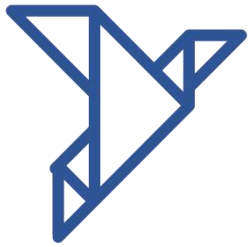


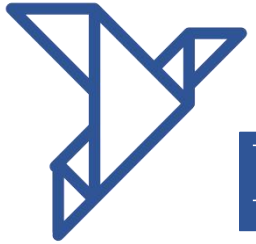
Origami: Efficient ML-Driven Metadata Load Balancing for Distributed File System

Yiduo Wang* Wenda Tang Linghang Meng Liang Li Jie Wu

China Telecom Cloud Computing Research Institute

ICPP`25 Paper Session 14: Software, AI & Performance;
Thursday, September 11

1. Why is metadata management important yet inefficient?

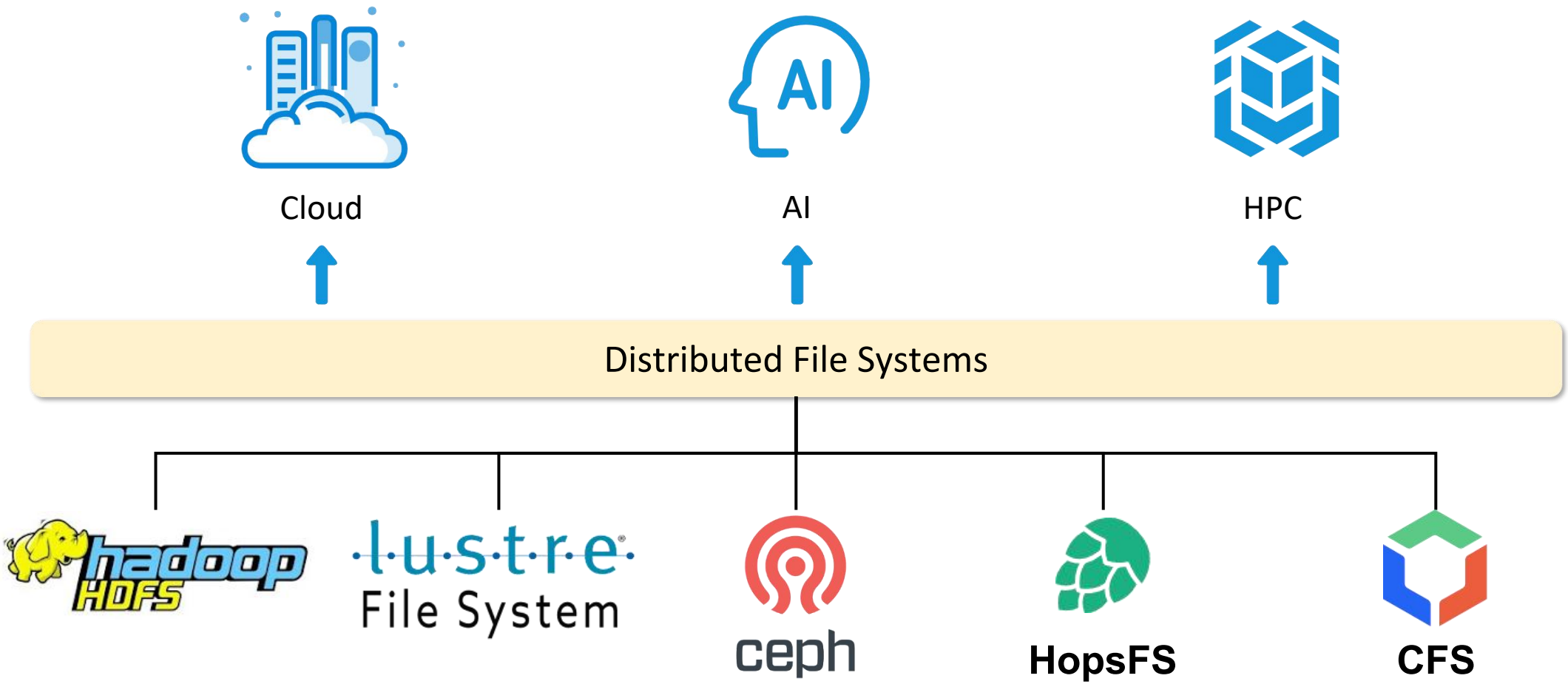


Origami: Efficient ML-Driven Metadata Load Balancing for Distributed File System

2. Do we really need metadata load balancing?

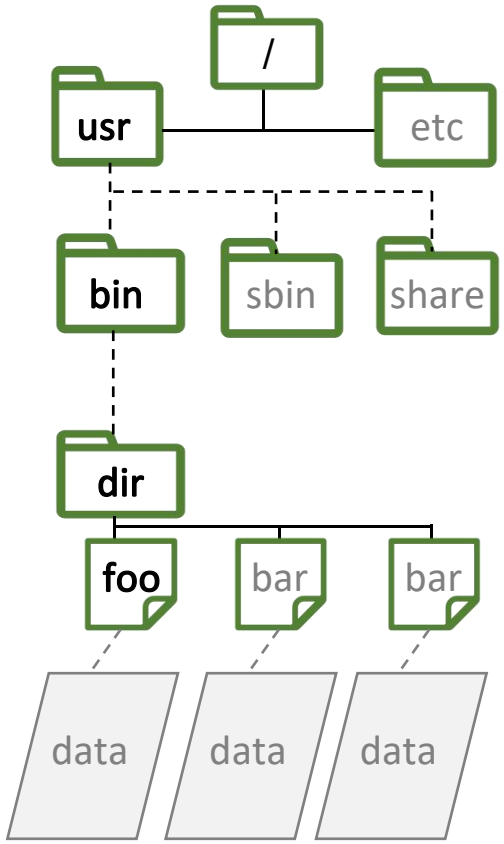
3. How to make load balancing efficient, via ML?

Distributed File System (DFS)



File System and Metadata

- The logic structure of file system: metadata and data



Metadata:
a tree/DAG

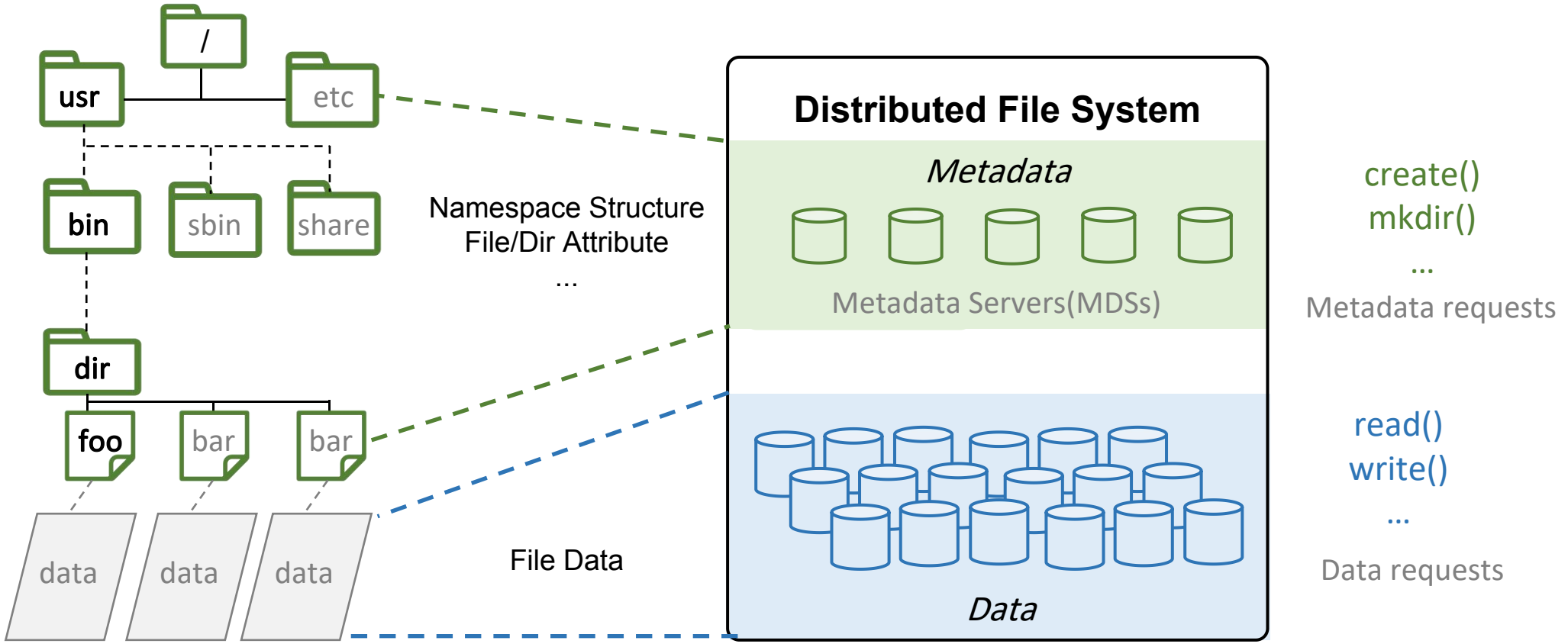
Namespace Structure (name, parent inode...)
File/Dir Attribute (access time, permission...)
...

Data

File Data

Modern Architecture of DFS

- Common architecture: metadata and data decoupled



Modern Architecture of DFS

- Distributed file system meets new challenges

New Scale

DFS capacity: hundreds of billions

DFS QPS: millions per second

New Hardware

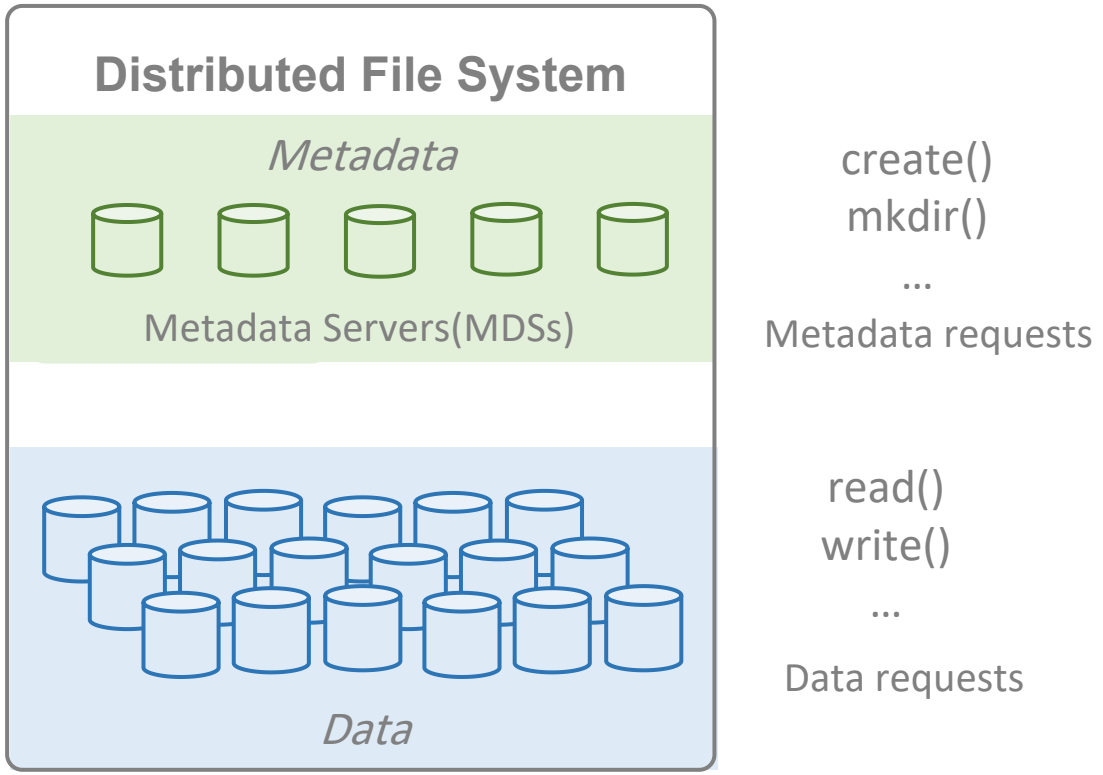
SSD Bandwidth: ~15GB/s

NIC Bandwidth: 400Gb+

New Pattern

Average file size: < 1MB

Metadata operation ratio: >90%



Modern Architecture of DFS

- Distributed file system meets new challenges

New Scale

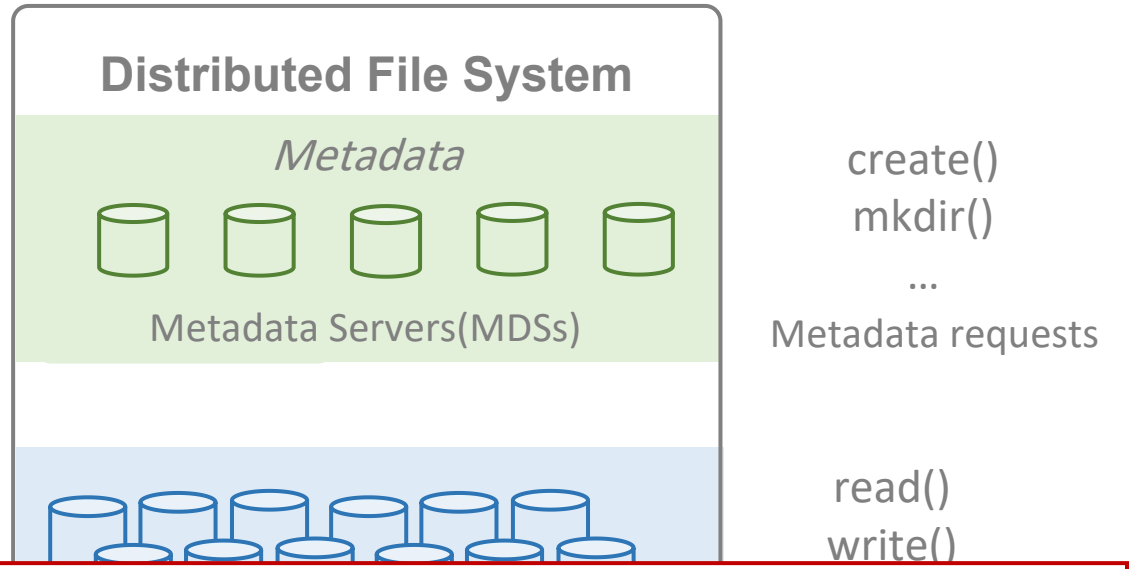
DFS capacity: hundreds of billions

DFS QPS: millions per second

New Hardware

SSD Bandwidth: ~15GB/s

NIC Bandwidth: 400Gb+

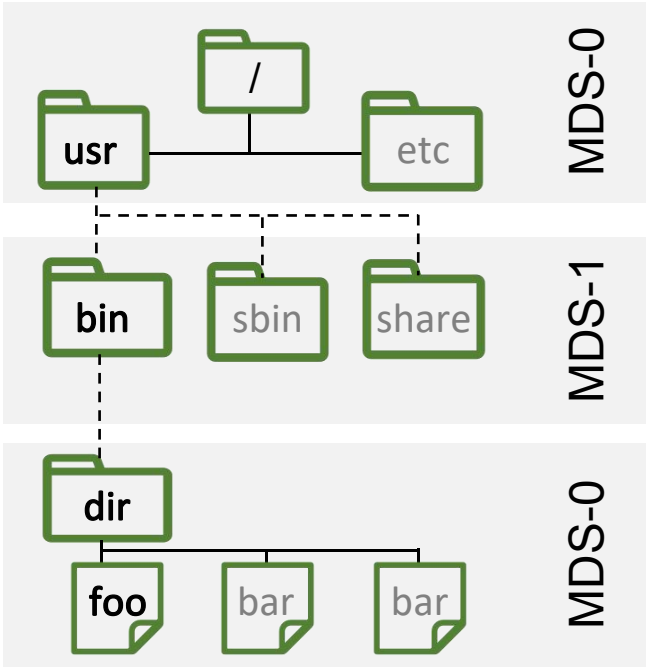


Metadata operation ratio: >90%

Metadata becomes the primary bottleneck of modren DFS

Scaling Metadata of DFS

- Breaking the metadata bottleneck: multi-MDSs collaboration



- ✓ Overcoming Single-MDS Capacity Limits
- ✓ Enabling Multi-MDS Parallel Processing
- ✓ Realizing Full In-Memory Metadata

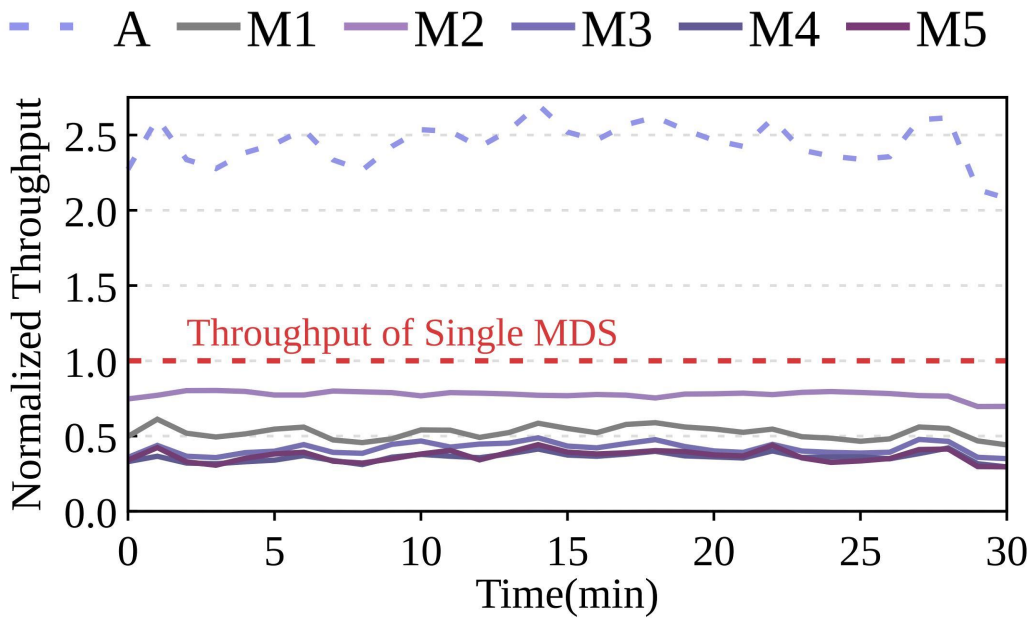
And, what must we give in return?

“It promises great power, but it exacts a terrible price.”

—Medivh

Scaling Out—But Not Efficiently

- Evaluation on a CephFS cluster with 5-MDSs configuration
  - Read-only metadata-heavy workload, saturate metadata service with many clients
  - Partition metadata evenly via hashing

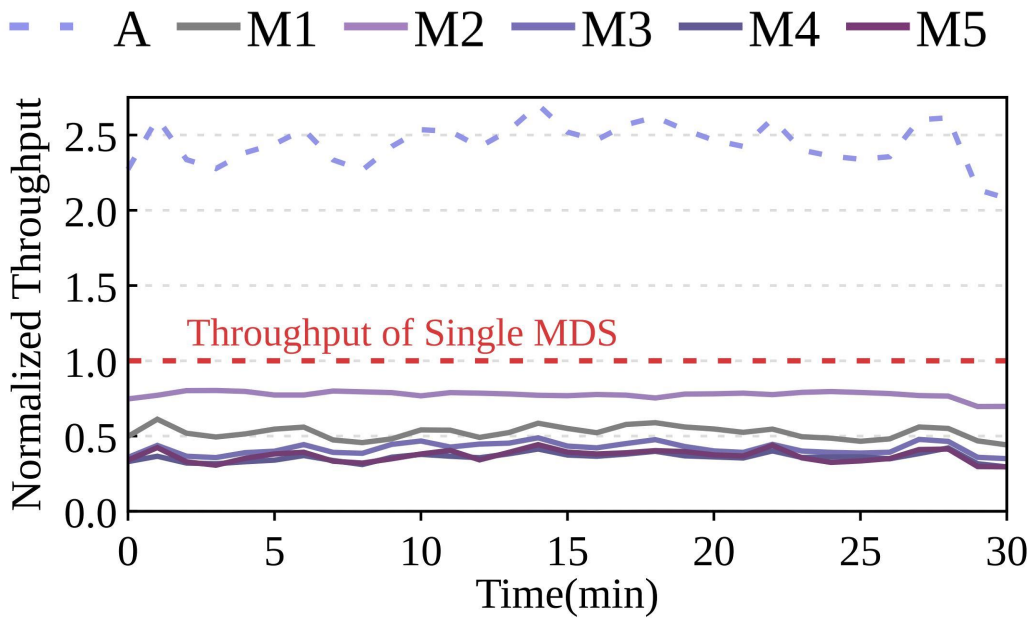


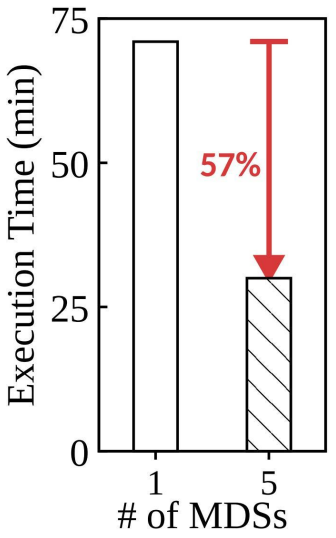
❖Known inefficiency:
metadata load imbalance.

❖More critical:
all MDS slow down.

Scaling Out—But Not Efficiently

- Evaluation on a CephFS cluster with 5-MDSs configuration
  - Read-only metadata-heavy workload, saturate metadata service with many clients
  - Partition metadata evenly via hashing





+400% hardware resource

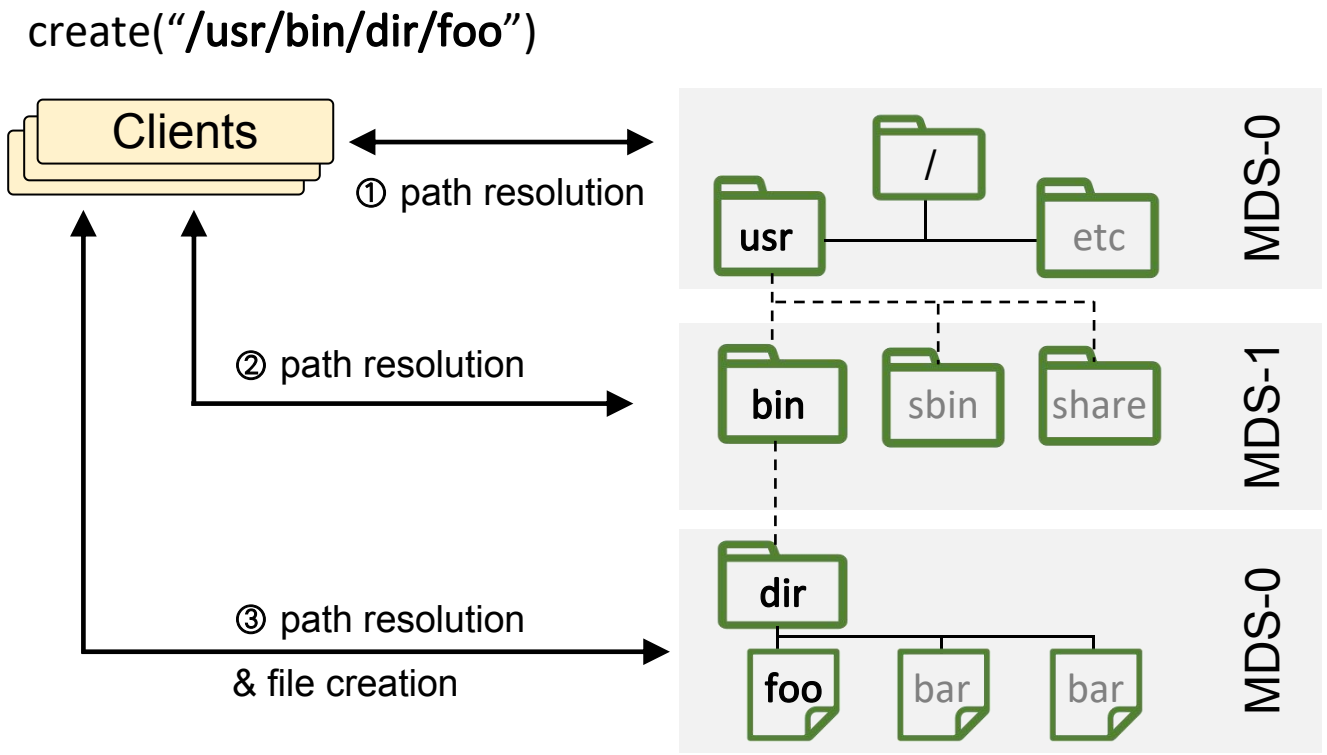
-57% Job completion time

The Cost of Scaling: Degraded Locality

- Metadata access pattern: **path resolution first**

The Cost of Scaling: Degraded Locality

- Metadata access pattern: **path resolution first**



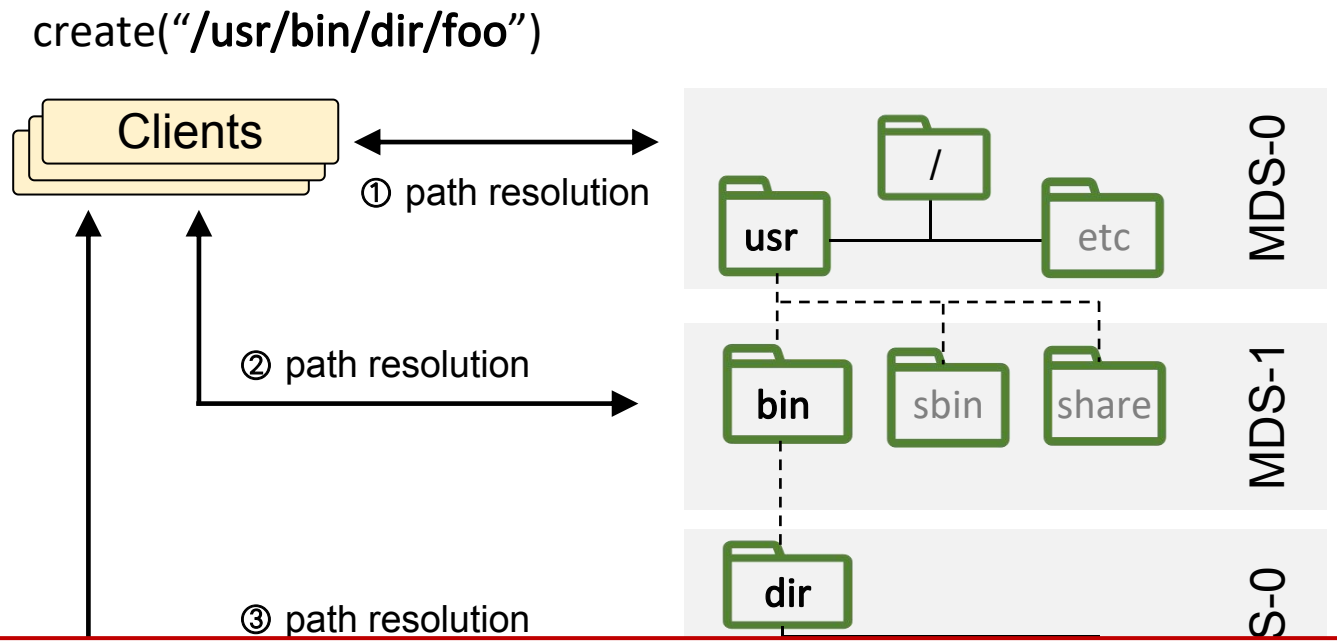
Partition Metadata in 3 MDSs

- ✓ 3x hardware process
- ✓ balanced load and capacity
- ✗ 3x RPC (remote process call)
- ✗ higher request completion time

The handling of metadata is similar, as cross-MDS processing requires additional coordination overhead such as distributed lock.

The Cost of Scaling: Degraded Locality

- Metadata access pattern: **path resolution first**



Partition Metadata in 3 MDSs

✓ 3x hardware process

✓ balanced load and capacity

✗ 3x RPC (remote process call)

Even Partitioning Considered Harmful!

Existing Work for Metadata Partitioning

Metadata partitioning

❖ Locality-preserved range partition

- CephFS SC'04
- IndexFS SC'14
- Lustre-DNE1 2014
- HopsFS FAST'17
- LocoFS SC'17
- ...

✓ light-weight coordination ✗ hotspots

❖ Hash-based partition

- Lustre-DNE2 2014
- Tectonic FAST'21
- CFS EuroSys'23
- CalvinFS FAST'15
- InfiniFS FAST'22
- ...

✓ load balance ✗ expensive path resolution

Metadata Load Balancing

❖ Programmability

- Mantle SC'15
- MUSE ICPADS'23

❖Spatio-Temporal Locality

- Lunule SC'21
- Lunule+ TOS'25

❖ ML-based Balancing

- LDPP ICPP'22
- LoADM DATE'24

When balance is prioritized and overhead is ignored, end-to-end gains remain limited.

Goal and Challengs

- **Our goal:** an efficient ML-driven metadata load balancing mechanism.
- **Key property: maximizing balance while minimizing additional overhead.**

- **We believe: balance should be a means, not the end**.

- **Challenge #1:** Measuring appropriate metrics.
- **Challenge #2:** Finding effective migration decisions.
- **Challenge #3:** Collecting statistics and validating models.

So, we want a best balancing strategy.

But what is “best balancing”?

What is the Best?

- Inspired by the classic *Bélády's algorithm* —
  for the **cache: if the access sequence is given, there exists an optimal cache replacement policy that maximizes the cache hit ratio.**

What is the Best?

- Inspired by the classic *Bélády's algorithm* —
  for the cache: if the access sequence is given, there exists an optimal cache replacement policy that maximizes the cache hit ratio.

- **Out insight:**
  for the **metadata**: if the future metadata access sequence is given, there should **also exist an optimal partitioning that minimizes end-to-end job completion time**(JCT).

We can train a model to learn it!

Measure RCT Instead of Balance

- To find the optimal strategy, we need to know
  how partitioning affect the request completion time(RCT)?
- For a single metadata request with:
  a path length of k and that is distributed among m distinct metadata partitions.

$$RCT = T_{meta} + m \cdot RTT + \sum_{i=1}^{m} Q_i$$

metadata **processing time**

RPCs incurred by **path resolution**

queuing at each partition

Measure RCT Instead of Balance

- To find the optimal strategy, we need to know
  how partitioning affect the request completion time(RCT)?
- For a single metadata request with:
  a path length of k and that is distributed among m distinct metadata partitions.

$$RCT = T_{meta} + m \cdot RTT + \sum_{i=1}^{m} Q_i$$

$$T_{meta} = T_{inode} \cdot (m + k) + T_{exec} + \begin{cases} RTT \cdot i, & * \\ T_{coor} \cdot \mathbb{I}(i > 0), \\ 0, \end{cases}$$

	list directory, lsdir
	namespace mutation, such as mkdir
	other operations

*migrating the sub-files/directories to i other MDSs

Measure RCT Instead of Balance

- To find the optimal strategy, we need to know how partitioning affect the request completion time(RCT)?
- For a metadata request calculate RCT by

$$RCT = T_{meta} + m \cdot RTT + \sum_{i=1}^{m} Q_i \qquad T_{meta} = T_{inode} \cdot (m + k) + T_{exec} + \begin{cases} RTT \cdot i, \\ T_{coor} \cdot \mathbb{I}(i > 0), \\ 0, \end{cases}$$

- How to determine the hyperparameters? **Sampling under high load**.
- Approximating JCT by casting it as a bin-packing problem.

Estimating the Benefits of Migration

- MetaOPT: Finding the near-optimal migration and benefit

Input: namespace partitioning + access sequence

Enumerate candidate subtree migrations

Comparing the JCT before/after migration

Output: Estimated benefit + best migration plan

Estimating the Benefits of Migration

- MetaOPT: Finding the near-optimal migration and benefit


Input: namespace partitioning + access sequence


Enumerate candidate subtree migrations

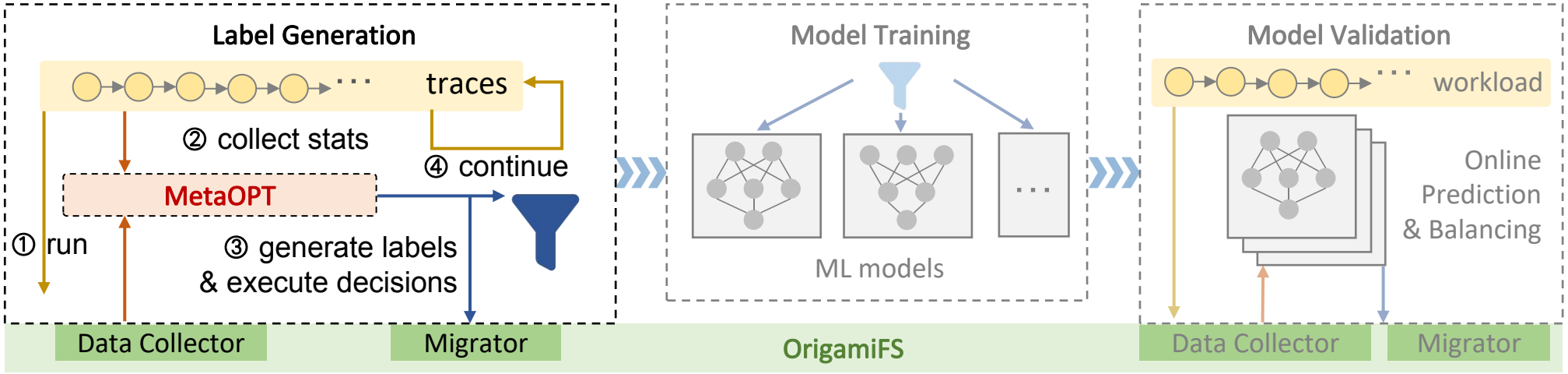

Comparing the JCT before/after migration


Output: Estimated benefit + best migration plan

- Is MetaOPT really Optimal?
- No, but the benefit gap to optimal < Δ

Origami Overview and Workflow

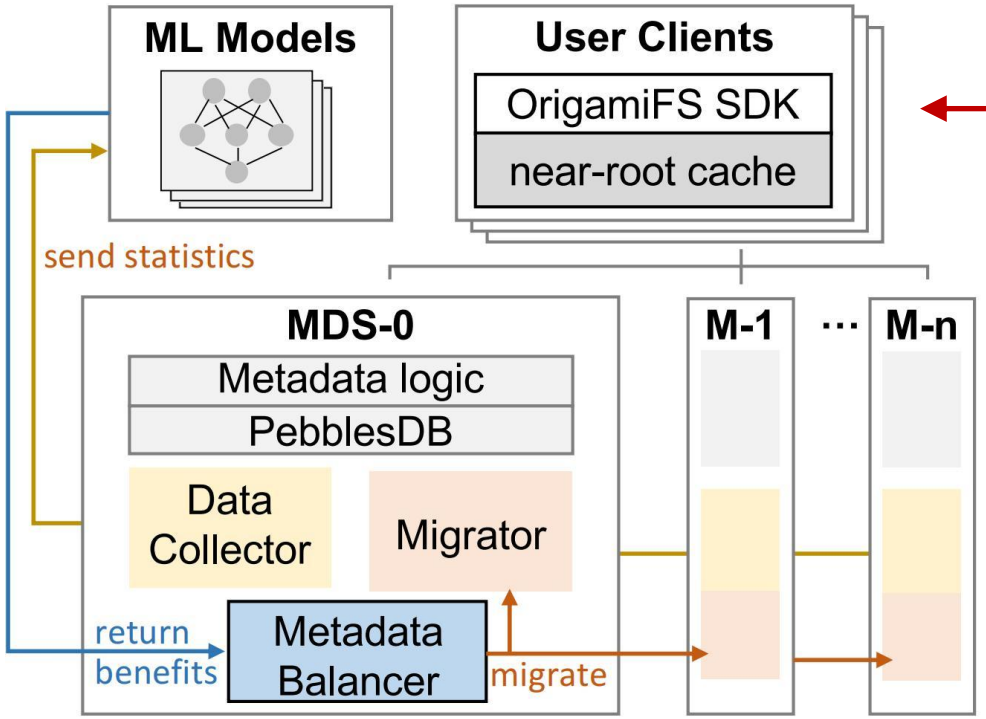
- Origami: a framework for training efficient balancing models



- **MetaOPT:** to offline generate labels and execute decisions
- **Data Collector:** to dump the runtime namespace state
- **Migrator:** execute external migration decisions

OrigamiFS Architecture

- A prototype of distributed metadata service



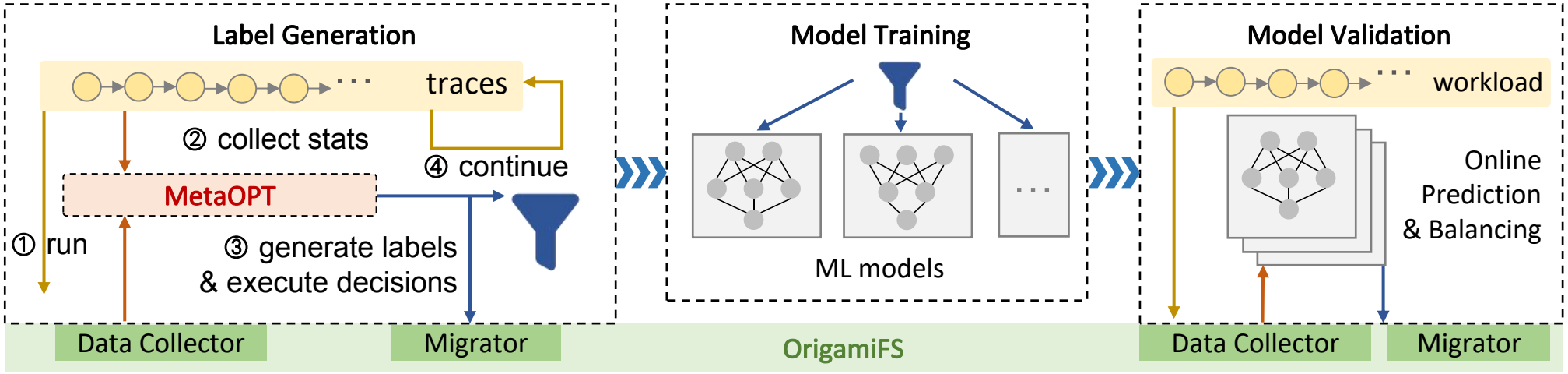
Cache for near-root hotspot mitigation

Metadata disaggregation for scalability

Specific interface for training and validation

Origami Overview and Workflow

- Origami: a framework for training efficient balancing models



- **Training feature:** namespace statistics
- **Training label:** migration benefit

About the Models

- Training feature

Type	Feature	Normalization
Namespace Structure	depth	by the max value
	# sub-files	
	# sub-dirs	
Metadata History	# read	by # total access in last epoch
	# write	
Derived Feature	read-write ratio	raw
	dir-file ratio	

- Model: LightGBM ~~GDBT, MLP~~

fastest yet precise enough

too slow

Questions to Answer

- How does Origami impact the metadata performance?
- Does Origami achieve better metadata load balance?
- Does Origami scale meta-ops better than SOTA?
- How does the near-root cache impact the performance?
- Does Origami more efficient than other methods?
- .....

Evaluation Setup

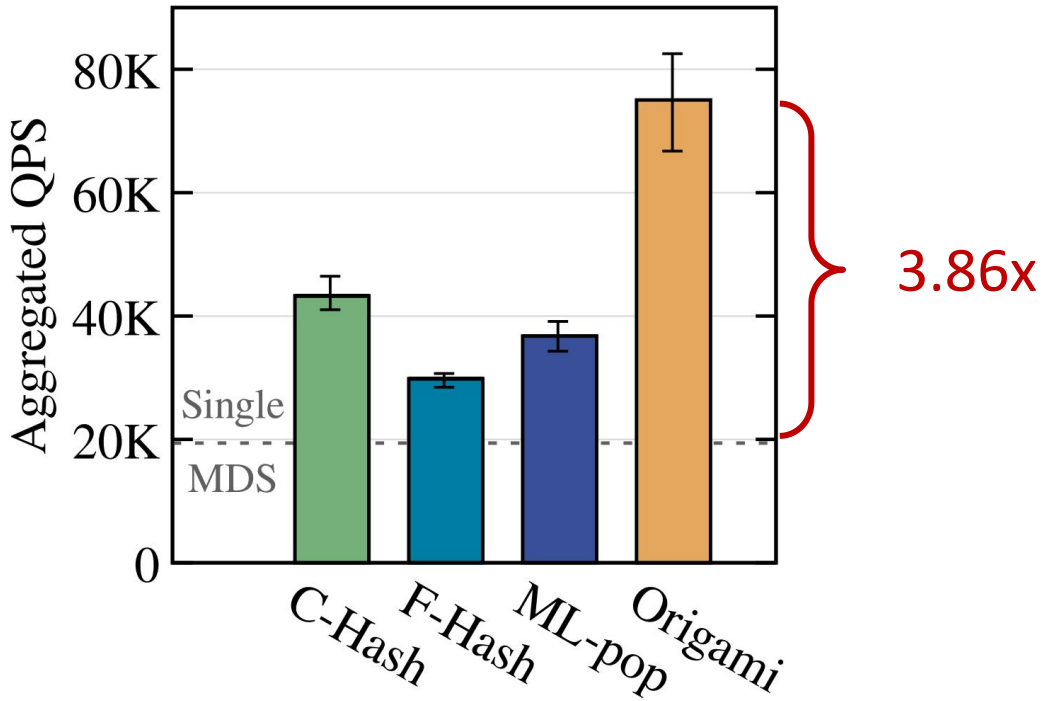
- Baseline Systems
  - **C-Hash**: coarse-grained hash the top level directory of namespace, like *HopsFS[FAST'17]* and ***CephFS-pin****[open-source]*;
  - **F-Hash**: a fine-grained approach that hashes all directory of the name-space, used in ***CFS****[EuroSys'23]*, ***InfiniFS****[FAST'22]* and ***Tectnoic****[FAST'21]*;
  - **ML-tree**: predicting the popularity of directory via lightGBM, and re-balancing by subtree*[DATE'24][ICPP'22]*.
- Deployment
  - 5MDSs, 5Clients with multiple threads to saturate metadata service

Evaluation Setup

- Workloads
  - Trace-RW: A large compilation task consisting of numerous complex metadata operations;
  - Trace-RO: A read-only web application access trace;
  - Trace-WI: A write-intensive trace from a productive cloud environment.

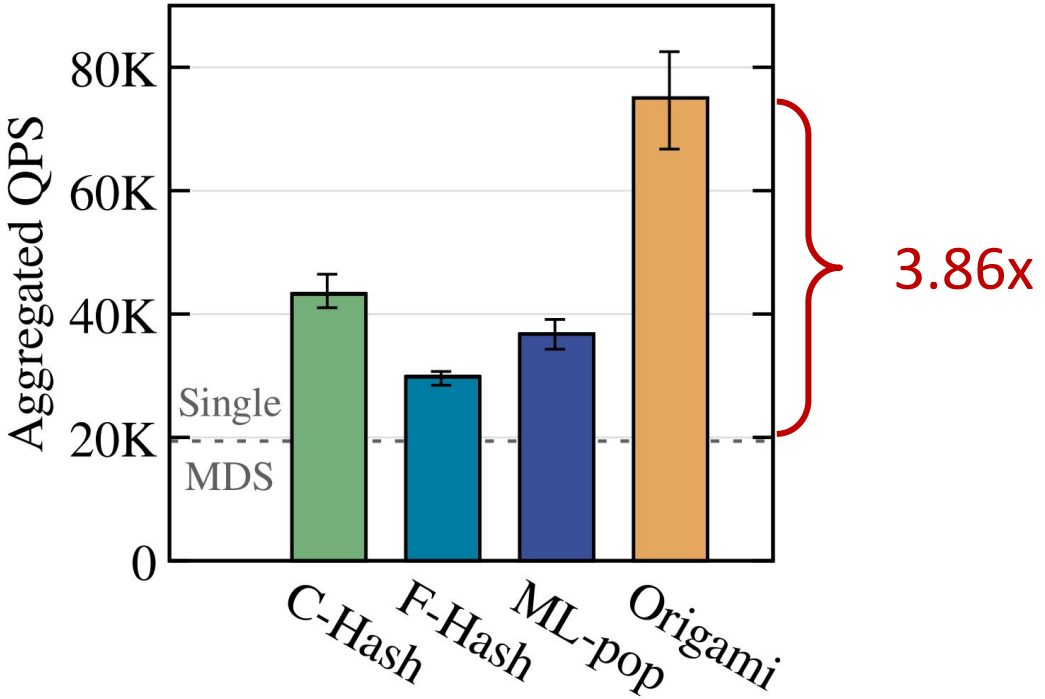
Evaluation: Overall Performance

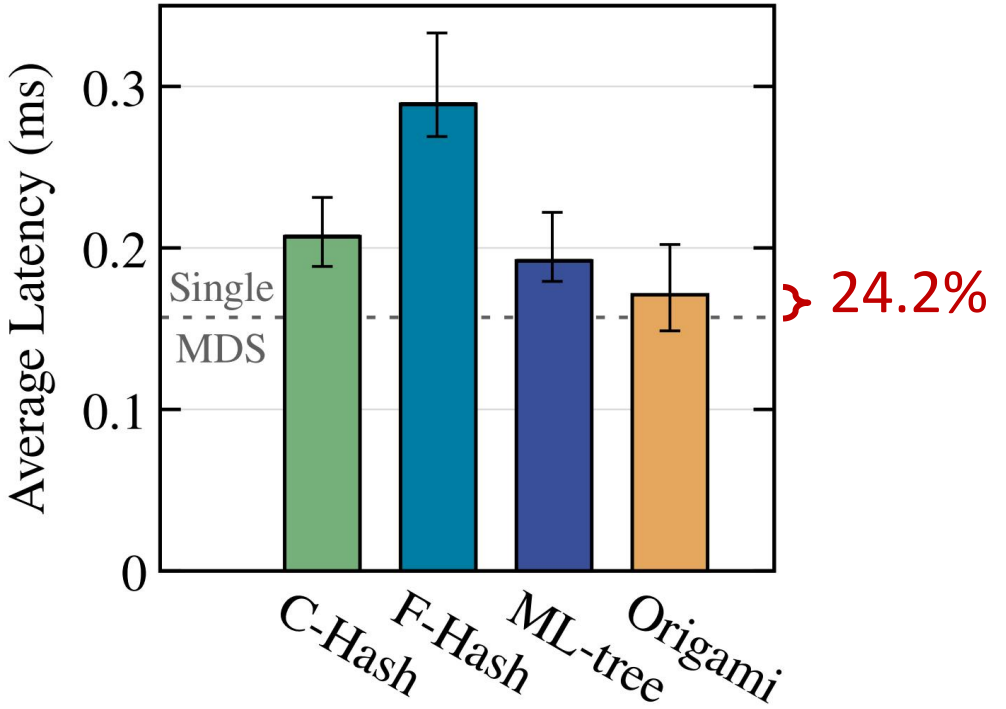
- Aggregated QPS under high load



Evaluation: Overall Performance

- Aggregated QPS under high load
- Average latency under sigle thread

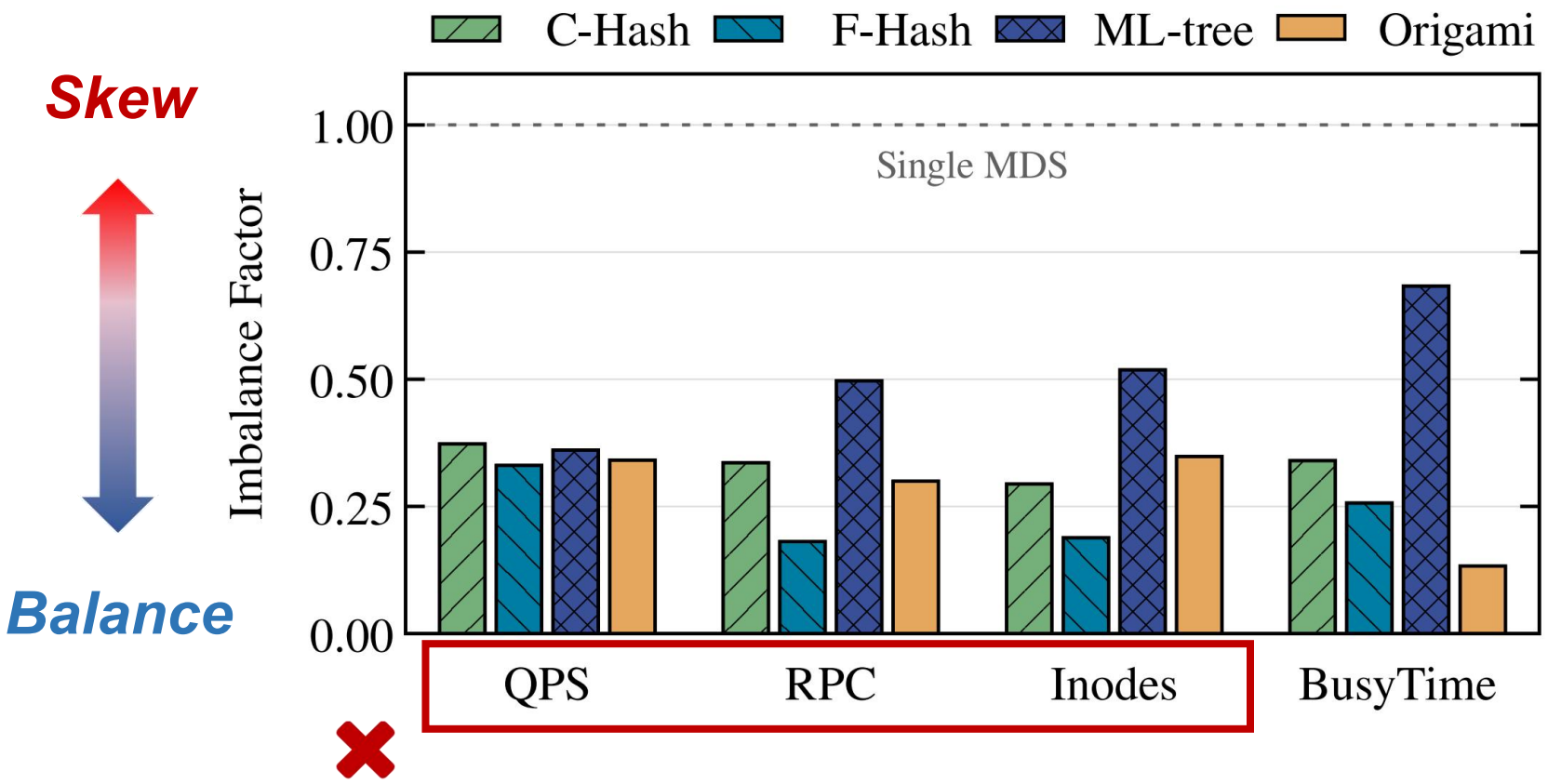




Near-linear scales under high load while
liitle overhead under light load

Evaluation: Balance Analysis

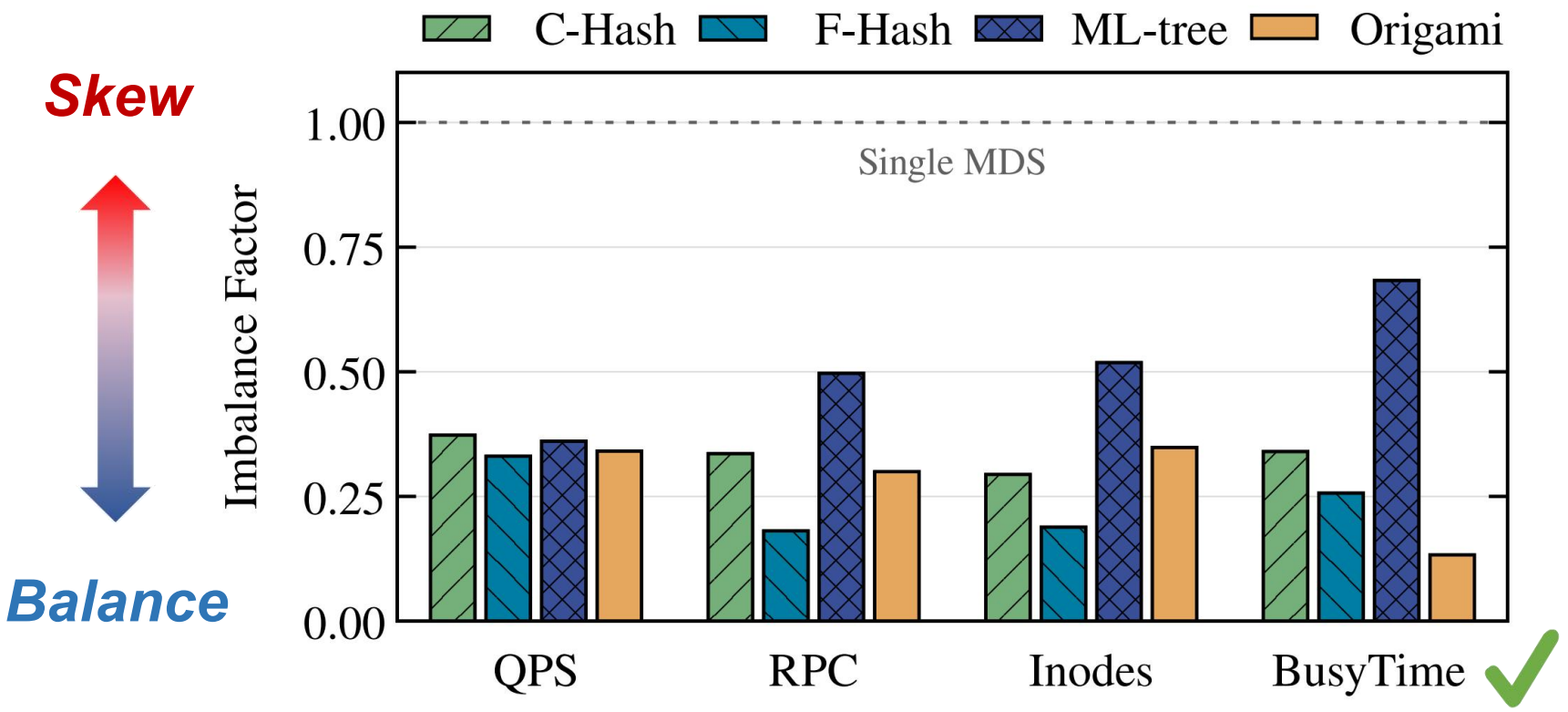
- Using *Imbalance Factor* to evaluate the balancing



Origami is NOT more balance than baselines

Evaluation: Balance Analysis

- Using *Imbalance Factor* to evaluate the balancing



Excellent balance not means excellent performance, keep all MDSs busy on the right way is the key.

Conclusion

- Problem: metadata load balancing is crucial but difficult
- Key insight: even partition (no matter inodes or requests) is NOT suit for metadata load balancing
- Key techniques of Origami
  - **MetaOPT algorithm** → **Find the near-optimal decision quickly**
  - **OrigamiFS** → **Collected statistic, train and validate models**
- Results
  - Improves end-to-end performance
  - Bring minimal overhead

Conclusion

- Problem: metadata load balancing is crucial but difficult
- Key insight: even partition (no matter inodes or requests) is NOT suit for metadata load balancing.
- Key techniques of Origami
  - **MetaOPT algorithm** → **Find the near-optimal decision quickly**
  - **OrigamiFS** → **Collected statistic, train and validate models**
- Results
  - Improves end-to-end performance
  - Bring minimal overhead

Thanks!

wangyd22@chinatelecom.cn

some other metadata work: https://yiduo.site

中国电信云计算研究院
China Telecom Cloud Computing Research Institute