ADVANCES IN OPERATING SYSTEMS DESIGN

Facebook's Tectonic Filesystem: Efficiency from Exascale

19th USENIX Conference on File and Storage Technologies, 2021

Presented by:

Bratin Mondal

Overview: What is Tectonic ?

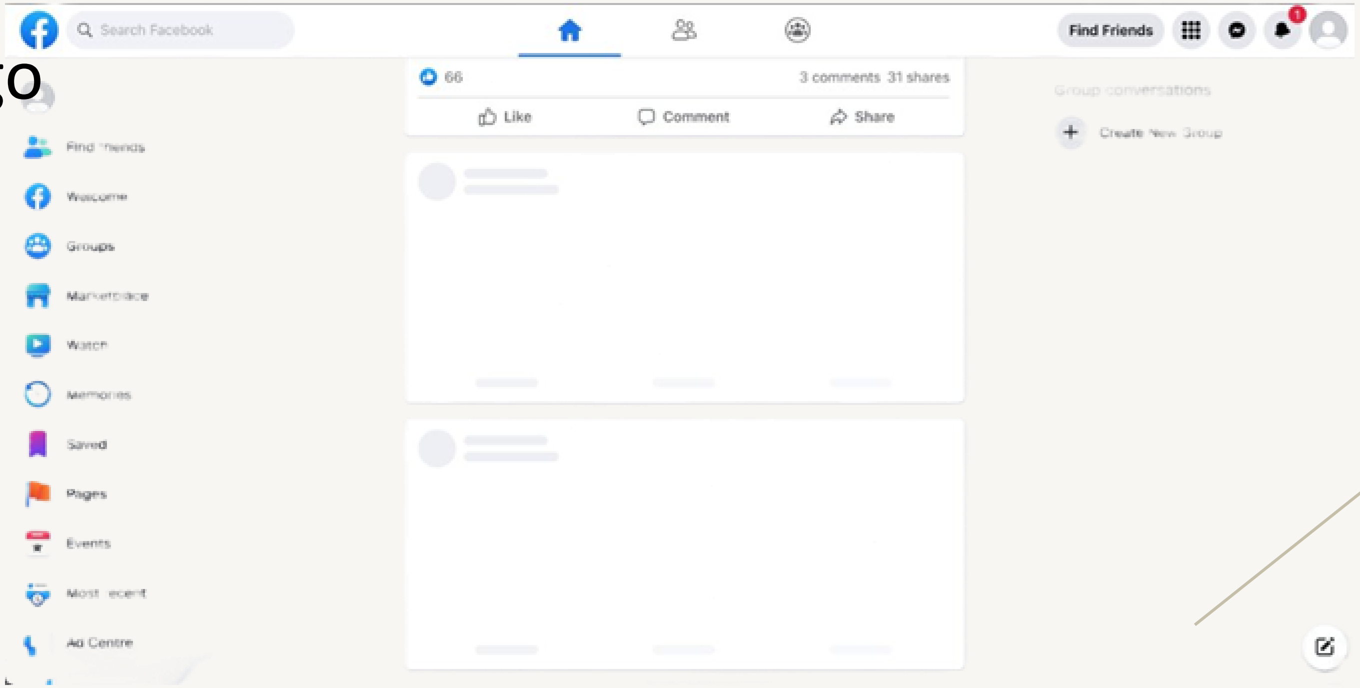
- Meta(Facebook)'s **exabyte-scale** distributed filesystem, append-only
- Serves around **ten tenants** with different requirements
  - Tenants are distributed systems that do not share data with each other and having **different set of requirements**
    - User media Storage (for serving user requests)
    - User history storage (for training analytics model)
- Not optimized for any particular workload but designed in such a way that **tenants can enable their optimizations**

Meta's Storage Infrastructure before Tectonic

- Tenants used specialized storage systems
- Majorly two types of tenants
  - Blob Storage
  - Data Warehouse

Blob (Binary Large Objects) Storage

- Immutable, varies from kilobytes(small photos) to megabytes(HD Videos)
- Low-latency reads and writes required
- Hot Blobs
  - Photo uploaded one minute ago
- Warm Blobs
  - Photo uploaded one year ago



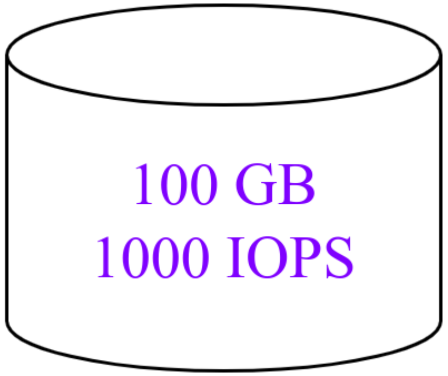
Haystack – Hot blob storage

- **Issue?**
  - Haystack needed very high IOPS
  - Need more devices to meet the IOPS requirements
  - Spare disk storage capacity
  - Replication factor 5.3x in practice (Ideal – 3.6)
  - **Can we use the surplus storage for others?**

f4 – Warm blob storage

- **Issue?**
  - Large number of storage nodes needed to store the vast amount of warm blobs
  - But it also provides IOPS capacity which is not needed for the less frequently-accessed data
  - **Can we use the surplus IOPS for others?**

- 500 GB storage, 8000 IOPS
- 8 Devices
- What do we do with the extra 300 GB?



- 5000 GB storage, 10000 IOPS
- 50 Devices
- What do we do with the extra 40000 IOPS?

Why don't we buy devices that meet our requirements more specifically?

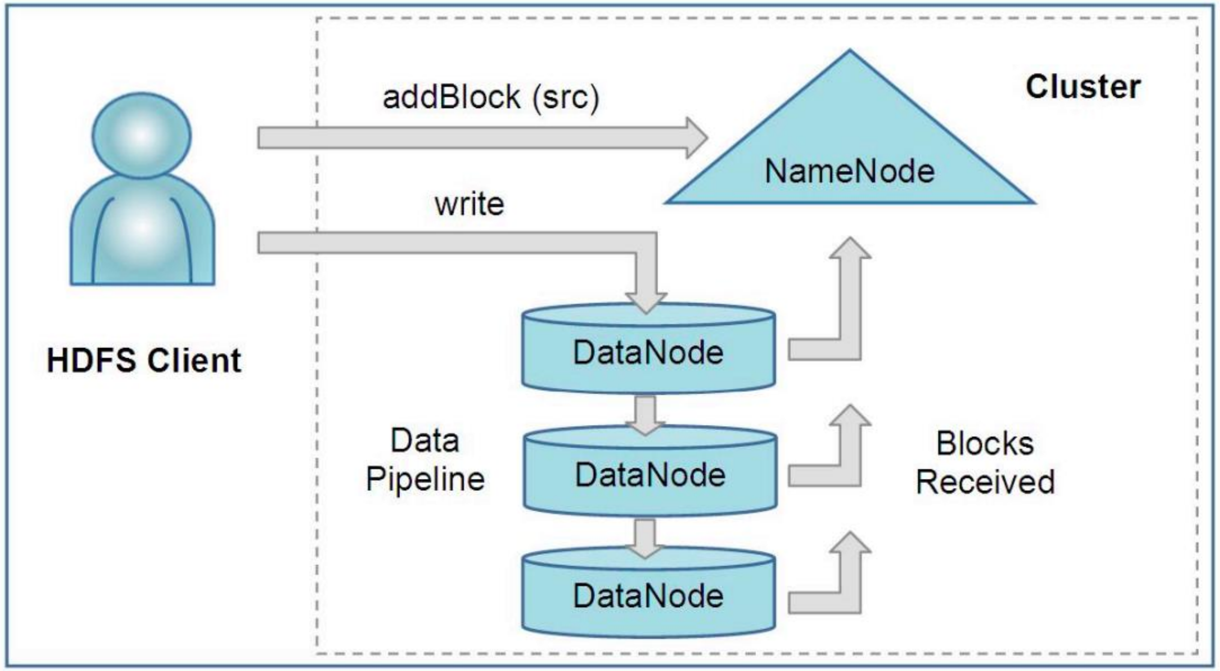
Data Warehouse

- Storage for data analytics (map-reduce tables, snapshots of social graph, etc.)
- Read/Write throughput prioritized over latency (Batch processing)
- Files in a same directory are likely to be accessed parallelly

Hadoop Distributed File System(HDFS)

- NameNode to store and serve metadata
- DataNode to serve read/writes

- **Issue?**
    - HDFS clusters are limited in size because of single NameNode
    - Data needs to be distributed among multiple clusters
    - 2D Bin Packing (NP-Hard) -
        - Cluster capacity
        - Cluster throughput
- **How to scale?**



Source: https://pages.cs.wisc.edu/~akella/CS838/F16/838-CloudPapers/hdfs.pdf

Challenges

1. Scale to exabyte scale
   - How to store the large amount of metadata?
   - How to serve the metadata effectively without being a bottleneck?
2. Performance isolation between tenants
   - In a storage system used by a large number of tenants, how to prevent one tenant's performance from being affected by others while enabling surplus resource sharing?
3. Tenant-specific optimizations
   - In a generalized filesystem, how to provide tenants with support that meets the guarantees typically offered by specialized filesystems?

Some existing Systems

- Federated HDFS and Windows Azure Storage (WAS)
  - **Merge multiple smaller storage clusters into larger clusters**
    - Multiple independent namespaces but Data nodes are shared – Still bin-packing at namespace level (**In which namespace to put what data?**)
- Ceph and Fault-tolerant Distributed Storage(FDS)
  - **Hash Data object to get data location** – Increase the Range of hash function to scale
    - Hash function needs to be updated on each data relocation(Too costly in large systems)

Tectonic Cluster

- Storage Nodes
- Metadata Nodes
- Stateless Nodes for background operations
- Client library interacts with tectonic

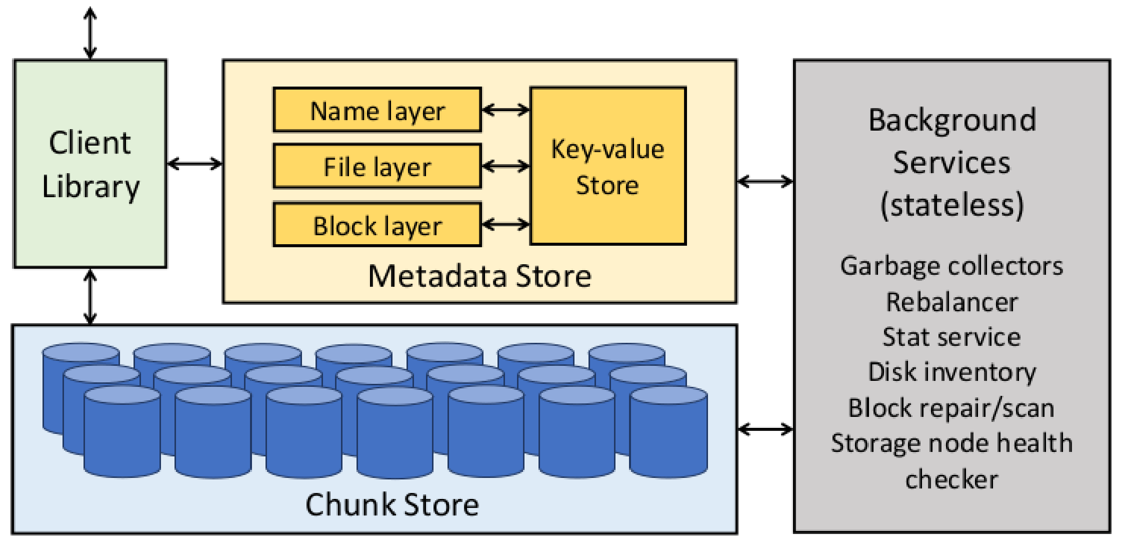


Figure 2: Tectonic architecture. Arrows indicate network calls. Tectonic stores filesystem metadata in a key-value store. Apart from the Chunk and Metadata Stores, all components are stateless.

Source: https://www.usenix.org/system/files/fast21-pan.pdf

Chunk Store

- Files divided into blocks and blocks are divided into chunks
- Abstraction created by the client library using metadata
- Chunks are stored as files in cluster's storage nodes using XFS
  1. Scalable – Linearly grows with the number of storage nodes
  2. Read/write to storage nodes can be **specialized to tenant's performance needs**
- Tenants can choose how they want to store the chunks
  1. Replication
  2. Reed-Solomon Encoding [RS(n,k): n data blocks, k parity blocks]

Metadata Store

Layer	Key	Value	Sharded by	Mapping
Name	(dir_id, *subdirname*)	subdir_info, subdir_id	dir_id	dir → list of subdirs (expanded)
	(dir_id, *filename*)	file_info, file_id	dir_id	dir → list of files (expanded)
File	(file_id, blk_id)	blk_info	file_id	file → list of blocks (expanded)
Block	blk_id	list<disk_id>	blk_id	block → list of disks (i.e., chunks)
	(disk_id, blk_id)	chunk_info	blk_id	disk → list of blocks (expanded)

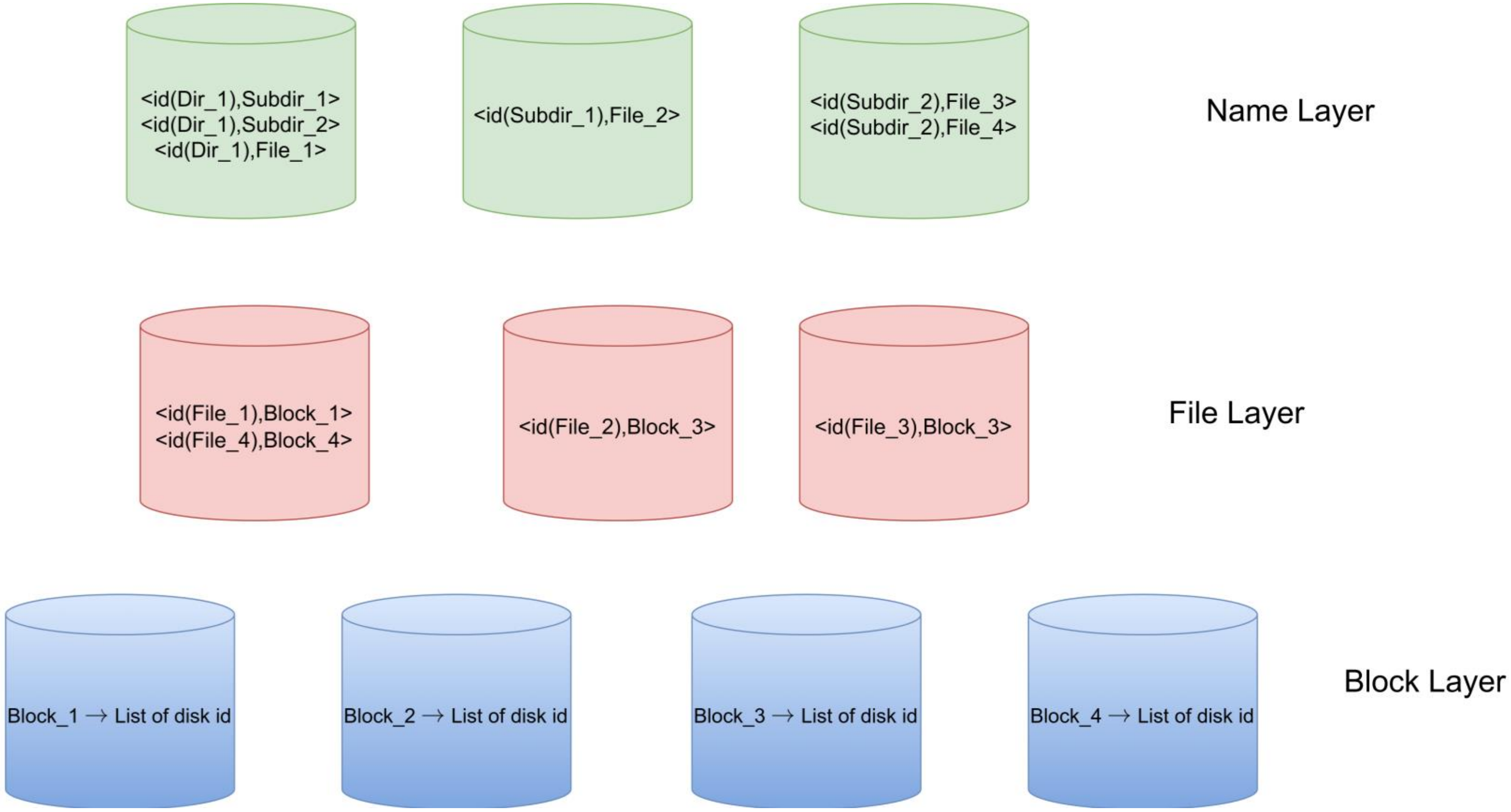
Table 1: Tectonic's layered metadata schema. *dirname* and *filename* are application-exposed strings. dir_id, file_id, and block_id are internal object references. Most mappings are expanded for efficient updating.

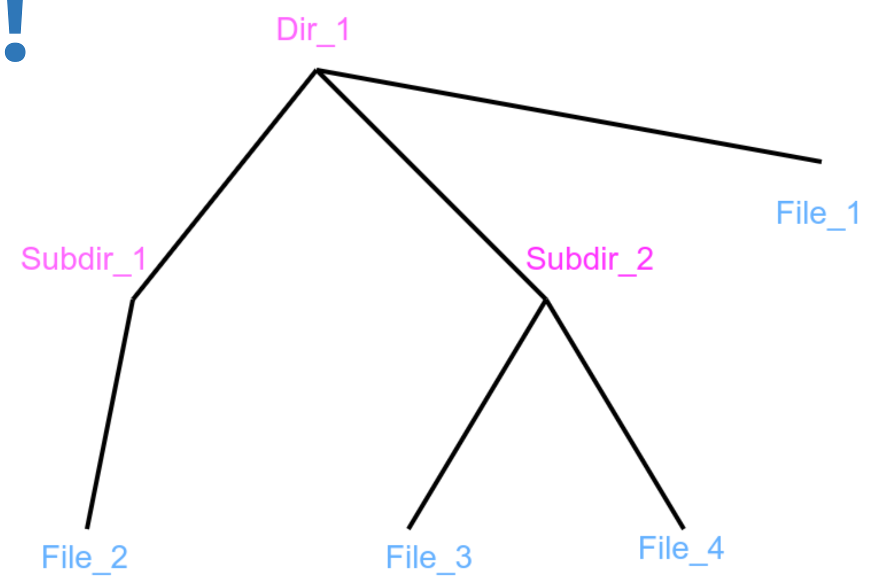
- Three metadata layers
  - Name Layer: Directory → Sub-directories/files
  - File Layer: File → Blocks
  - Block Layer: Block → List of disks(chunks), Disk → Block
- Sharded by Key using **Hash Partitioning**
- Stored in **expanded format** (Fast updates)

Break Metadata into Layers !

1. Expanded Key-Value Pairs
2. Hash Based Partitioning

Metadata Storage





Namespace

- Managed by ZippyDB at a shard granularity
- Metadata store nodes internally runs RocksDB and shards are replicated using Paxos

- Advantages
  1. Scalable – Metadata storage grows with number of metadata nodes
  2. Layered metadata approach and hash-partitioning avoids hotspots

- Issue
  1. Higher latency
     - Cache Name and File layer once sealed
     - Block layer cache needs to be updated
  2. No support for atomic cross-directory move operations
  3. No whole directory content listing support. Client Library needs to build it

Note: Initially, Name Layer and Block Layer were merged but it turned out to be a hotspot

Client Library

- Provides filesystem abstraction to applications
- Client library operates read/write operations at chunk granularity

Single Writer Semantics

- Token provided to client when it opens the file for appending
- Updates to metadata must contain the token
- Last process having opened the file is the only writer

- Tenants needing multiple-writer semantics can build their own serialization semantics on top of tectonic

Background Services

- Operate on one shard at a time
- Garbage collector
  - Removes inconsistencies between metadata layers

- Rebalance and Repair
  - Identify lost chunks and repair them
  - Uses the reverse index mapping from disks to blocks to scale horizontally

Multitenancy

- Non-ephemeral resource: Predictable and slow changes and No dynamic adaptation
  Storage Capacity: Pre-allocated capacity
- Ephemeral resource: Real-time automated management
  - IOPS and Metadata Query

- At what granularity to manage?
  - Tenants?
    - Too much generalization – Tenants have different applications with different workloads
  - Applications?
    - Too complex and resource intensive

- **TrafficGroups** and **TrafficClass**
  - Applications within same Trafficgroup have similar latency and resource requirements
  - TrafficClass – Gold(Latency-Sensitive), Silver(Normal), Bronze(Background applications)

- **Global Resource Sharing**
  - **Aim:**
    - Limit resource usage
    - Share surplus resources effectively

  - Modified Leaky bucket algorithm – High-Performance, Near-Realtime Distributed Counters
    - Increment the counter – If spare capacity, use it
      - Check in other TrafficClass in same tenant
      - Check in other tenants
    - Using other TrafficClass resource – Resulting Traffic Class gets the minimum class

- **Local Resource Sharing at Metadata and Storage Nodes**
  - **Aim:**
    - Avoid it being hotspot
    - Meet latency requirement for Gold class

  - Weighted Round-Robin(WRR): Skip already if quota is used or will exceed if granted
  - Some optimizations:
    - Lower request class can give its turn to higher request if it will have enough time after the higher request gets serviced
    - Limit number of non-Gold traffic in the queue for a disk if there are pending Gold requests
    - Disks may re-arrange requests! Stop scheduling non-gold requests if some gold request is pending for some threshold amount of time

- **Note:** Resource sharing need not be done again at Nodes because you have already done it globally

Access Control

- Token Based Access Control Mechanism
- Each Layer give the token for the next layer
- Tokens are piggybacked

Tenant Specific Optimizations – Client Library Driven Design

- **Data Warehouse Write Optimizations –** Read only after complete write semantics
  1. **Asynchronous writes**:
     - Applications themselves buffer writes until block size data available
     - RS-encode (less memory and network overhead)
     - Write the chunks in parallel
     - No inconsistency since metadata only update after complete write
  2. **Hedged Quorum Writes**:
     - First send reservation requests to more than the numbers of nodes needed
     - Don't wait for successful completion of writes to all the nodes. Wait only until majority of the blocks have been written and then reply to client.
     - Improves 99th percentile latency

- **Blob Storage Optimizations –**
  1. **Consistent partial block appends:**
     - Wait only till quorum size appends
     - Only block creator can create append
     - Metadata updated with post-append block size and checksum
  2. **Reencoding blocks for storage efficiency:**
     - Enabled by Client Library-driven design. Generally filesystems need you to preconfigure the way you want to store files.
     - From replicated to Reed-Solomon(RS) encoding

Copysets -

Cidon et al. Copysets: Reducing the Frequency of Data Loss in Cloud Storage, Stanford University.

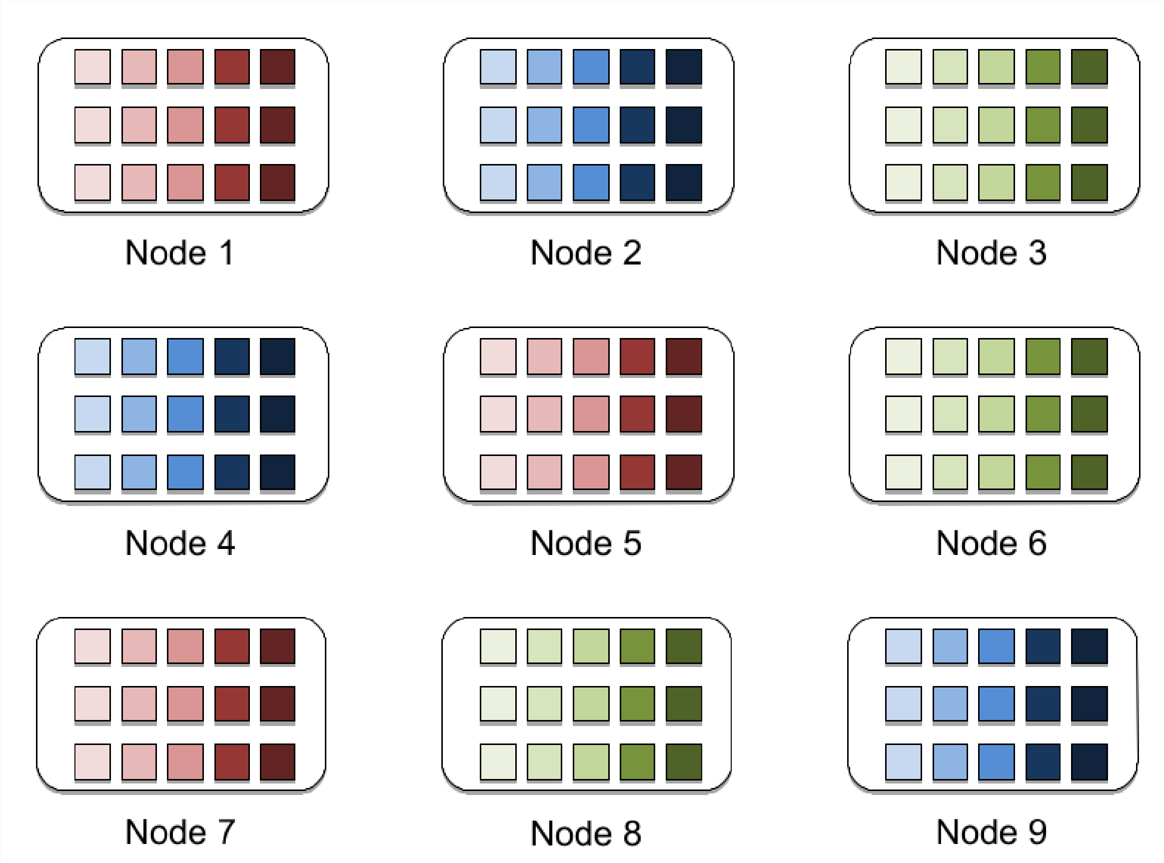
- Set of nodes for which simultaneous crash will lead to data loss
- What if we put copies of data in randomly chosen storage nodes?

- Huge number of copysets
- Data is lost if any three of the nodes go down

Source: https://www.usenix.org/conference/atc13/technical-sessions/presentation/cidon

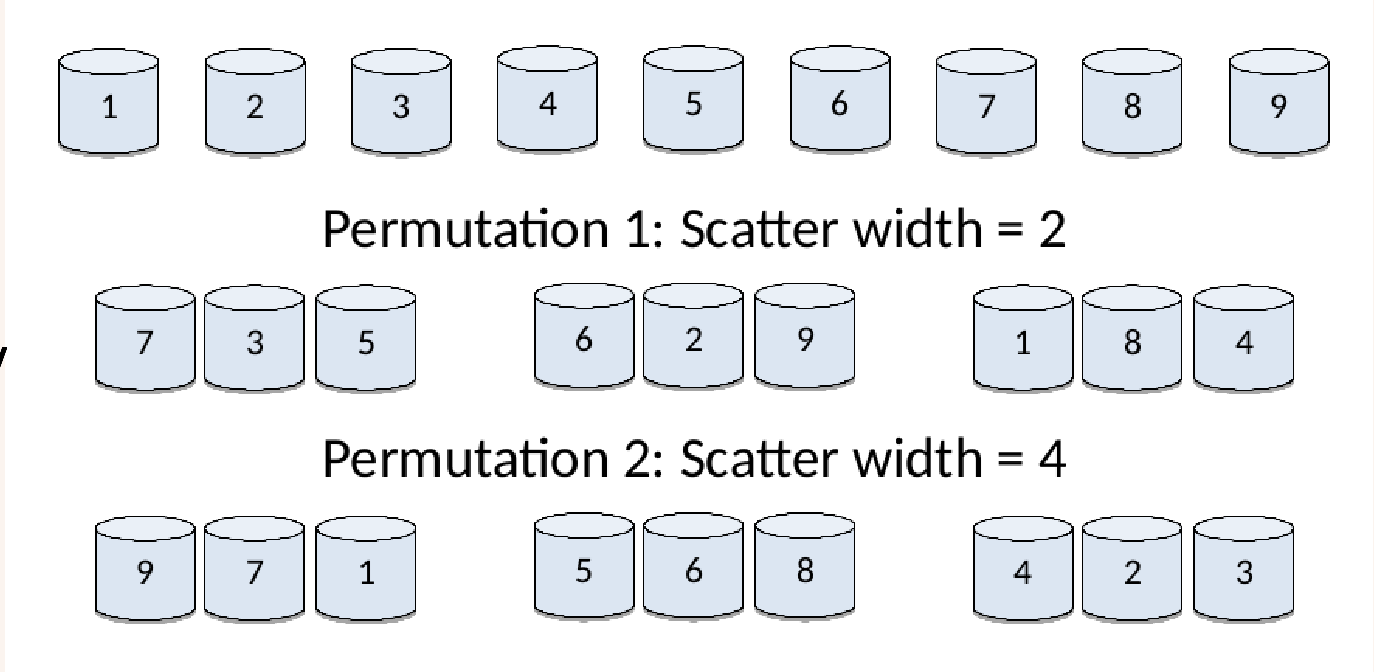
- Only 3 copysets - {1,5,7}, {2,4,9}, {3,6,8}
- Probability of data loss is very low

- **What it the issue?**
- We lose data very less frequently, but we would lose large amount of data
- Reconstruction can be done only from the fellow nodes in the copyset



Source: https://www.usenix.org/conference/atc13/technical-sessions/presentation/cidon

- Scatter Width – Number of nodes from which reconstruction can be done
- Create random permutations and group consecutive disks in a copyset
- While writing a block, choose a primary node randomly
- Choose a permutation corresponding to a block id, and use the copyset for the primary node



Source: https://www.usenix.org/conference/atc13/technical-sessions/presentation/cidon

- Reconstruction load is now distributed

- In tectonic, the block layer and the rebalancer services together attempt to maintain a fixed copyset count.

Tectonic in Production: Observations and Lessons

- Performance comparable to specialized filesystems
- Data warehouse request spikes handled by surplus resources
- Only name layer metadata was seen to be hotspot (1% time)
  - Solution: Master uses the metadata using list API and informs the worker nodes
- Client Library access data directly:
  - Alternative design would use a frontend proxy – Avoid an extra network node hop
  - But bugs in library become bug in application
- Reconstruction Storm:
  - Prevent reconstruction to 10%(tuned) of total reads

Tradeoff

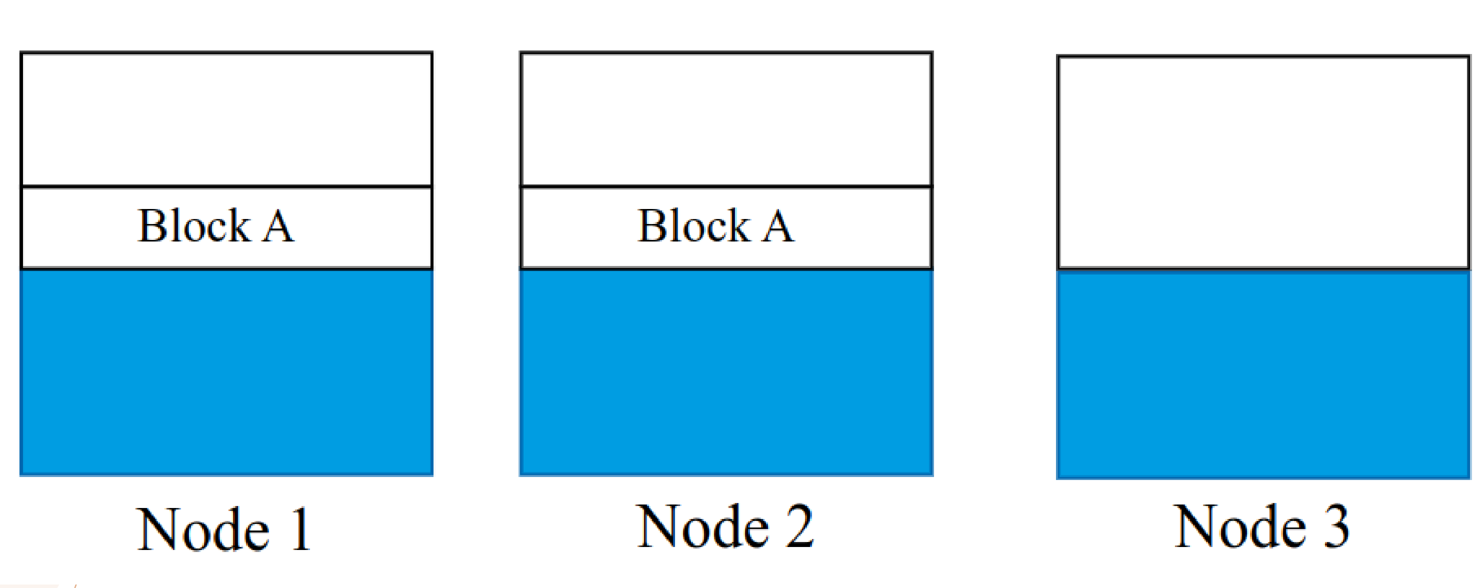
- Higher Metadata Latency
  - As such no direct solution
  - Applications need to adjust how certain metadata operations are handled
    - Parallelize if possible
- Hash partitioned metadata – avoids hotspot
  - No recursive list API – Client needs to build a wrapper for it
    - You don’t have updated directory usage information (updated periodically)
      - Commands like du(directory utilization) not available

Does every Application in Meta use Tectonic?

- Not every service at Meta use Tectonic (Graph storage, Key-Value Storage)
- Many applications communicate with tenants which use Tectonic underneath.
    - Not useful to design a client library for each application

Observations

- Token based single writer semantics:
  - May lead to deadlock (Alternating retry)
- The eventual consistency model for the partial block appends has performance/consistency issues



THANK YOU