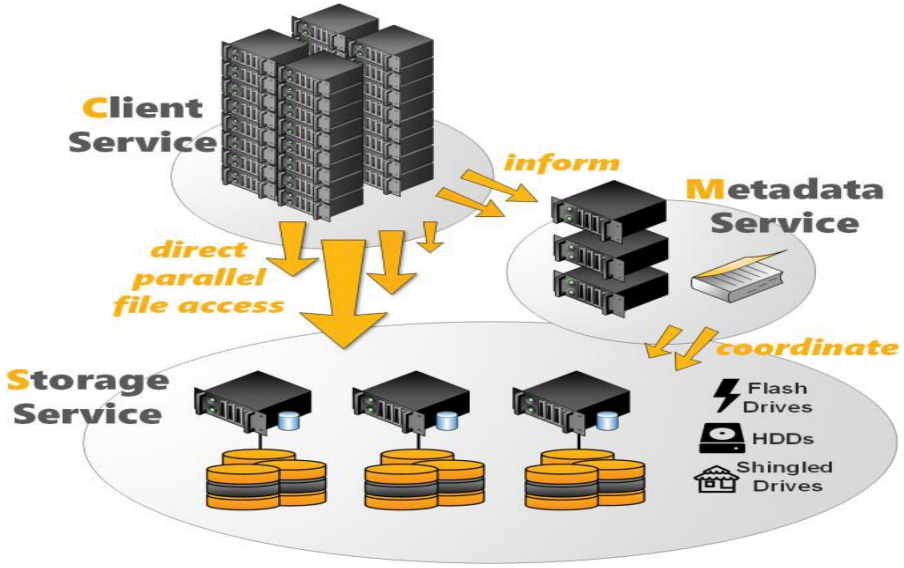


- Metadata is distributed across the metadata nodes (NameNodes) on a per directory basis (Name Spaces).
- The content of a directory is always stored on one metadata node.
- A metadata node is chosen manually.

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/Federation.html

Related work: BeeGFS

Per-Directory Distribution and Local File System Storage



- Metadata is distributed across the metadata nodes on a per directory basis.
- The content of a directory is always stored on one metadata node.
- A metadata node is chosen at random for each directory.
- Uses a unique approach, creating a file per inode on top of local Linux file systems (extended attributes).

https://doc.beegfs.io/latest/architecture/overview.html

Related Work

Database-Centric approaches to distributed filesystem metadata

- Shifts the complexity of consistency, replication, and fault tolerance away from the filesystem itself onto a specialized, battle-tested external service.
- MDSs act as a lightweight, scalable API/protocol gateway.
- Translates filesystem operations into database transactions.
- Could maintain a hot cache for performance, but the database is the ultimate source of truth.
- Can be scaled out horizontally to handle more client connections.

Related Work

Database-Centric approaches to distributed filesystem metadata

- Manages the "Hard Problems":
  - **Transactions (ACID)**: Ensures atomic operations, making complex actions like renaming a large directory safe and consistent.
  - **Consistency**: Provides strong consistency guarantees across the entire namespace.
  - **Replication & HA**: Manages data replication for high availability and disaster recovery.
  - **Sharding**: Automatically partitions and distributes the data across database nodes to scale.

Solution

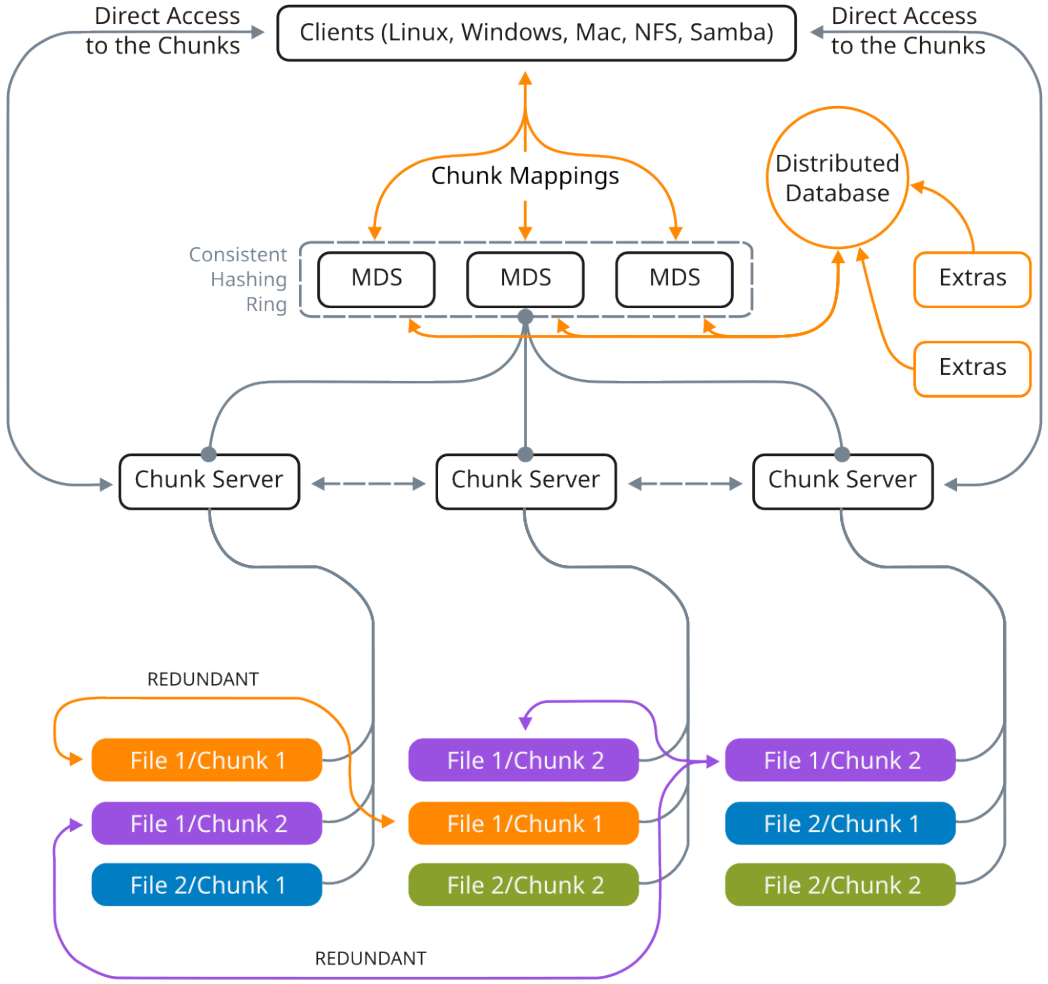
New multi-active MDS architecture on top of distributed database with **ACID** transactions.

Atomicity (mkdir foo && cd foo)

Consistency (mkdir foo creates '.' & '..')

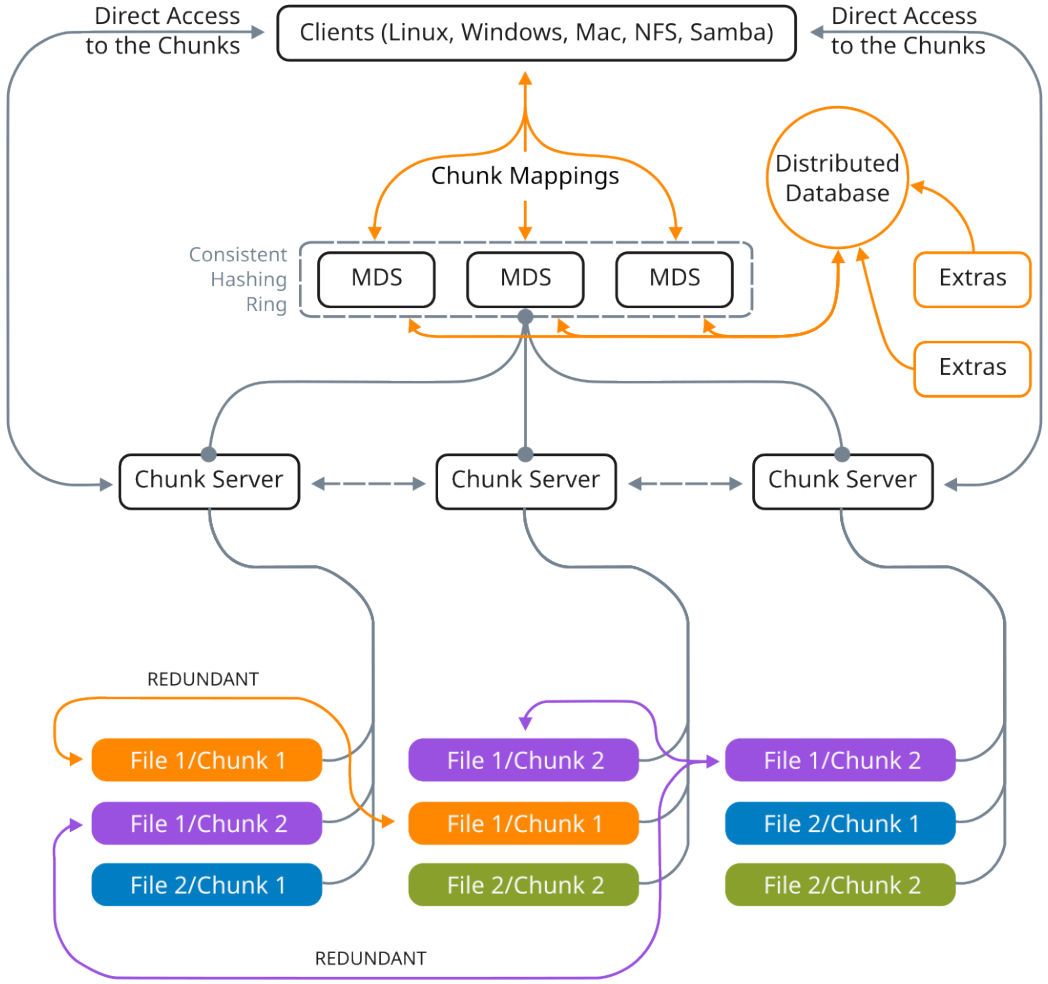
Isolation (mkdir foo & mkdir foo)

Durability (sync)



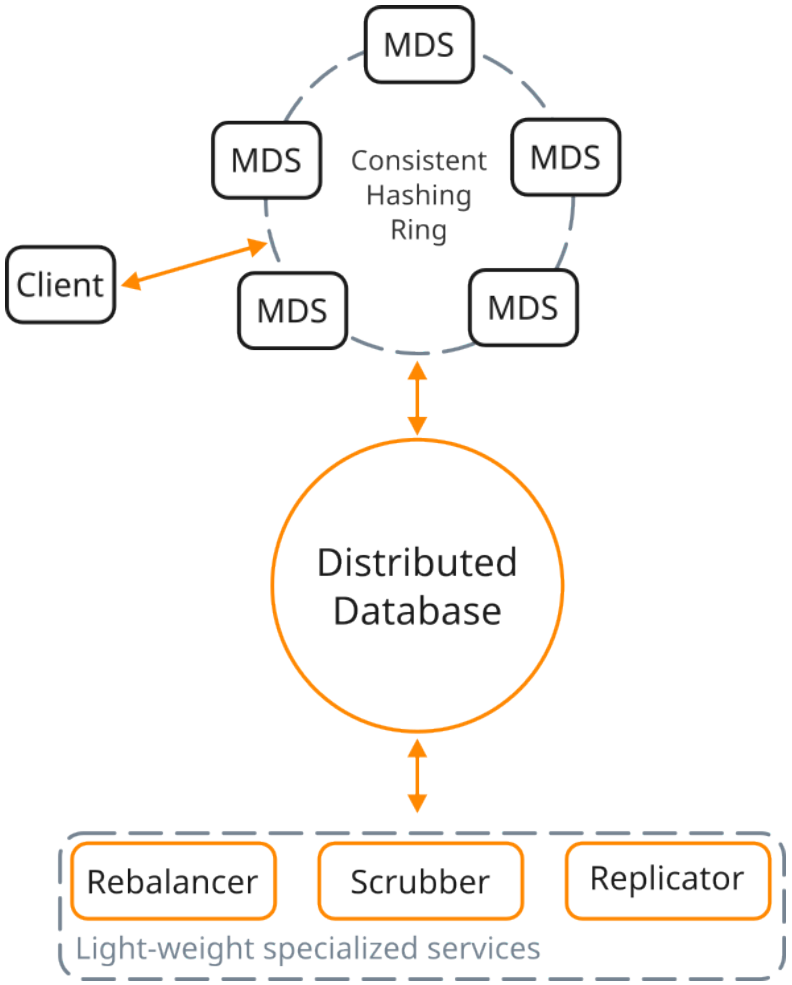
Solution

- Single source of trust.
- Increased Scalability for metadata operations (by adding nodes).
- No Downtime during handover from Master to ~~Shadow~~.
- No forking needed, which means MDSs can now be multi-threaded without quirks.
- No need for time consuming Chunk re-registration.
- Light-weight stateless extras (replication, rebalancing, etc.).



Solution

One Ring to rule them all...



- The information about the Ring members is stored in the database.
- MDSs use leases or heartbeats to keep their entry alive in the database.
- Any MDS can provide the information about the Ring.
- MDSs don't own the data, they are *responsible for handling operations for specific subsets of the namespace*.
- The hashing function deterministically maps any given file or directory to a specific MDS.

Distributed Databases

SQL & NewSQL

- CockroachDB
- **TiDB**
- YugabyteDB
- **MySQL**
- …

NoSQL

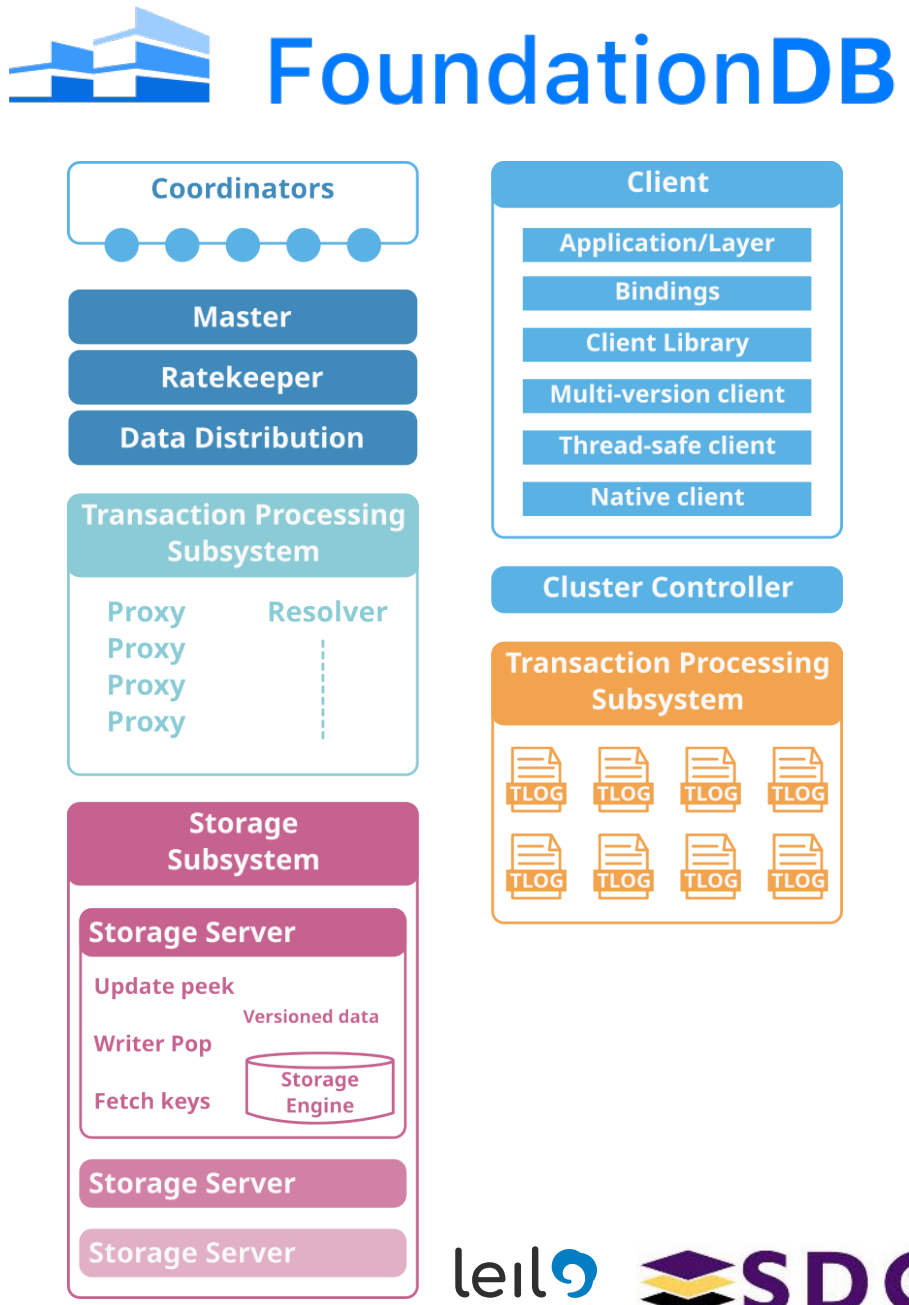
- MongoDB
- Couchbase
- Apache Cassandra
- ScyllaDB
- …

Key-Value

- **FoudationDB**
- DynamoDB
- …

FoundationDB

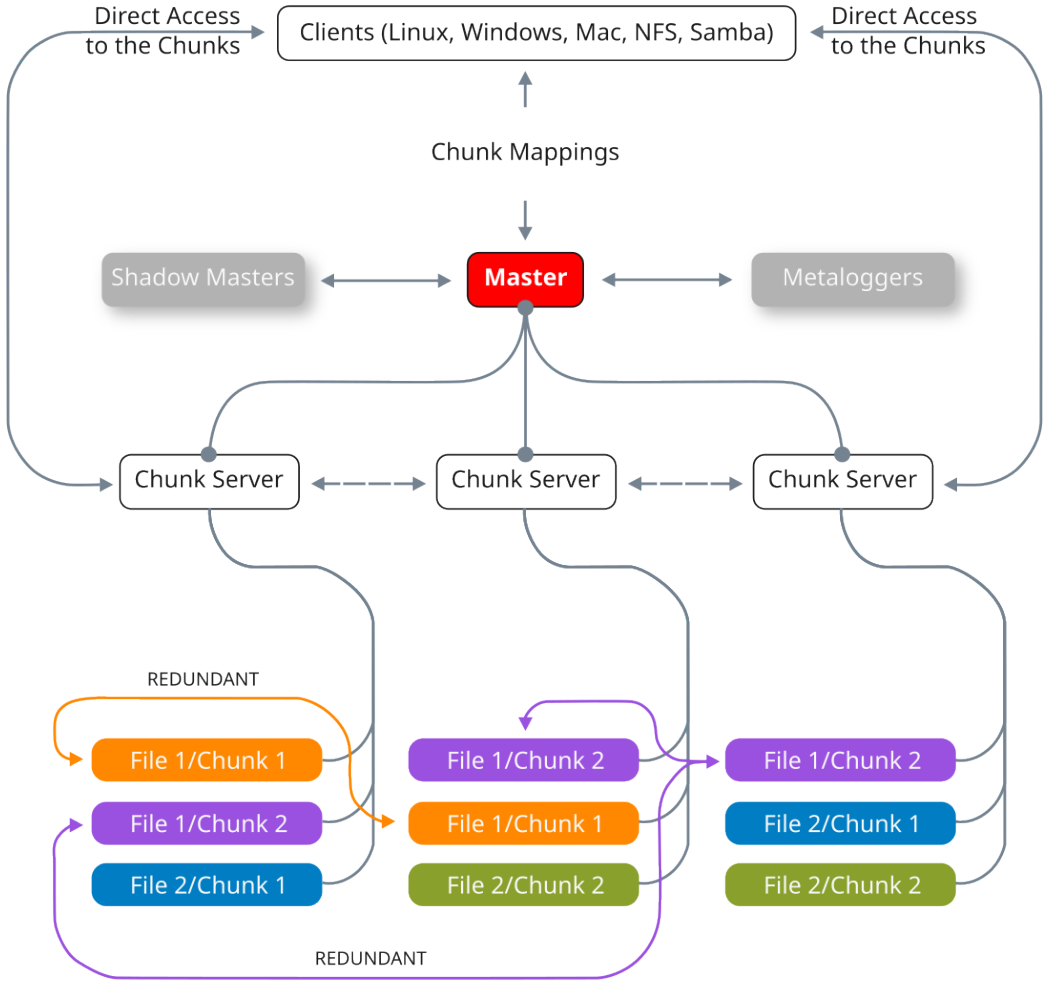
- FoundationDB is an **open-source**, **distributed**, **transactional** key–value store designed for **strong consistency**, high **fault tolerance**, and **multi-model** layers on top of a single core database.
- All data is safely stored, distributed, and replicated in the Key-Value Store component.



Current Metadata Implementation

SaunaFS

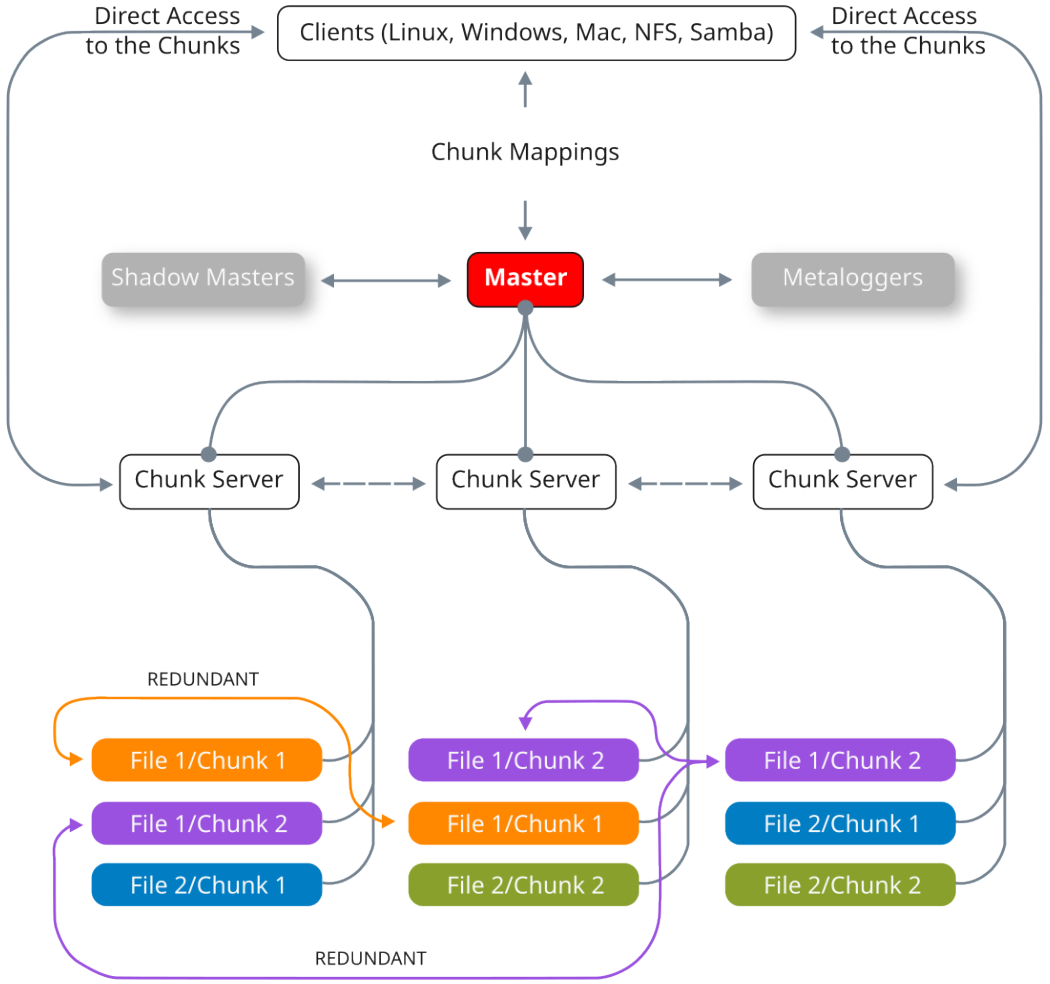
- The original implementation was tied to the file **metadata.sfs**, with the following sections:
  - HEADER
  - NODE 1.0
  - EDGE 1.0
  - FREE 1.0
  - XATR 1.0
  - ACLS 1.2
  - QUOT 1.1
  - FLCK 1.0
  - CHNK 1.0
  - EOF



Current Metadata Implementation

SaunaFS

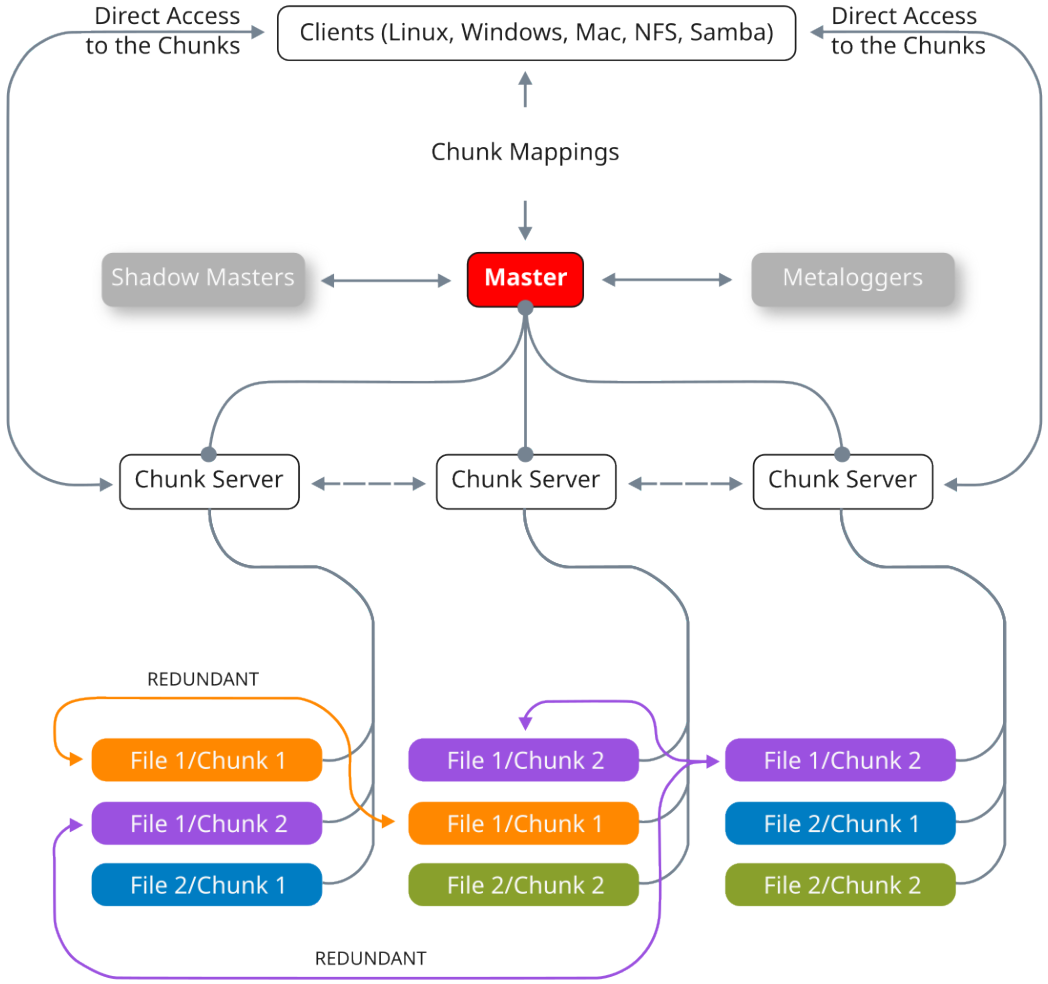
- Metadata servers (Master, Shadow and Metalogger) generates changelogs (journals) on mutable operations.
- On startup Master server reads the metadata.sfs file and applies the changelogs if needed.
- Then dumps the new metadata to persistent storage and sends/syncs it to Shadows and Metaloggers.



Current Metadata Implementation

SaunaFS

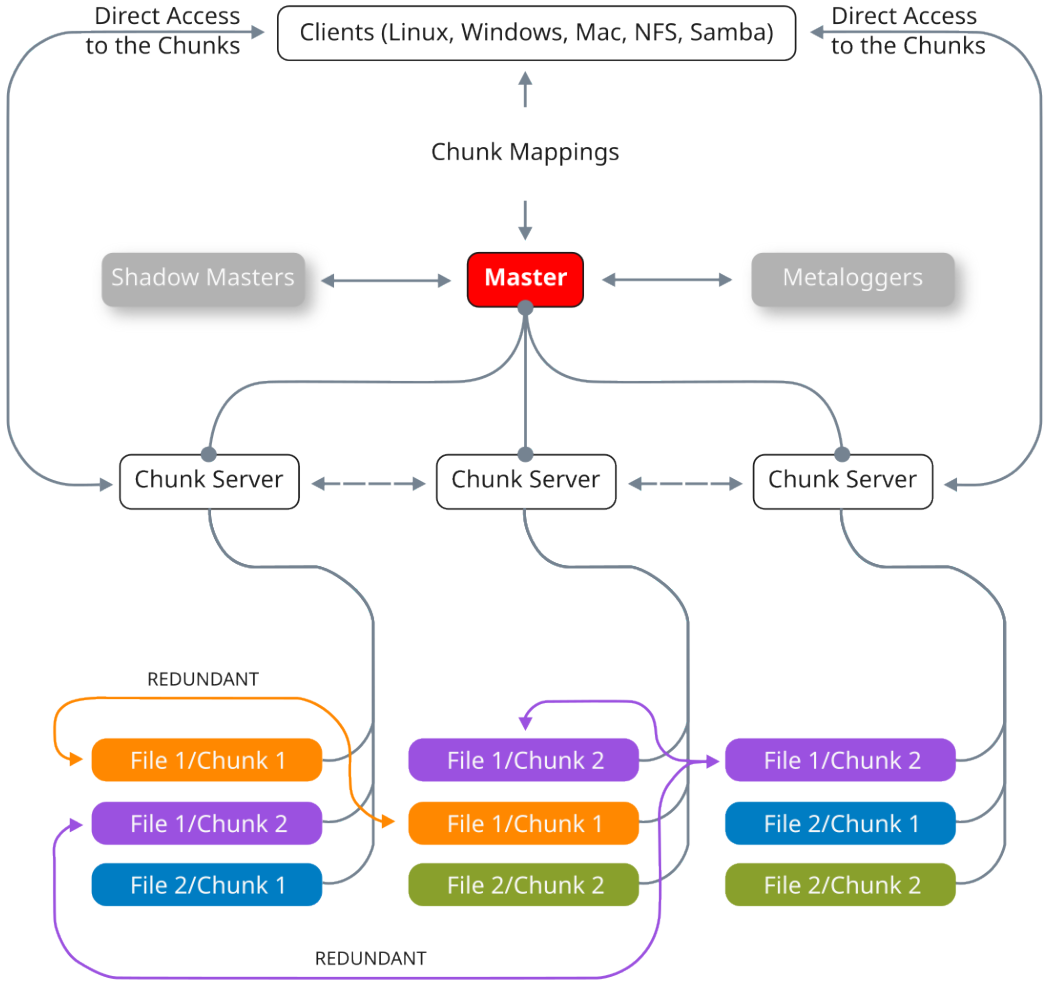
- Mutable operations are broadcasted to Shadows and Metaloggers.
- Every hour (default), MDS **forks** and dumps the new metadata.sfs.
- The forking process temporarily duplicates the RAM usage.



Current Metadata Implementation

SaunaFS

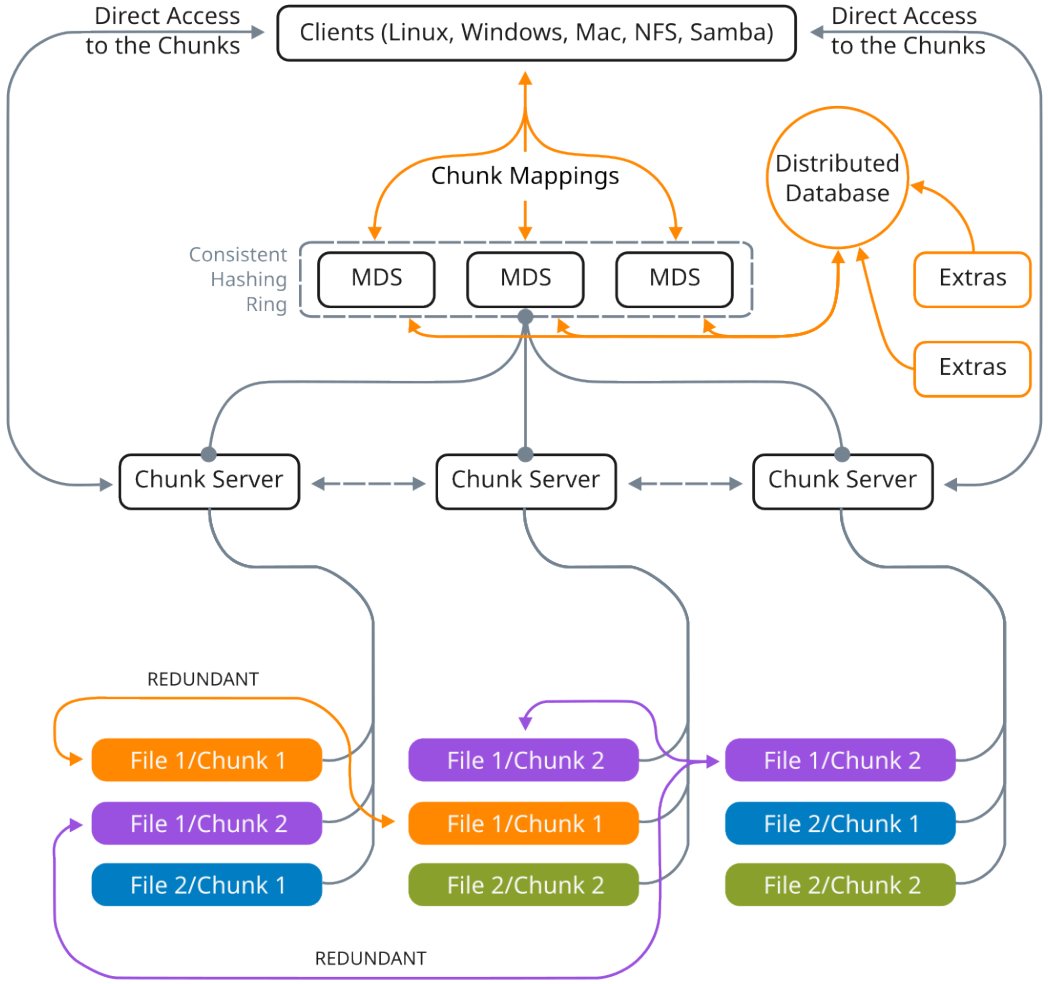
- Some integral variables are also synchronized between metadata server processes:
  - Metadata version (incremental).
  - Max inode id (incremental*).
  - Next session id (incremental*).
  - Number of nodes, directories, links, etc..
  - Metadata checksums.



New Metadata Implementation

SaunaFS

- Provide extension points for metadata related backends.
  - IMetadataBackend
    - MetadataBackendFile (metadata.sfs)
    - MetadataBackendKV
      - MetadataBackendFDB
      - Other backends
  - IMetadataDumper
    - File (with forking)
    - FDB (dummy to keep compatibility)

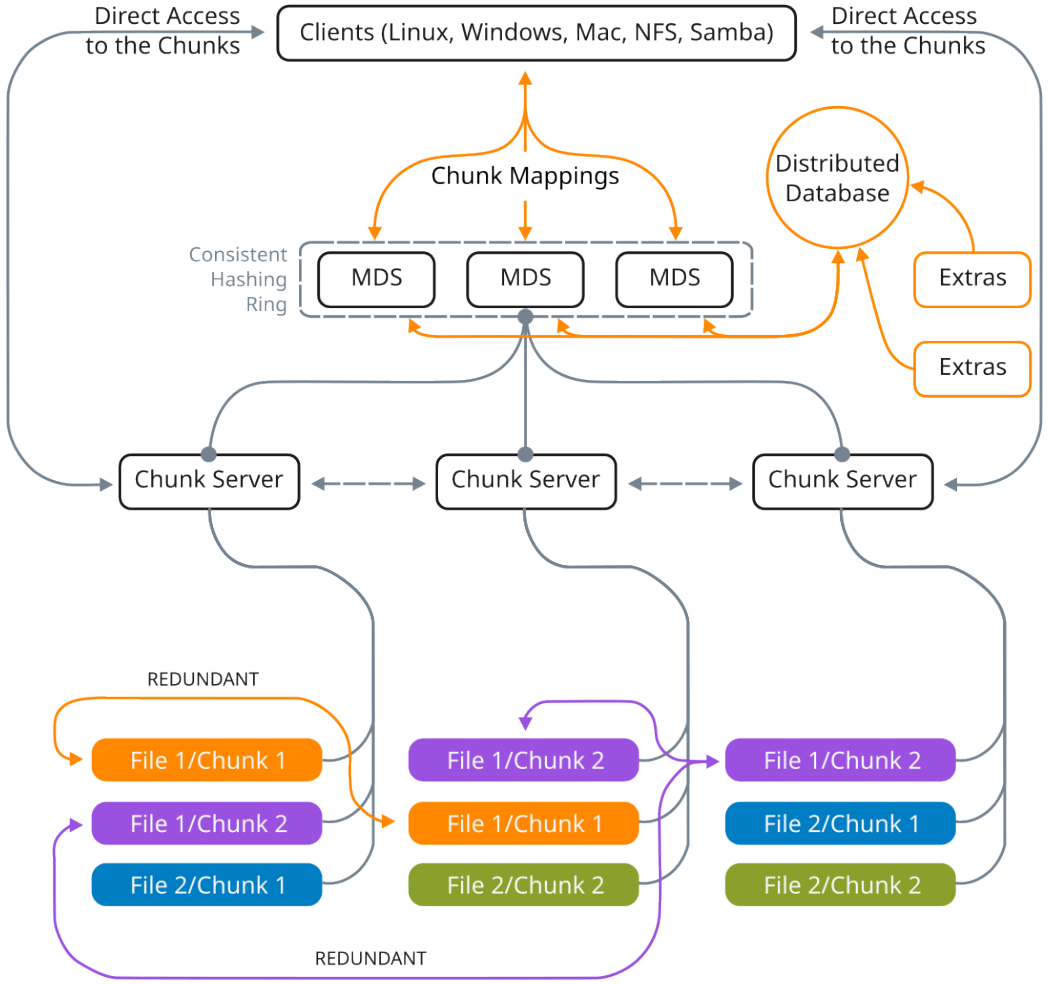


New Metadata Implementation

SaunaFS

- Provide an extensible abstract infrastructure based on transactions.
  - Key-Value engine library based on transactions.
    - IKVEngine
    - ITransaction

The abstractions allows to easily change the underlying database (KV) by implementing the new interfaces.



New Metadata Implementation

SaunaFS

- Provide an event driven approach to trigger transactions on selected metadata properties.
  - Signals-Slots based observer pattern.
    - Signal.
    - ObservableProperty.
    - ObservableIntegralProperty.

This way allows to react on property changes without affecting the existing production code (file based).

```
gMetadata->nextSessionId().connect([this](uint32_t oldSessionId, uint32_t newSessionId) {
    (void)oldSessionId;  // Unused parameter

    auto transaction = kvEngine_->createReadWriteTransaction();
    kv::Key sessionKey{kv::toU8Vector(gMetadata->nextSessionId().getName())};
    kv::Value sessionValue;
    serialize(sessionValue, newSessionId);
    transaction->set(sessionKey, sessionValue);

    if (!transaction->commit()) {
        safs::log_err("Failed to store session ID: {}", newSessionId);
    }
});
```

Example of connection to the Property nextSessionId.

leil STORAGE SDC 25

New Metadata Implementation

SaunaFS

- Provide testing framework support for FDB backends.
  - On-demand FDB cluster configuration and start on integration tests execution.
  - Automatic cluster cleaning on test end.
  - Shared cluster-file among main components.

```
CHUNKSERVERS=1 \
    METADATA_BACKEND=FDB \
    USE_RAMDISK=YES \
    MOUNT_EXTRA_CONFIG="sfscachemode=NEVER" \
    CHUNKSERVER_EXTRA_CONFIG="GARBAGE_COLLECTION_FREQ_MS = 0|HDD_TEST_FREQ = 100000" \
    SFSEXPORTS_EXTRA_OPTIONS="allcanchangequota,ignoregid" \
    setup_local_empty_saunafs info

cd "${info[mount0]}"

metadata_generate_all
```

Selecting FDB as metadata backend for specific tests.

leil9 STORAGE SDC 25

Database Schema

SaunaFS

Keys are sorted lexicographically based on their byte representation.

- `META_`
- `NODE_<INODE>:<SERIALIZED_FSNode>`
- `EDGE_`
- `FREE_`
- `XATR_`
- `ACLS_`
- `QUOT_`
- `FLCK_`
- `CHNK_`

This way, keys are logically separated and grouped, to be efficiently retrieved using ranges.

```
Range limited to 1000 keys
`NODE_\x00\x00\x00\x01' is `d\x00\x00\x00\x01\x01\x01\xff\x00
`NODE_\x00\x00\x00\x02' is `f\x00\x00\x00\x02\x01\x01\xb4\x00
`NODE_\x00\x00\x00\x03' is `f\x00\x00\x00\x03\x01\x01\xb4\x00
`NODE_\x00\x00\x00\x04' is `f\x00\x00\x00\x04\x01\x01\xb4\x00
`NODE_\x00\x00\x00\x05' is `f\x00\x00\x00\x05\x01\x01\xb4\x00
`NODE_\x00\x00\x00\x06' is `f\x00\x00\x00\x06\x01\x01\xb4\x00
`NODE_\x00\x00\x00\x07' is `f\x00\x00\x00\x07\x01\x01\xb4\x00
```

FSNodes information in FDB.

leil STORAGE SDC 25

Database Schema

SaunaFS

Keys are sorted lexicographically based on their byte representation.

- `META_`
- `NODE_`
- `EDGE_<IPARENT><ICHILD>:NAME`
- `FREE_`
- `XATR_`
- `ACLS_`
- `QUOT_`
- `FLCK_`
- `CHNK_`

This way, keys are logically separated and grouped, to be efficiently retrieved using ranges.

Edges in FDB.

Database Schema

SaunaFS

Keys are sorted lexicographically based on their byte representation.

- `META_`
- `NODE_`
- `EDGE_`
- `FREE_<INODE>:<TIME_STAMP>`
- `XATR_`
- `ACLS_`
- `QUOT_`
- `FLCK_`
- `CHNK_`

This way, keys are logically separated and grouped, to be efficiently retrieved using ranges.

```
Range limited to 1000 keys
`FREE_\x00\x00\x00\x09' is `h\xa84o'
`FREE_\x00\x00\x00\x0b' is `h\xa84o'
```

Free inode ids information in FDB.

leil STORAGE SDC 25

Database Schema

SaunaFS

Keys are sorted lexicographically based on their byte representation.

- `META_`
- `NODE_`
- `EDGE_`
- `FREE_`
- `XATR_`
- `ACLS_`
- `QUOT_`
- `FLCK_`
- `CHNK_<ID><VER>:<TIMEOUT><LOCKID>`

This way, keys are logically separated and grouped, to be efficiently retrieved using ranges.

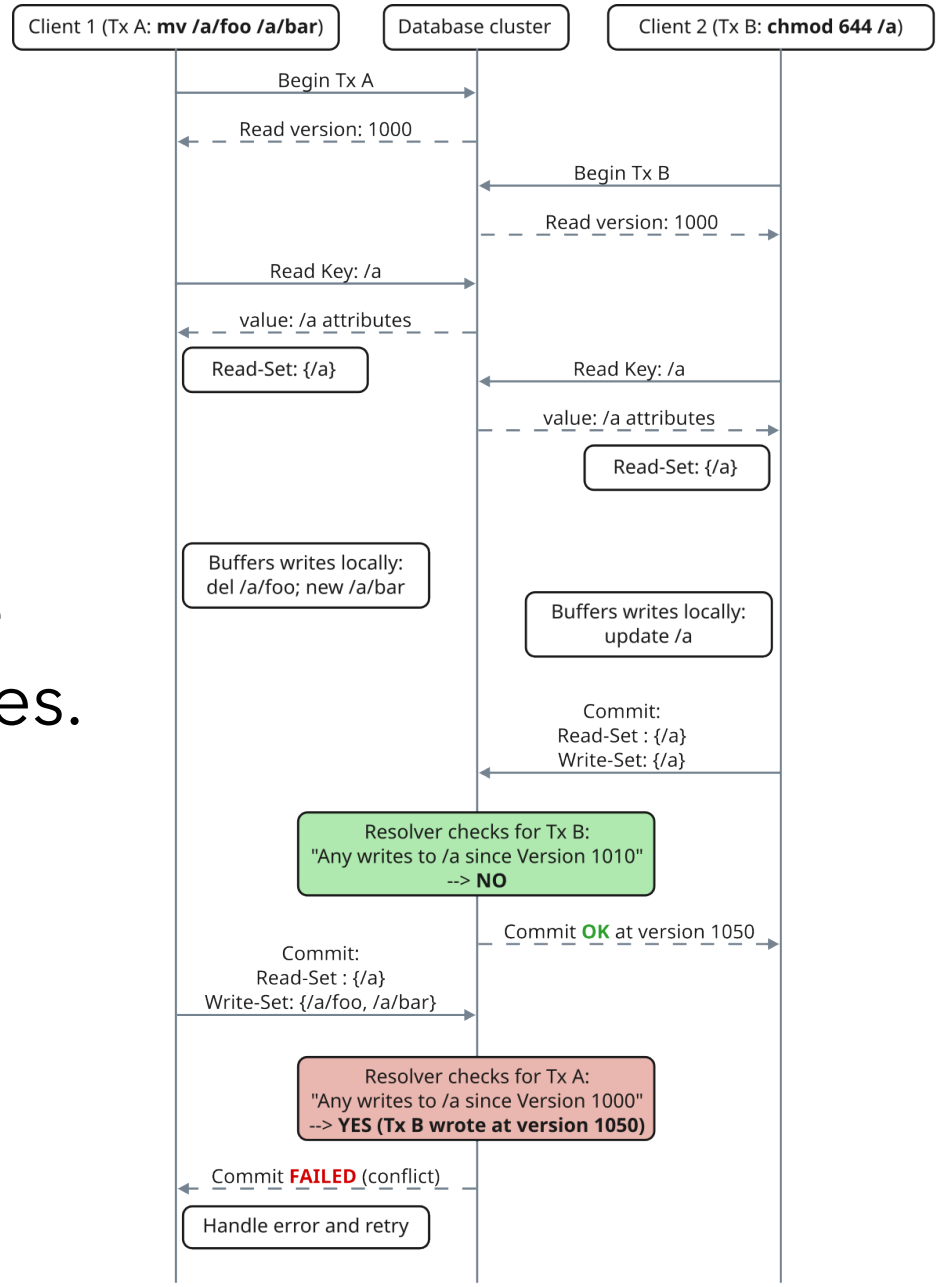
```
Range limited to 1000 keys
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01' is `\x00\x00\x00\x00\x0bdH\xd1'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x01' is `\x00\x00\x00\x00\xcd\xa1\xe0}'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x03\x00\x00\x00\x01' is `\x00\x00\x00\x00/\xac\x8e\xc9'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x01' is `\x00\x00\x00\x00\xcc7\xcf\xe9'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x05\x00\x00\x00\x01' is `\x00\x00\x00\x00}+Mi'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x06\x00\x00\x00\x01' is `\x00\x00\x00\x00zD\xd0\x97'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x07\x00\x00\x00\x01' is `\x00\x00\x00\x00\xc2\x87\x05\xcf'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x08\x00\x00\x00\x01' is `\x00\x00\x00\x00\xf9\x9c\xed\xf6'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x01' is `\x00\x00\x00\x00\x88\x0e\xc1R'
`CHNK_\x00\x00\x00\x00\x00\x00\x00\x0a\x00\x00\x00\x01' is `\x00\x00\x00\x00\xcan,b'
```

Chunks information in FDB.

leil STORAGE SDC 25

Conflict resolution in FDB

- OCC (Optimistic Concurrency Control): Conflicts are checked only at commit time, avoiding heavy locking.
- Abort + Retry: If a conflict is detected, the transaction is aborted, and the client retries.



https://apple.github.io/foundationdb/read-write-path.html

Prototype Results

- Most of the metadata sections were already transformed to key-value format.
- Easy to use Observer-based pattern implemented to keep the metadata in sync with the database.
- Loading the full metadata takes reasonable time, including the insertion in the MDS data structures.

```
/mnt/mount206:
 inodes:          978Ki
  directories:     1003
  files:           977Ki
  links:              0
 chunks:          977Ki
 length:          977GiB
 size:            981GiB
 realsize:        1.3TiB
```

CHUNK STATISTICS

GOAL	SAFE CHUNKS	ENDANGERED CHUNKS	MISSING CHUNKS	ALL CHUNKS
EC62SMR	1000000	0	0	1000000
ALL	1000000	0	0	1000000

CHUNKS WHICH NEED REPLICATION

GOAL	0 COPIES TO REPLICATE	1 COPY TO REPLICATE	2 COPIES TO REPLICATE	3 COPIES TO REPLICATE	4 COPIES TO REPLICATE	5 COPIES TO REPLICATE	6 COPIES TO REPLICATE	7 COPIES TO REPLICATE	8 COPIES TO REPLICATE	9 COPIES TO REPLICATE	10+ COPIES TO REPLICATE
EC62SMR	1000000	0	0	0	0	0	0	0	0	0	0
ALL	1000000	0	0	0	0	0	0	0	0	0	0

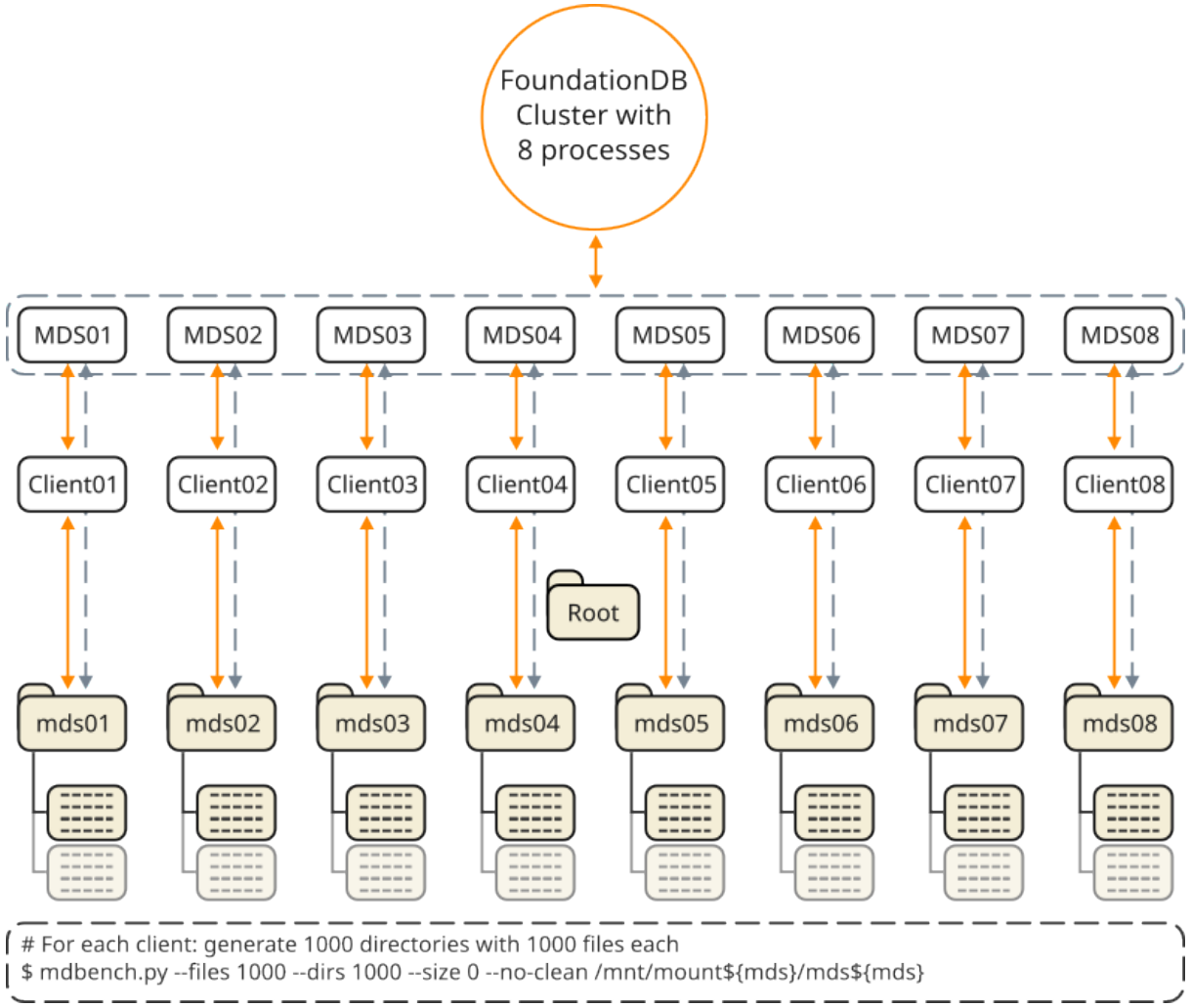
CHUNKS WHICH NEED DELETION

GOAL	0 COPIES TO DELETE	1 COPY TO DELETE	2 COPIES TO DELETE	3 COPIES TO DELETE	4 COPIES TO DELETE	5 COPIES TO DELETE	6 COPIES TO DELETE	7 COPIES TO DELETE	8 COPIES TO DELETE	9 COPIES TO DELETE	10+ COPIES TO DELETE
EC62SMR	1000000	0	0	0	0	0	0	0	0	0	0
ALL	1000000	0	0	0	0	0	0	0	0	0	0

```
08/28/25 22:52:40.155 [info] [3155836:3155836] : metadata read (1002004 inodes including 1004 directory inodes, 1001000 file inodes, 0 symlink inodes and 1001000 chunks)
08/28/25 22:52:40.155 [info] [3155836:3155836] : metadata load time: 2s
```


Prototype Results

Sharding with 8 MDSs and using 8 clients



- For this initial test, each MDS was generating inode numbers in a configured range, to avoid conflicts:
  - MDS01: [2, 10M].
  - MDS02: [10M + 1, 20M].
  - …
  - MDS08: [70M + 1, 80M].

Prototype Results

Sharding with 8 MDSs and using 8 clients

```
Command (merged)
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster04.cfg
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster01.cfg
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster02.cfg
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster03.cfg
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster05.cfg
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster06.cfg
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster07.cfg
/home/developers/leil/saunafs-pro/saunafs/install/saunafs/sbin/sfsmds restart -c /etc/saunafs/sfsmaster08.cfg
```

```
SSH.207 developers@JBOD-207 /mnt $ tree -L 2 /mnt/mount206.05
/mnt/mount206.05
├── mds01
│   └── mdbench.JBOD-207.3729637
├── mds02
│   └── mdbench.JBOD-207.3729886
├── mds03
│   └── mdbench.JBOD-207.3730061
├── mds04
│   └── mdbench.JBOD-207.3730106
├── mds05
│   └── mdbench.JBOD-207.3730112
├── mds06
│   └── mdbench.JBOD-207.3730143
├── mds07
│   └── mdbench.JBOD-207.3730190
└── mds08
    └── mdbench.JBOD-207.3730200
```

Each client was pointing to a specific MDS for these initial tests.

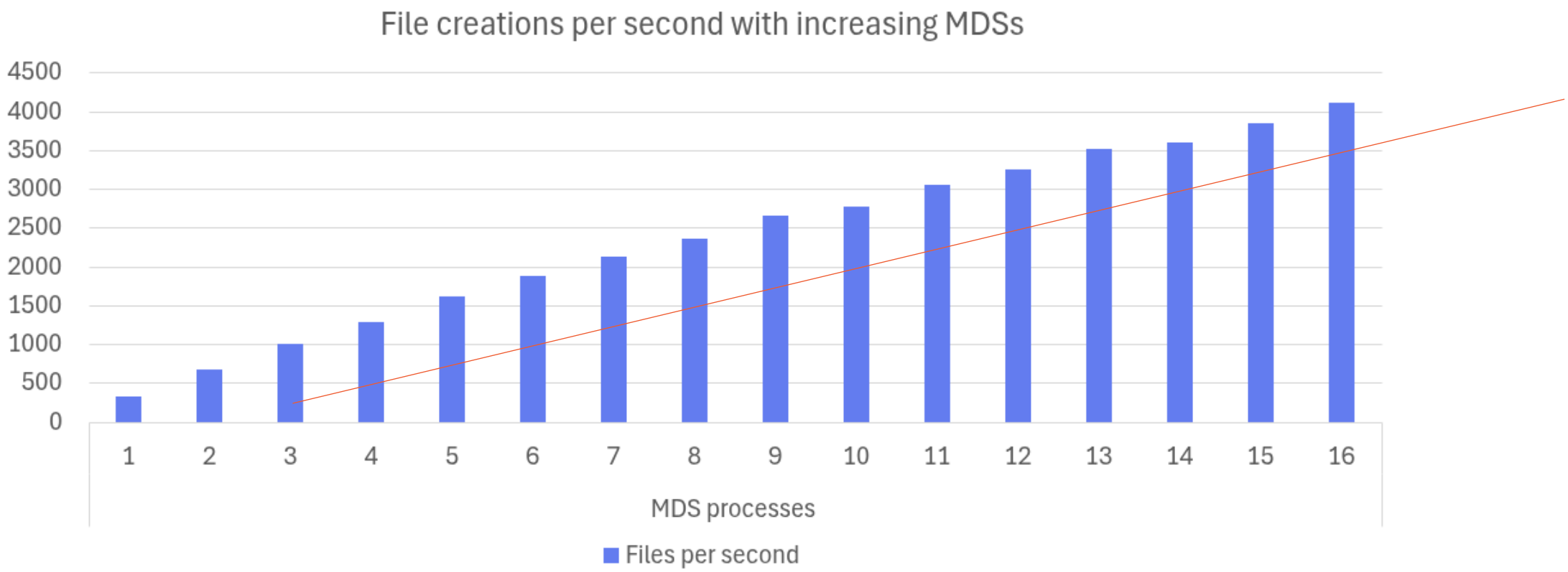
Each client created 1000 directories with 1000 files inside (1 Million files per client).

Metadata operations scaled linearly with the number of MDSs.

```
SSH.207 developers@JBOD-207 /mnt $ export mds=01; sudo /opt/mdbench/mdbench.py --files 1000 --dirs 1000 --size 0 --no-clean /mnt/mount206.${mds}/
Starting at:    : 2025-08-30 19:54:20.678010

dir creates     :   2.45ms ±  0.30ms, 408.64 op/s
file creates    :   3.14ms ±  0.62ms, 318.63 op/s
file stats      :   0.14ms ±  0.05ms, 7299.71 op/s
dir stats       :   0.00ms ±  0.00ms, 299401.20 op/s

Finished at:    : 2025-08-30 20:49:06.478120, runtime: 0:54:45.800110
✓ (54m45s) 20:49:06
SSH.207 developers@JBOD-207 /mnt $

SSH.207 developers@JBOD-207 ~ $ export mds=02; sudo /opt/mdbench/mdbench.py --files 1000 --dirs 1000 --size 0 --no-clean /mnt/mount206.${mds}
Starting at:    : 2025-08-30 19:54:28.070367

dir creates     :   2.61ms ±  0.18ms, 383.36 op/s
file creates    :   3.14ms ±  0.63ms, 317.97 op/s
file stats      :   0.14ms ±  0.05ms, 7104.44 op/s
dir stats       :   0.01ms ±  0.02ms, 137475.94 op/s

Finished at:    : 2025-08-30 20:49:24.649080, runtime: 0:54:56.578713
✓ (54m56s) 20:49:24
SSH.207 developers@JBOD-207 ~ $

SSH.207 developers@JBOD-207 ~ $ export mds=03; sudo /opt/mdbench/mdbench.py --files 1000 --dirs 1000 --size 0 --no-clean /mnt/mount206.${mds}
Starting at:    : 2025-08-30 19:54:38.684014

dir creates     :   2.48ms ±  0.41ms, 403.75 op/s
file creates    :   3.14ms ±  0.63ms, 318.37 op/s
file stats      :   0.14ms ±  0.05ms, 7060.01 op/s
dir stats       :   0.03ms ±  0.04ms, 32820.24 op/s

Finished at:    : 2025-08-30 20:49:32.479303, runtime: 0:54:53.795289
✓ (54m53s) 20:49:32
SSH.207 developers@JBOD-207 ~ $

SSH.207 developers@JBOD-207 ~ $ export mds=04; sudo /opt/mdbench/mdbench.py --files 1000 --dirs 1000 --size 0 --no-clean /mnt/mount206.${mds}
Starting at:    : 2025-08-30 19:54:43.673285

dir creates     :   2.60ms ±  0.31ms, 384.55 op/s
file creates    :   3.14ms ±  0.63ms, 318.68 op/s
file stats      :   0.14ms ±  0.05ms, 7101.60 op/s
dir stats       :   0.02ms ±  0.03ms, 50158.00 op/s

Finished at:    : 2025-08-30 20:49:33.287026, runtime: 0:54:49.613741
✓ (54m49s) 20:49:33
SSH.207 developers@JBOD-207 ~ $
```


Prototype Results

Sharding with 16 MDSs and using 16 clients



Prototype Results

Loading the complete metadata in a single MDS

```
08/31/25 17:12:13.540 [info] [1404734:1404734] : Metadata backend: MetadataBackendFDB
08/31/25 17:12:13.607 [info] [1404734:1404734] : Inode shard: [2 - 10000000]
08/31/25 17:12:13.632 [info] [1404734:1404734] : MetadataBackendFDB::loadall
08/31/25 17:12:13.632 [info] [1404734:1404734] : Loading nodes...
08/31/25 17:12:19.089 [info] [1404734:1404734] : Nodes loaded: 5s
08/31/25 17:12:19.089 [info] [1404734:1404734] : Loading edges...
08/31/25 17:12:23.688 [info] [1404734:1404734] : Edges loaded: 4s
08/31/25 17:12:23.688 [info] [1404734:1404734] : Loading free nodes...
08/31/25 17:12:23.690 [info] [1404734:1404734] : Free nodes loaded: 0s
08/31/25 17:12:23.690 [info] [1404734:1404734] : Loading chunks...
08/31/25 17:12:23.691 [info] [1404734:1404734] : Loaded 0 chunks
08/31/25 17:12:23.691 [info] [1404734:1404734] : Chunks loaded: 0s
08/31/25 17:12:23.691 [info] [1404734:1404734] : connecting files and chunks
08/31/25 17:12:23.847 [info] [1404734:1404734] : calculating checksum of the metadata
08/31/25 17:12:24.514 [info] [1404734:1404734] : metadata read (8008017 inodes including 8017
  directory inodes, 8000000 file inodes, 0 symlink inodes and 0 chunks)
08/31/25 17:12:24.514 [info] [1404734:1404734] : metadata load time: 10s
```

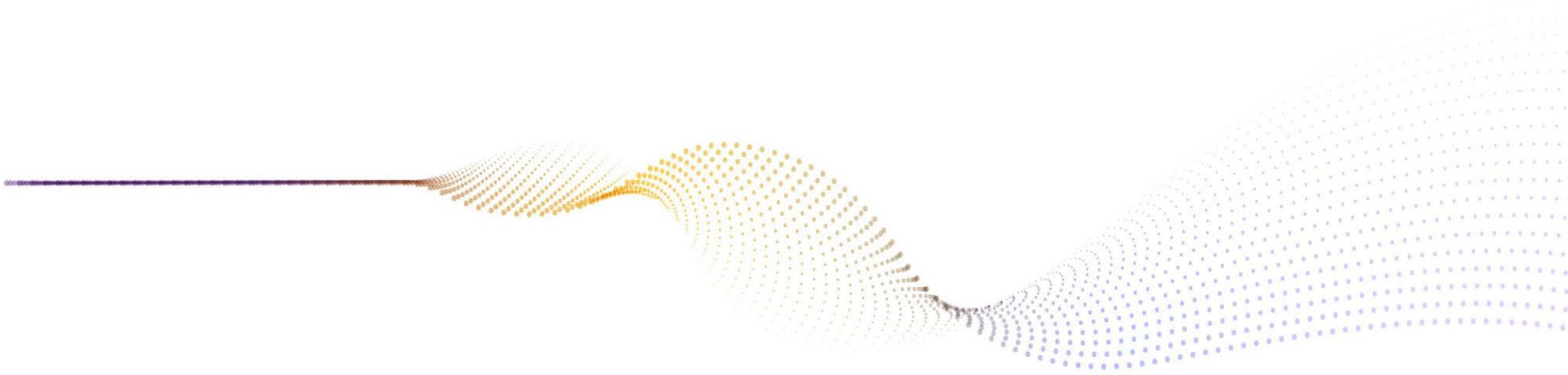

Conclusions

We introduced a scalable metadata service for GFS-inspired file systems, built on top of a distributed database, that provides:

- **Elimination of the single-MDS bottleneck** through multiple active MDS processes backed by FoundationDB.
- **High availability** under node failures, achieved with a Consistent Hashing Ring design.
- **Horizontal scalability** in both metadata throughput (IOPS) and capacity.
- **Support for massive namespaces**, enabling practical management of very large clusters.

Future work

- Continue exploring a **shared-nothing architecture** with a small number of independent MDS processes (like FoundationDB or ScyllaDB or Nginx) rather than relying on thread pools.
  - Potential benefits: better CPU affinity, fewer context switches, reduced contention.
- Enable **elastic scaling** of MDS processes, with automatic spawning or reduction based on workload demand.
- **Benchmark the new architecture at scale** to validate performance and scalability.



Thank you for attending!

Please remember to rate this session. You get access the presentations at
http://sniadeveloper.org/conference

