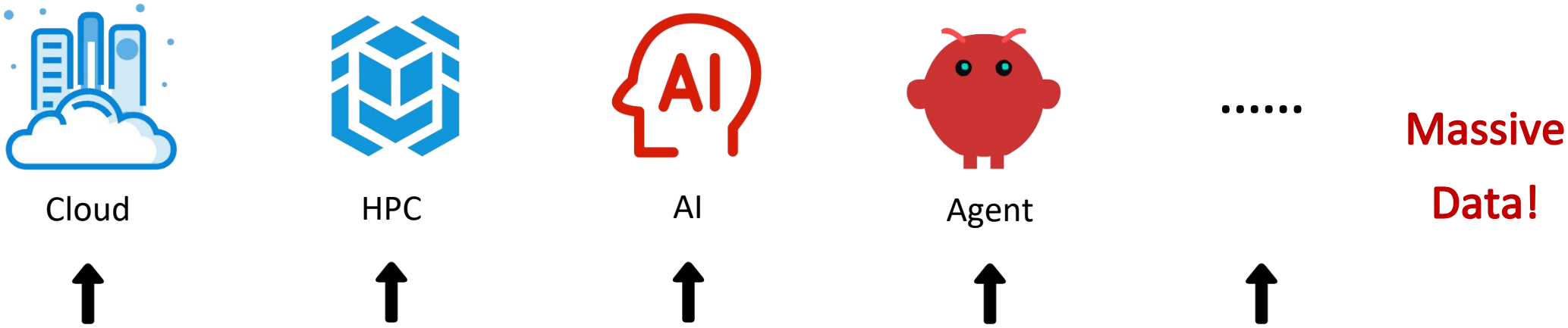
Accelerating Metadata Management of DFS via *Speculative Permission Checking*

Yiduo Wang* Linghang Meng Liang Li Jie Wu

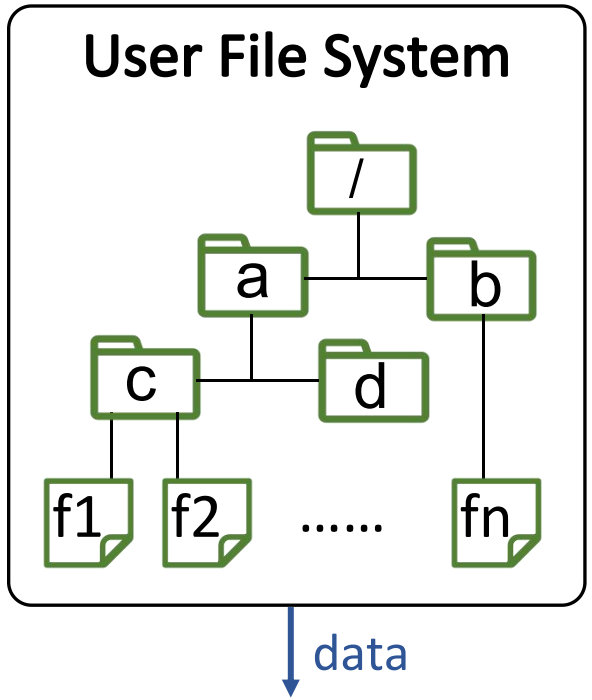
China Telecom Cloud Computing Research Institute

Distributed File System



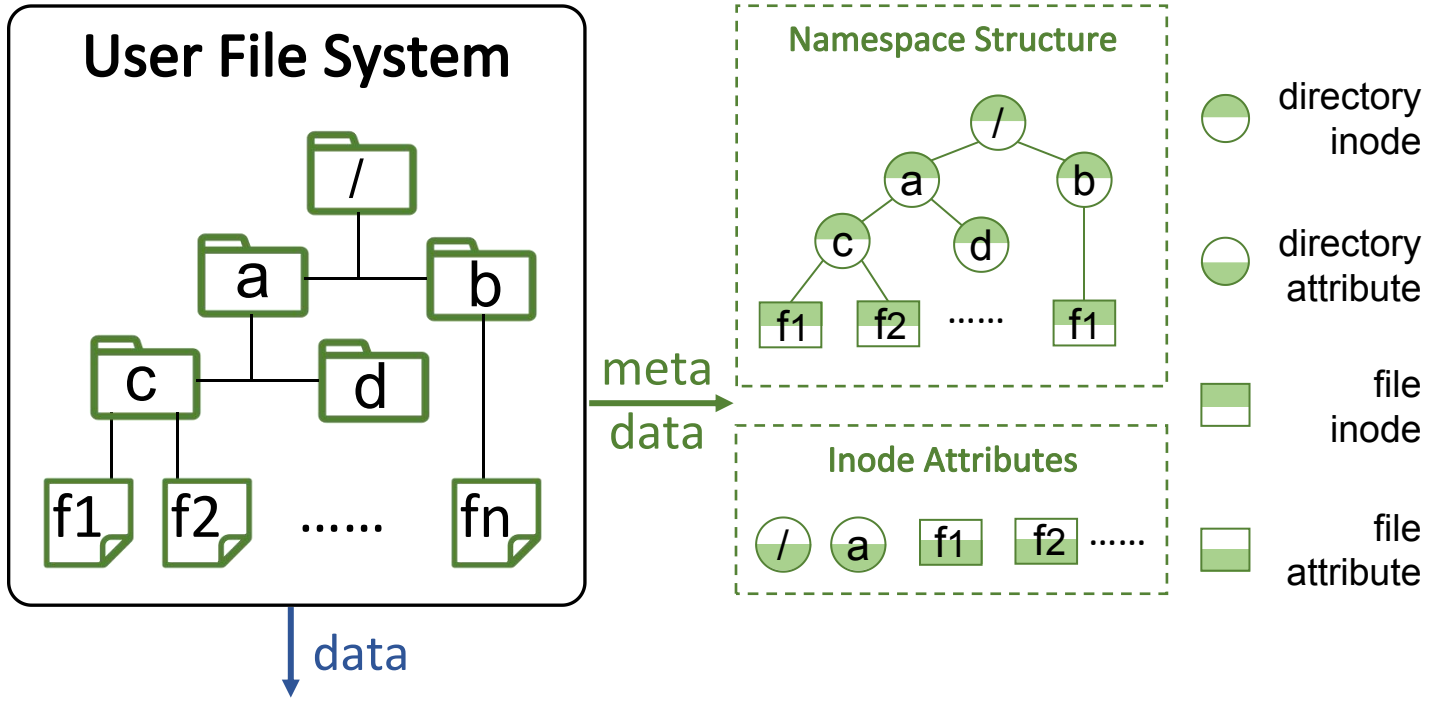
Distributed file systems are now critical infrastructure.

FS, Metadata, and the Bottleneck



Store file data across many data servers.

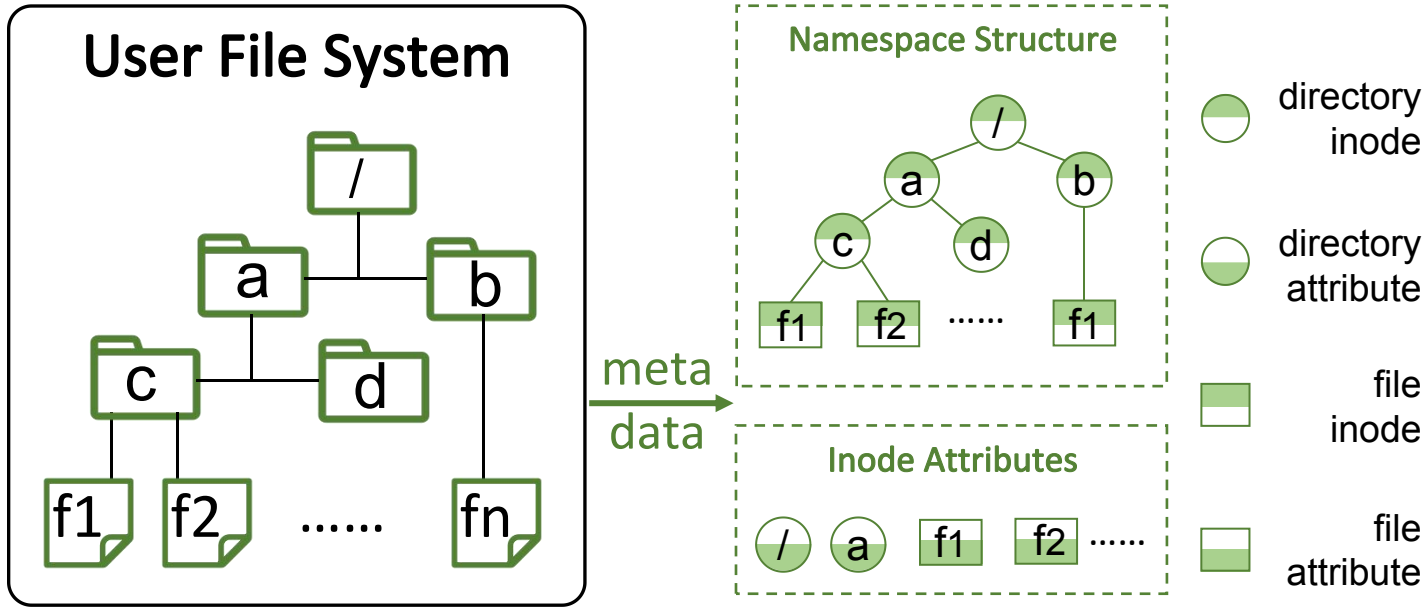
FS, Metadata, and the Bottleneck



Store metadata in distributed KV/DB for scalability.

Store file data across many data servers.

FS, Metadata, and the Bottleneck



Metadata has become the primary bottleneck.

Large Scale

Capacity: 100 billions+

QPS: millions per second

Fast Hardware

SSD Bandwidth: ~15GB/s

NIC Bandwidth: 400Gb+

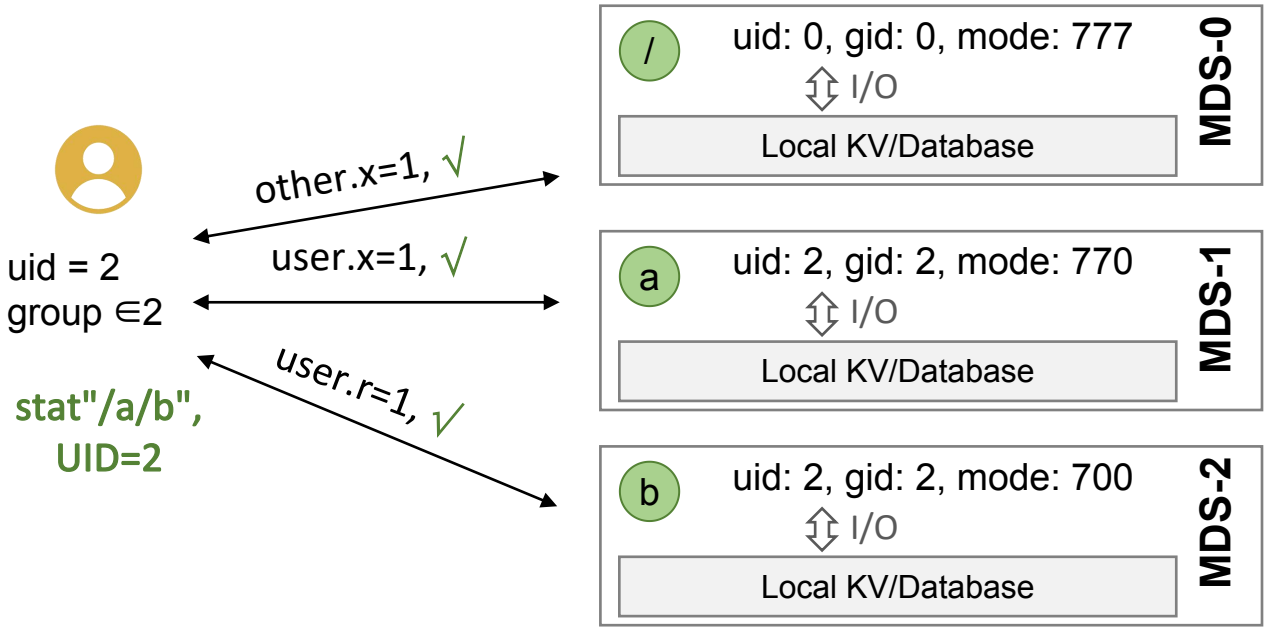
New Trend

File size: < 1MB

Meta-op: >90%

Metadata Bottleneck: Permission Checking

- Metadata operations require step-by-step path resolution for *Inode Indexing* and *Permission Checking*.

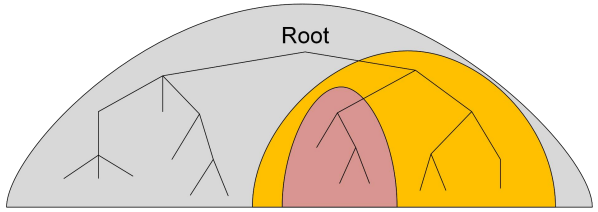


- match only 1 class
  (User → Group → Other)
- Verify Permission
  the 'x' bit

O(N) RPCs, DB queries, I/O...

We've Tried Many Ways...

Subtree-Based Partitioning

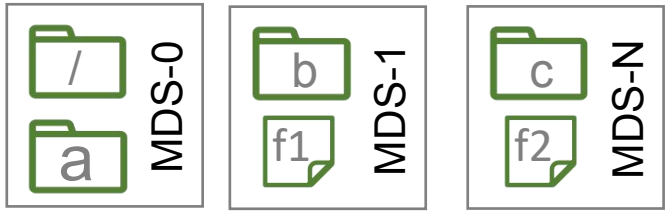


CephFS, HDFS, HopsFS...

better locality, less RPCs

hotspots

Hash-Based Partitioning

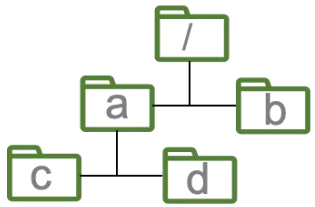


Tectonic, CFS, InfiniFS...

better scalability

high overhead

Or Namespace-Centric?



Single directory server

Many metadata/file servers

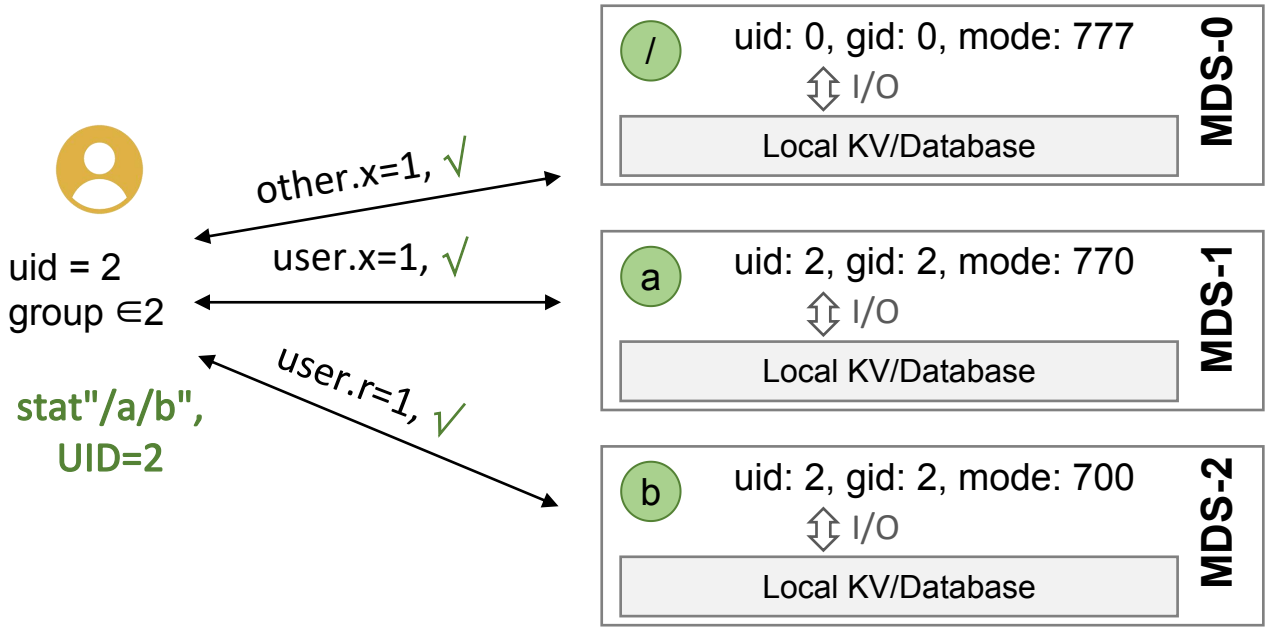
LocoFS, Mantle...

No matter the trade-off, we always pay the price for permission checking.

Is it possible to bypass the step-by-step checks, and resolve paths with $O(1)$ overhead?

Wait... Do We Really Need to Check Every Directory?

- The UID of a single request is **invariant** during metadata operation!



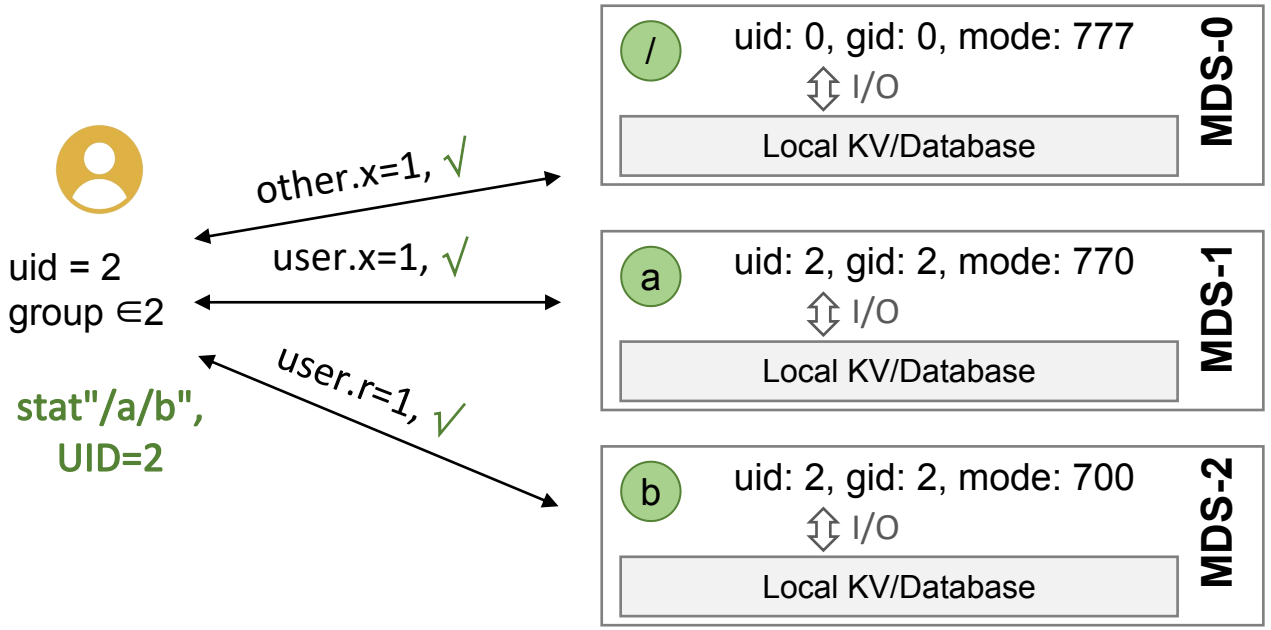
- Can we precompute whether $UID = b.uid$ can traverse all parent directories?
  We call it owner-class Reachability.
- For each directory i, class:

$$Check(i) \approx \begin{cases} i.user.x & if\ i.uid == b.uid \\ i.other.x & otherwise \end{cases}$$

- Similarly for Group and Other classes.

Wait... Do We Really Need to Check Every Directory?

- The UID of a single request is **invariant** during metadata operation!



- Can we precompute whether $UID = b.uid$ can traverse all parent directories?
  We call it owner-class Reachability.
- For each directory i, class:

$$Check(i) \approx \begin{cases} i.user.x & if\ i.uid == b.uid \\ i.other.x & otherwise \end{cases}$$

- Similarly for Group and Other classes.

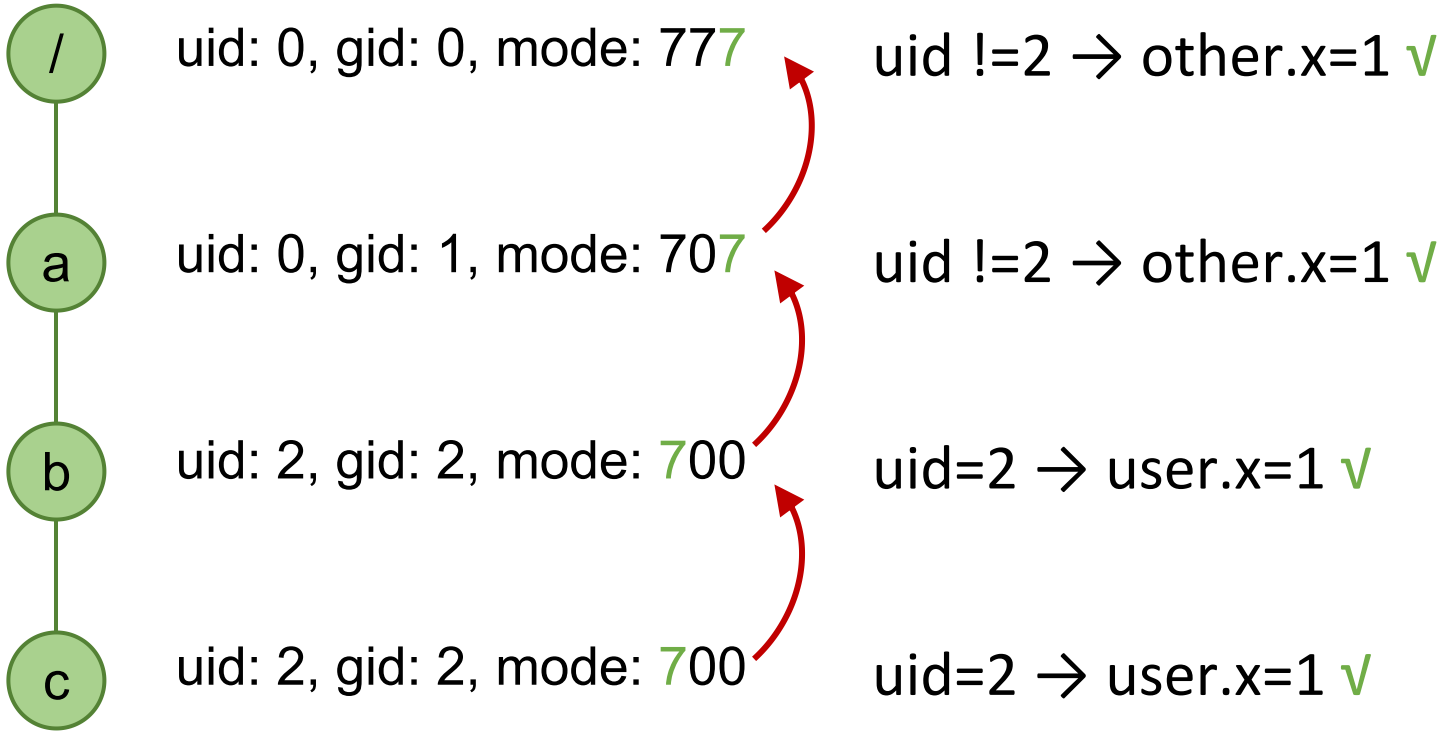
Maybe we can precompute speculative Reachability,
using just 3 bits—one for each permission class?

If only it were that simple…

Here's the Catch

- A naive pre-computation can lead to dangerous false positives, incorrectly granting access to unauthorized users in corner cases.

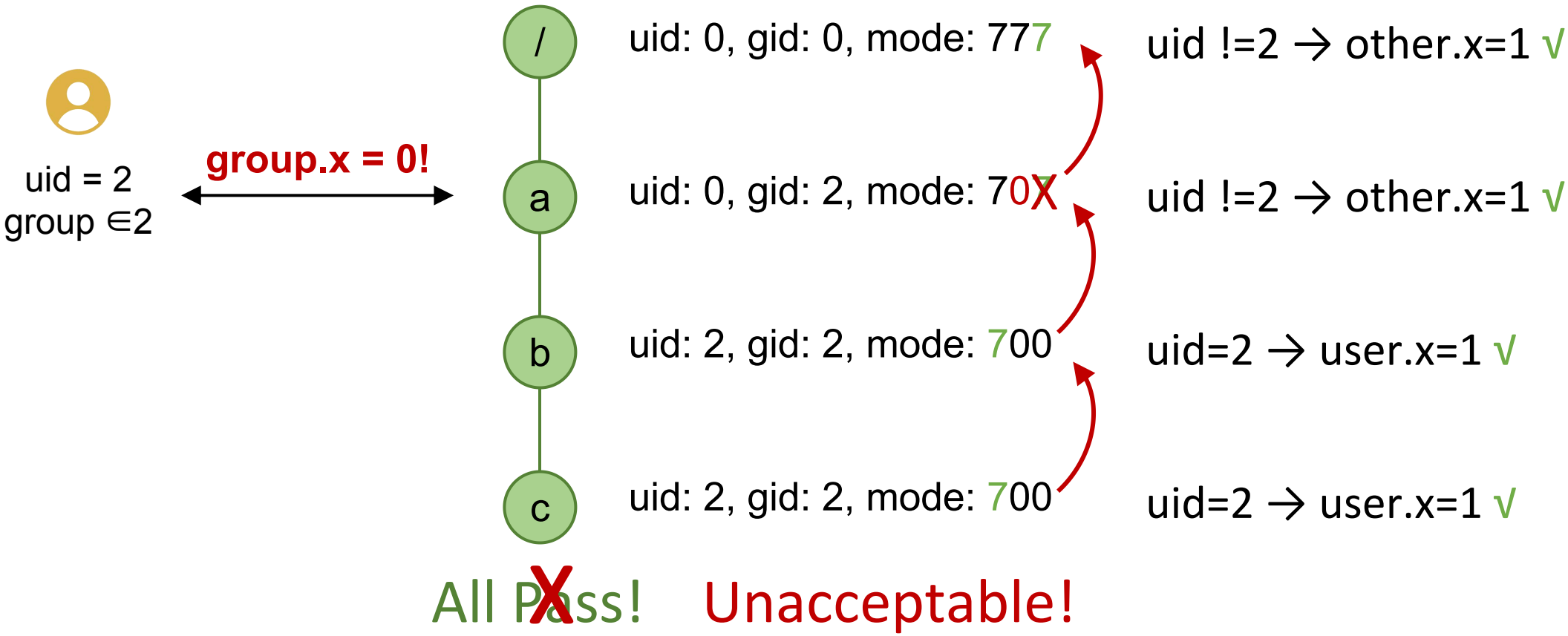
E.g., Pre-compute whether **/a/b/c** is reachable by its Owner



All Pass! We set User.Reachability to True

Here's the Catch

- A naive pre-computation can lead to dangerous false positives, incorrectly granting access to unauthorized users in corner cases.



Is It Truly a Dead End?

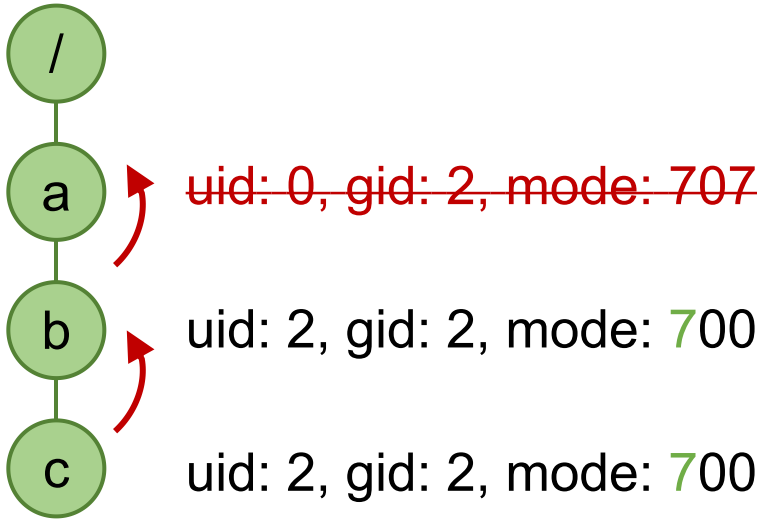
- **What we found**: Most directories follow a Permission Descending Pattern (owner.x ≥ group.x ≥ other.x).

Metadata permission analysis of 4 real-world DFS images.

Dataset	# Dir	Depth	PDP	Reachability		
				User	Group	Other
Users	1.6M	9.22	99.90%	99.22%	11.75%	0.00%
Annoy	3.9M	11.52	99.96%	97.59%	33.29%	4.94%
SCR4	5.0M	10.27	99.71%	99.26%	21.68%	0.02%
Projs	14.0M	13.65	99.98%	50.56%	52.01%	19.17%

The Good News: PDP Makes Speculation Safe

- **What we found**: Most directories follow a Permission Descending Pattern (owner ≥ group ≥ other).
- **What we prove**: Under this constraint, the precomputation is False-Positive-Free (Zero risk of unauthorized access).
- **We also found**: False-Negatives are Safe (Just fall back).



Violate the PDP,
Set all reachability to 0!

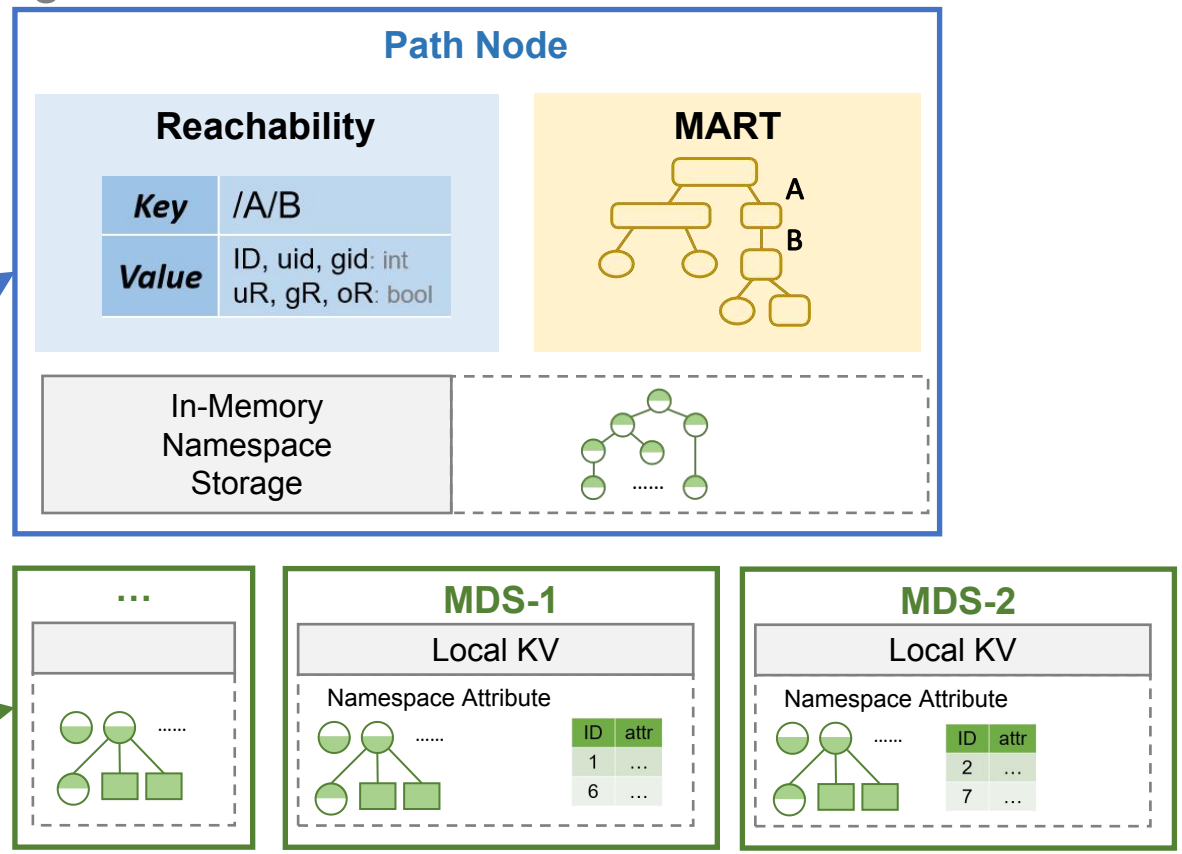
Key idea: an O(1) fast path for most metadata requests, with safe fallback for the rest.

SPMeta: Accelerating Metadata Management via Speculative Permission Checking

- Hybrid Metadata Organization
- Speculative Permission Checking
- Efficient Indexing with MART

caches simplified namespace structure and speculative reachability in memory.

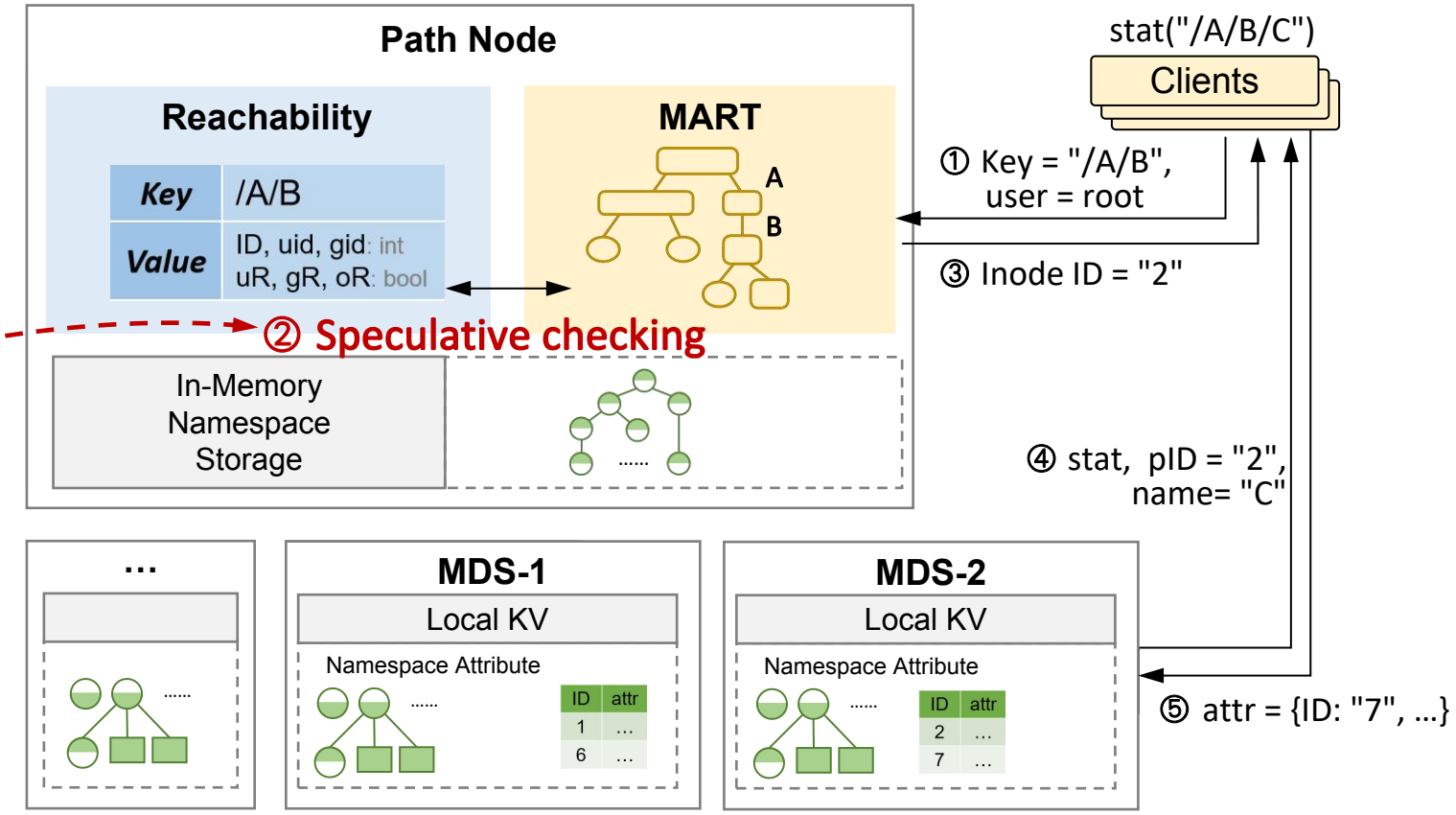
persists metadata in local KV stores.



SPMeta: Accelerating Metadata Management via Speculative Permission Checking

- Hybrid Metadata Organization
- Speculative Permission Checking
- Efficient Indexing with MART

Most lookups take the O(1) fast path.

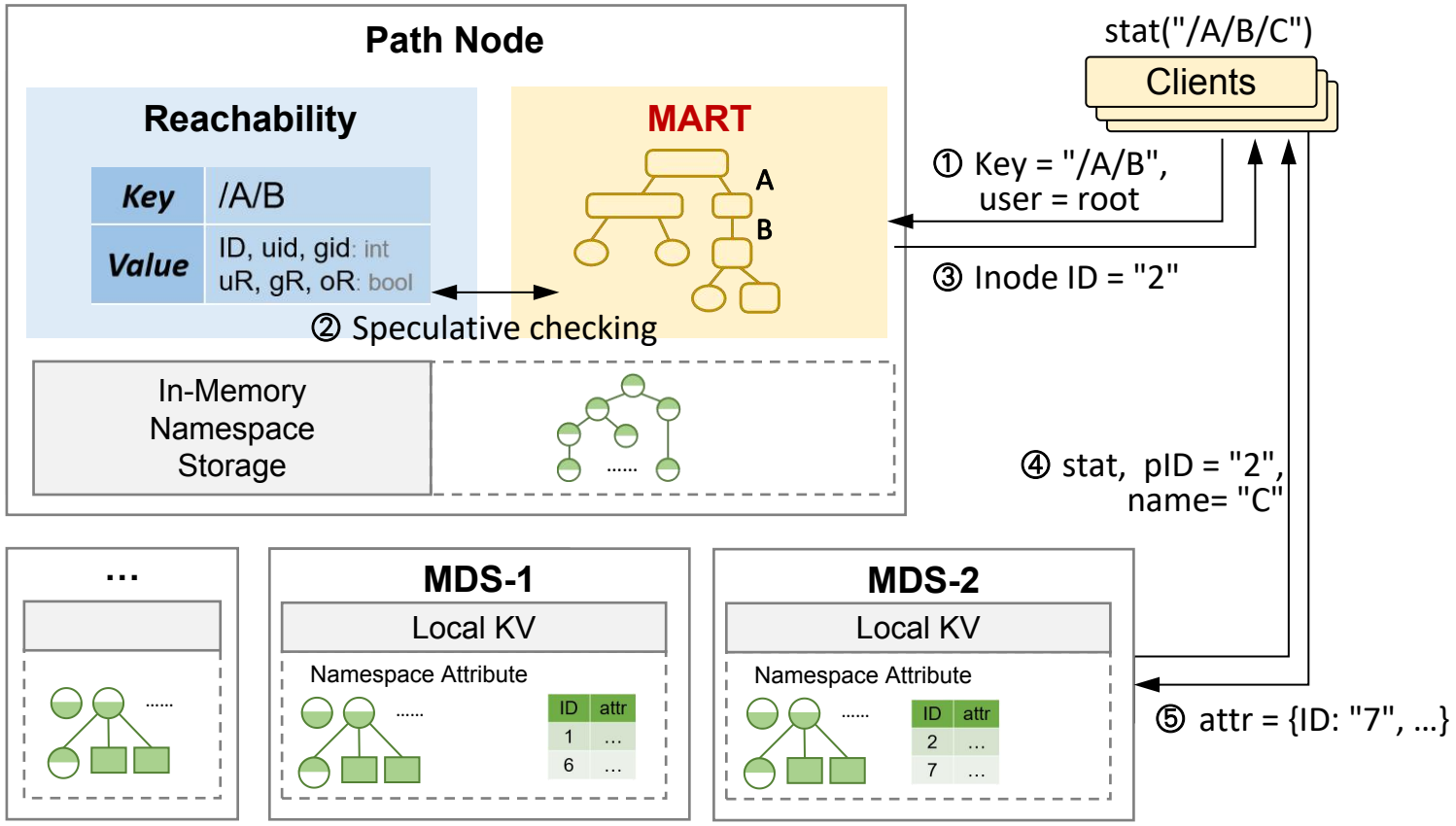


SPMeta: Accelerating Metadata Management via Speculative Permission Checking

- Hybrid Metadata Organization
- Speculative Permission Checking
- Efficient Indexing with MART:
  Metadata-Orient
  Adaptive Radix Tree

➢ Value-Embedded Inner Nodes
reduce memory overhead, since directories are often both prefixes and valid entries.

➢ Adaptive Tree Relocation
that enables efficient dir-rename.



Questions to Answer

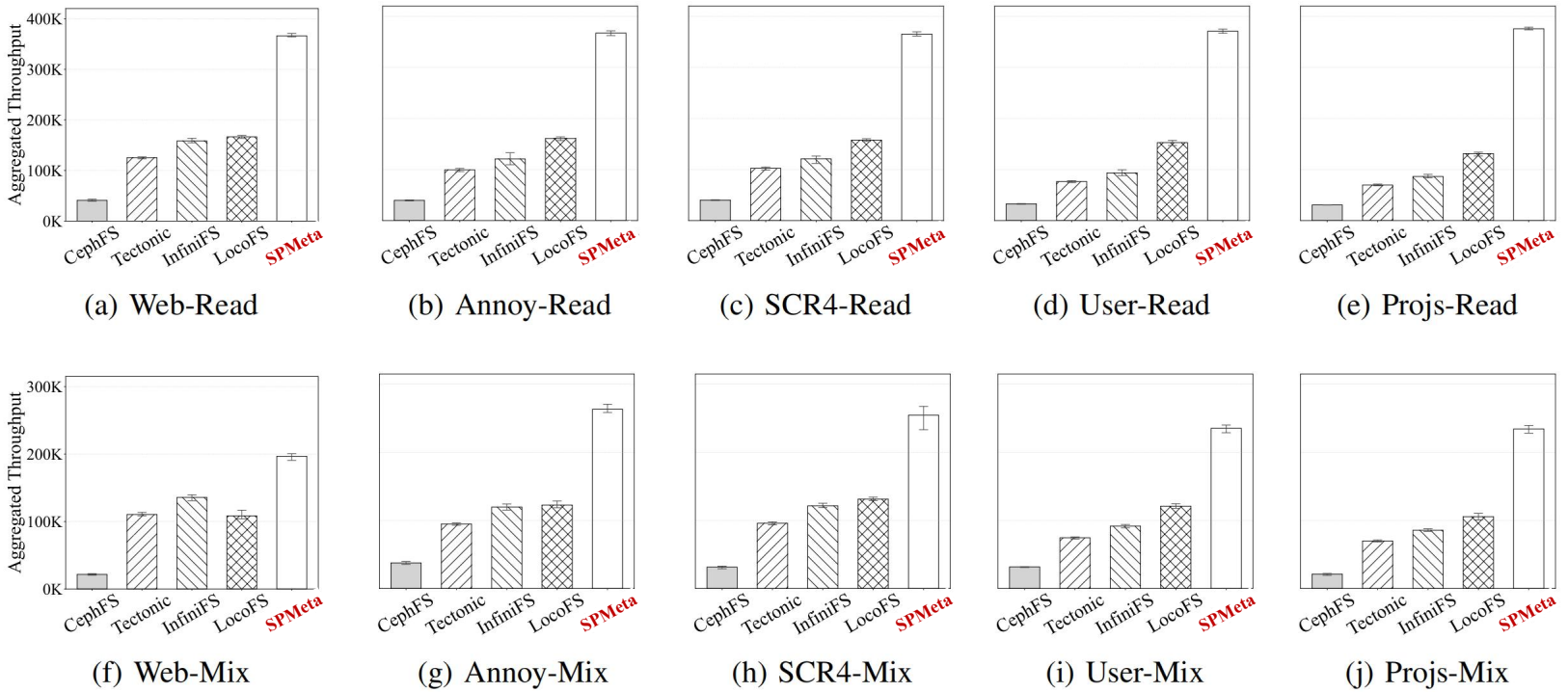
- How does SPMeta perform on real-world workloads?
- Does SPMeta eliminate the path-resolution bottleneck?
- Does SPMeta remain robust under larger namespaces?
- How sensitive is SPMeta to write ratio and fallback ratio?
- What is the memory overhead of MART?
- Does SPMeta still help when data I/O is enabled?
- ……

Evaluation Setup

- Baseline Systems
  - CephFS: subtree-based metadata partitioning;
  - Tectonic: hash-based metadata partitioning;
  - InfiniFS: hash partitioning with concurrent path resolution;
  - LocoFS/Mantle: centralized path resolution with distributed metadata storage.
- Deployment
  - 5 client nodes + 5 server nodes;
  - multiple threads to saturate metadata service.
- Workload
  - 5 real-world namespaces;
  - Read-only and read-write mixed workloads.

Evaluation: Improving Overall Performance

- Overall metadata throughput under read-only and read-write mixed workloads.



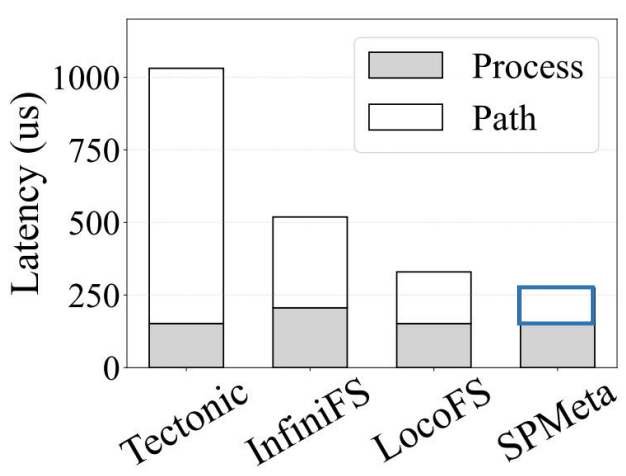
SPMeta consistently outperforms all baselines,

delivering up to 2.86 × higher throughput than the second-best approach.

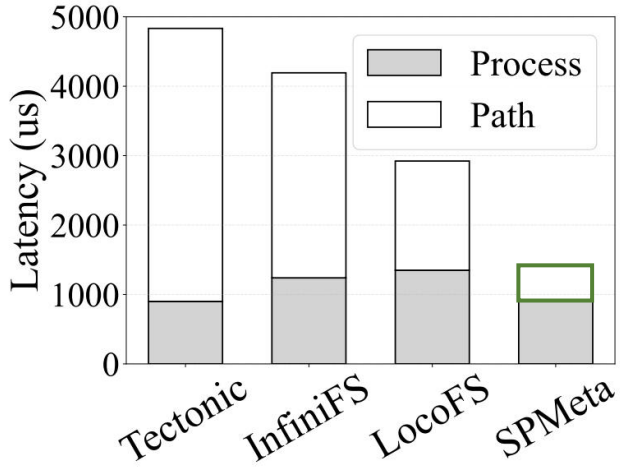
Evaluation: Eliminating the Path-Resolution Bottleneck

- Break down metadata latency to understand where SPMeta gains come from.

Achieves the lowest path-resolution latency under light load.



(a) Light Load



(b) Heavy Load

Maintains low overhead even under heavy load.

SPMeta removes most path-resolution overhead via speculative permission checking and significantly lowers metadata latency.

Conclusion

- Problem: path resolution incurs high overhead in DFS metadata services.
- Key insight: step-by-step permission checks for most metadata requests can be safely avoided.
- Key techniques of SPMeta
  - Hybrid Metadata Organization
  - Speculative Permission Checking
  - Metadata-Oriented ART
- Results
  - Eliminates most path-resolution overhead.
  - Delivers up to 2.86× higher throughput than the second-best approach.

Conclusion

- Problem: path resolution incurs high overhead in DFS metadata services.
- Key insight: step-by-step permission checks for most metadata requests can be safely avoided.
- Key techniques of SPMeta
  - Hybrid Metadata Organization
  - Speculative Permission Checking
  - Metadata-Oriented ART
- Results
  - Eliminates most path-resolution overhead.
  - Delivers up to 2.86× higher throughput than the second-best approach.

Thanks!

wangyd22@chinatelecom.cn

yiduo.site