Modification of Data Structure for Metadata Storage of a Distributed File System

Agnes Natasya

Supervisor: Jialin Li

Outline

- Motivation
- Background
- Implementation
- Result and Analysis
- Future Work
- Conclusion

Motivation

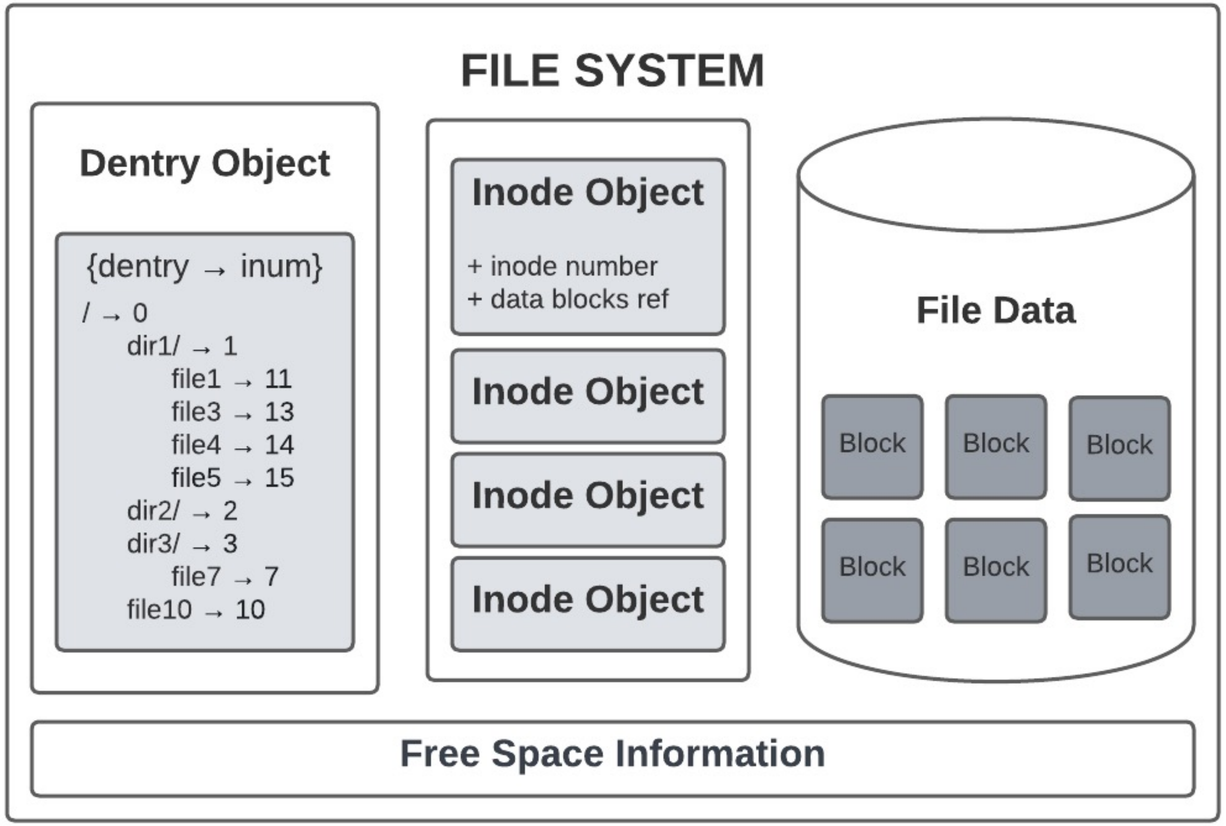
Motivation

- Distributed File System provides remote access to filesystems
- Usually designed for certain workload characteristics
  - In the last decade, focus: scaling of large data operations
  - Not so much on scaling of metadata-intensive operations 1
- Newer hardware technology emerged
  - Non-Volatile Memory (NVM)
  - Remote Direct Memory Access (RDMA)
- **Exploring efficient data structure for metadata-intensive operations, leveraging on NVM and RDMA**

Ren et. al. (2013)

File System Data Structure Design

File System Data Structure

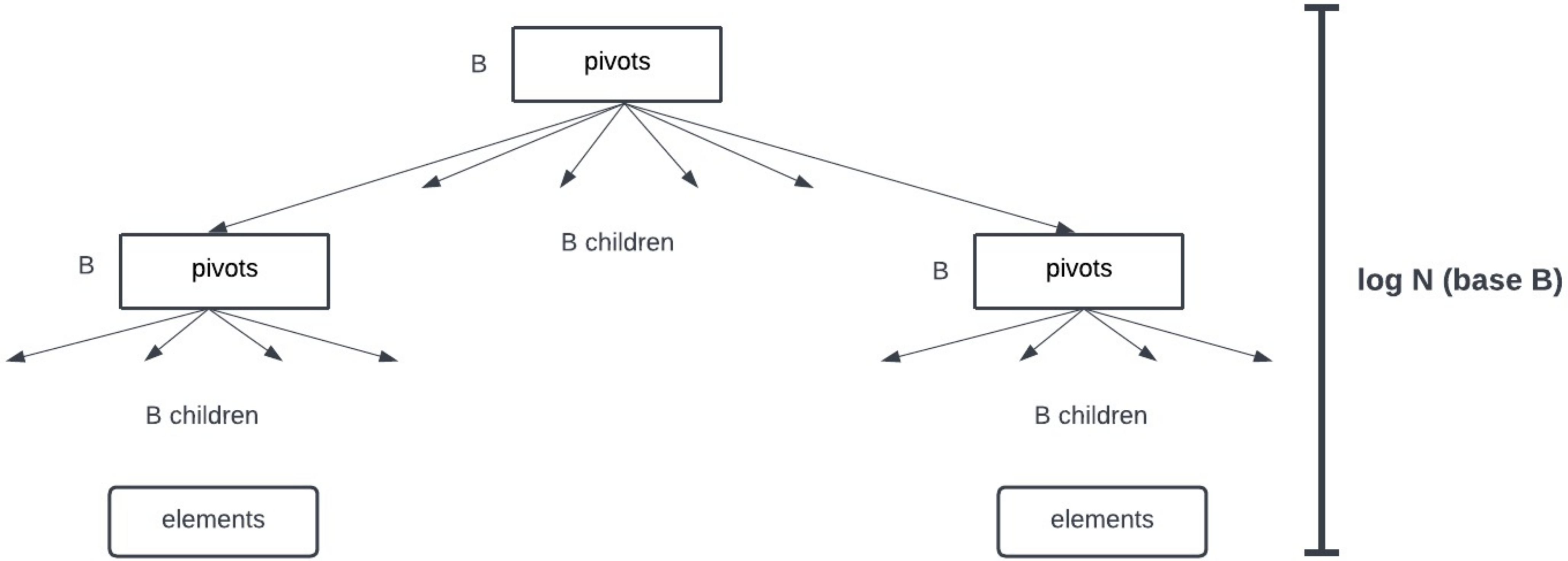


- The data structure to organize the data impact performance
- Examples of diff data structures to store dentry
  - Ext3 → array list of entries
  - Ext4 → HTree (a specialized Btree)
  - BetrFS → Be-Tree (modification of Btree)

Data Structure Tradeoffs

- Unorganised data *(logs)* → write-optimized
- Organised data *(index)* → read-optimized
- Write-optimized index → best of both worlds
  - similar read performance as an indexed data
  - has an asymptotically better write performance than unorganised

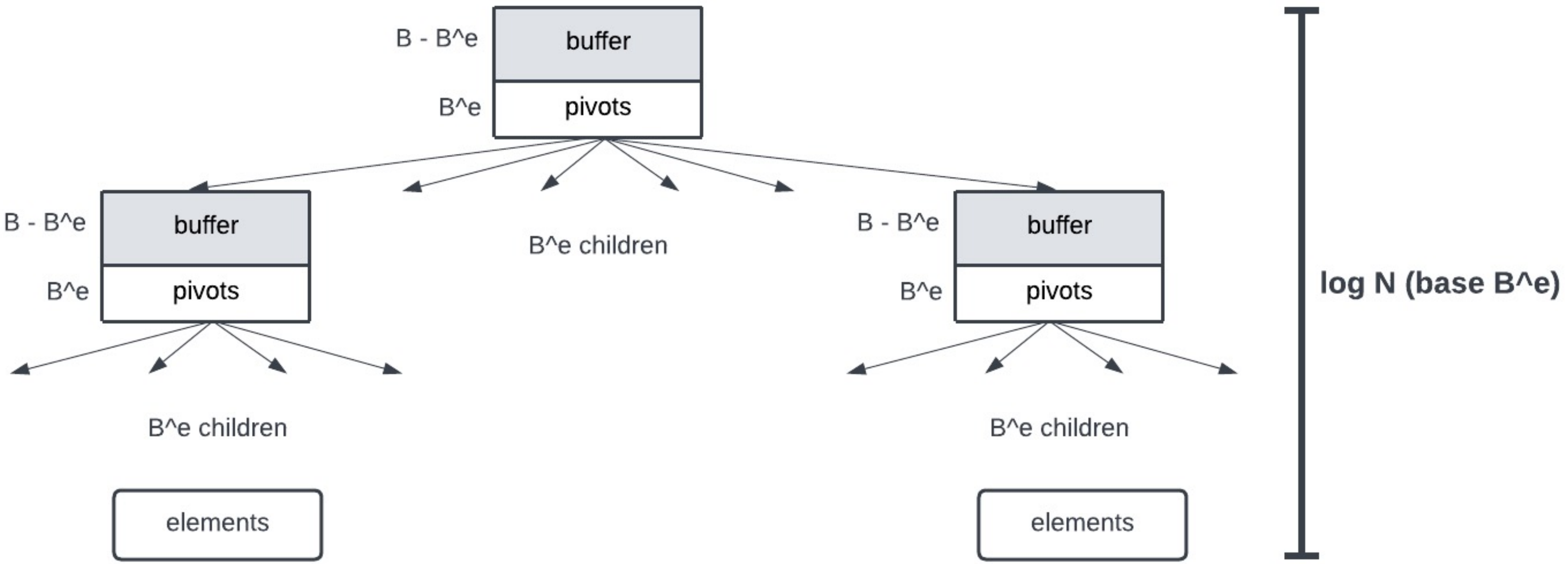
B-Tree (Indexed, Non-Write Optimized)



- Balanced tree → for indexing purpose

Rodeh et. al. (2013)

B^e-Tree (WOI)

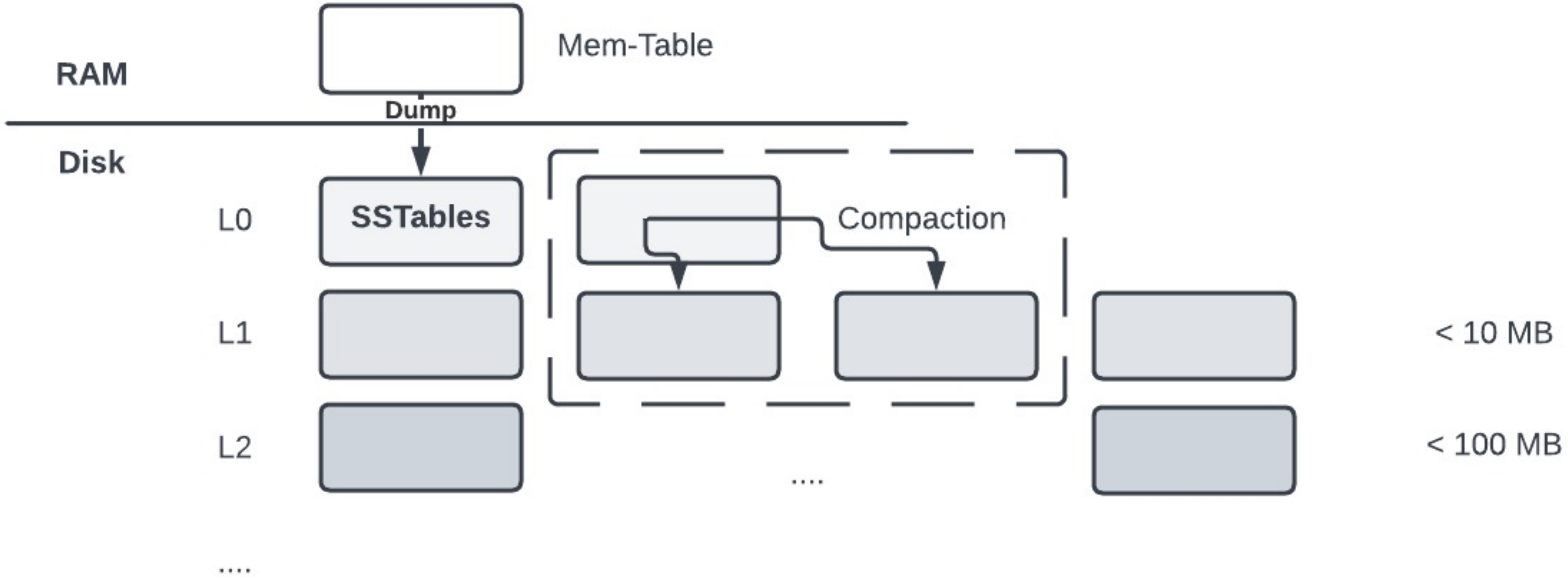


- Buffer amortizes write cost
  - Comparable read performance with B-Tree
  - Asymptotically better write performance than BTree

Jannen et. al. (2015)

Log-Structured Merge Tree (WOI)

LevelDB



- Mem-Table helps write cost
- SSTables helps indexing

Ren et. al. (2013)

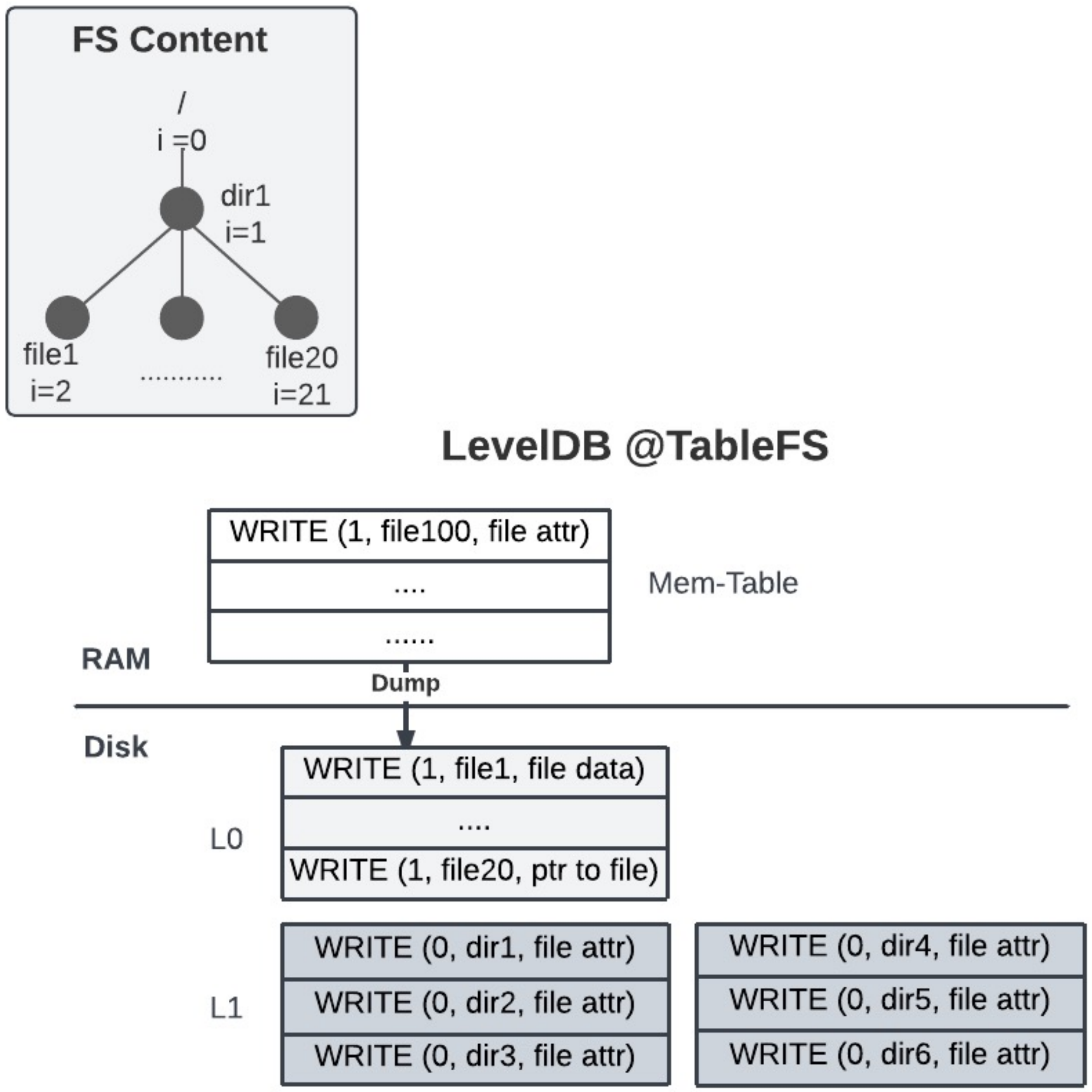
WOI on File System

- WOI → memory ‘log-like’ structure + delayed organization process
- Widely adopted in scalable distributed databases
- Extend to File System
- Some local file systems that uses WOI
  - **TableFS, KVFS → use LSM tree**
  - **BetrFS → use Be-Tree**
  - Not widely adopted
  - Not in distributed file system

TableFS (WOI)

TableFS (local)

- Uses LevelDB (LSM Tree)
- DB table stores:
  - File metadata → dir info
    - KV {Path → file attr}
  - File data → file info
    - Small files: {Path → file data}
    - Large files: {Path → ptr to file}
- In Ext4 (non-WOI), write can change the structure of the parent inode tree

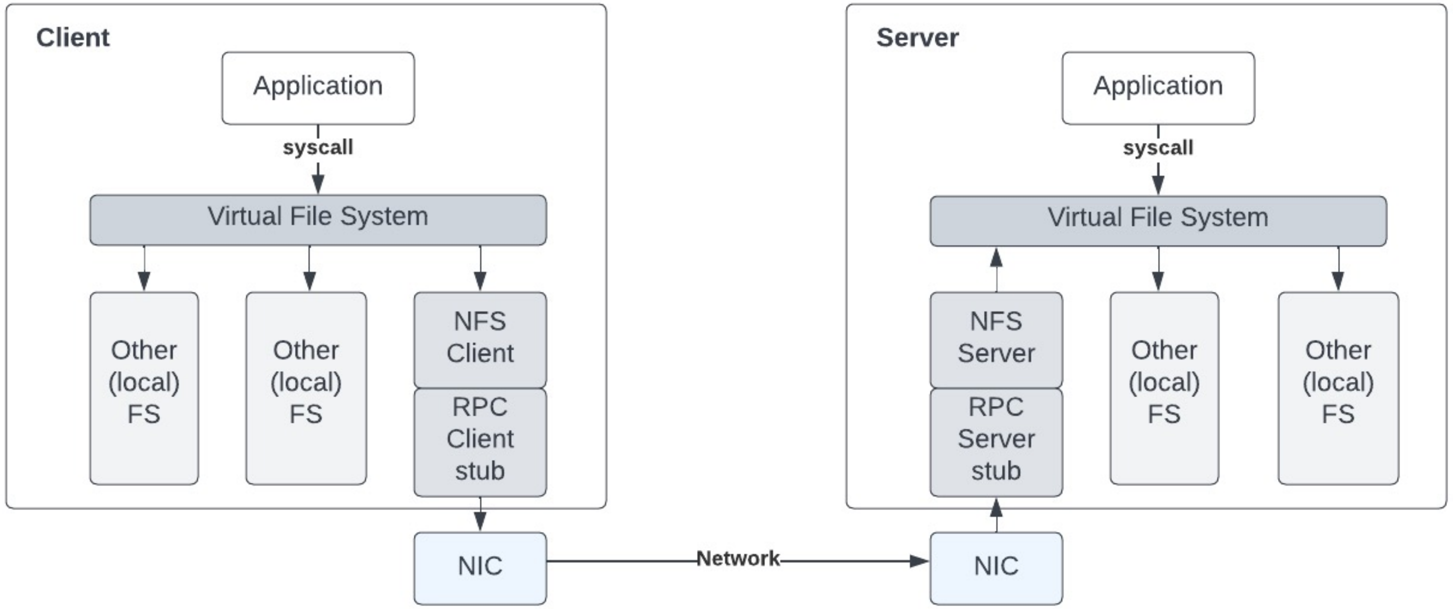


Ren et. al. (2013)

Distributed File System Design

Network File System

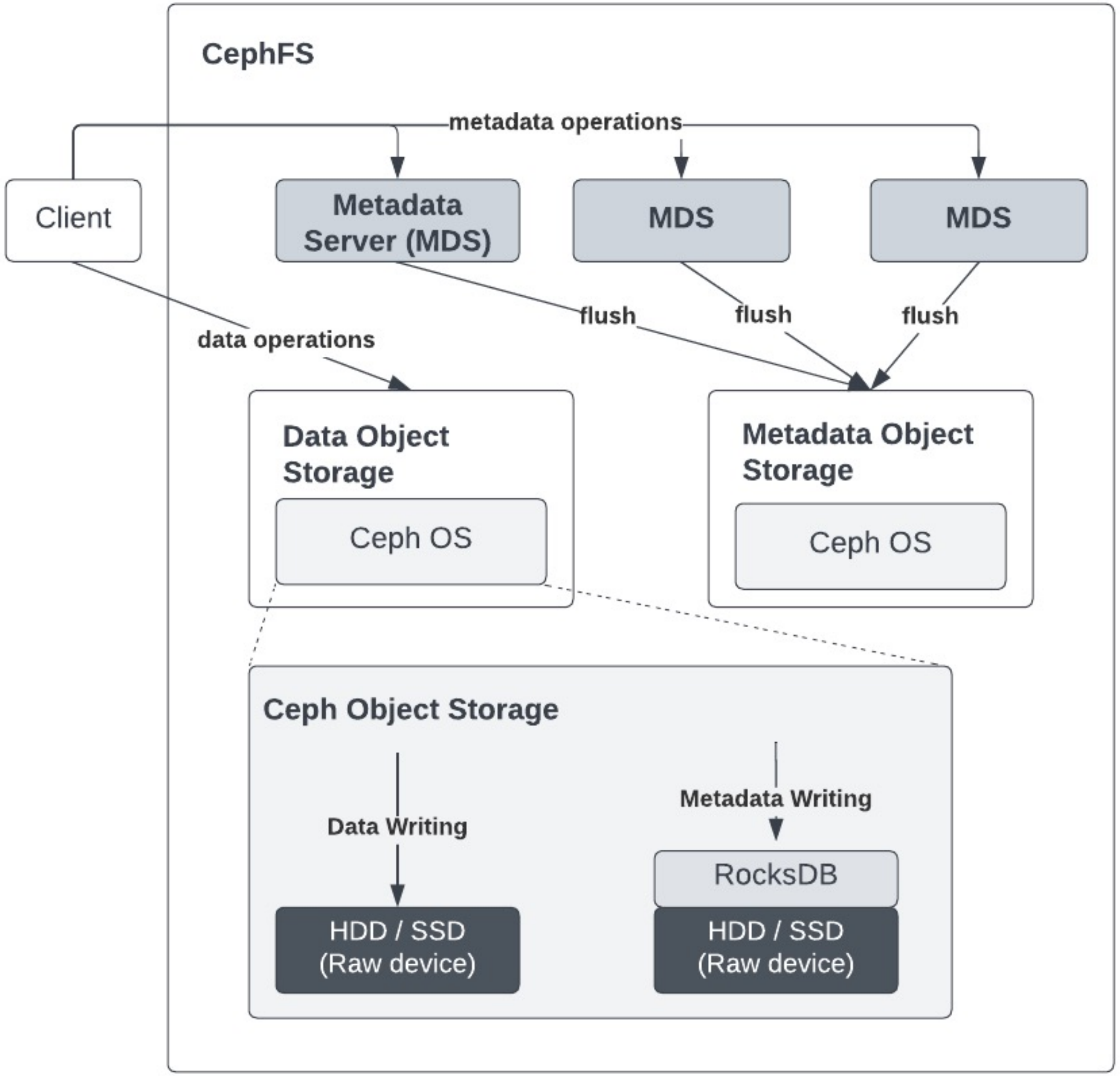
- Server’s directory is mounted by clients
- Performance relies on the underlying file system on the server



Pawlowski et. al. (1994)

Ceph File System

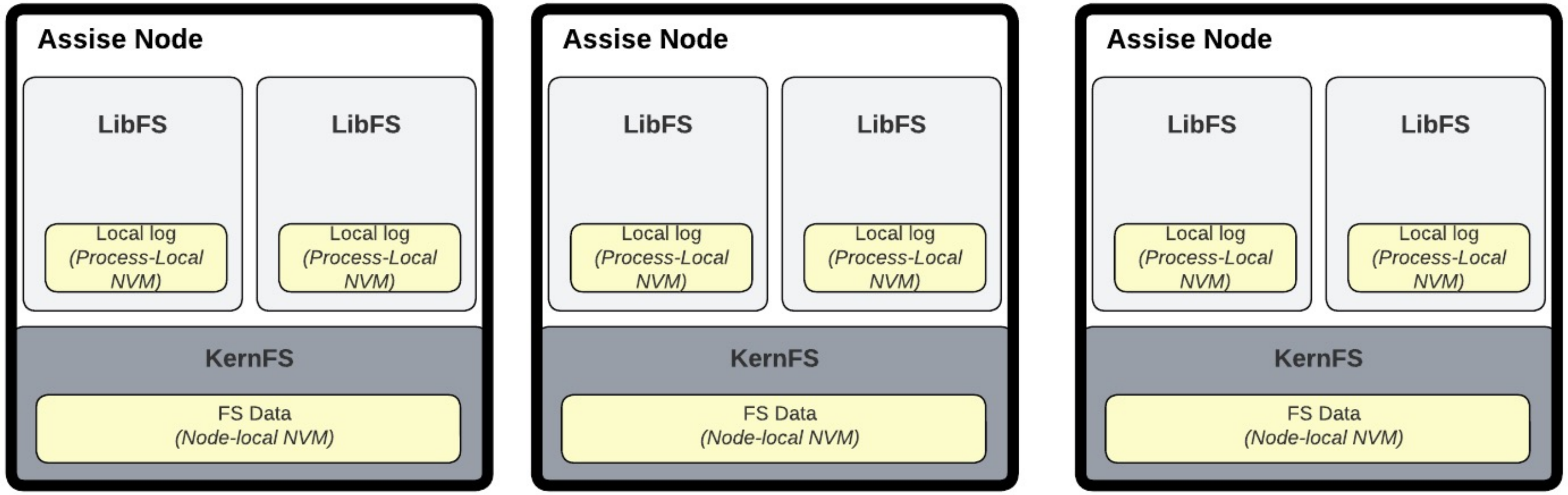
- Replicated object storage (OS) storing file data and metadata
- Metadata is cached by MDS, large-distributed cache for metadata OS



Weil et. al. (2006)

Assise File System

- A collection of servers replicating their local file system log @ NVM
  - Significantly faster than disk but persisted

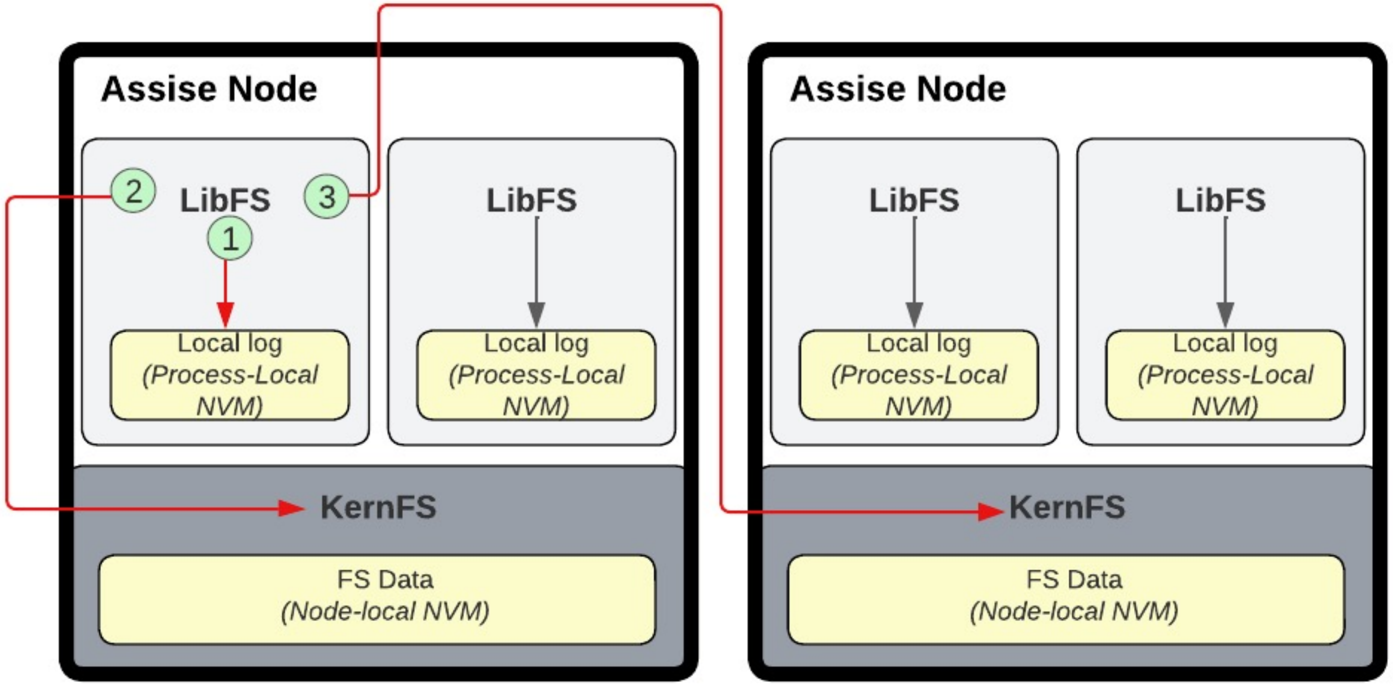


Anderson et. al. (2020)

IO Paths

Each server

- 1 KernFS
- 1 (or more) LibFS



- LibFS → lib (user) process

  Issues FS command

  1. Writes to local log
  2. Request digest to local KernFS
  3. Request replicate to remote KernFS

- KernFS → kernel process
  - Organize FS information
    - Replication and sync of all LibFS
    - Digest LibFS log to indexed FS Data

Anderson et. al. (2020)

Implementation

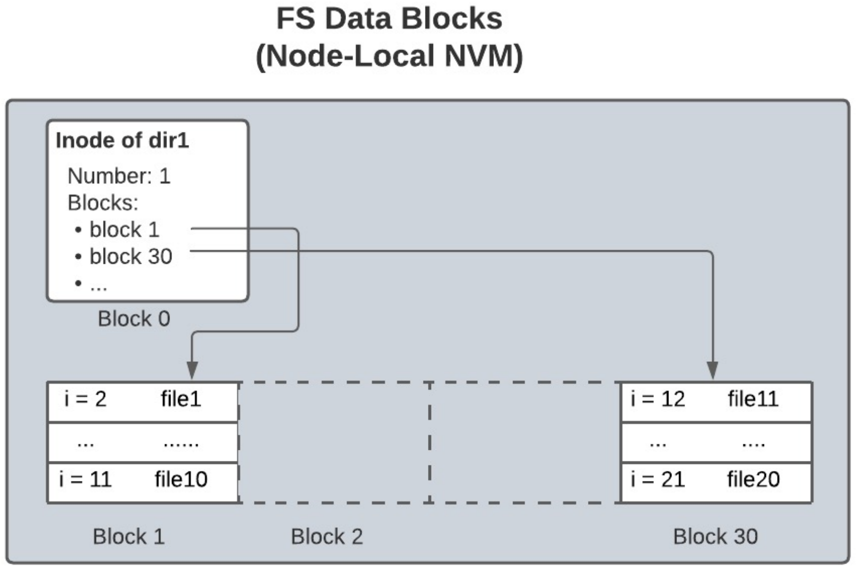
Modifying Assise to Leverage on WOI

- Explore the effect of leveraging LSM Tree index in Assise
- Assise is open-source, built from scratch, not in-kernel, POSIX-compliant
- Assise leverages on
  - NVM as storage system
  - RDMA as communication protocol
  - NVM + RDMA is great → can directly access remote's NVM

Metadata Storage in Assise

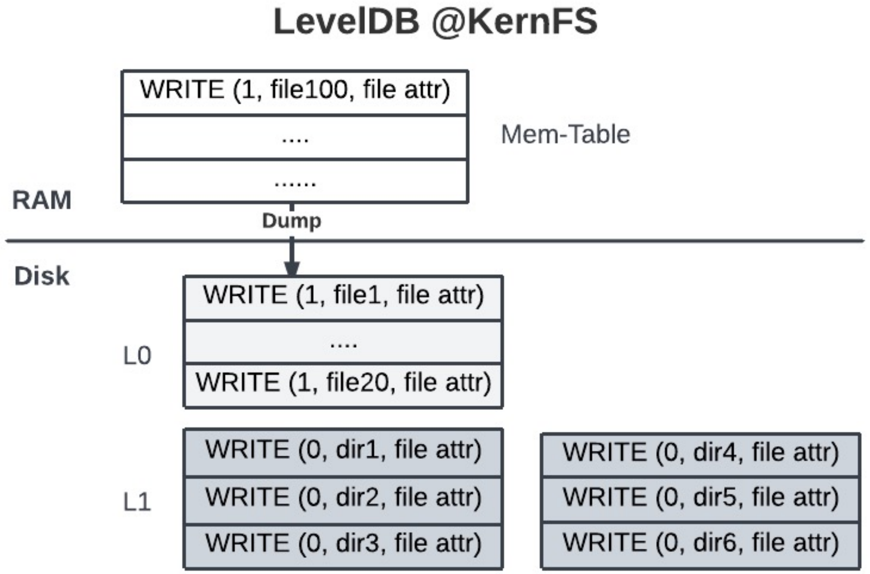
Assise does not leverage on LSM Tree

- File data uses extent tree (like Ext4)
- Dir data uses list (like Ext3)



Modify Assise to leverage LSM Tree

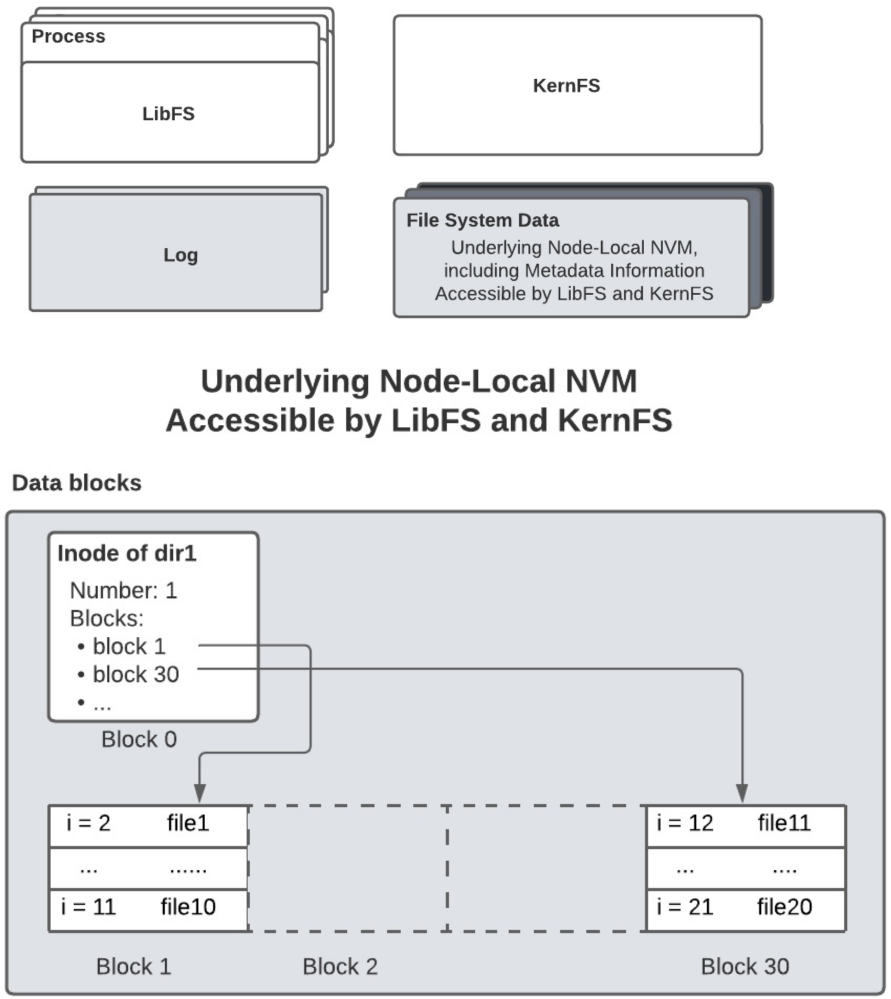
- Stores dir data in LevelDB
- Similar to TableFS



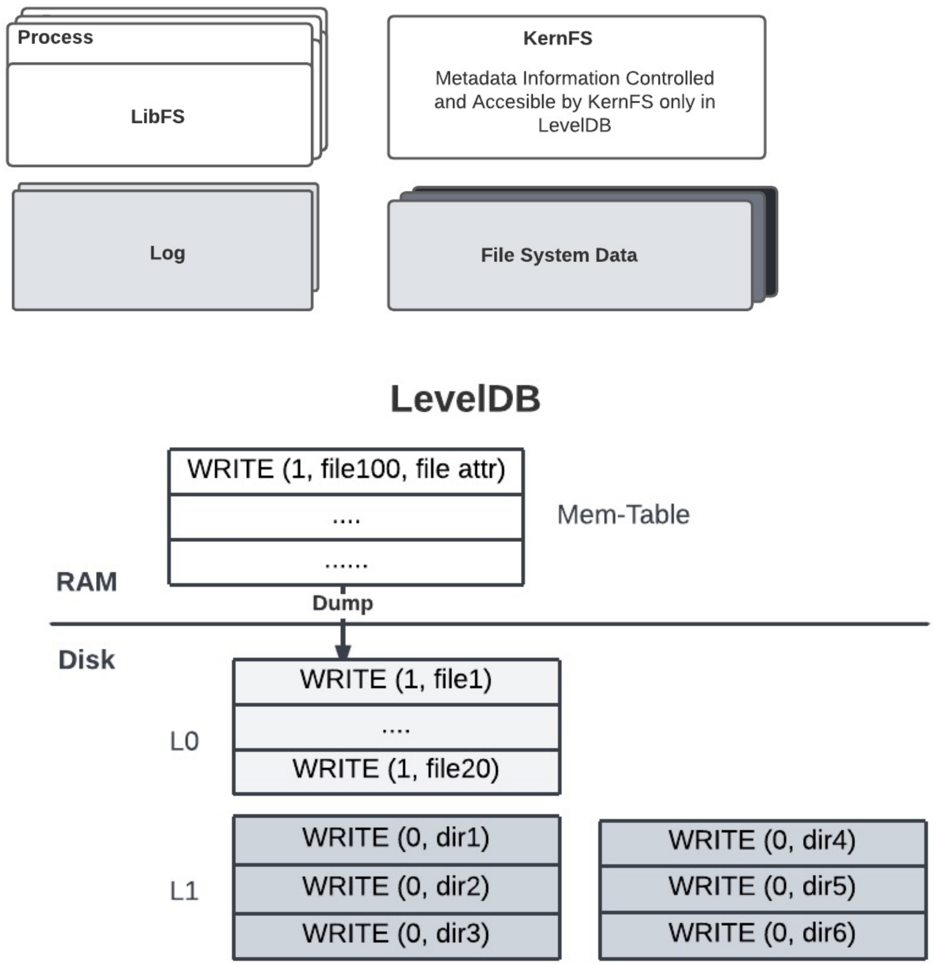
LevelDB Implementation Details

- LevelDB has an in-memory that flushes to disk
  - It disallows 2 processes to access it at the same time, to prevent race condition
- Every node: multiple LibFS processes
  - LibFS cannot access LevelDB
  - KernFS run and access the LevelDB instance
- LibFS (the user) contacts KernFS (stores FS data) to read / write to LevelDB

Fundamental Difference



Original Assise

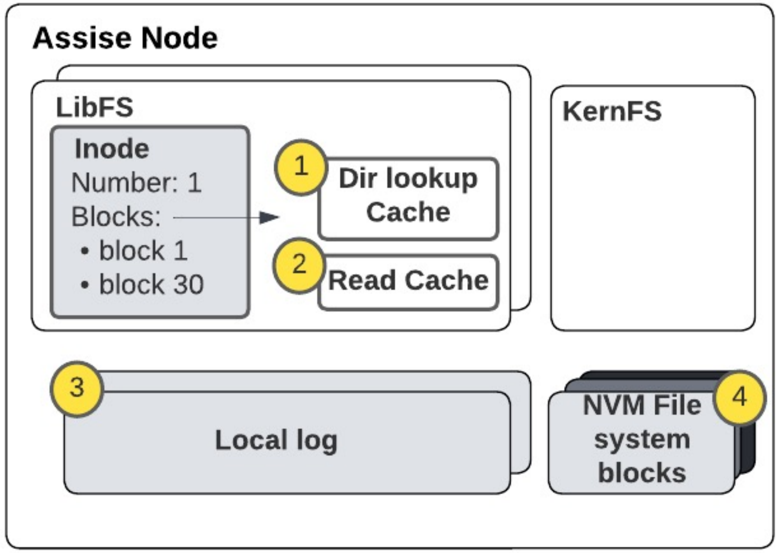


Modified Assise

Directory Lookup Flow

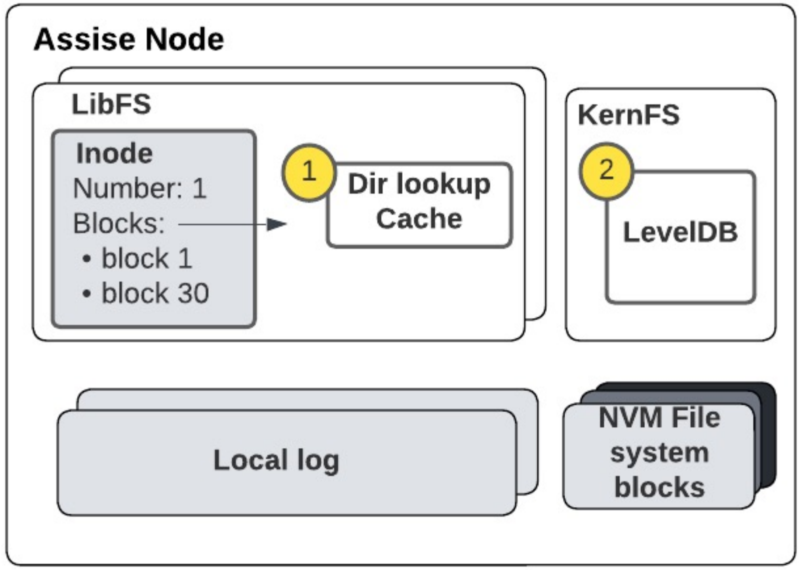
Lookup (name, parent inode)

- For every dir entry inum in the parent inode:
    - Read the content by going through the read layer cache



Lookup (name, parent inode)

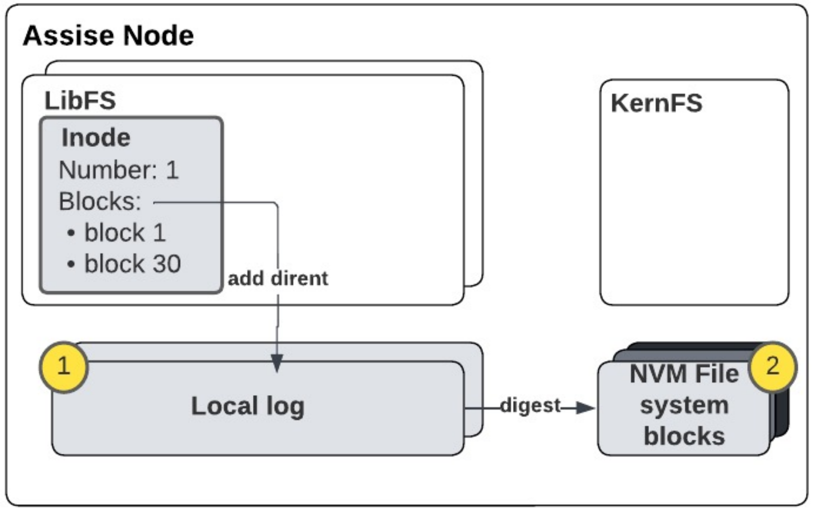
- RPC Lookup (parent inode, name) to local KernFS
- Block until RPC finishes



Directory Entry Addition Flow

Add Entry (name, parent inode)

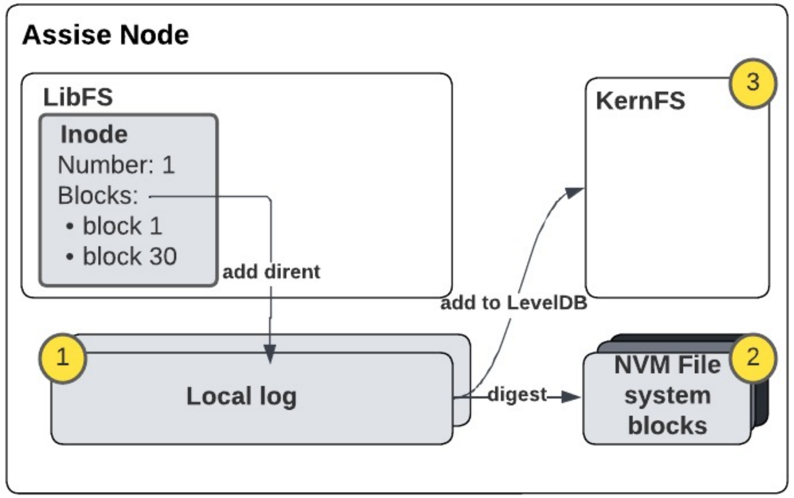
- Log directory entry addition at (parent inode, name)
- Digest in due time



Add Entry (name, parent inode)

Same, but additionally:

- Add to LevelDB
  - Read log content from NVM
  - Put full path → inum

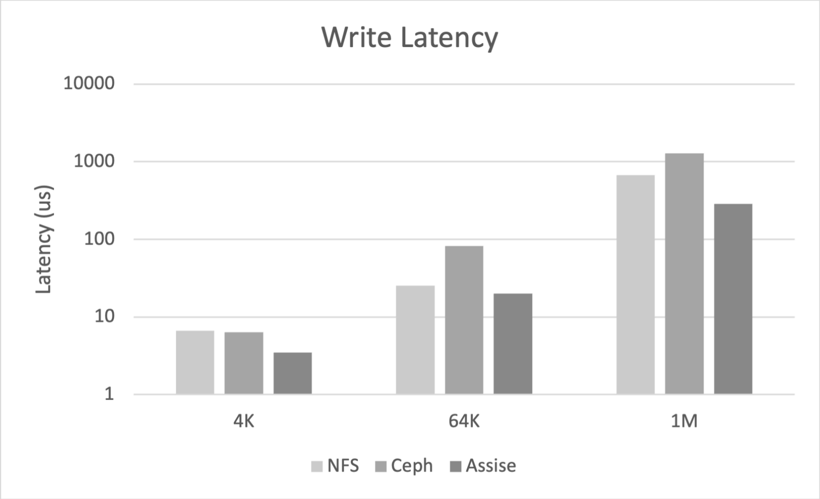


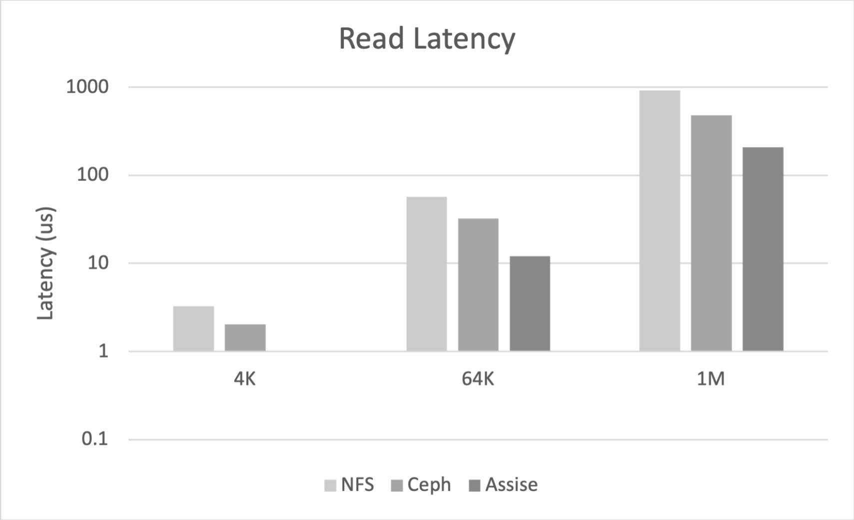
Result and Analysis

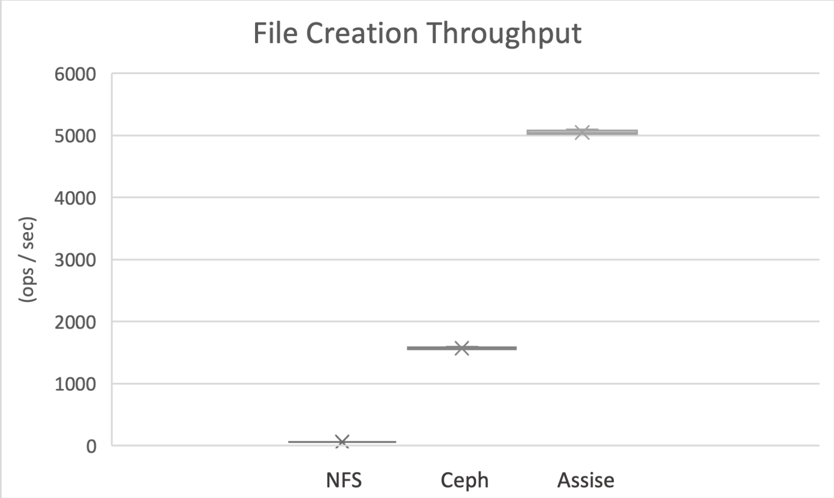
Machines Configuration

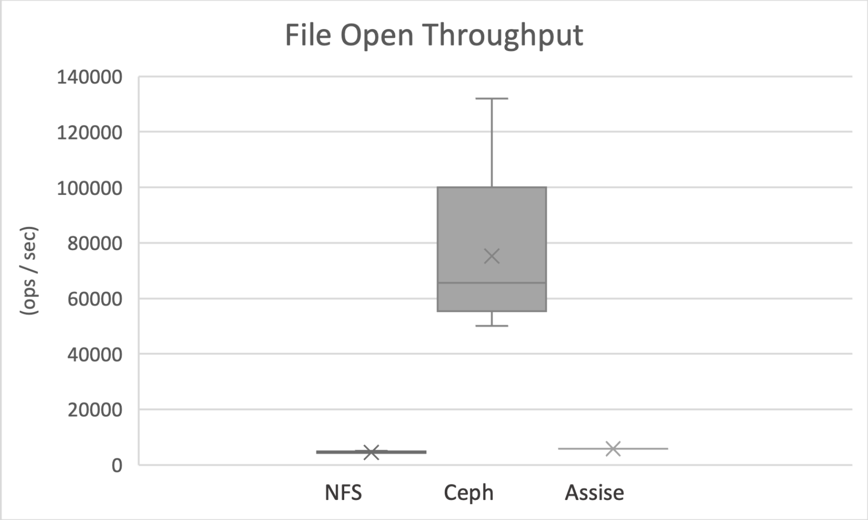
- Xeon E5-2450 processor →16 cores @2.1GHz.
- DRAM capacity → 16GB
  - A subset of the DRAM, 8GB, is used to emulate NVM
- Infiniband NIC : A single Mellanox MX354A Dual port
  - Used for RDMA communication @Assise
  - Used as IPoIB communication @Ceph & NFS
- Ethernet NIC: 1GbE Dual port Broadcom NIC
- 4 x 500GB HDD

Initial Benchmarks



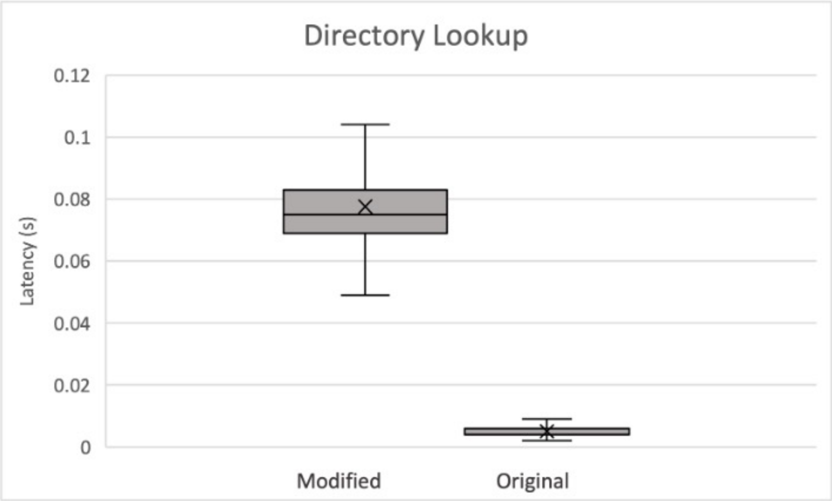






Directory Lookup: Read From LevelDB

Directory Lookup Latency



File Open Throughput

- The modified version includes
  - Blocking RPC call
  - Reading from LevelDB
- Depend on RDMA and LevelDB
- LibFS gets little ‘indexing’ advantage of LevelDB

Directory Entry Addition: Write to LevelDB

Directory Entry Addition Latency

- The modified version includes
  - Reading data from NVM
  - Putting to LevelDB

  on top of the digestion process

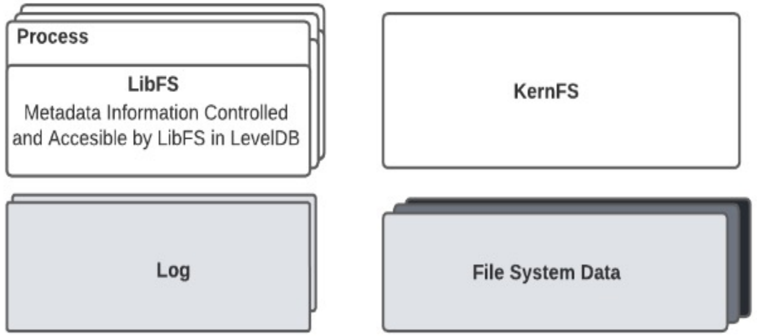
File Creation Throughput

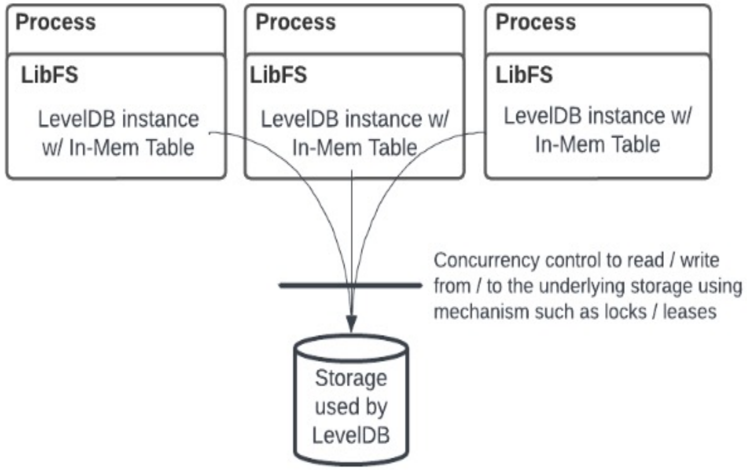
- LSM Tree → low write throughput?
  - Logs to process-local NVM. When full, digest request to KernFS.
  - Limited by the process-local NVM
  - Higher write latency

Future Work

Ideal Design

- In the original version, LibFS can read from the node-local NVM storage shared between LibFS and KernFS → tightly coupled
- LibFS should be able to read directly from the LSM Tree, the LevelDB
  - Modify the LevelDB to allow concurrent access
  - Implement an LSM Tree that allow concurrent access
  - Requires synchronization on flushing process





Conclusion

- The effect of performance optimisation using advanced data structures depends on the implementation and design of the FS.
- Assise user process can directly read from local-node NVM (efficient)
  - Because LibFS and KernFS of Assise is tightly coupled in FS knowledge
- Introducing LevelDB, with KernFS having an exclusive knowledge on it, goes against this advantage