Fire-Flyer File System (3FS)

A high-performance distributed file system designed to address the challenges of AI training and inference workloads

DeepSeek-AI

Presenter: Chongzhuo Yang, Jiahao Li

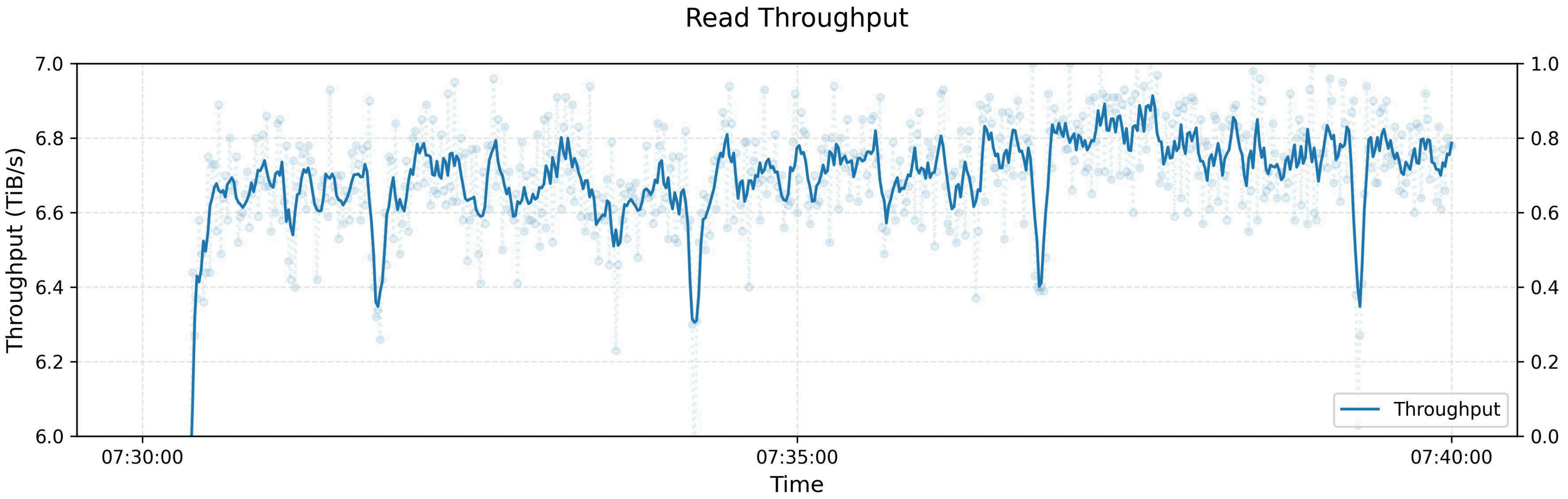
1.1. Fire-Flyer File System

The 3FS is a distributed file system designed for **AI workloads**.

Targeted Workloads:

- Data Preparation
- Dataloaders
- Checkpointing
- KV Cache

1.1. Fire-Flyer File System



The final aggregate read throughput reached approximately **6.6 TiB/s** with 180 storage nodes.

1.1. Fire-Flyer File System

Node Configuration:

- 200Gbps InfiniBand NICs $\times$ 2
- 14TiB SSDs PCIe 4.0x4 $\times$ 16

The 3FS achieves **75%** of the ideal read throughput with 180 storage nodes.

The final aggregate read throughput reached approximately **6.6 TiB/s**.
Each SSD delivers an average bandwidth of **2.347 GiB/s**.

Bandwidth Reference

- RDMA bandwidth for each SSD: **3.125GiB/s**
- NVMe read throughput: 6.33GiB/s (SEQ), **3.78GiB/s**(RND)

1.2. Diverse Workloads

The 3FS is a distributed file system designed for **AI workloads**.

Workloads	R/W	Descriptions
Data Preparation	Mixed R/W	3.66 TiB/min sort throughput with 25 nodes
Dataloaders	Read	approximately 6.6 TiB/s with 180 nodes
Checkpointing	Write	estimated 1.63 TiB/s with 180 nodes[1]
KV Cache	Mixed R/W	up to 40 GiB/s read throughput

The 3FS is **high-performance** and **scalable**.

- Use disaggregated chunk servers for data.
- Use distributed key-value store for metadata.

[1] estimated based on read throughput

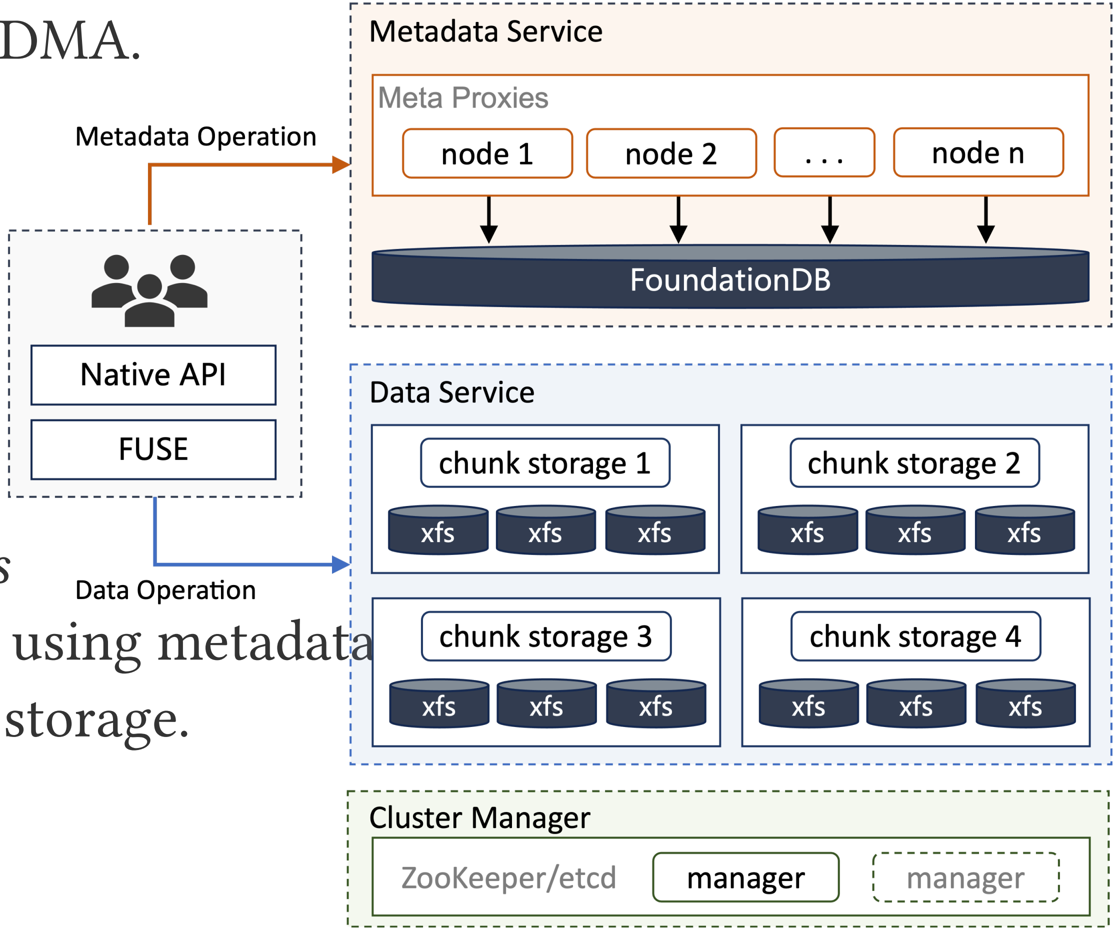
1.3. System Architecture

University of Science and Technology of China

The components are all connected via RDMA.

Components

- Client
- Metadata Service
- Storage Service
- Cluster Manager



Key Points: *disaggregated chunk servers*

1. Client can get chunk/replica location using metadata
2. Client can read data from any chunk storage.

2.1. Overview

The data chunk service needs to provide **load balance** and **data consistency**.

Load Balance:

- Chunking and striping
- Balanced chain table

Data Consistency:

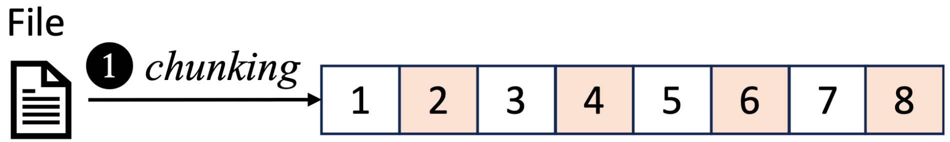
- Use CRAQ (chain replication) to replicate chunks

2.2. Data Write Pipeline

Data write can be divided into *three* steps.

Step 1: *chunking*

Files are divided into equally sized *chunks*.



2.2. Data Write Pipeline

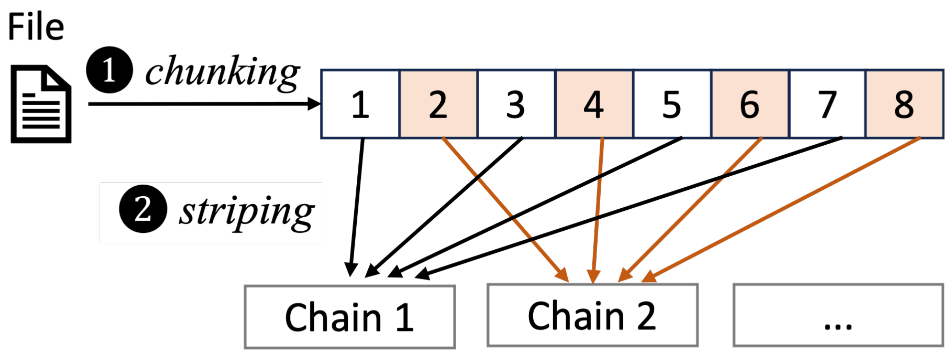
Data write can be divided into *three* steps.

Step 1: *chunking*

Files are divided into equally sized *chunks*.

Step 2: *striping*

Chunks stripe across multiple *chains*.



2.2. Data Write Pipeline

Data write can be divided into *three* steps.

Step 1: *chunking*

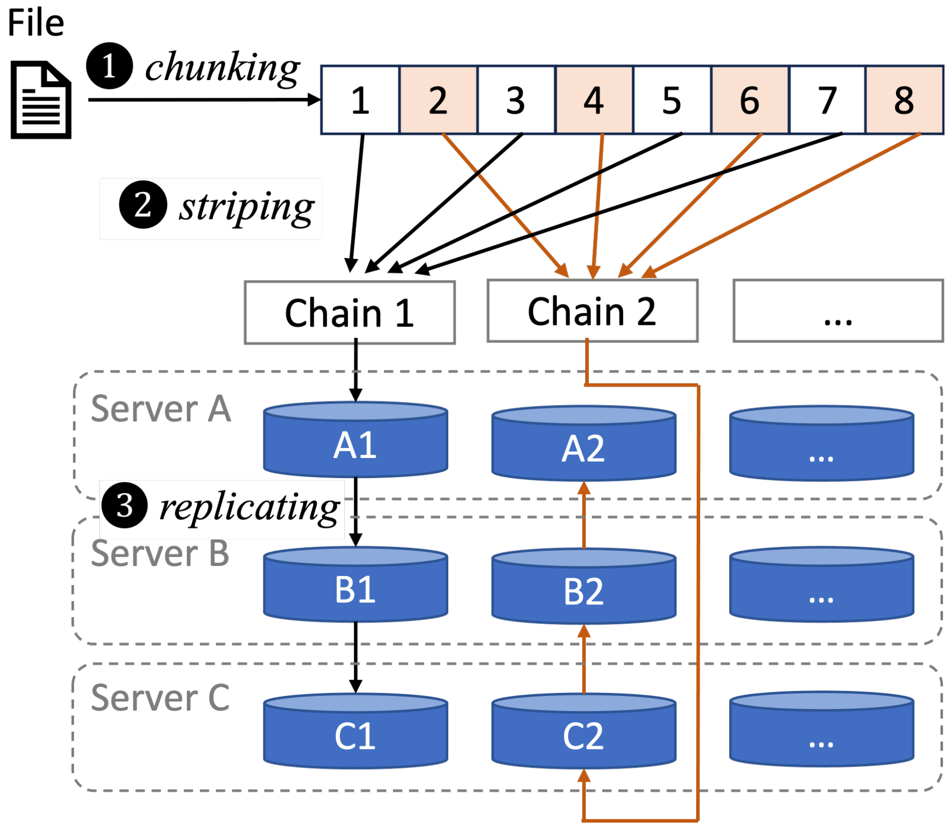
Files are divided into equally sized *chunks*.

Step 2: *striping*

Chunks stripe across multiple *chains*.

Step 3: *replicating*

Chunks are replicated using CRAQ (3 replicas).



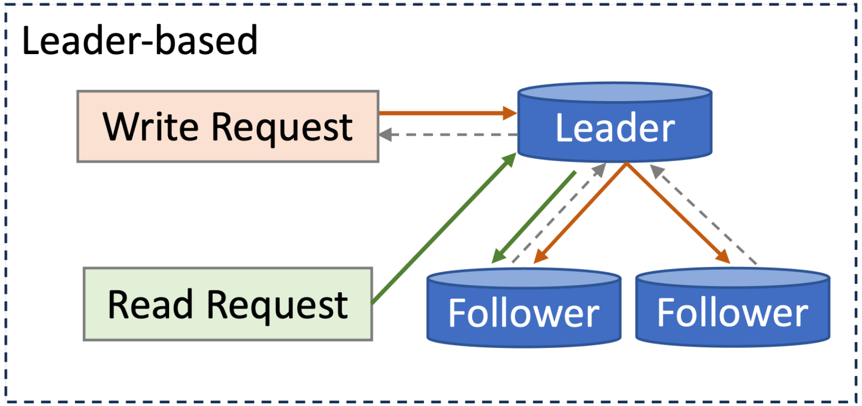
2.3. Data Replication

Comparison between two replication methods:

Leader-Based:

The leader will forward chunks to followers.

- ❌ Read needs the leader for replica info.



2.3. Data Replication

Comparison between two replication methods:

Leader-Based:

The leader will forward chunks to followers.

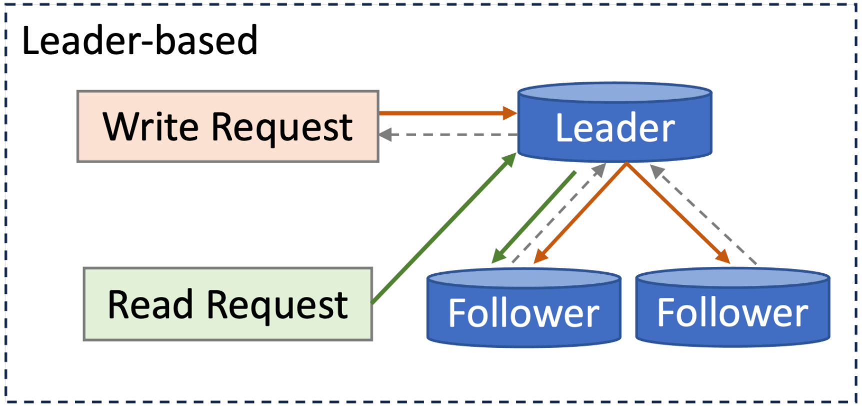
- ❌ Read needs the leader for replica info.

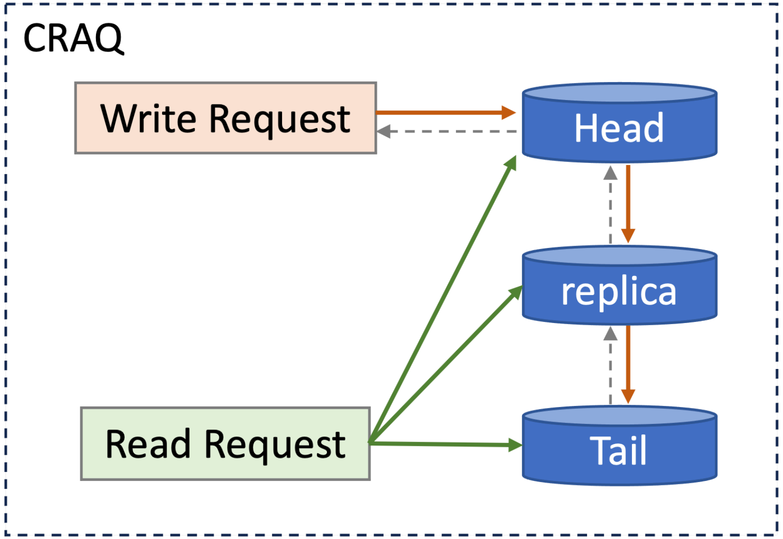
CRAQ: *Chain Replication with Apportioned Queries.*

Chunks are replicated over a chain of storage targets.

- ✅ Chunks can be read from any storage target.

CRAQ can **save one request** to the leader!

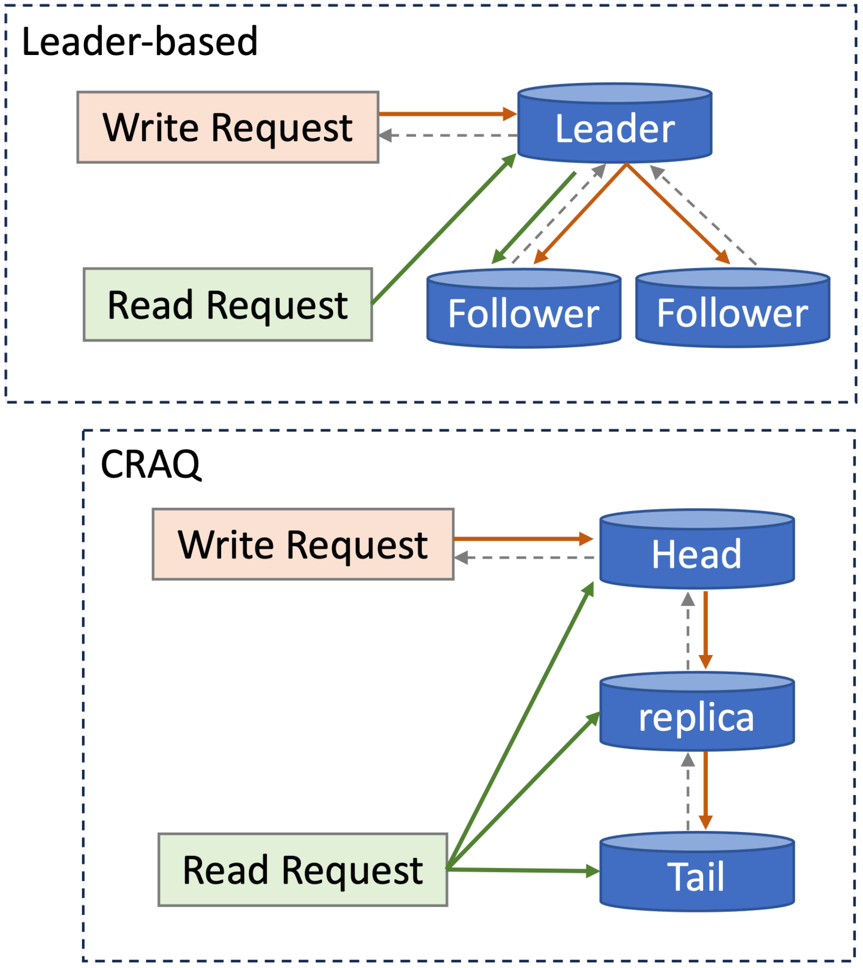




2.3. Data Replication

The CRAQ is ***read-friendly.***

	Failed	Write	Read
Leader-based	leader	❌	❌
Leader-based	other	✅	✅
CRAQ	head	❌	✅
CRAQ	other	❌	✅



2.4. Load Balance during Recovery

中国科学技术大学
University of Science and Technology of China

The chain consists of multiple **storage targets**.

Chain Table Example

- 6 SSDs (A, B, C, D, E, F),
- 5 targets in each SSD (1, 2, 3, 4, 5),
- 10 chains,
- 3 replicas.

Chain	Version	Target 1 (head)	Target 2	Target 3 (tail)
1	1	A1	B1	C1
2	1	D1	E1	F1
3	1	A2	B2	C2
4	1	D2	E2	F2
5	1	A3	B3	C3
6	1	D3	E3	F3
7	1	A4	B4	C4
8	1	D4	E4	F4
9	1	A5	B5	C5
10	1	D5	E5	F5

2.4. Load Balance during Recovery

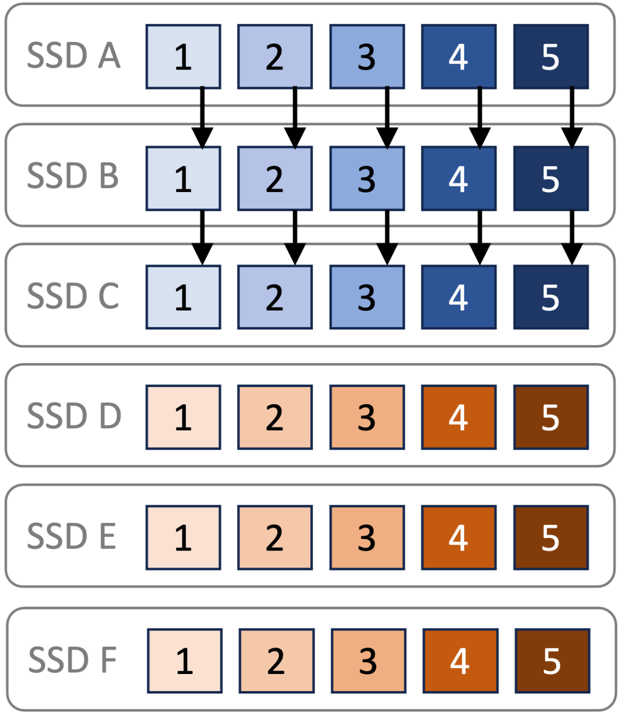
中国科学技术大学
University of Science and Technology of China

The chain consists of multiple **storage targets**.

Chain Table Example

- 6 SSDs (A, B, C, D, E, F),
- 5 targets in each SSD (1, 2, 3, 4, 5),
- 10 chains (different colors),
- 3 replicas.

Each SSD will handle 1/6 requests **evenly**.



2.4. Load Balance during Recovery

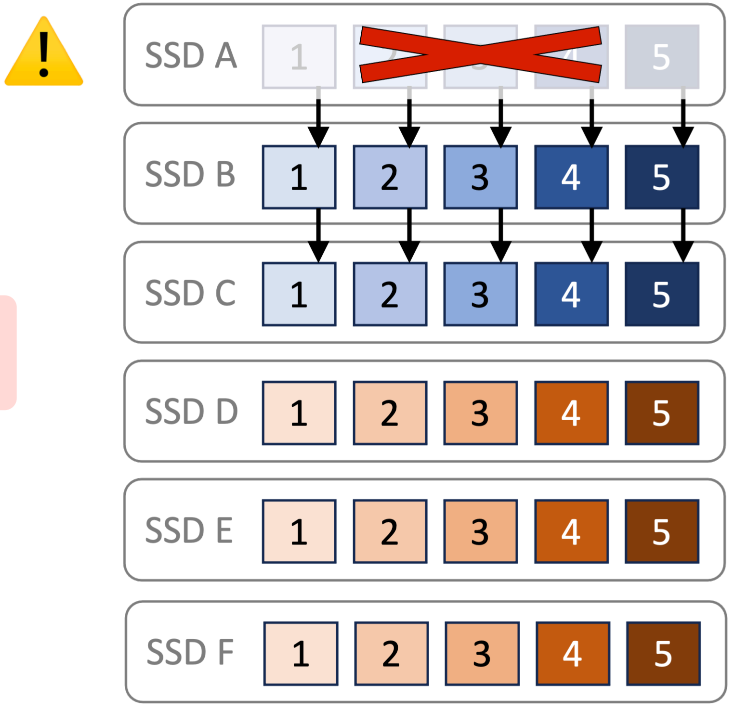
University of Science and Technology of China

For CRAQ, chunks remain readable if a target is down.

How about when SSD A is broken ?

Read requests to SSD A will be redirected evenly to SSD B and C.

The load of SSD B and C will be ***increased by 50%***.

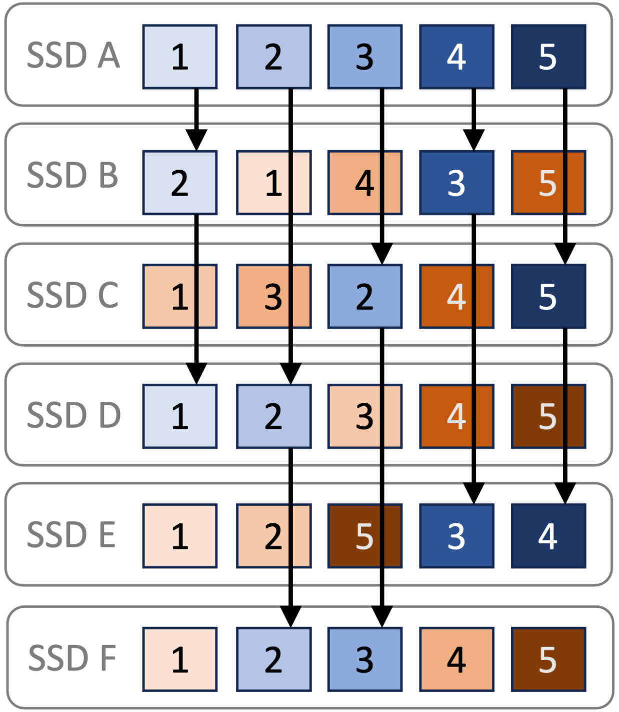


2.4. Load Balance during Recovery

The 3FS uses a special chain table to achieve balanced traffic during recovery.

Balanced Chain Table

Each SSD will handle requests **evenly** before failure.



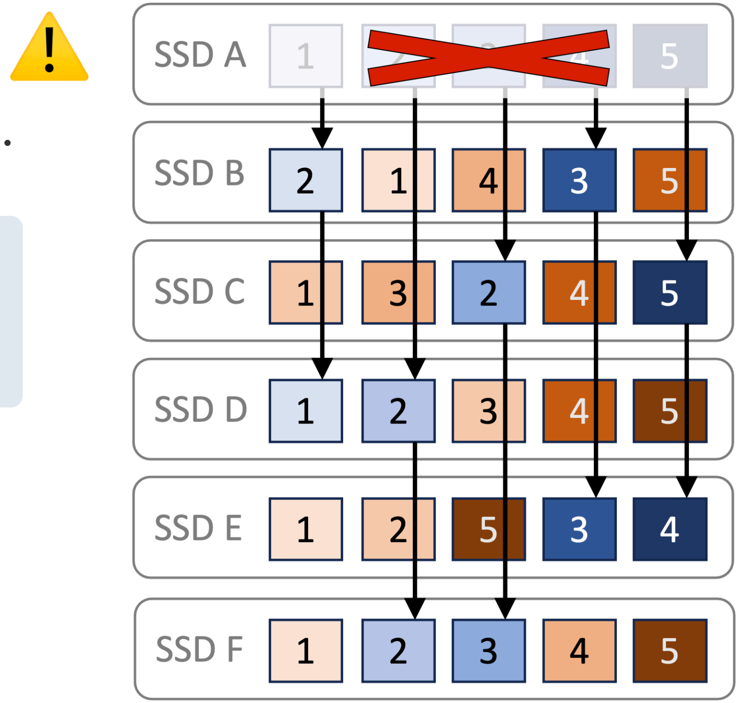
* *The target order within SSD is adjusted for a clearer chain view.*

2.4. Load Balance during Recovery

The 3FS uses a special chain table to achieve balanced traffic during recovery.

Balanced Chain Table

The chains that include SSD A are all marked in blue.



The load on remaining SSDs will remain balanced, increasing by only **20%**.

* *The target order within SSD is adjusted for a clearer chain view.*

2.5. CRAQ based Chunk Storage

中国科学技术大学
University of Science and Technology of China

The 3FS use the CRAQ as replication method of chunk storage which is more friendly on **read** workloads.

Pros.

- The chunks are **readable** when any node is down.
- The CRAQ can avoid the bottleneck of leader's network.

2.5. CRAQ based Chunk Storage

The 3FS use the CRAQ as replication method of chunk storage which is more friendly on **read** workloads.

Pros.

- The chunks are **readable** when any node is down.
- The CRAQ can avoid the bottleneck of leader's network.

Cons.

- Chunk servers use local file system to manage SSDs.
- The write throughput may be much lower than other design.

中国科学技术大学
University of Science and Technology of China

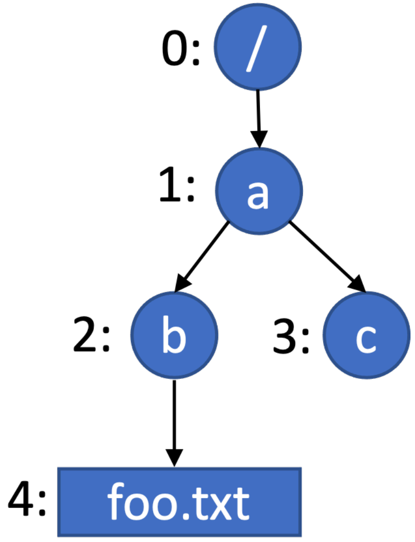
The metadata is stored in FoundationDB.

Tree layout

The path `'/a/b/foo.txt'` will be resolved as follows:

1. find `'/a'` by key = `'DENT'+0+'a'`.
2. find `'/a/b'` by key = `'DENT'+1+'b'`.
3. find `'/a/b/foo.txt'` by key = `'DENT'+2+'foo.txt'`.

Key			Value	
prefix	dir_id	name	id	type
'DENT'	0	'a'	1	DIR
'DENT'	1	'b'	2	DIR
'DENT'	1	'c'	3	DIR
'DENT'	2	'foo.txt'	4	FILE



3.1. Metadata Layout

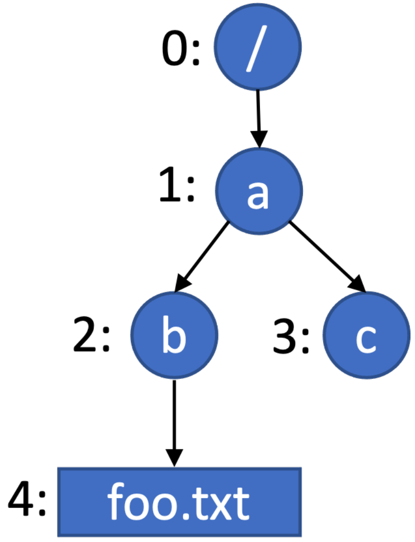
The metadata is stored in FoundationDB.

Inode Attributes: permissions, file size, dir layout, …

The inode attributes can be found using ID.

- find metadata of `foo.txt` by key = `'INOD'+4`

Key		Value					
prefix	id	type	length	name	chain table	chunk size	...
'INOD'	0	DIR	-	'/'	-	512	
'INOD'	1	DIR	-	'a'	-	512	
'INOD'	2	DIR	-	'b'	-	512	
'INOD'	3	DIR	-	'c'	-	512	
'INOD'	4	FILE	4096	'foo.txt'	02	512	



The metadata design of 3FS is **simple** and **practical** based on FoundationDB.

3.2. Metadata Design

The metadata design of 3FS is **simple** and **practical** based on FoundationDB.

Pros.

- Using inode id as the key can be well adapted with FUSE.
- The little-endian byte order of inode ids provides **better** load balance.
- The FoundationDB can support SSI, so it's easy to implement the `rename`.

3.2. Metadata Design

The metadata design of 3FS is **simple** and **practical** based on FoundationDB.

Pros.

- Using inode id as the key can be well adapted with FUSE.
- The little-endian byte order of inode ids provides **better** load balance.
- The FoundationDB can support SSI, so it's easy to implement the `rename`.

Cons.

- The inodes within a directory can **not** be listed by range query because the directory entry only contains the inode id.
- Some operations (e.g., `create/unlink`) may use cross-shard transactions and introduce high overheads.

4. Conclusion

The 3FS is a distributed file system designed for **AI workloads**.

Workloads	**R/W**	**Descriptions**
Data Preparation	Mixed R/W	3.66 TiB/min sort throughput with 25 nodes
Dataloaders	Read	approximately 6.6 TiB/s with 180 nodes
Checkpointing	Write	estimated 4.9 TiB/s with 180 nodes[1]
KV Cache	Mixed R/W	up to 40 GiB/s read throughput

The 3FS is **high-performance** and **scalable**.

- Use decentralized chunk servers for data.
- Use distributed key-value store for metadata.

[1] estimated based on read throughput

Thanks