RLSM: A Disaggregated LSM-Based Key-Value Store with Resource-Aware Compaction

Zhiyuan Lou, Guoli Wei, Qingyang Zhang, Yongkun Li, Yinlong Xu

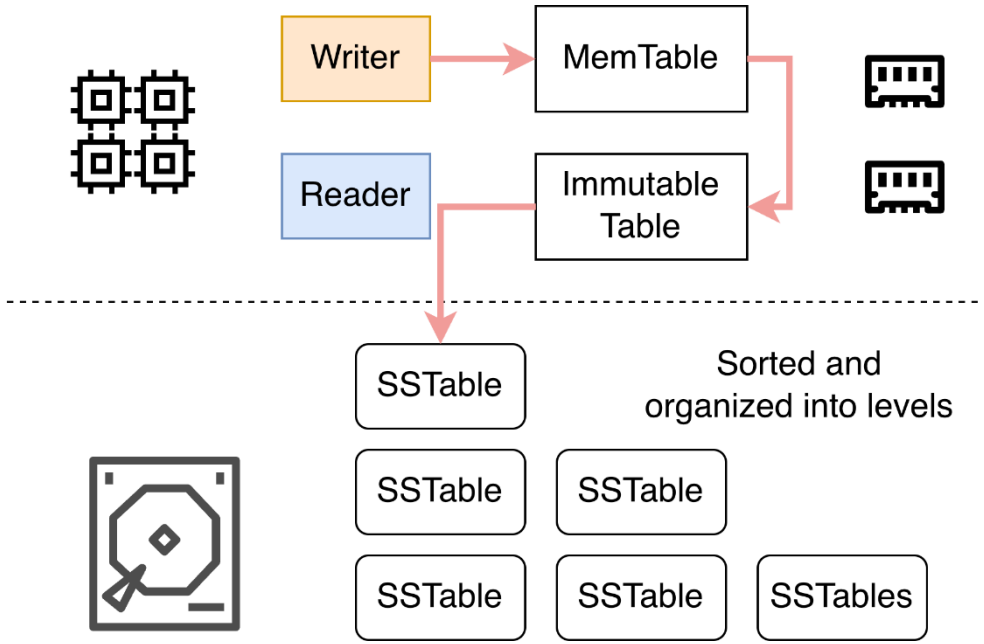


中国科学技术大学
University of Science and Technology of China

The 18th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '26)

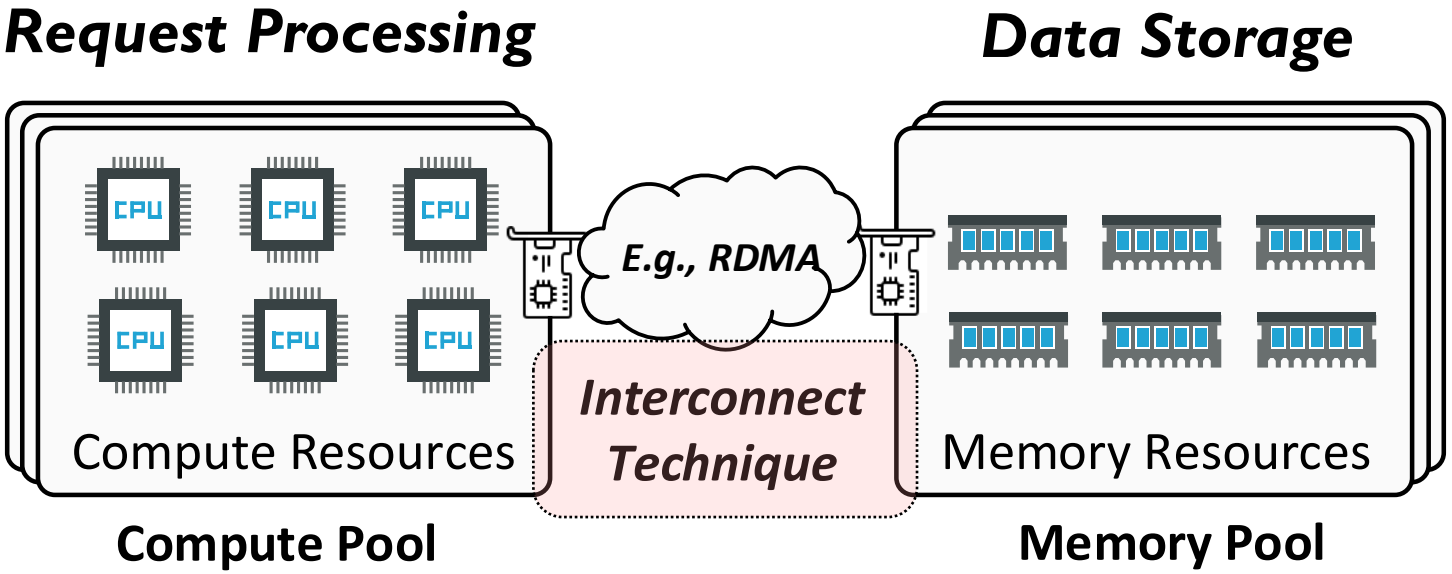
Background

- LSM-Based Key-Value Store
  - Simple and efficient Put / Get / Delete operations
  - Convert random writes into sequential appends for **high write throughput**
  - Compaction merges immutable sorted data files (SSTables)



Background

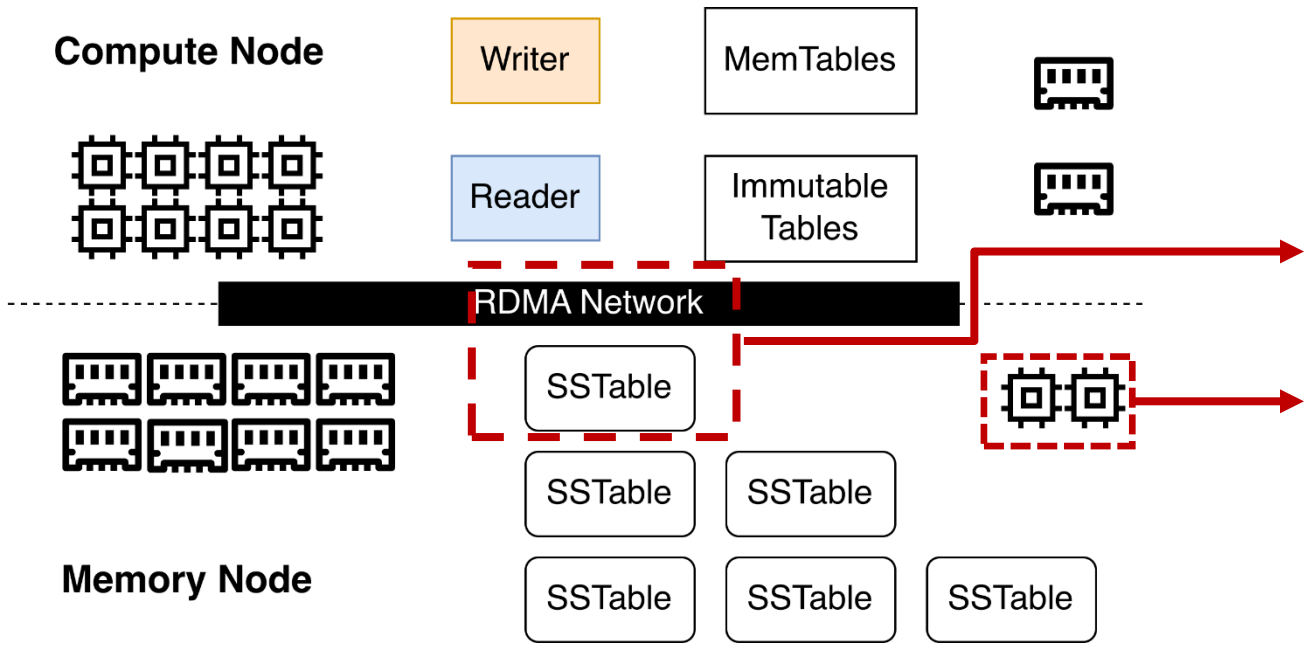
- Disaggregated memory (DM)
  - Resource separation and pooling



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

Background

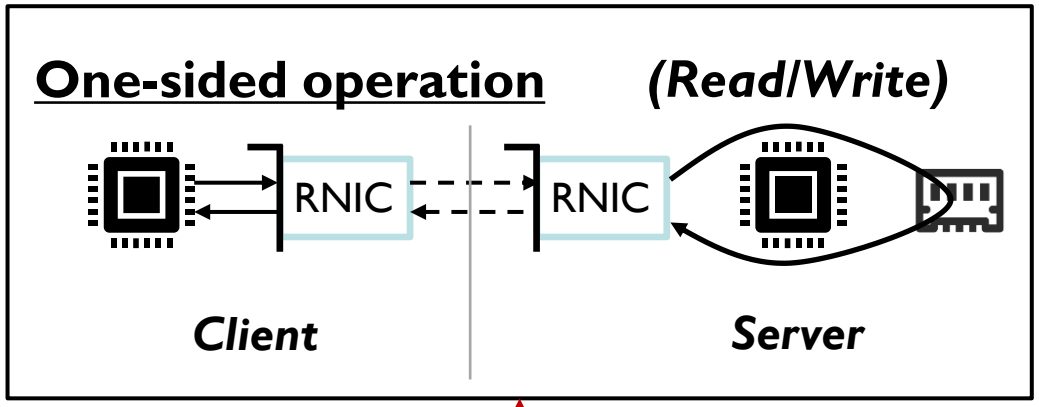
- LSM-tree in Disaggregated memory
  - One LSM-tree spans **multiple nodes**
  - Compute nodes handle requests and maintain in-memory write buffers
  - Memory nodes store large volumes of SSTables

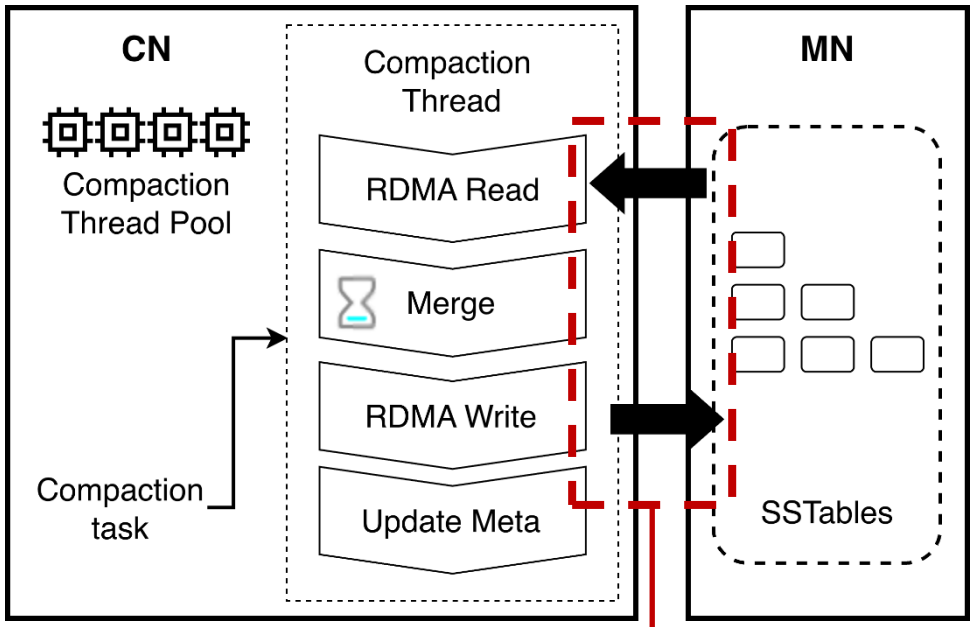


- Supports one-sided RDMA reads and writes.
- Enables near-data computation.

Background and Problem

- Compaction for reorganizing SSTables
  - CN manages LSM metadata and triggers compaction tasks
  - ① **CN-side compaction** remotely reads and writes large SSTables over RDMA

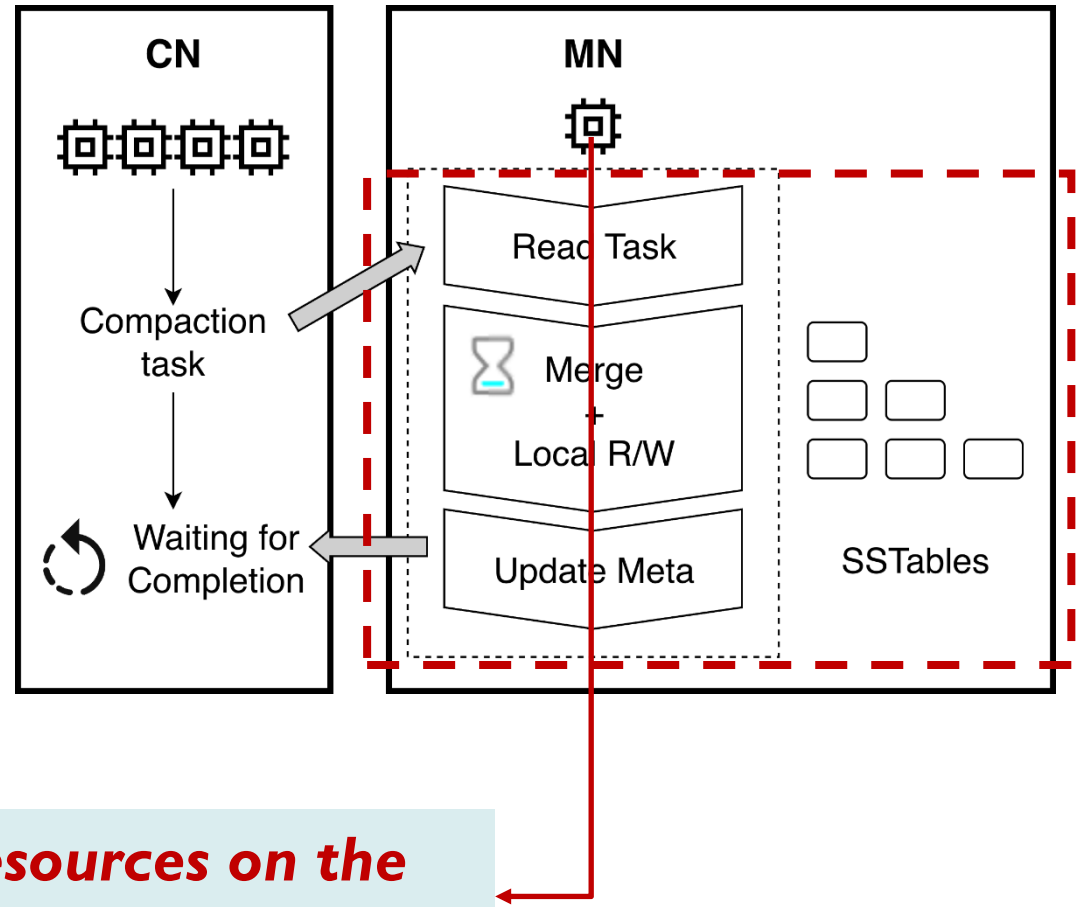




***Bandwidth Bottleneck*: data transfer rate (e.g., 100 Gbps, Mellanox ConnectX-6)**

Background and Problem

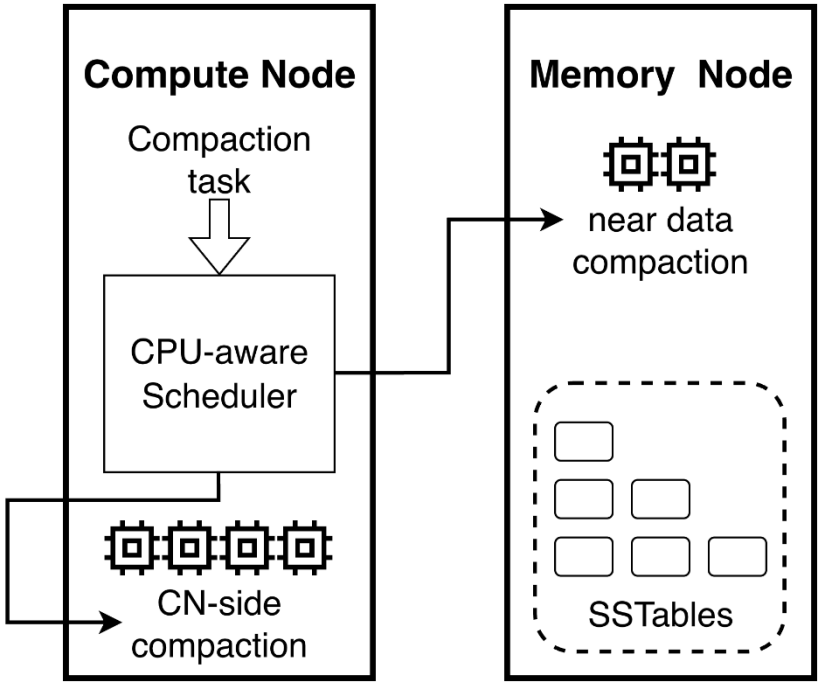
- Different Compaction Paths
  - CN manages LSM metadata and triggers schedulable compaction tasks
  - ② **MN-side (near-data) compaction** merges SSTables using CPU close to the data
  - Local access to SSTables



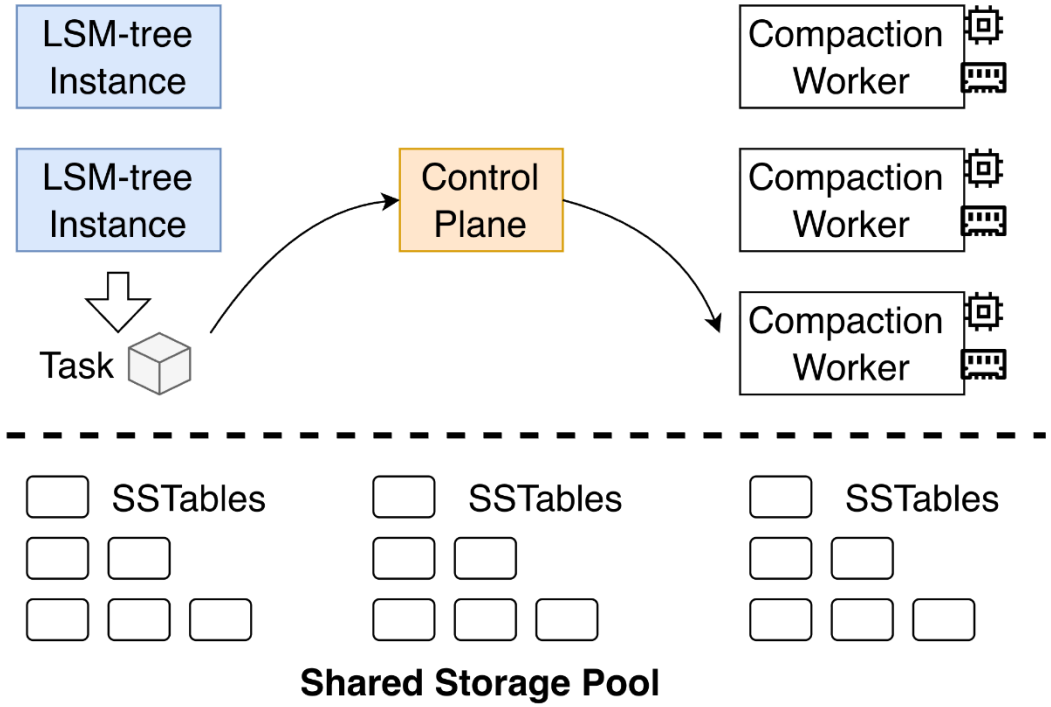
Limited CPU resources on the memory-side

Existing Designs

- CPU-Aware Scheduling



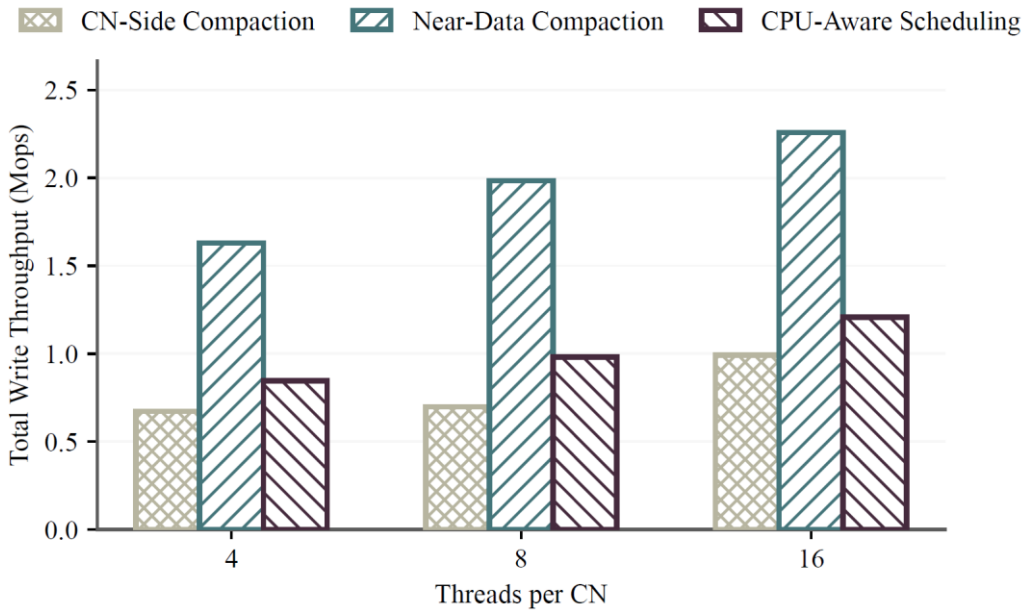
dLSM-dynamic@VLDBJ'24



HailStorm@ASPLOS'20, CaaS-LSM@SIGMOD'24

Existing Designs

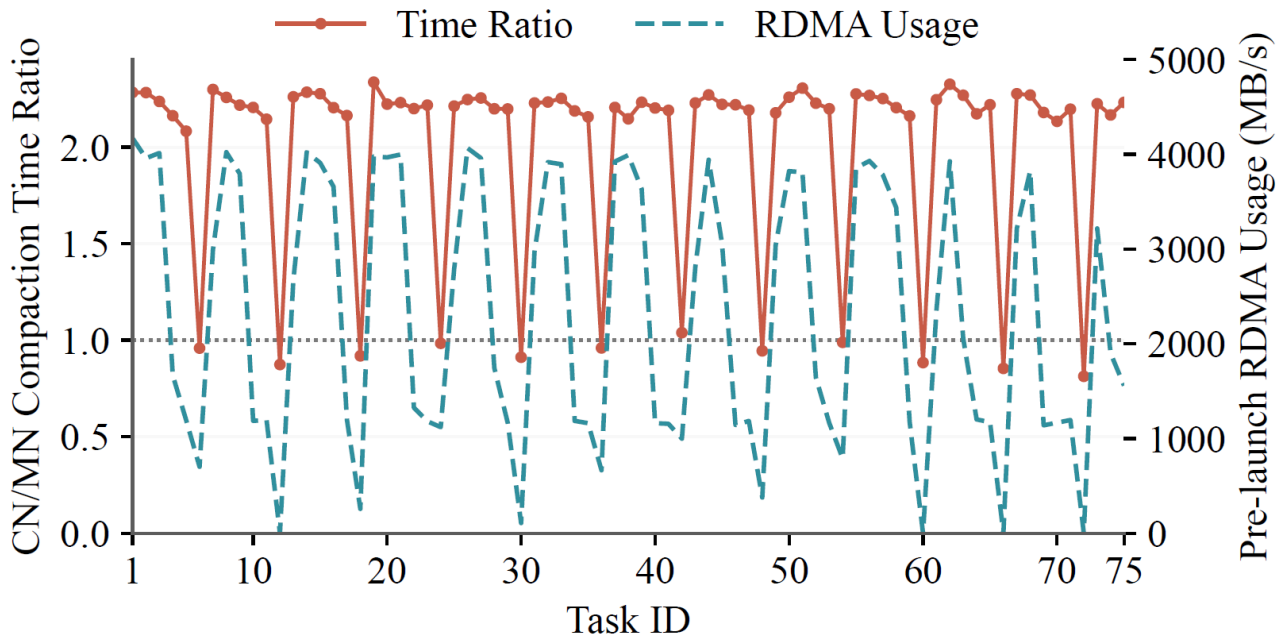
- Convergence toward CN-Side Compaction
  - RDMA bandwidth is the primary bottleneck for CN-Side Compaction
  - CN CPUs remain underutilized
  - **CPU-aware scheduling favors CN-side compaction**



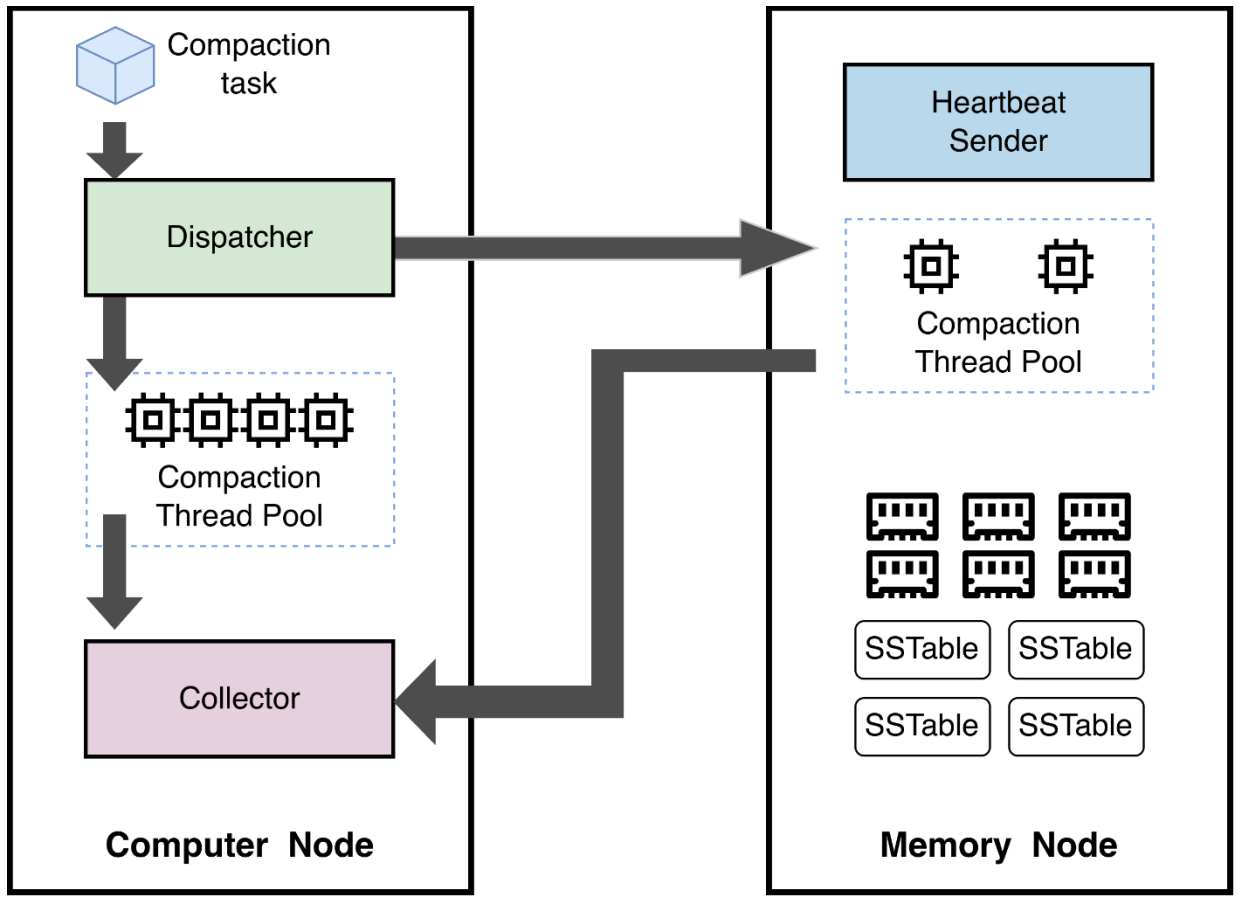
CPU-Aware Scheduling: Fails to exploit both compaction paths effectively

Motivation

- How to Schedule Compaction Effectively ?
- **Insight:** Schedule by **critical resources**, not CPU utilization
  - Near-data compaction as the preferred path
  - Dispatch to CN only with sufficient RDMA headroom



RLSM Compaction Management



Critical Resource Reporting

Resource-Aware Dispatcher

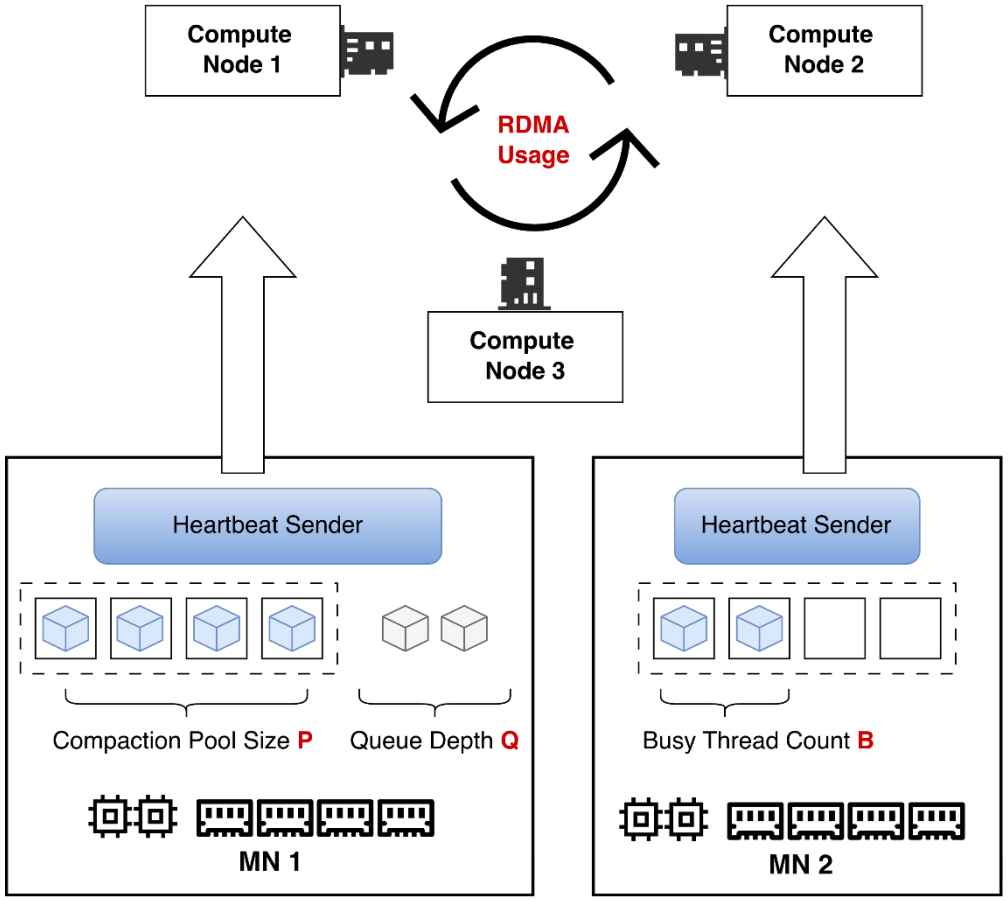
- Multi-stage admission control
- Priority-aware scheduling

Independent Collector

- Decouples completion handling
- Batches metadata updates

Resource Heartbeat

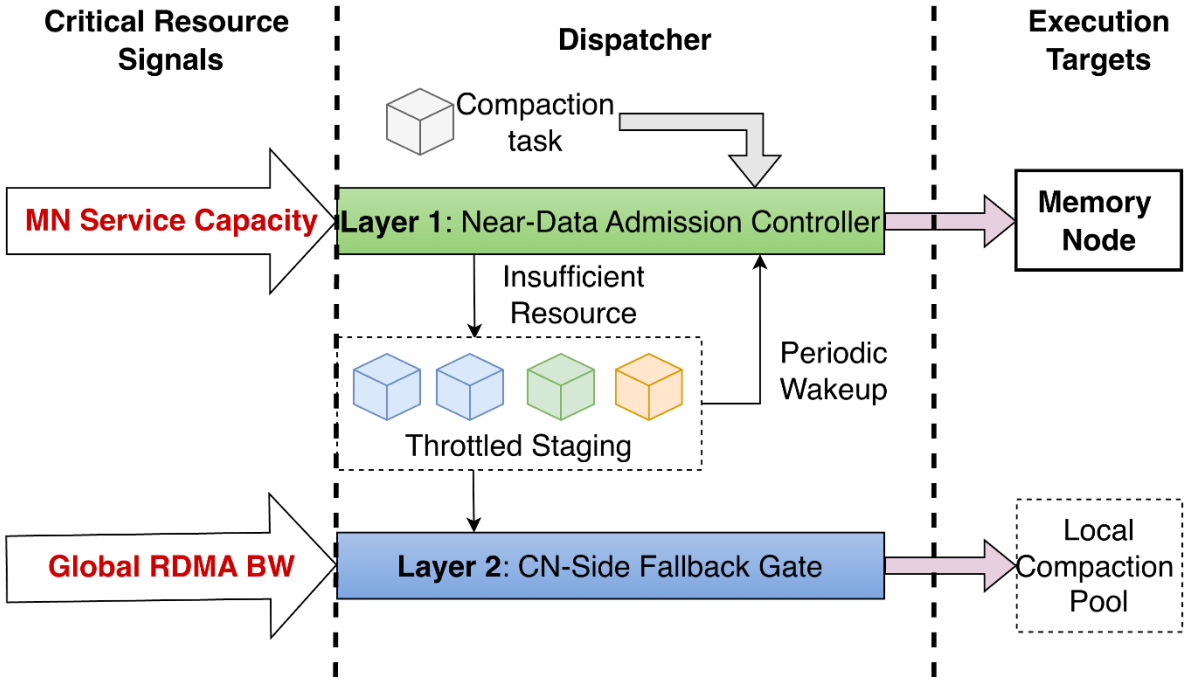
➢ Critical resource information aggregated at CNs



- Report only **compaction-critical resources**
- CNs exchange shared RDMA usage
- MN reports worker load and queue depth
- CN caches **the latest resource snapshot**

Two-Stage Dispatcher

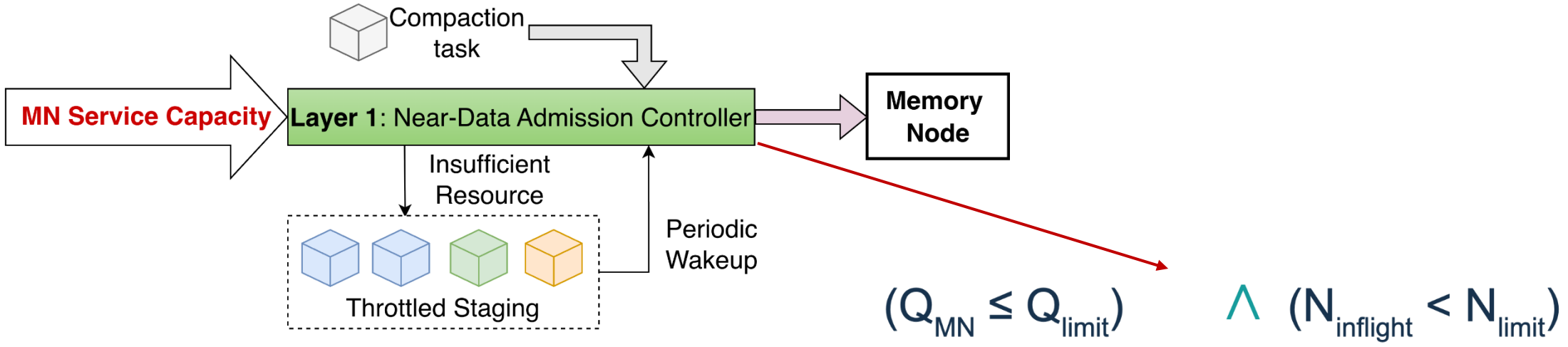
➢ Schedule compaction based on resource snapshots



- Avoid single-resource decisions
- Throttle and retry
- Ensure effective, resource-safe scheduling

Two-Stage Dispatcher

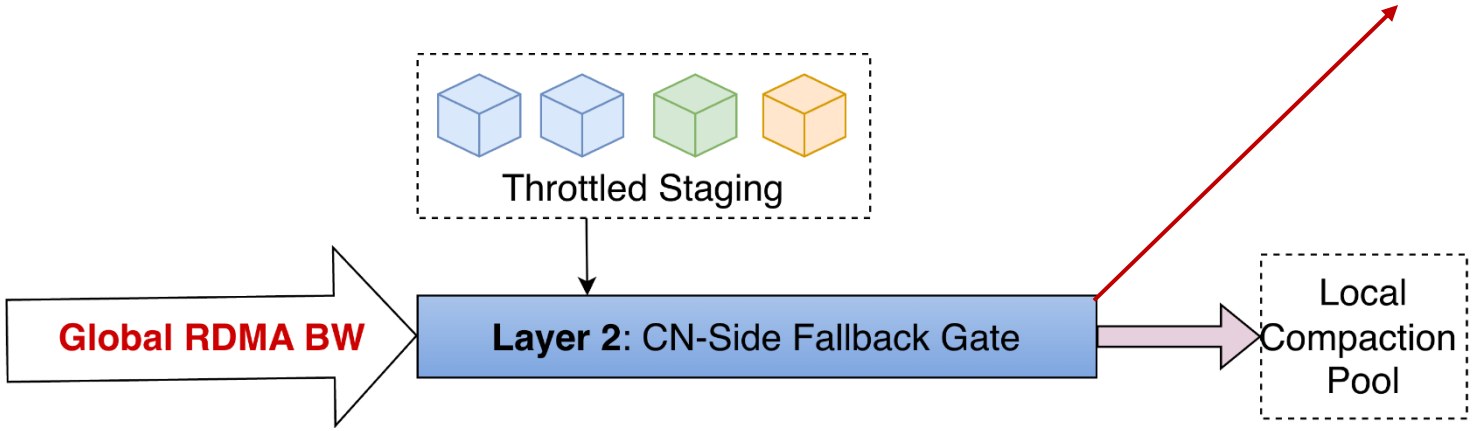
- Near-Data Admission Controller
  - Near-data compaction is the preferred path for efficiency
  - Based on **queue depth** and local **in-flight task** count
  - The thresholds depend on task classification.



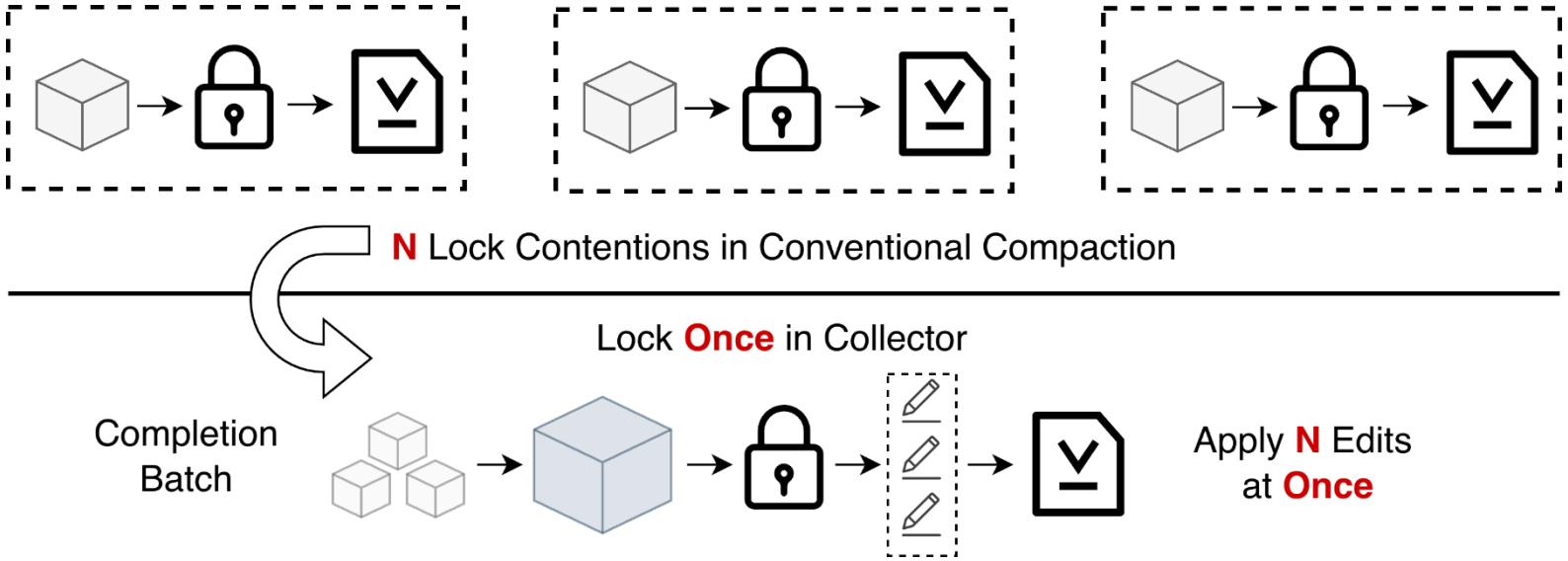
Two-Stage Dispatcher

- CN-Side Fallback Gate
  - Failed tasks in Layer 1 are staged and wait for another opportunity
  - Staged tasks may fall back to CN-side execution
  - Fallback only with **sufficient RDMA headroom**

$$(B_{local} + \Delta B_t < B_{local}^{max}) \quad \wedge \quad (B_{global} + \Delta B_t < B_{global}^{max})$$



Collector



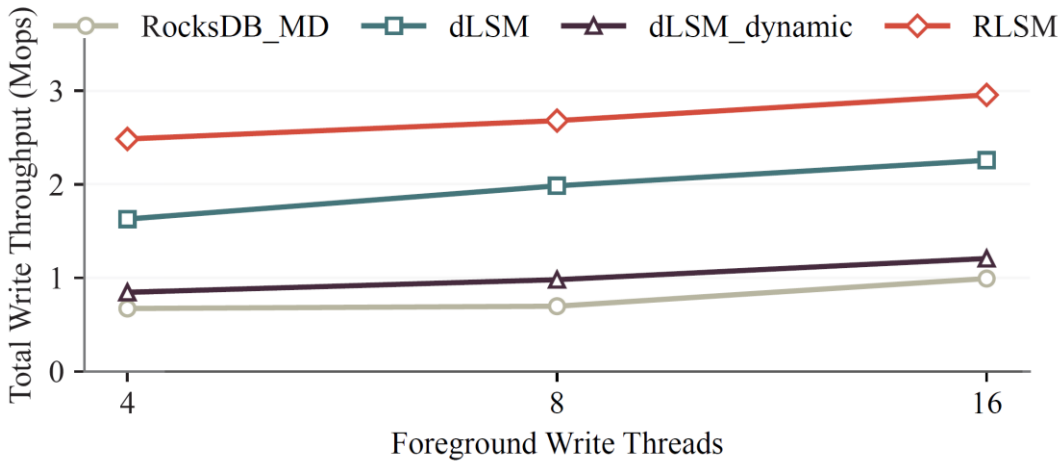
- Completion Batching
  - Handle completed tasks from both CN and MN.
  - Single-threaded management **reduces lock contention.**
  - Batch multiple metadata updates and publish **a single latest version**.

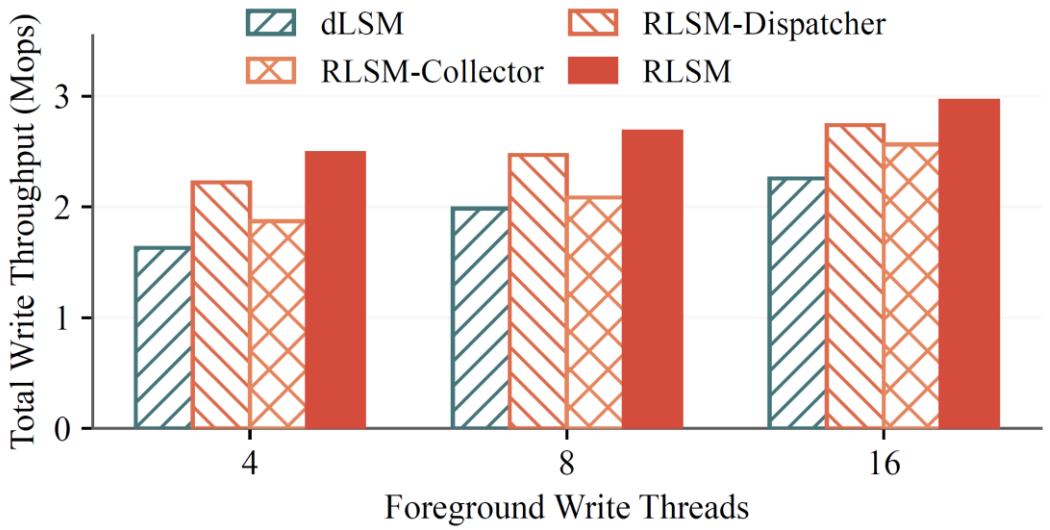
Experiments

- Testbed:
  - 4-node cluster, ***3 compute nodes + 1 memory node***
  - AMD EPYC 7543 CPUs, 256 GB DRAM, Mellanox ConnectX-6 100 Gbps RNIC
- Workloads:
  - RocksDB-style ***db_bench*** workload, 4–16 foreground read/write threads per CN
  - ***fillrandom*** write workloads and ***readrandomwriterandom*** mixed workloads.
- Comparisons:
  - Prototype Implementation: **RLSM**
  - *RocksDB_MD* represents CN-side compaction
  - *dLSM@ICDE'23* represents Near-Data Compaction
  - *dLSM-dynamic@VLDBJ'24* represents CPU-Aware Scheduling

Performance Evaluation

➢ Write Performance & Ablation Study





Conclusion: RLSM delivers ***31%–53%*** higher aggregate write throughput than the strongest baseline. Ablation shows that both designs **independently improve** write performance through complementary mechanisms.

Mechanism Analysis

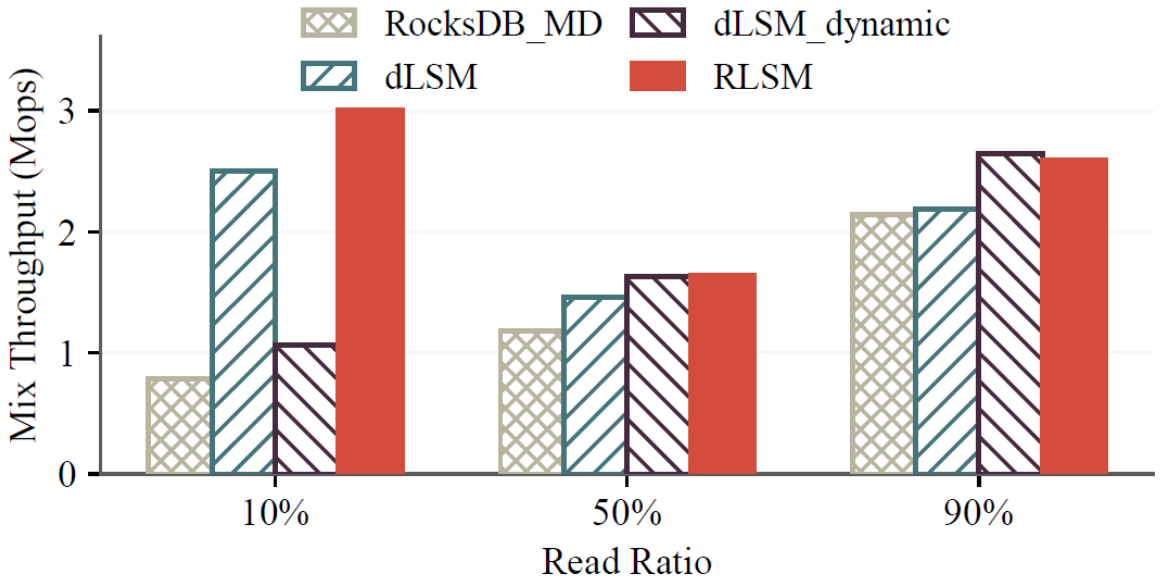
➢ Resource Usage

Strategy	CN compaction Util/Queue	MN compaction Util/Queue	RDMA BW Avg/Peak
RocksDB_MD	39.0/13.4	–	3.74/12.78
dLSM	–	94.8/29.2	1.08/2.17
dLSM_dynamic	34.0/8.2	20.6/8.6	2.79/11.15
RLSM	6.1/0.0	92.6/1.2	1.58/5.52

Conclusion: Critical-resource-aware scheduling reduces *near-data queueing* and suppresses *RDMA bandwidth spikes.*

Performance Evaluation

➢ Mixed Read/Write Workloads



Conclusion: Although optimized for write performance, RLSM maintains ***competitive performance*** under read-heavy workloads.

Conclusions

- **RLSM:** Dynamic placement in disaggregated LSMs must be driven by path-critical resources rather than CPU utilization alone.
- Core Mechanisms of RLSM:
  - Near-data-first two-stage scheduling
  - Each CN runs a single dispatcher and collector
  - Thread decoupling for accurate resource sensing
- More evaluation results and analysis are in the paper

Thanks for your attention!

Q&A

Contact email:
zhiyuan_lou2002@mail.ustc.edu.cn
weiguoli@mail.ustc.edu.cn