DMTree: Towards Efficient Tree Indexing on Disaggregated Memory via Compute-side Collaborative Design

Guoli Wei[†], Yongkun Li[†], Haoze Song[‡], Tao Li[†], Lulu Yao[†], Yinlong Xu[†], Heming Cui[‡]

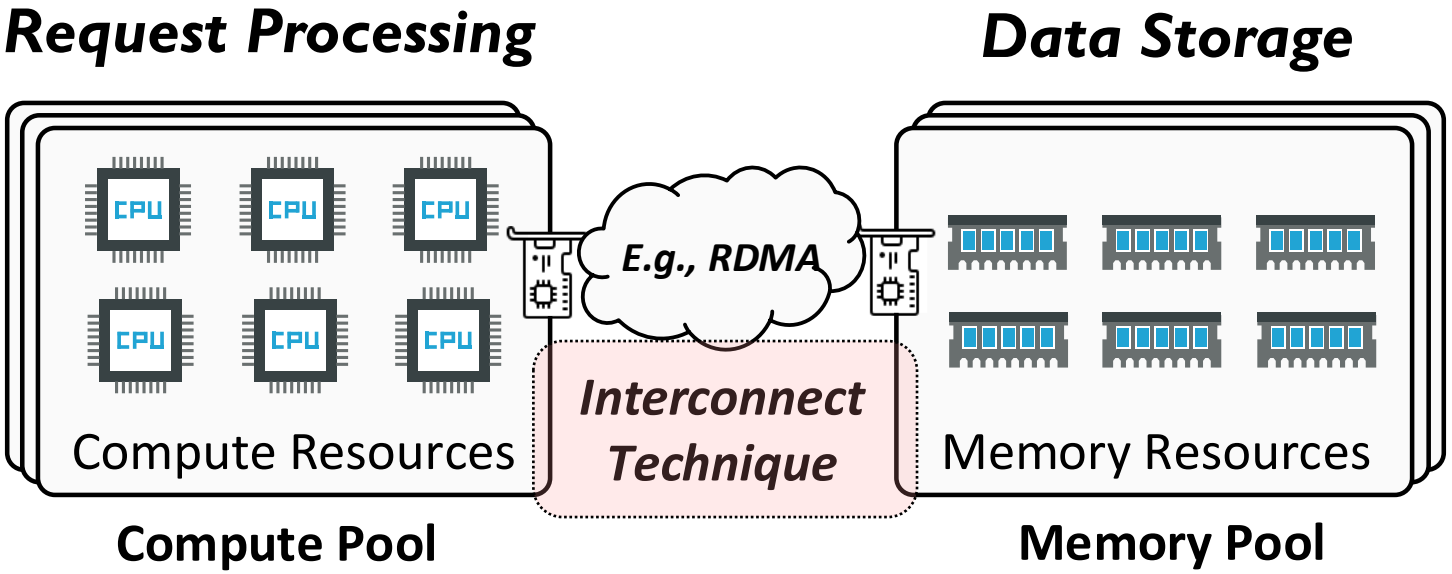
[†]University of Science and Technology of China

[‡]The University of Hong Kong

USENIX FAST 2026

Background

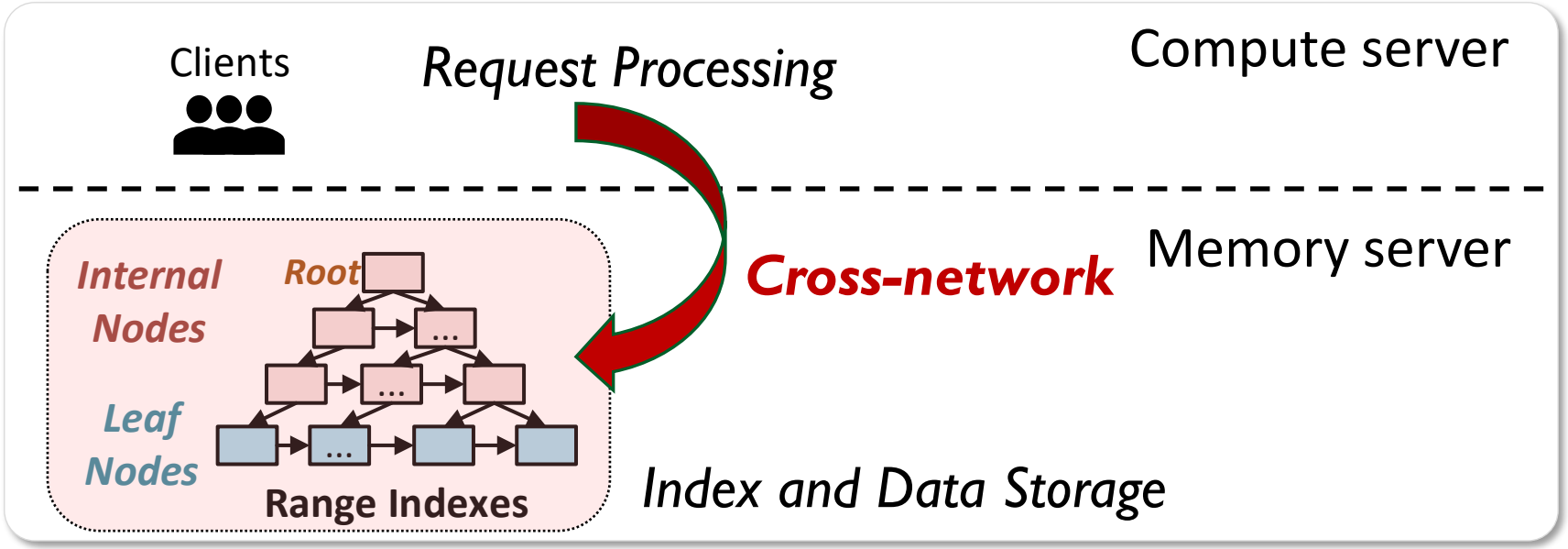
- Disaggregated memory (DM)
  - Resource separation and pooling



- **High resource utilization**: resources allocated independently
- **High resource scalability**: resources expanded independently

Background

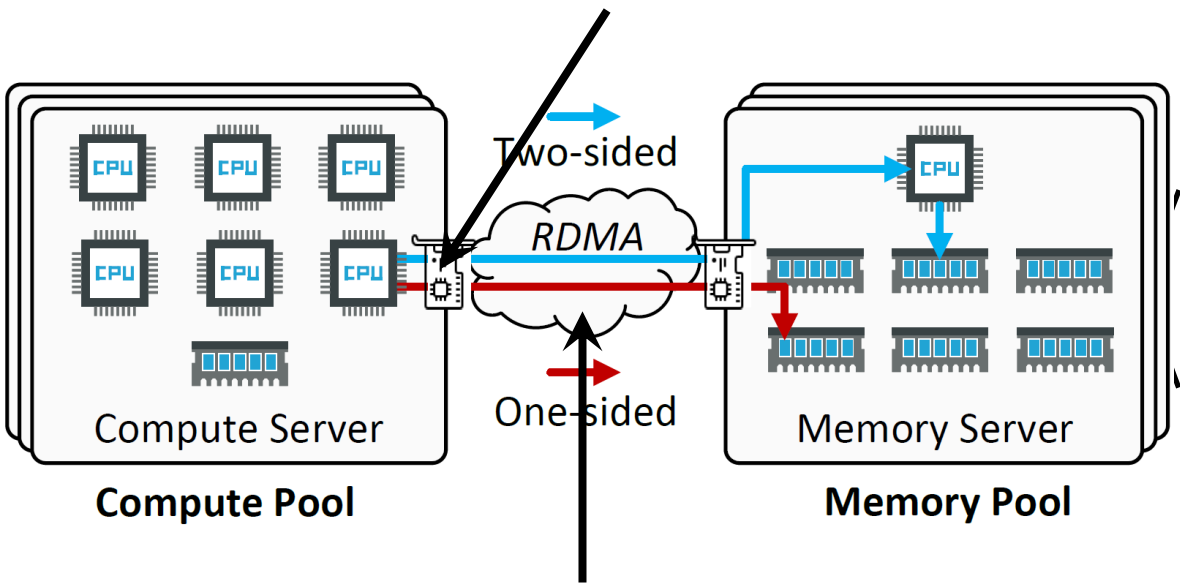
- Tree index in disaggregated memory
  - Key component to accelerate data access *(B+ tree, ART, …)*
- Index access pattern
  - Storing index on memory server
  - Compute servers issue ***cross-network accesses***



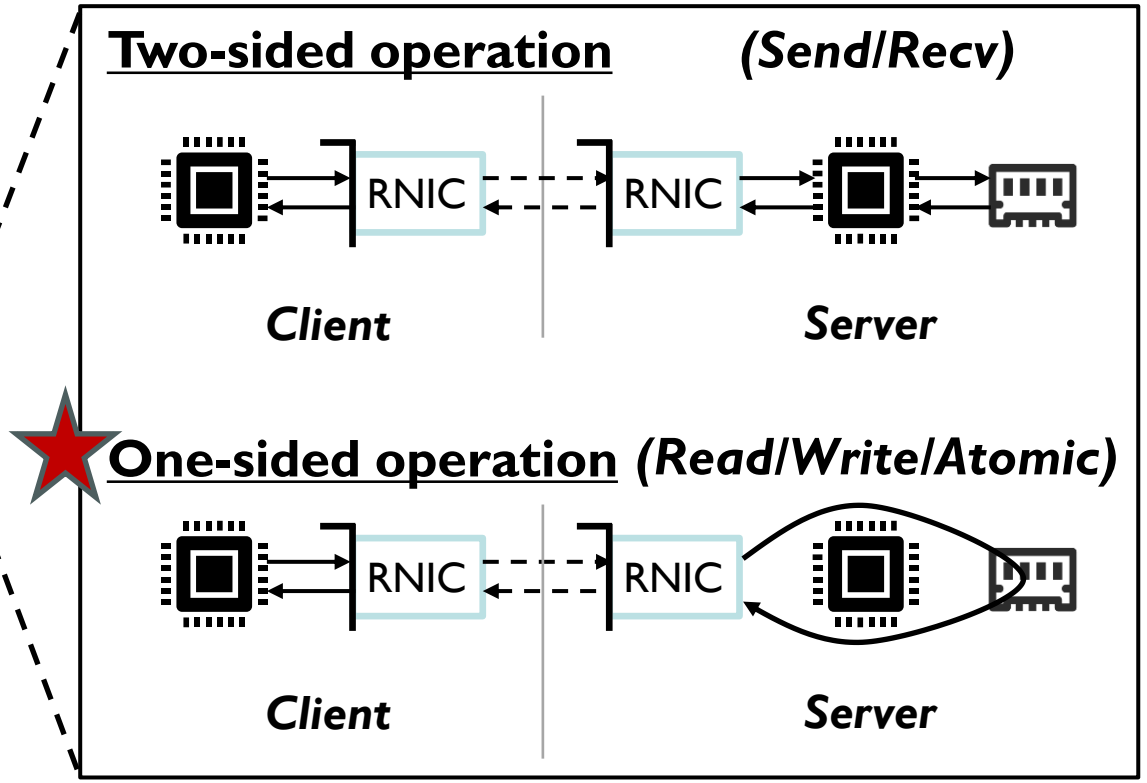
Background

- Remote Direct Memory Access (RDMA)

IOPS Resource: RNIC processing rate
(e.g., 60 Mops, Mellanox ConnectX-6)

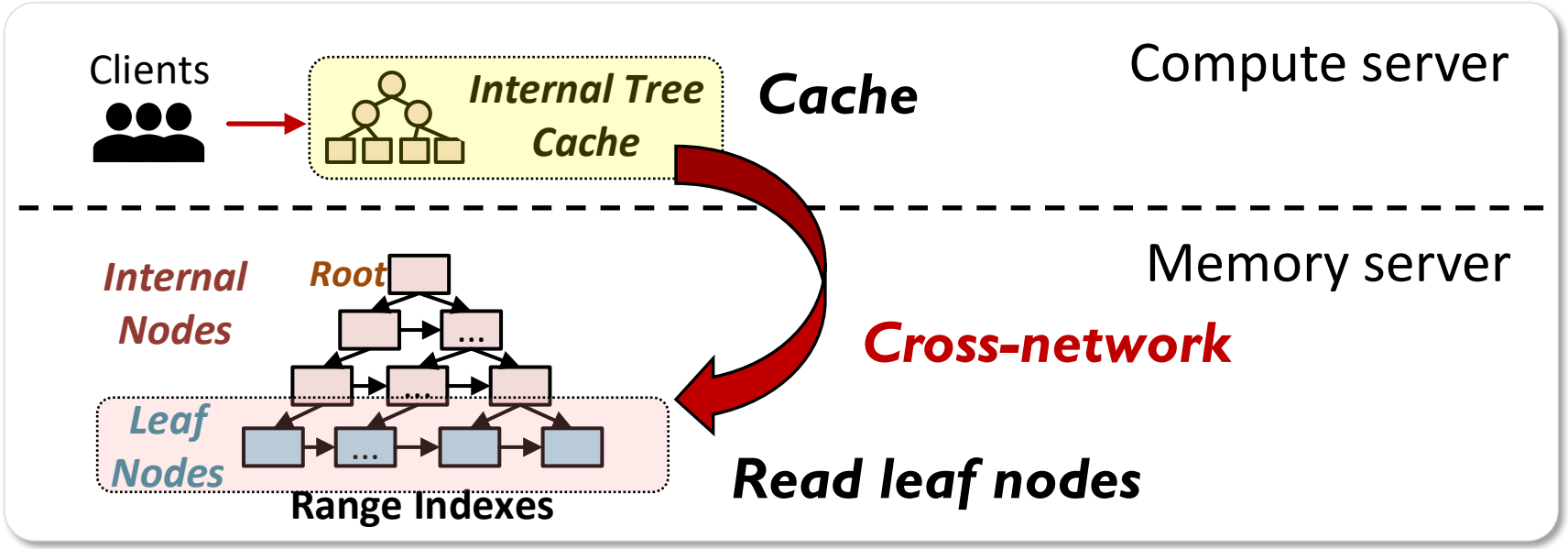


Bandwidth Resource: data transfer rate
(e.g., 100 Gbps, Mellanox ConnectX-6)



Problem

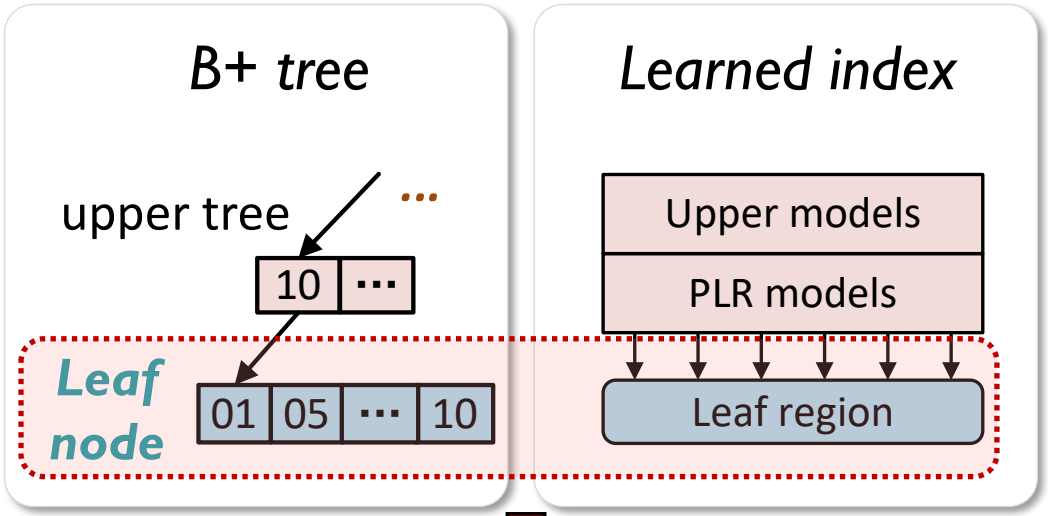
- Cross-network overhead
  - Common solution: cache upper-level tree, remotely access leaf nodes



- Tradeoffs in RDMA resource utilization
  - Existing works: ***contiguous range storage*** and ***precise key-value locating***

Existing Designs

➢ Index with contiguous range storage



- **Coarse-grained leaf node**
- Store contiguous range of key-values in leaf
- SOTA:
- Sherman@SIGMOD'22, ROLEX@FAST'23

Cross-network

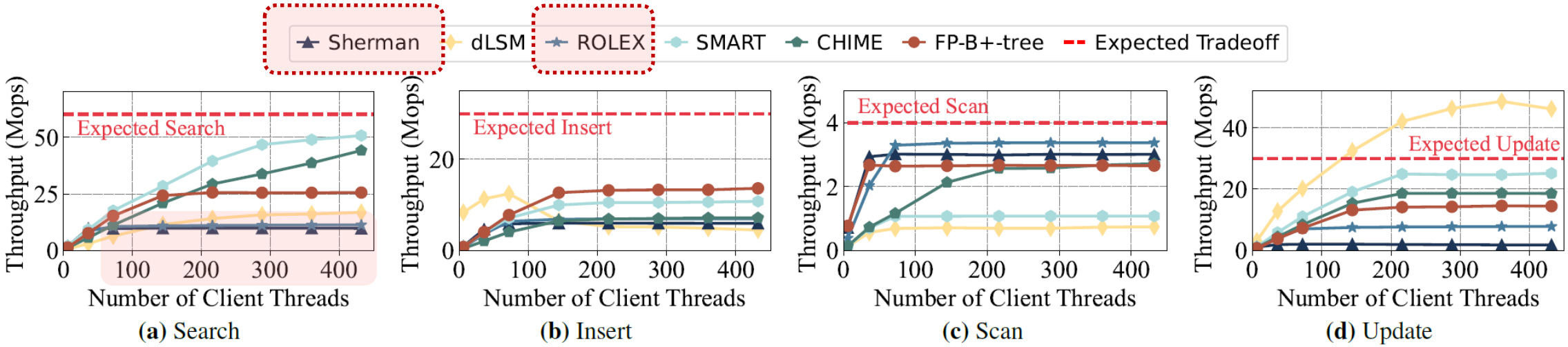
Read Leaf Node

- Compute server locate target leaf node
- Read/write the entire node
- **Cause read amplification**

Existing Designs

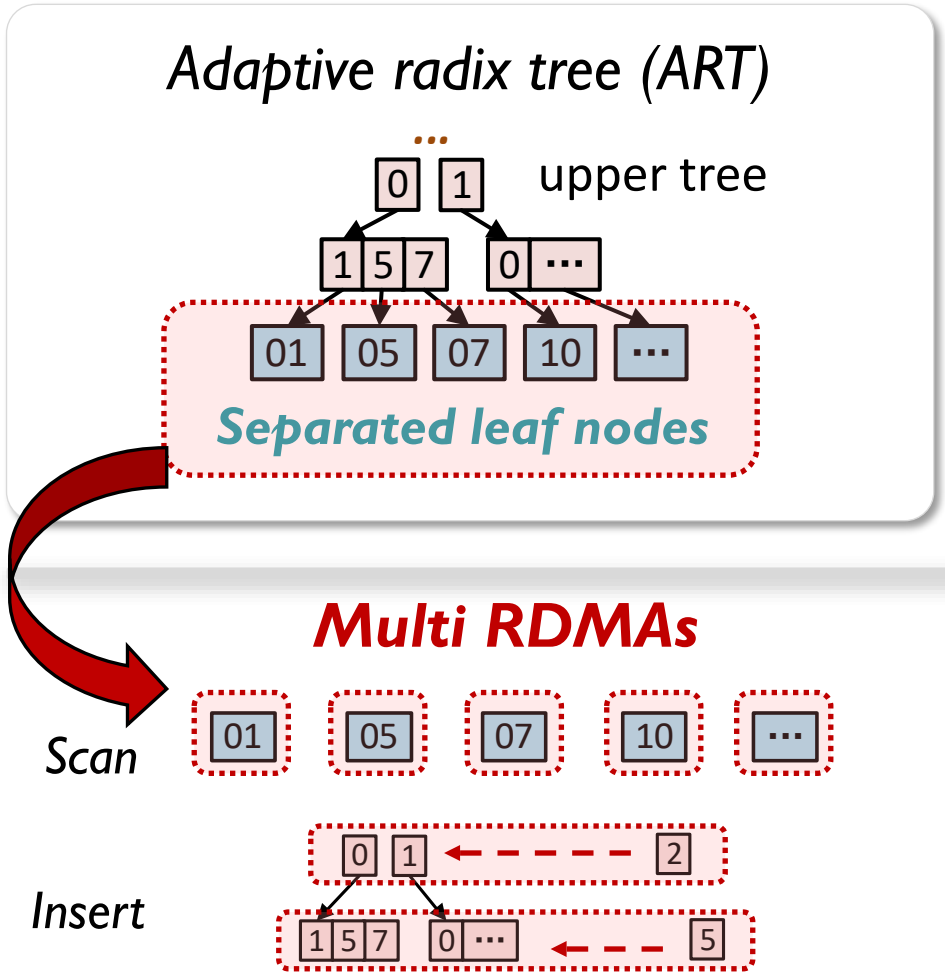
➢ Index with contiguous range storage

- Coarse-grained reads cause read amplification
- Waste bandwidth resources
- Leading to bottlenecks due to insufficient bandwidth resources



Existing Designs

➢ Index with precise key-value locating



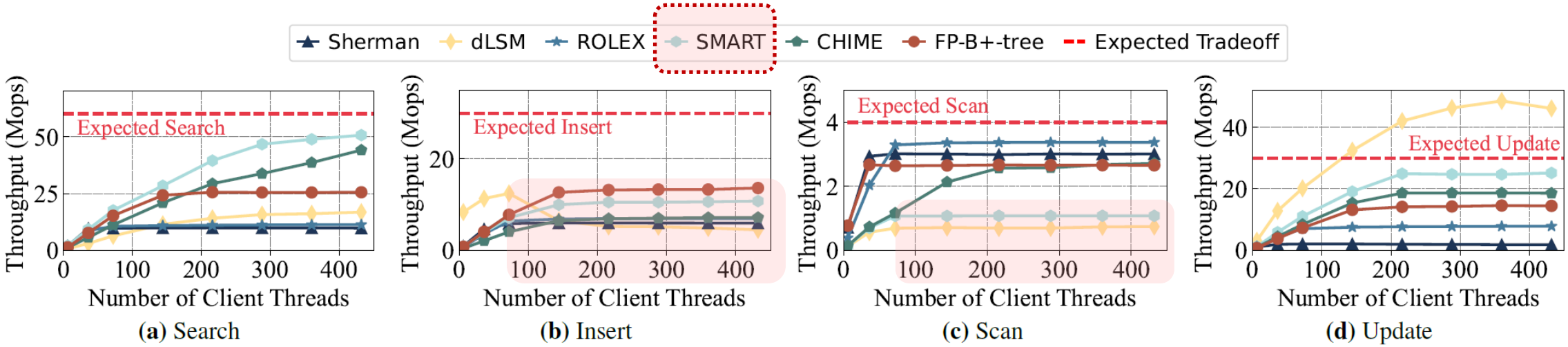
- **Fine-grained leaf nodes**
- Separated key-value in separated leaf
- No read amplification for leaf node
- SOTA: SMART@OSDI'23

- Multi RDMAs for reading separated leaf
- Multi RDMAs for updating upper-tree
- **Increase RDMA requests**

Existing Designs

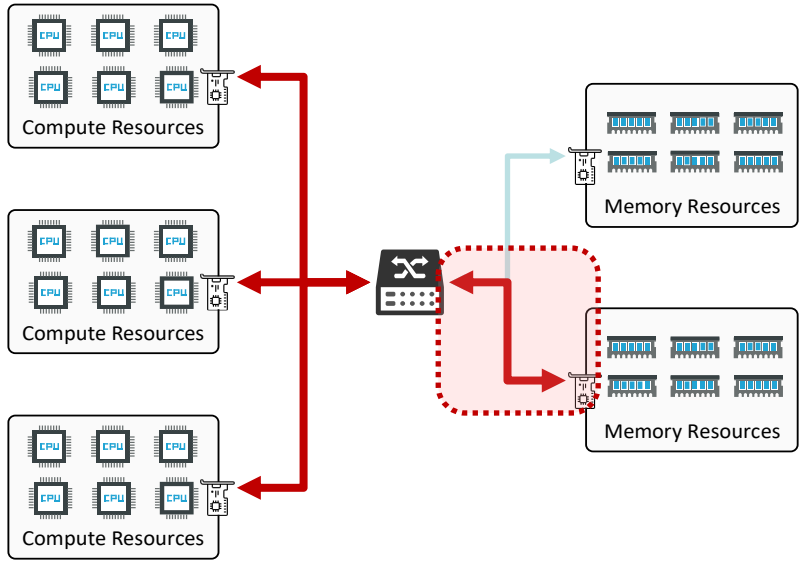
➢ Index with precise key-value locating

- Fine-grained nodes increase RDMA requests
- Consume IOPS resources
- Leading to bottlenecks due to insufficient IOPS resources

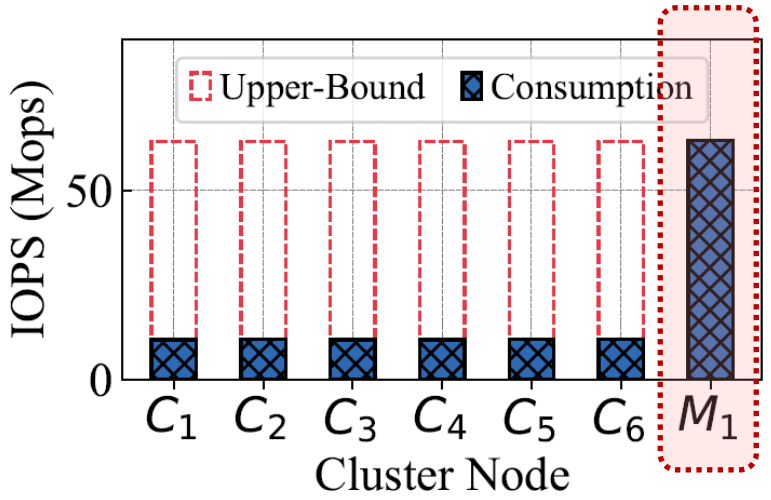


Motivation

- How to balance the inevitable resource consumption?
- **Insight:** Index access pattern in disaggregated memory
  - RDMA traffic converges at memory server
  - Compute-side RDMA remains underused

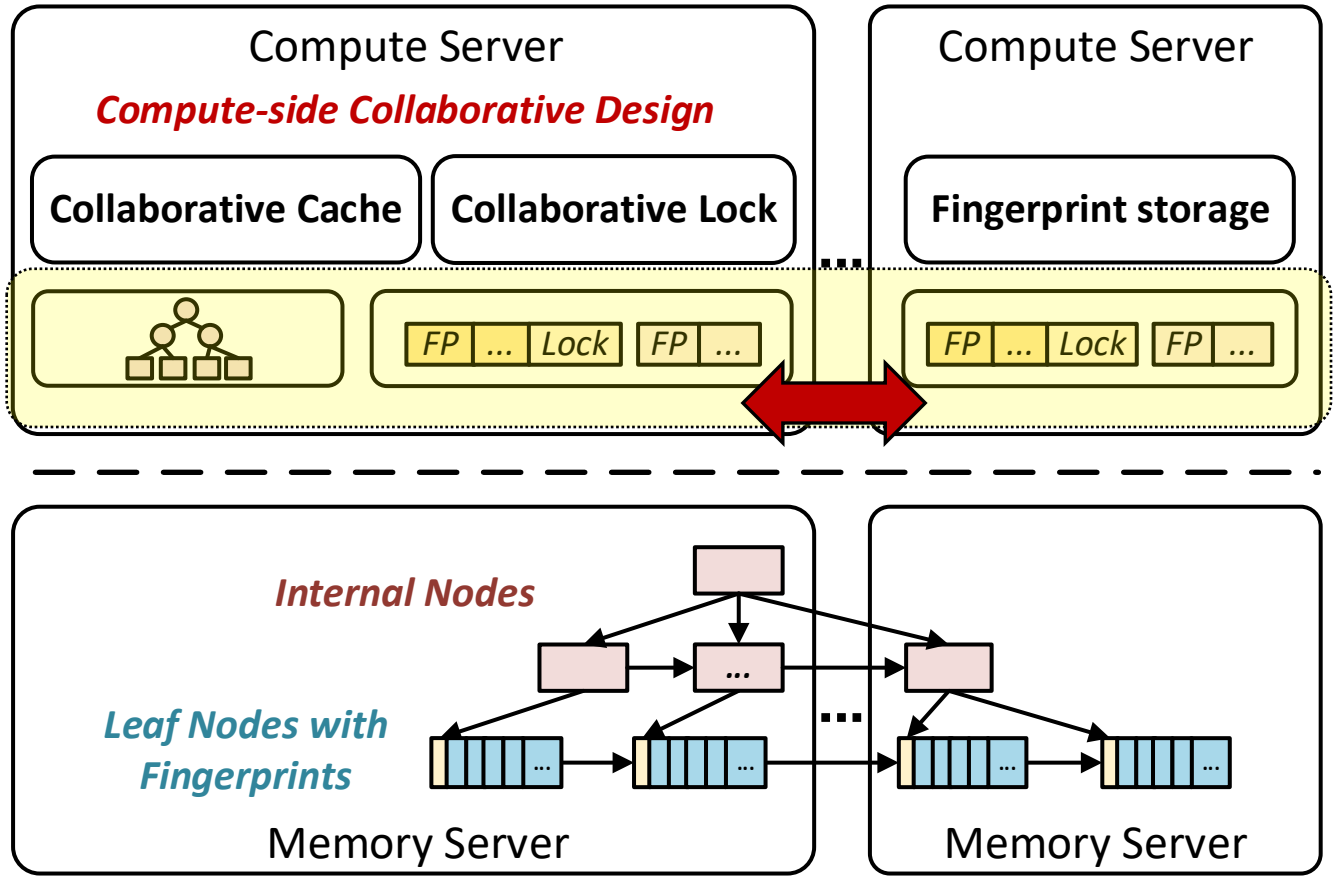


Memory-side bottleneck
[OSDI'23, SOSP'24]



Main Idea

Leverage **Compute-side RDMA** to Offload Index Operations Aggregate at Memory-side



DMTree Overview

Compute-side Collaborative Design

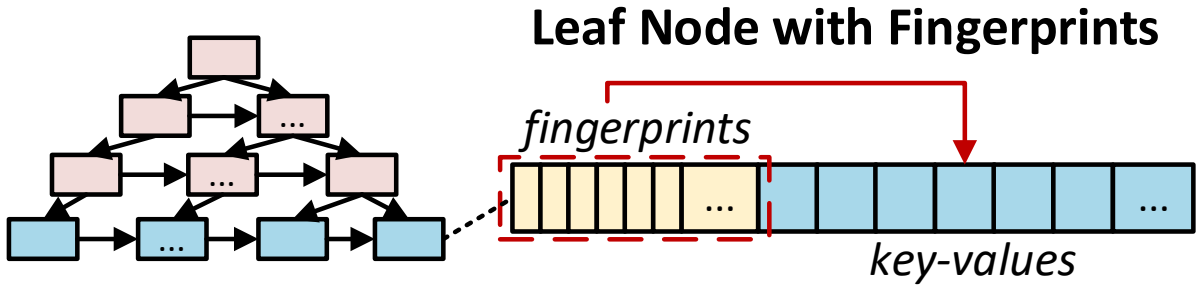
- Offload key–value locating
- Compute-side collaborative cache
- Offload concurrent locking

Alleviate Memory-side Bottleneck

- Distribute RDMA network traffic

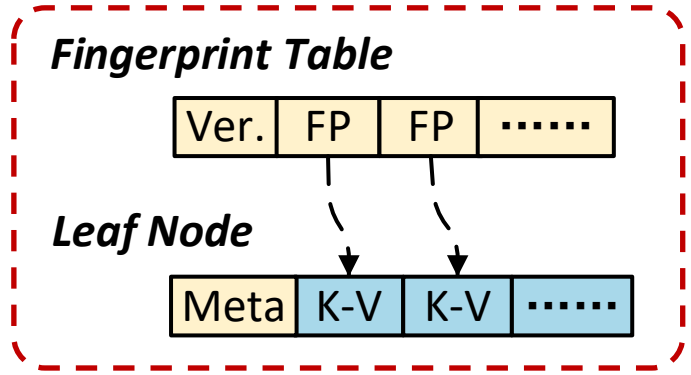
Basic Structure

- Precise key–value locating via **Fingerprint Table**
  - Before reading leaf, first traverse fingerprints
  - Locate target key-value in leaf nodes



Leaf Node ***+Fingerprint Table***

Fingerprint: FP = Hash(key)

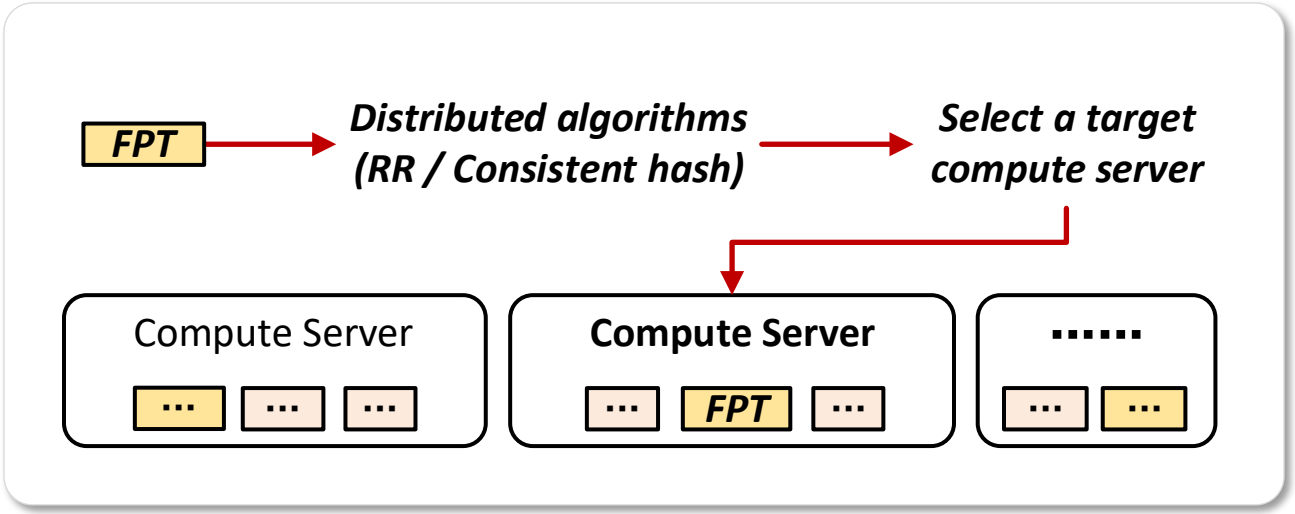


Locate target key

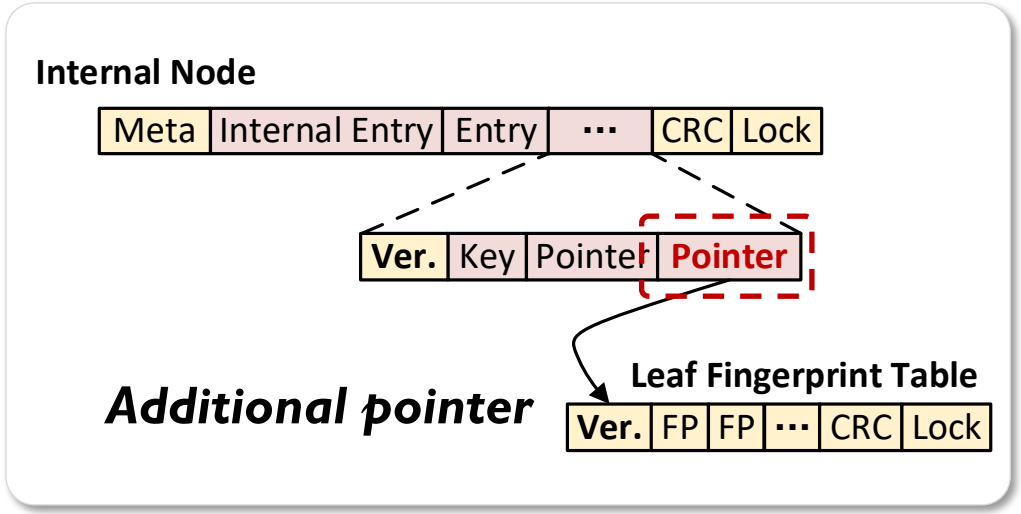
Offload Key–value Locating

- Distributed Fingerprint Tables on Compute Servers

Distributed algorithms



Retrieve fingerprint pointer



Distribute and access the fingerprint tables of all leaf nodes ***across compute servers.***

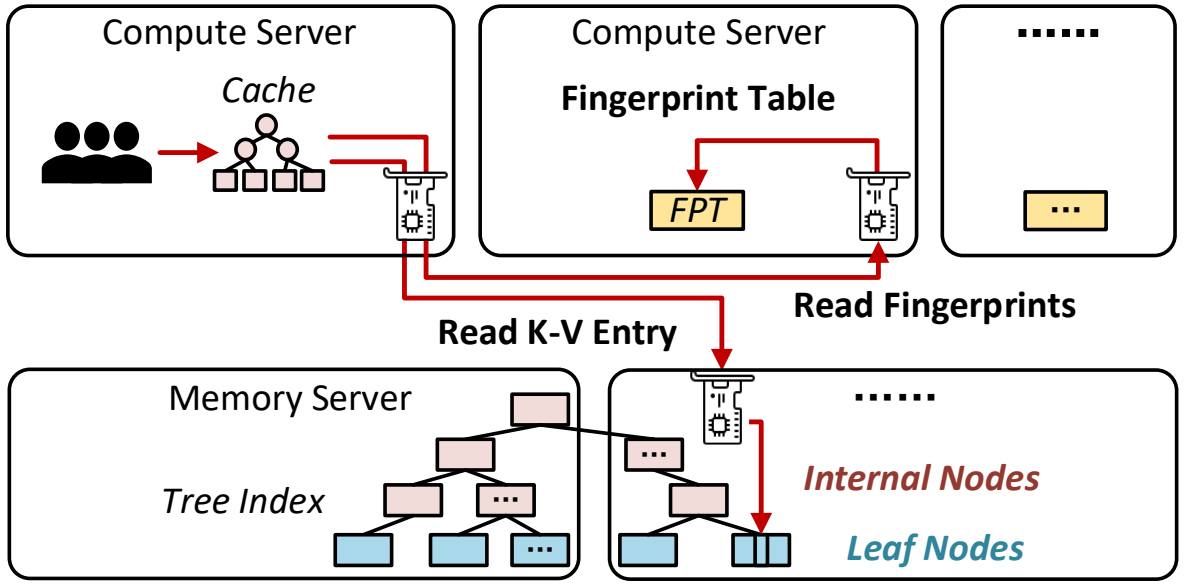
Offload Key–value Locating

➢ Offload Precise Key–Value Locating to Compute-side

Step1. Retrieve fingerprint pointer

Step2. Fetch fingerprints ***across compute servers***

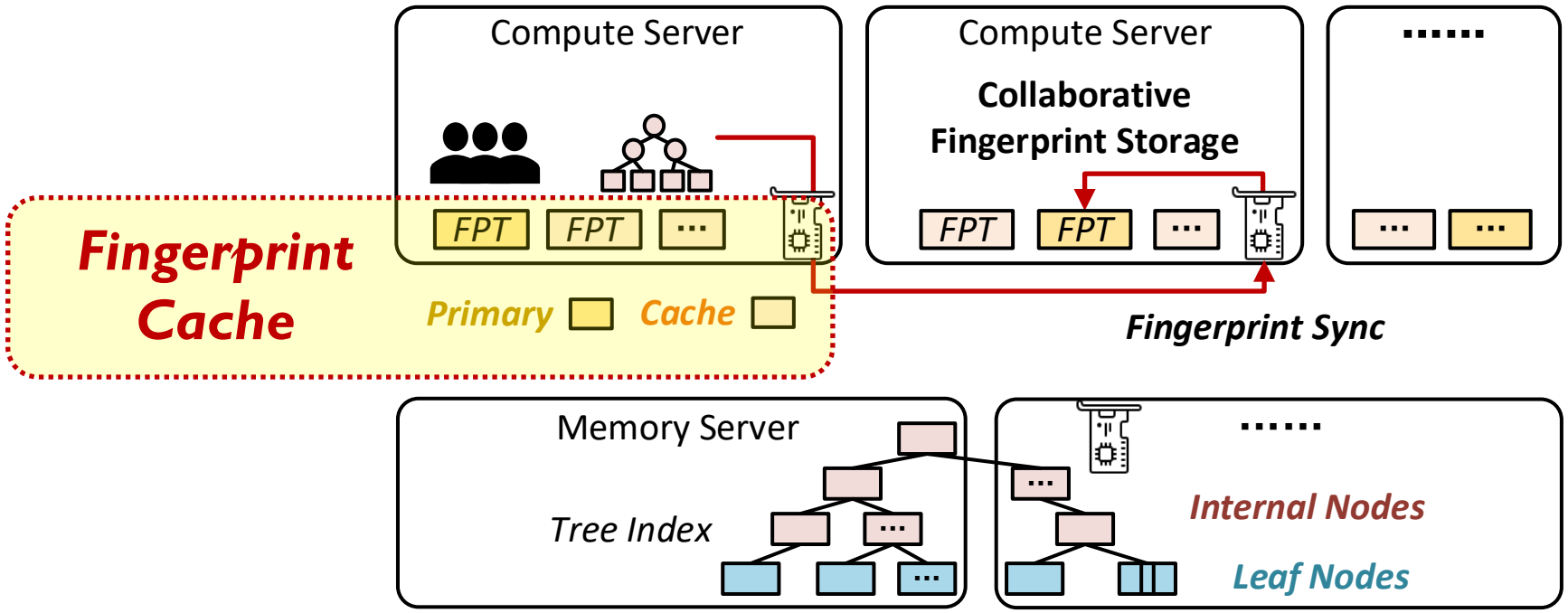
Step3. Locate target key, fetch key–value from memory server



Avoid additional RDMAs to fingerprints concentrated on memory server, ***alleviate IOPS bottleneck in precise key–value locating.***

Compute-Side Collaborative Cache

- Collaborative Fingerprint Storage on Compute-side
  - Each fingerprint table → primary storage on one server
  - ***Cached replicas on other servers***

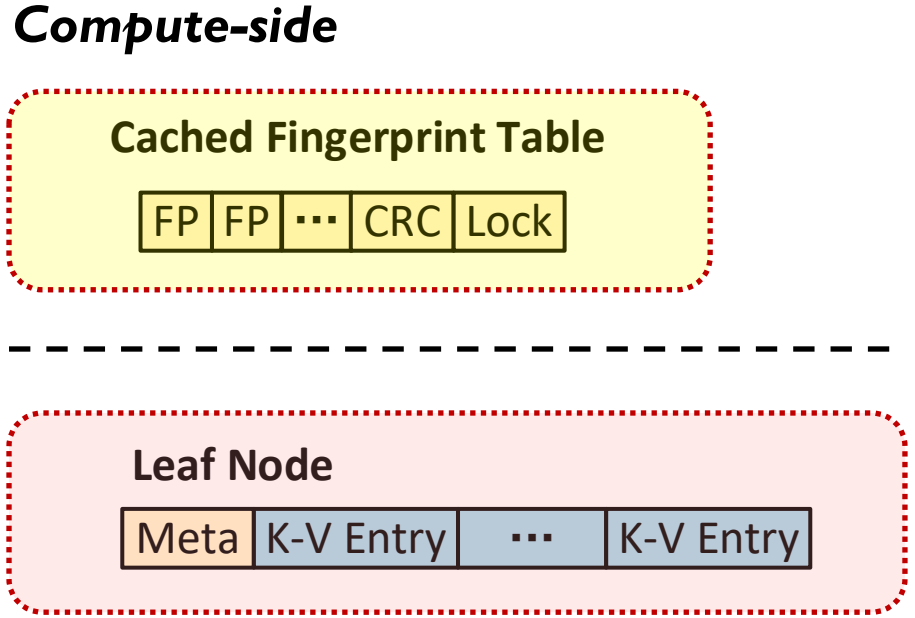


To accelerate compute-side fingerprint access

Compute-Side Collaborative Cache

- Detect and Resolve Fingerprint Inconsistency

  Inconsistency between ***Cached Fingerprints*** and ***Remote Storage***
  - Fingerprint misses due to outdated cache
  - Other mismatches issues: e.g., leaf node split

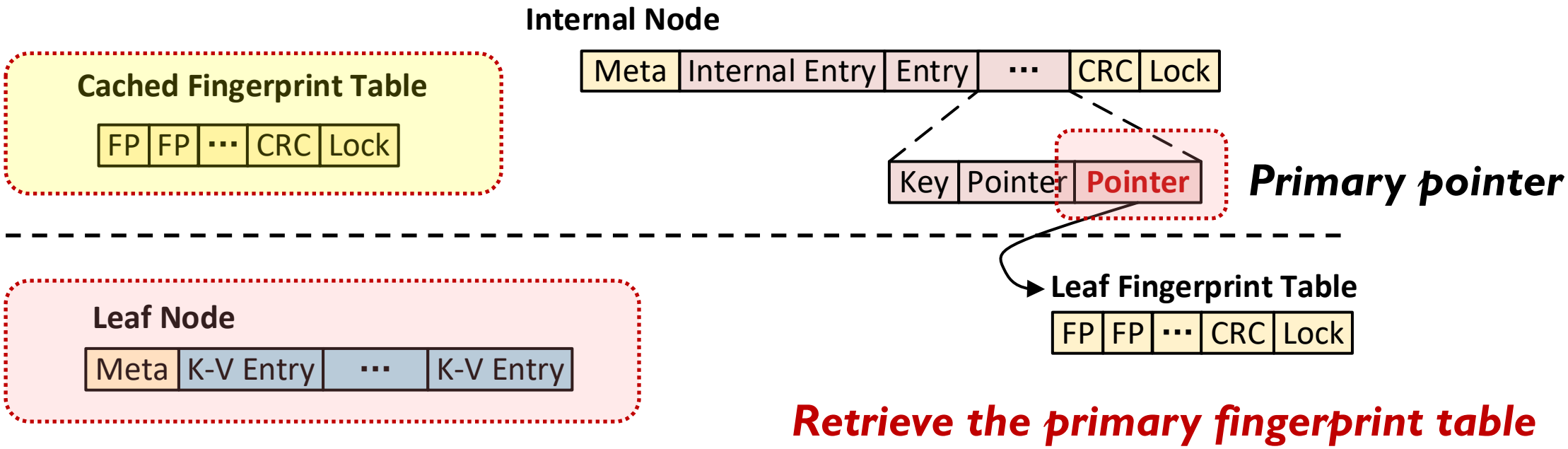


Compute-Side Collaborative Cache

➢ Detect and Resolve Fingerprint Inconsistency

Handling fingerprint misses due to ***Outdated Cache***

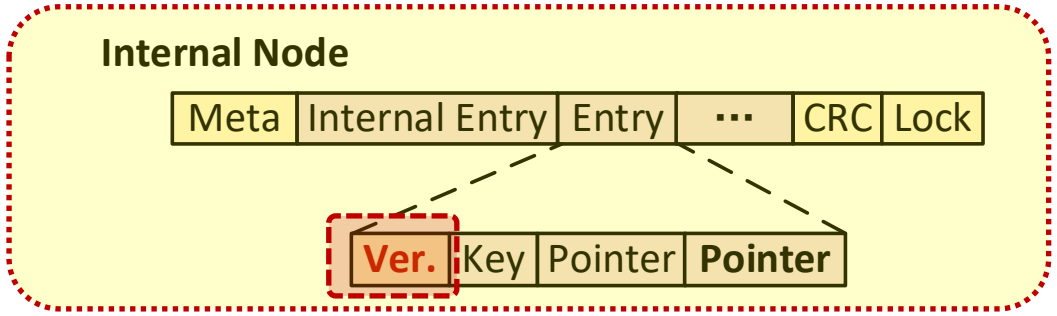
- redirect to primary via stored pointer
- retrieve latest fingerprints, update local cache



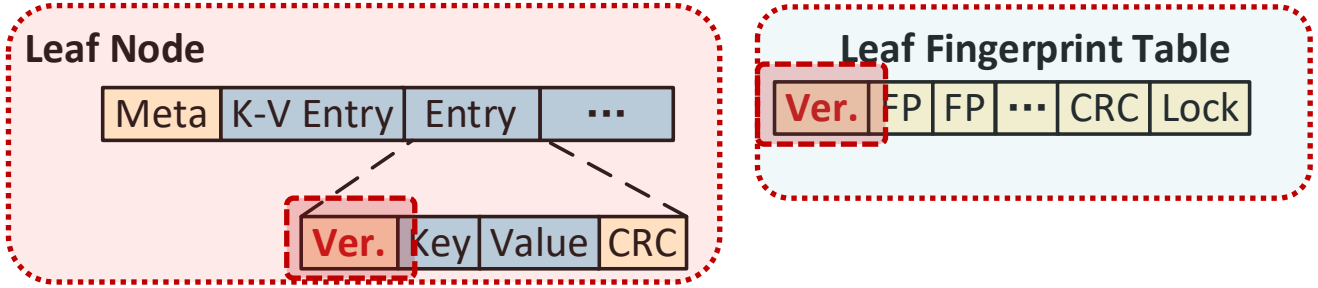
Compute-Side Collaborative Cache

➢ Detect and Resolve Fingerprint Inconsistency

Handling fingerprint mismatches caused by ***Node Split***



- ***Version fields validation***
- (cache version == primary version)

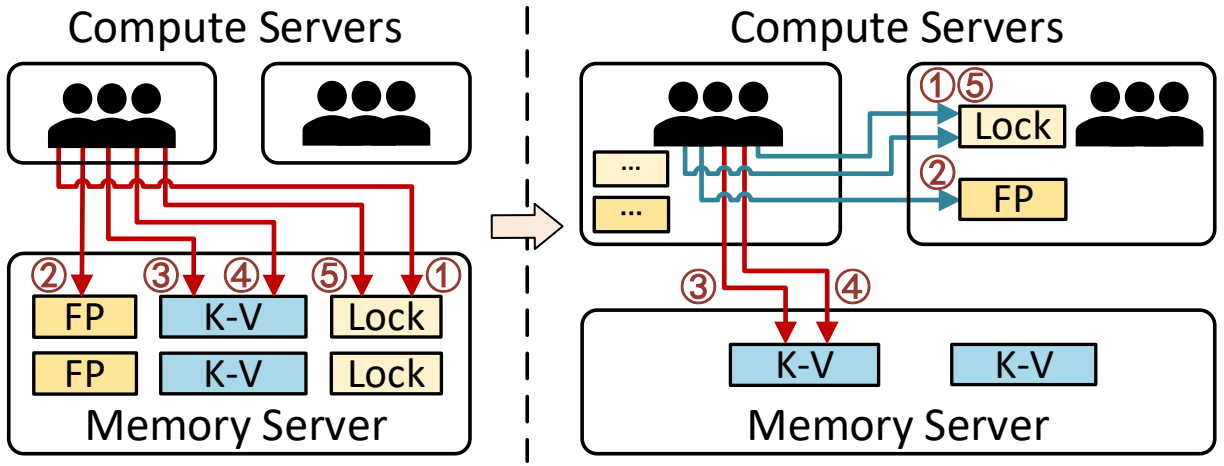


- If not match
- Refresh internal-tree and fingerprint cache

Further accelerate fingerprint access across compute servers,
with consistency ensured through validation fields.

Offload Locking Operations

- Distributed Lock Fields on Compute Servers
- Offloading Concurrent Lock Operations to the Compute-side



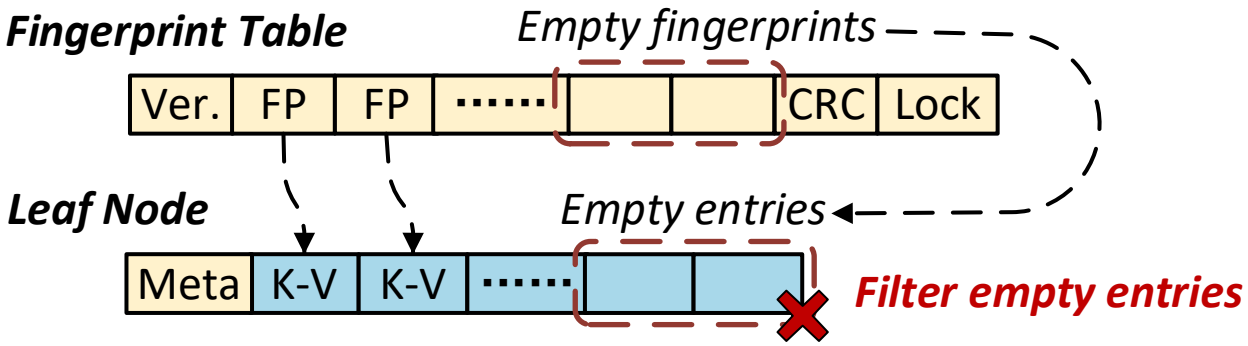
Offload Locking to Compute Servers

①⑤: Lock & unlock ②: Fingerprint read ③④: Entry read & update

Avoid additional RDMAs for locking concentrated on memory server,
further alleviate memory-side IOPS bottleneck.

Other Optimization

- Other ***RDMA Resource Optimization*** through Compute-Side Collaborative Design
- E.g., Filtering Empty Entries During Scan Operations Using Fingerprint Tables



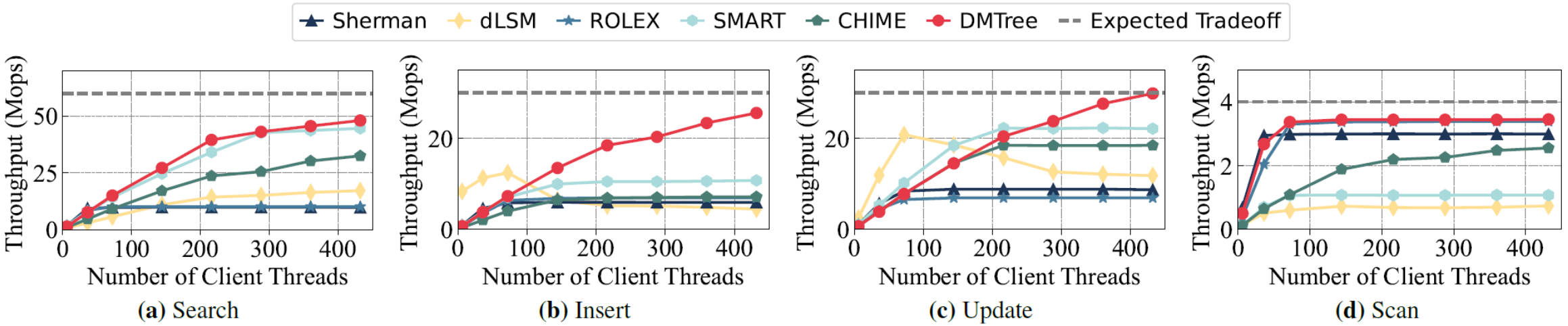
Further details can be found in the paper.

Experiments

- Testbed:
  - 7-node cluster, ***6 compute servers + 1 memory server***
  - Intel Xeon Gold CPUs, 128 GB DRAM, 100 Gbps Mellanox ConnectX-6 RNIC
- Workloads:
  - YCSB workloads with ***Uniform & Zipfian*** distributions
  - Microbenchmarks: search, insert, update, scan & YCSB A/B/C/D/E/F
- Comparisons:
  - Prototype Implementation: **DMTree**
  - Sherman@SIGMOD'22 (B+-tree), ROLEX@FAST'23 (Learned)
  - SMART@OSDI'23 (ART), CHIME@SOSP'24 (Hybrid), dLSM@ICDE'23 (LSM)

Performance Evaluation

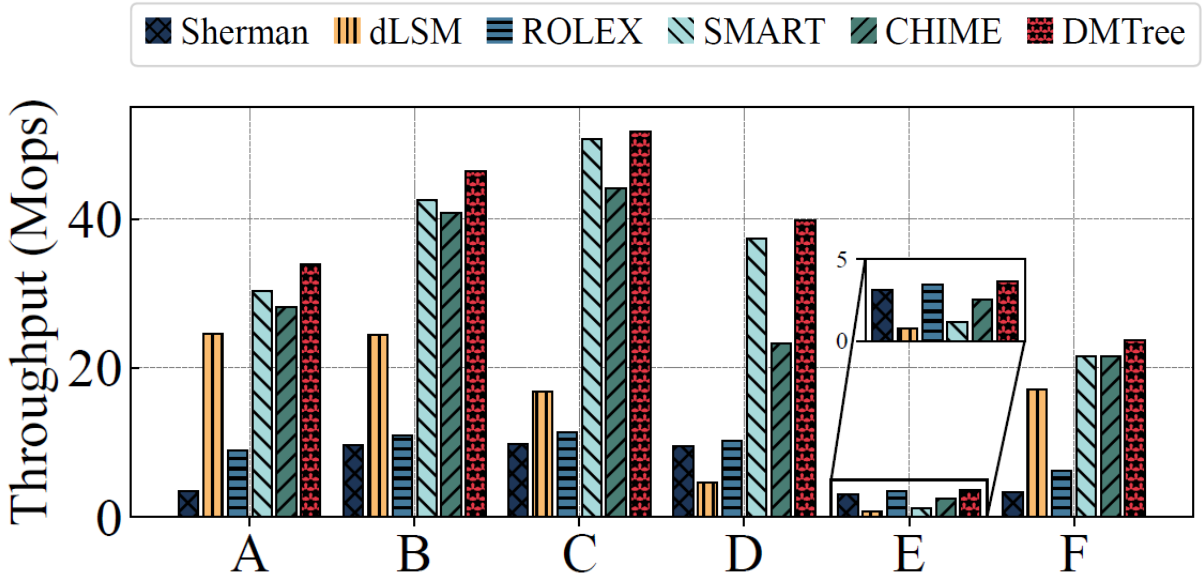
➢ Microbenchmarks with Varying Numbers of Client Threads



Conclusion: Under microbenchmarks, compared to baselines, DMTree achieves high performance across search/insert/update/scan, ***improving overall performance by up to 5.7×.***

Performance Evaluation

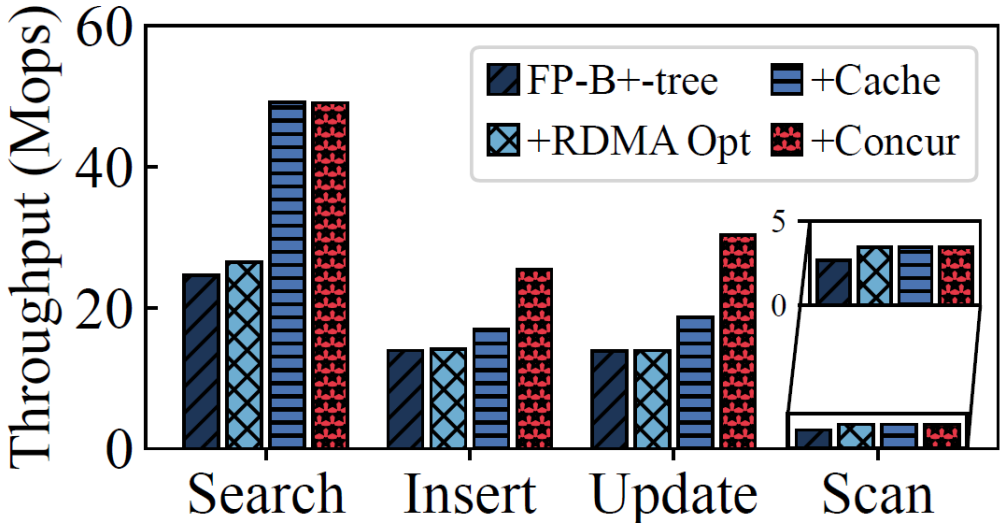
➢ YCSB A/B/C/D/E/F with Varying Numbers of Client Threads



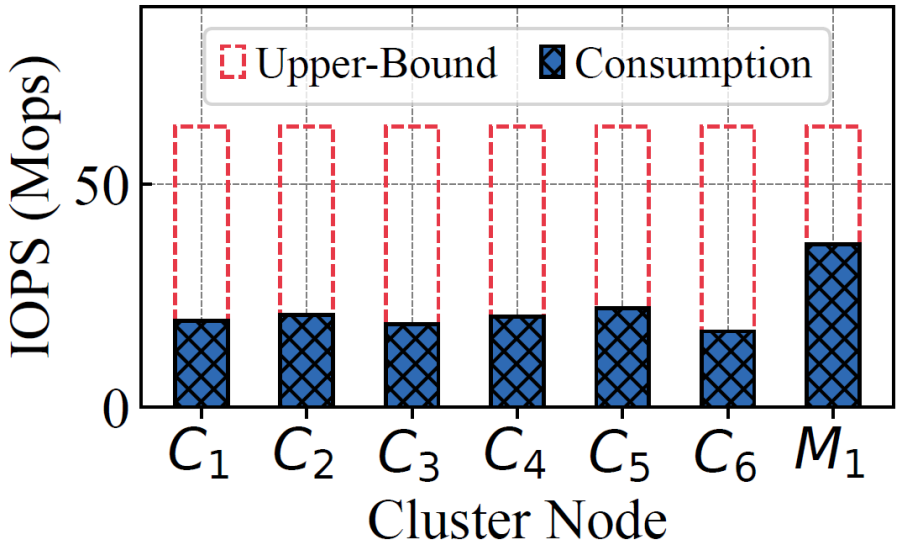
Conclusion: Under diverse YCSB workloads, DMTree consistently outperforms existing disaggregated-memory range indexes in overall performance.

Performance Evaluation

➢ Effectiveness of the Design Techniques



With compute-side collaborative design



Conclusion: By leveraging RDMA resources across compute servers, DMTree mitigates memory-side bottlenecks and improves indexing performance.

Conclusions

- **DMTree:** Towards Efficient Tree Indexing on Disaggregated Memory via Compute-side Collaborative Design
  - Offloading precise key–value locating, Compute-side collaborative cache, Offloading concurrent locking operations
- More evaluation results and analysis are in the paper
- The source code is at https://github.com/muouim/dmtree
  - For artifact evaluation materials and setup instructions, please refer to the following repository: https://github.com/muouim/aefast26

Thanks for your attention!

Q&A

Contact email:
weiguoli@mail.ustc.edu.cn