ROLEX: A Scalable RDMA-oriented Learned Key-Value Store for Disaggregated Memory Systems

Pengfei Li, Yu Hua, Pengfei Zuo, Zhangyu Chen, and Jiajie Sheng,

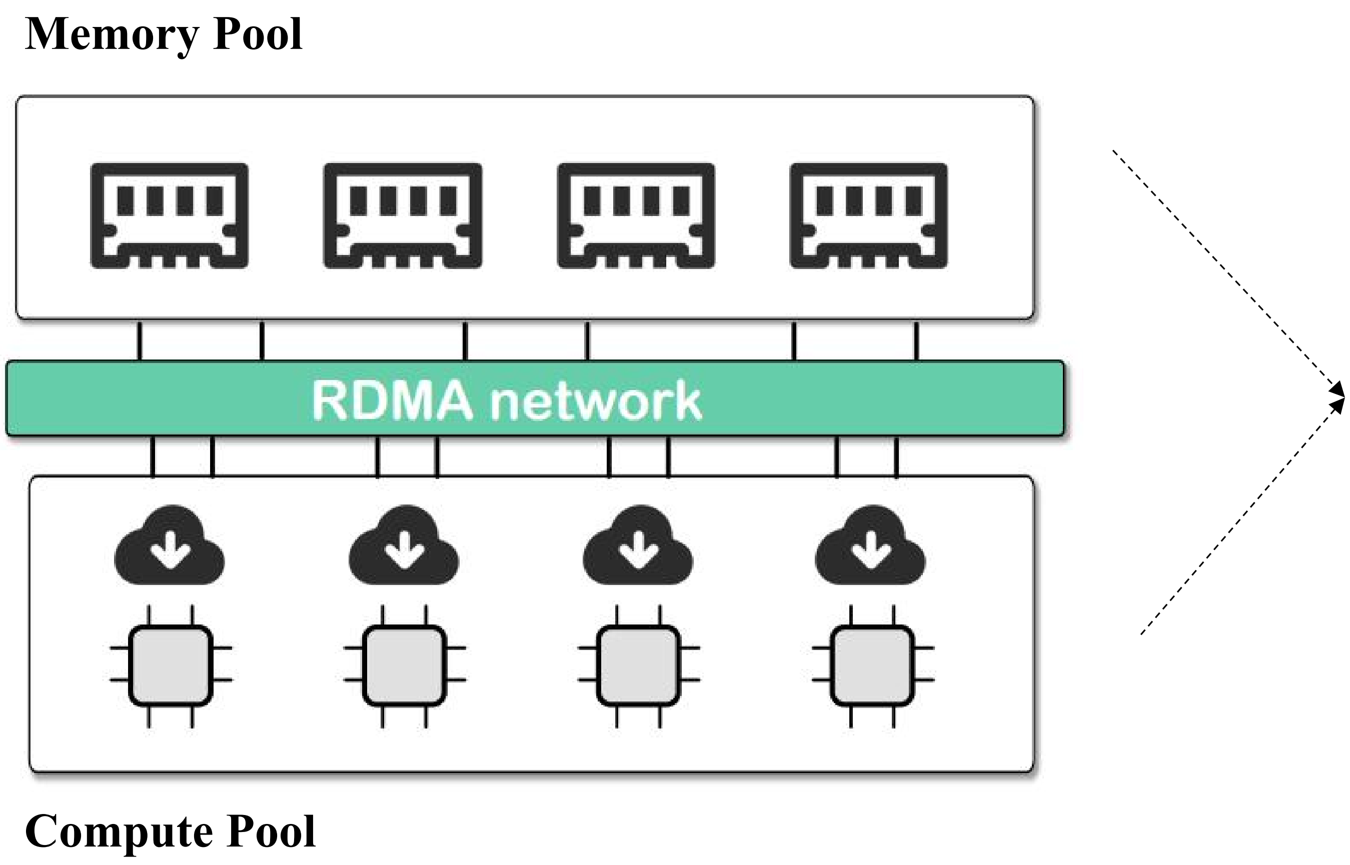
Huazhong University of Science and Technology

FAST'23 Best Paper

Speaker: Jun Wu

Background

Disaggregated Memory Systems



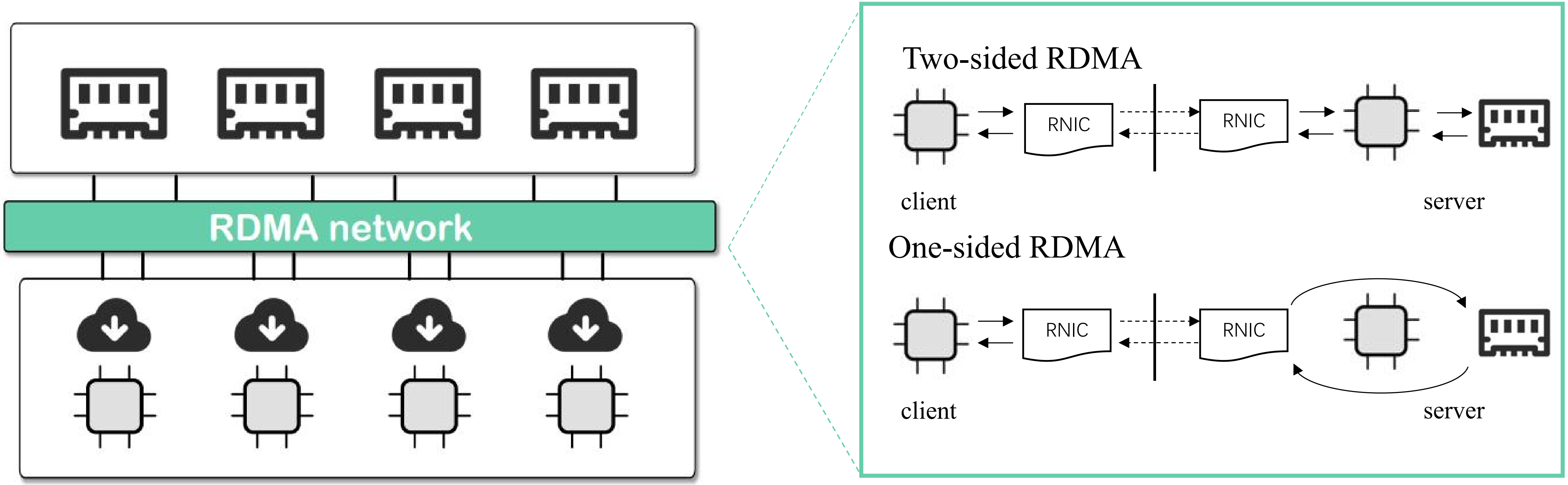
- ✓ **High resource Utilization**
- ✓ **Scalability**
- ✓ **Fault Isolation**
- ✓ **Data Sharing**
- ✓ **…**

Background

Disaggregated Memory Systems

Remote Direct Memory Access(RDMA)

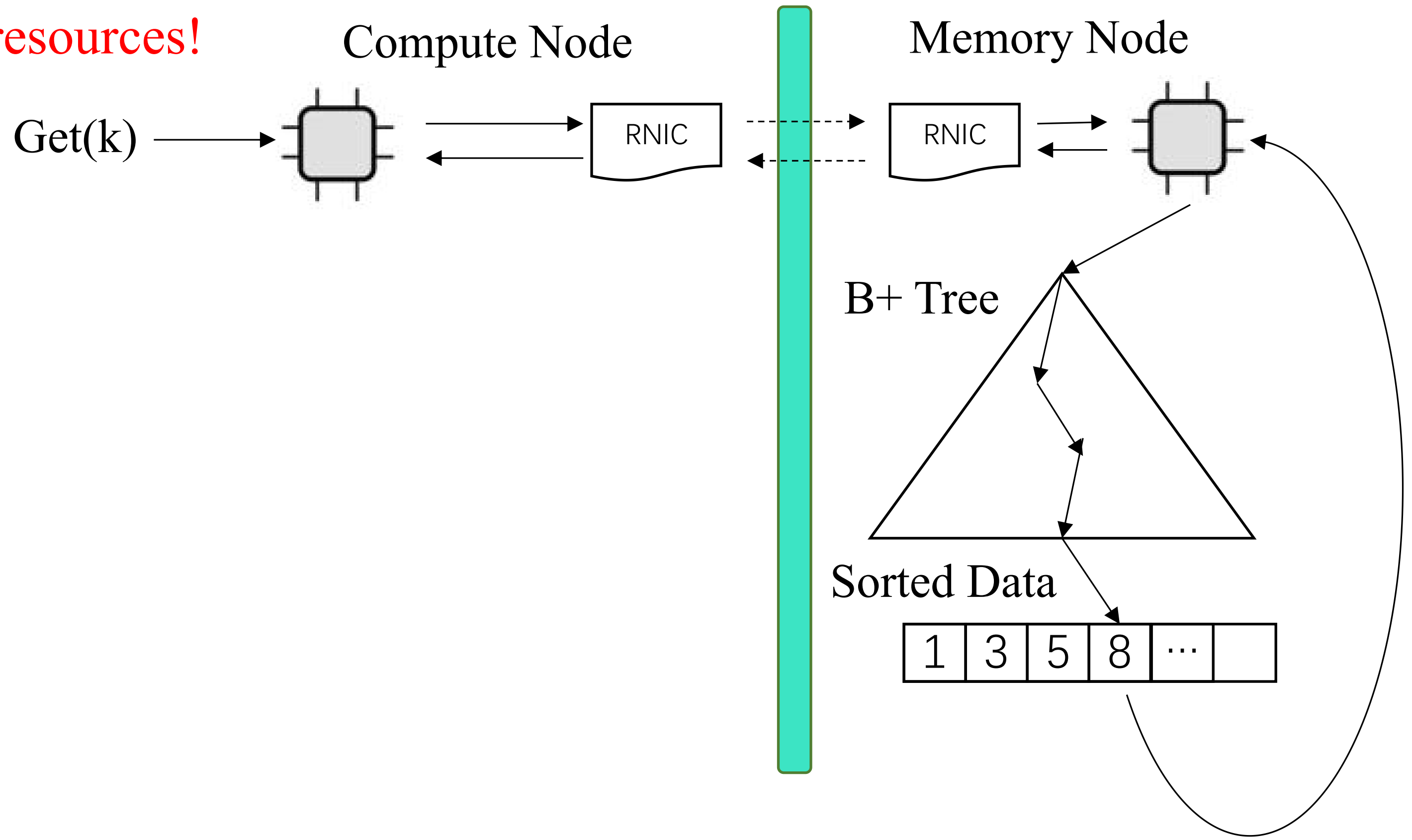
Memory Pool



Compute Pool

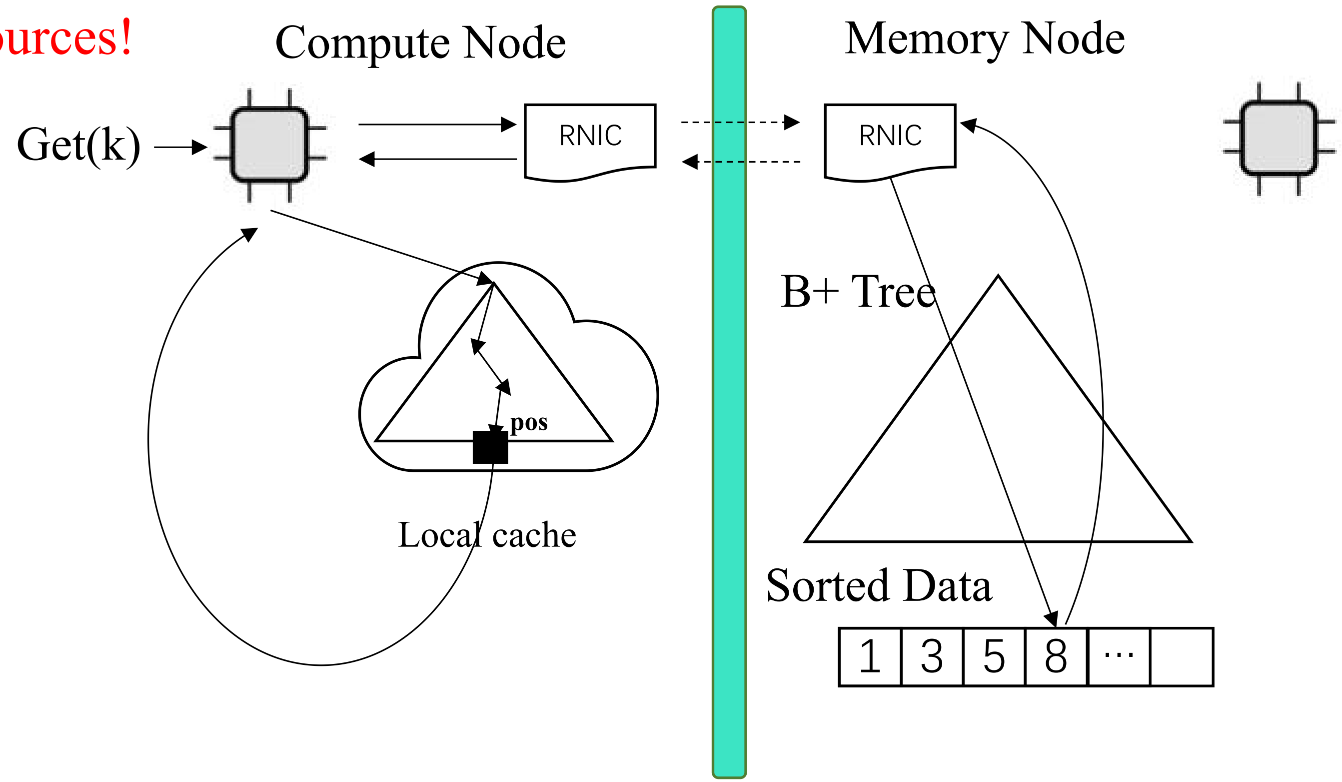
Background

- Deploying tree-based structure(ordered KV-store) in the disaggregated memory system (Two-sided)
  - Memory Node has limited computing resources!



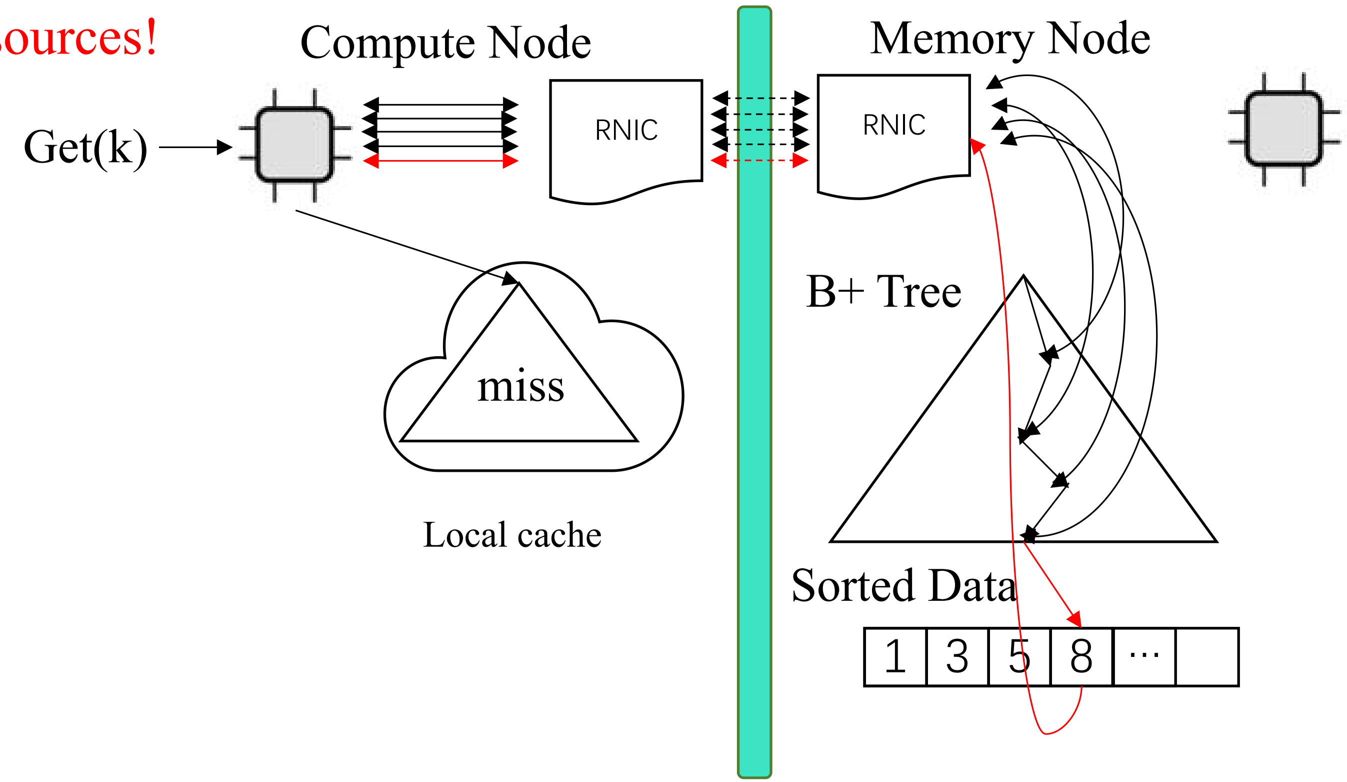
Background

- Deploying tree-based structure(ordered KV-store) in the Disaggregated Memory System (One-sided)
  - Memory Node has limited computing resources!
  - Memory consumption on compute node!



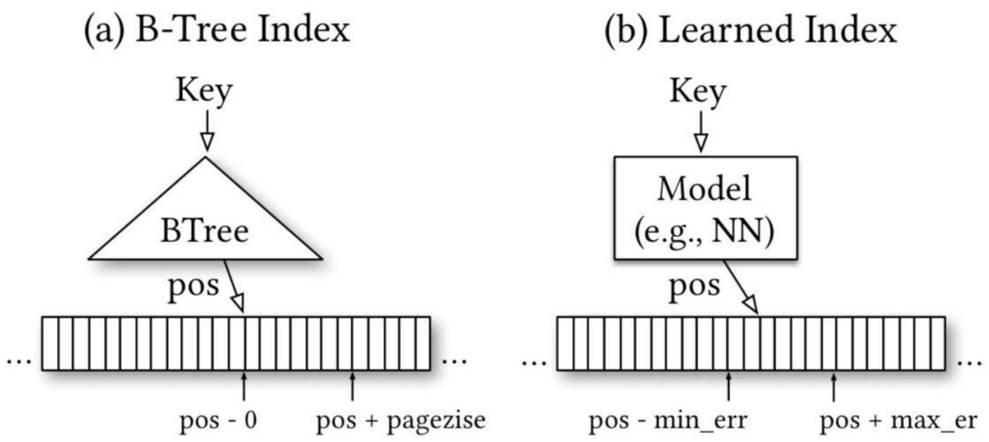
Background

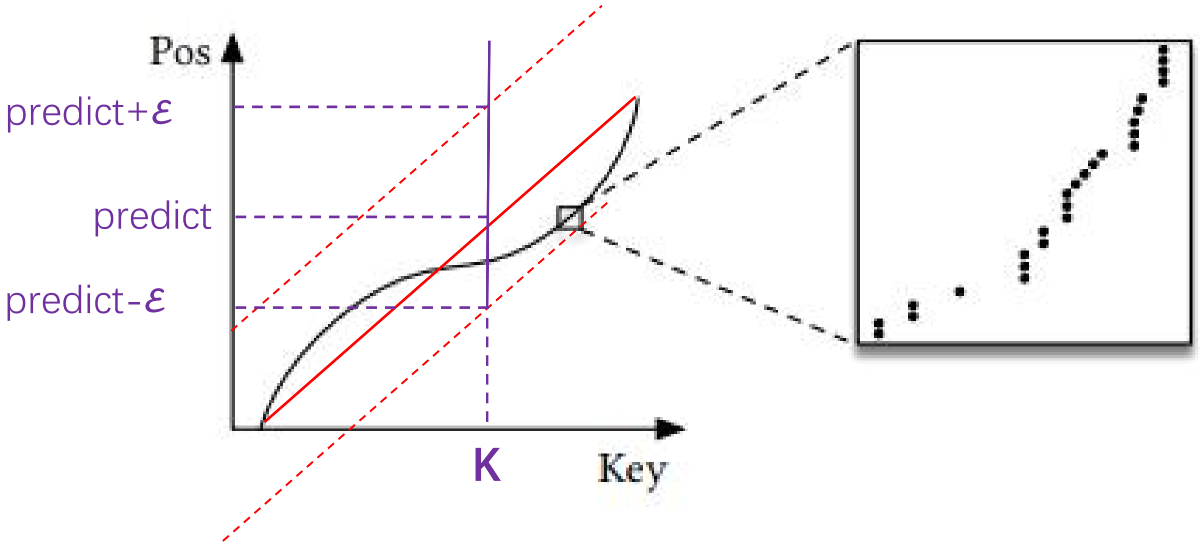
- Deploying tree-based structure(ordered KV-store) in the disaggregated memory system (One-sided)
  - Memory Node has limited computing resources!
  - Memory consumption on compute node!
  - Multi RTT reduce performance!



Background

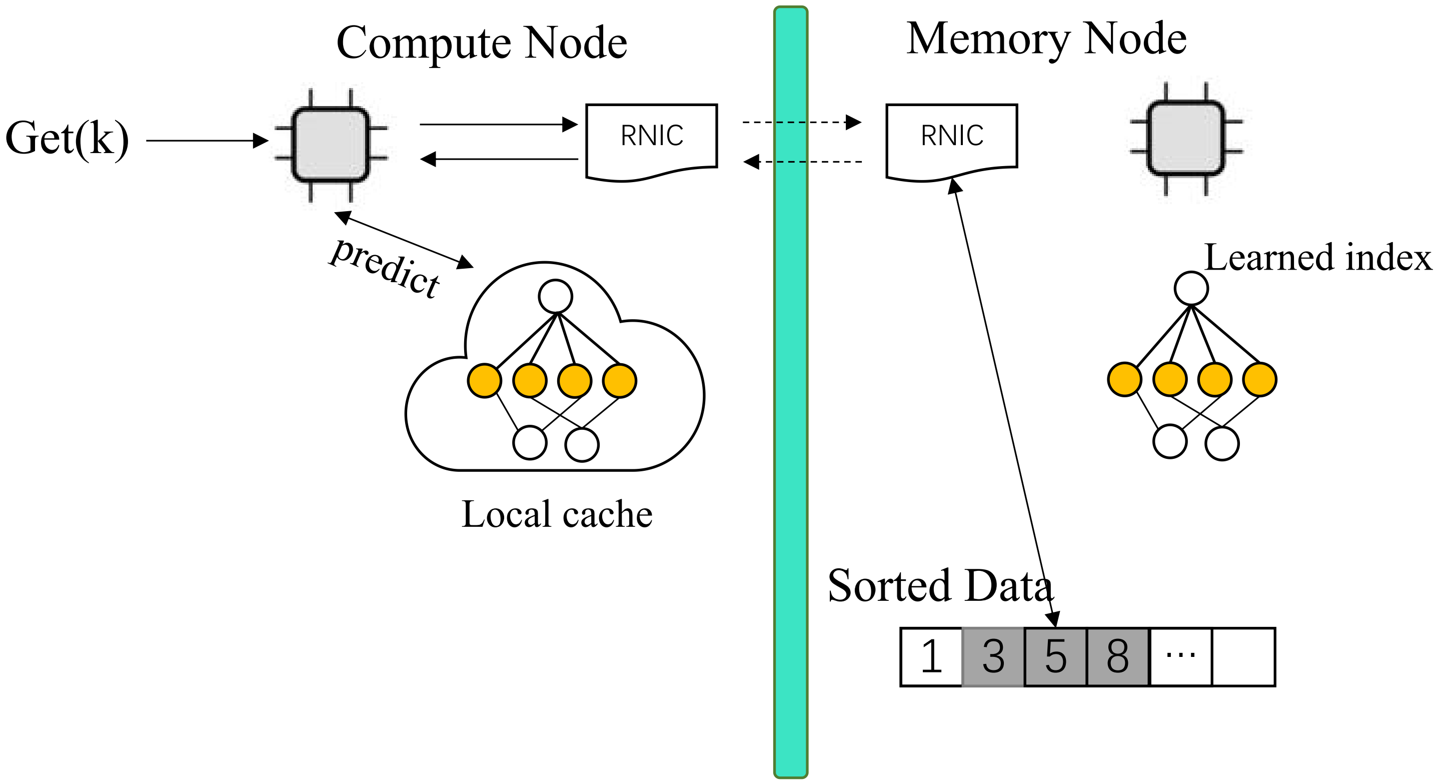
- Learned Indexes
  - Easy-to-use and small-sized learned models
  - 2-4 space-saving than tree-structured indexes
  - High searching speed than B+ tree indexes





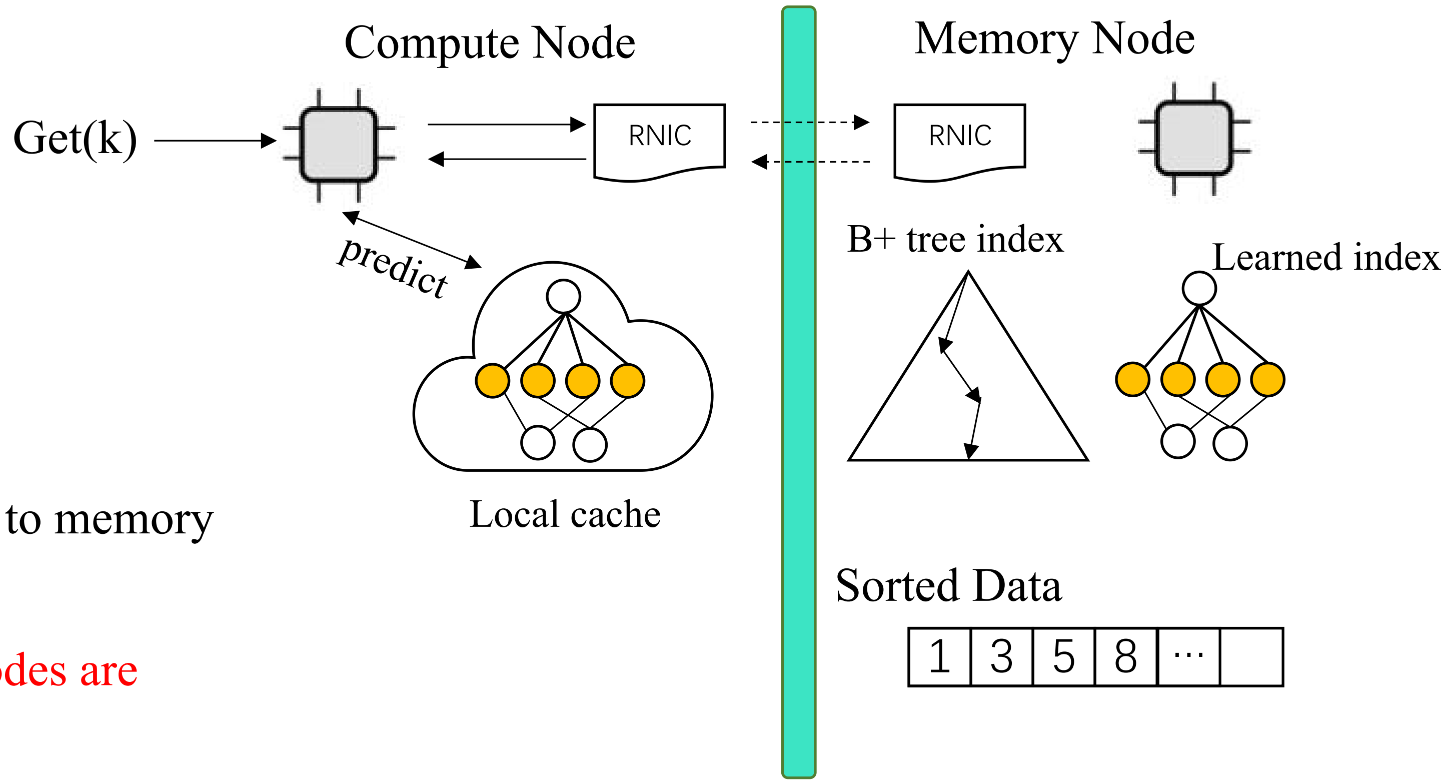
Background

- Deploying Learned Index structure(ordered KV-store) in the disaggregated memory system
  - Works well in Get(k)
  - How to manage Put(k, v)?



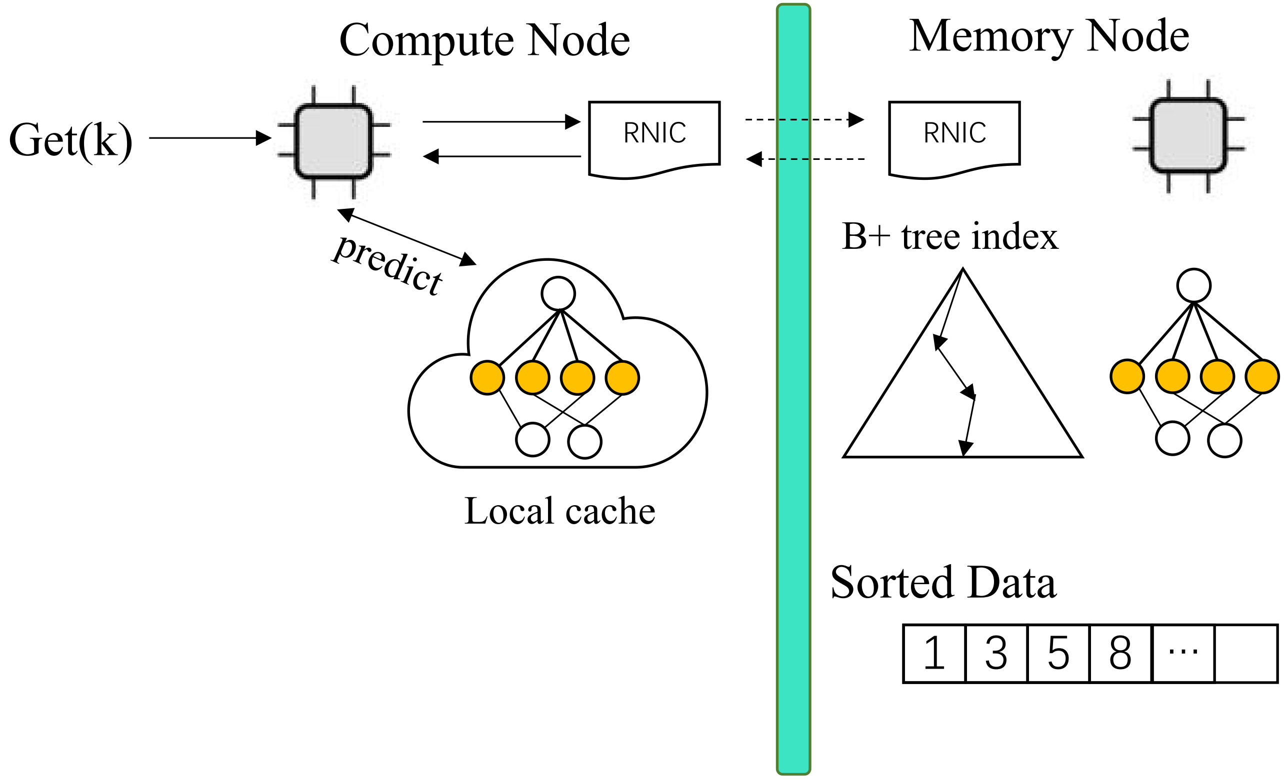
Background

- Deploying tree-based structure(ordered KV-store) in the disaggregated memory system
  - Works well in Get(k)
  - How to manage Put(k, v)?
    - Xstore @ OSDI'20
      - Read via learned index
      - Write via B+ Tree index
  - Xstore-D
    - transferring data modification requests to memory nodes
    - computing resources in the memory nodes are insufficient



Challenges

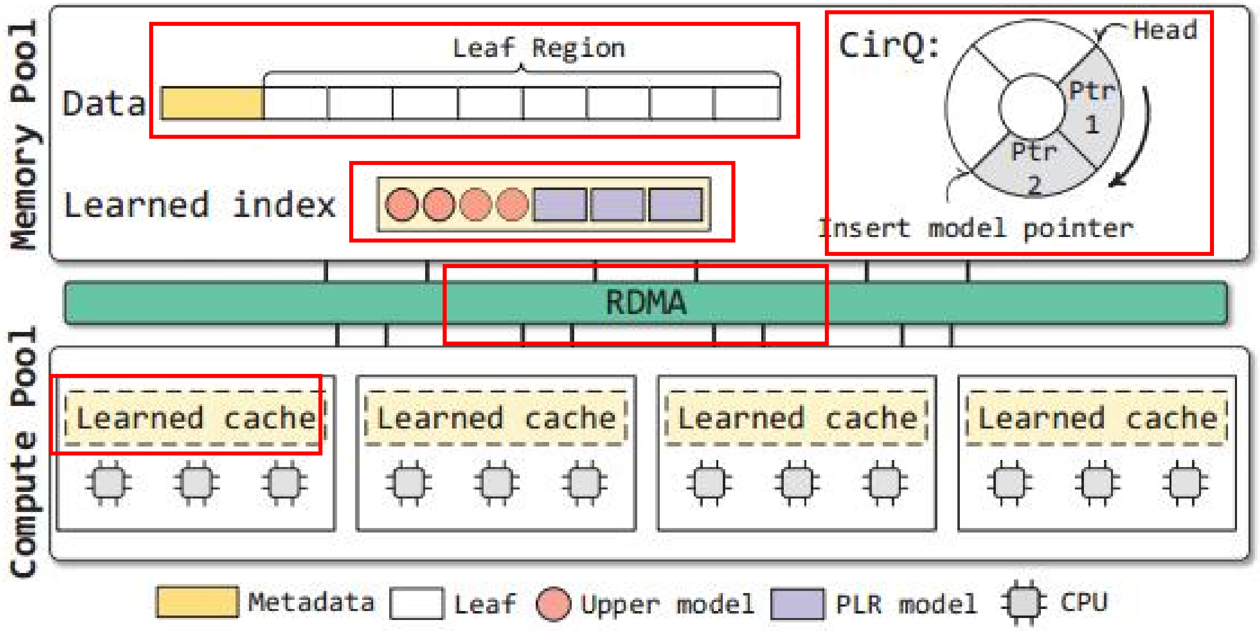
- Limited computing resources on memory nodes
- Overloaded bandwidth for data transferring
- Inconsistency issue among different nodes



ROLEX Design Overview

- Main Insight: Execute index operations with atomic designs and Asynchronously retrain models by decoupling the insertion and retraining operations with consistency guarantees.

- Design 1
  - Retraining-decoupled Learned Indexes
- Design 2
  - One-sided Index Operations
- Design 3
  - Asynchronous Retraining



Retraining-decoupled Learned Indexes(ch1)

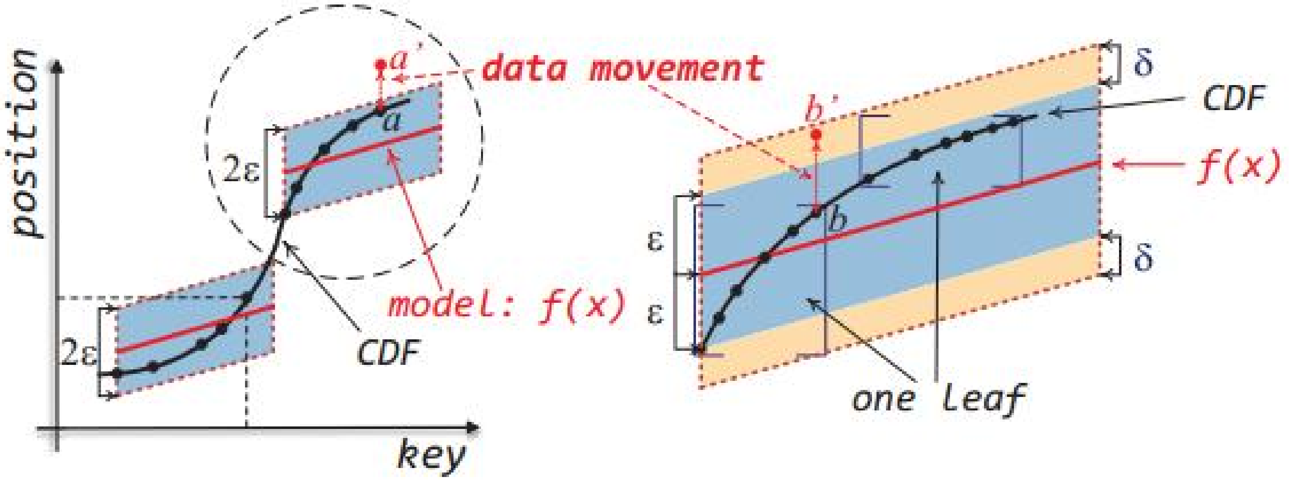
➢ Key idea: Modify training algorithm and add some constraints on data movements.

➢ Train the piecewise linear regression (PLR) models

➢ Adding a bias (represented as δ) to the prediction calculation

➢ Moving data within fixed-size(δ) leaves

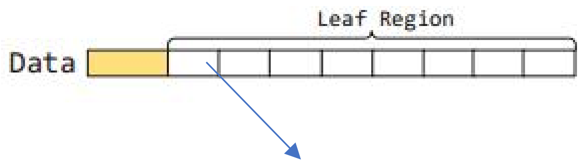
➢ Synonym-leaf sharing



$$\varepsilon >= max|f(X_i) - Y_i| \quad \forall i \in (0, N)$$

$$P_{range} = [f(X_i) - \varepsilon - \delta, f(X_i) + \varepsilon + \delta]$$

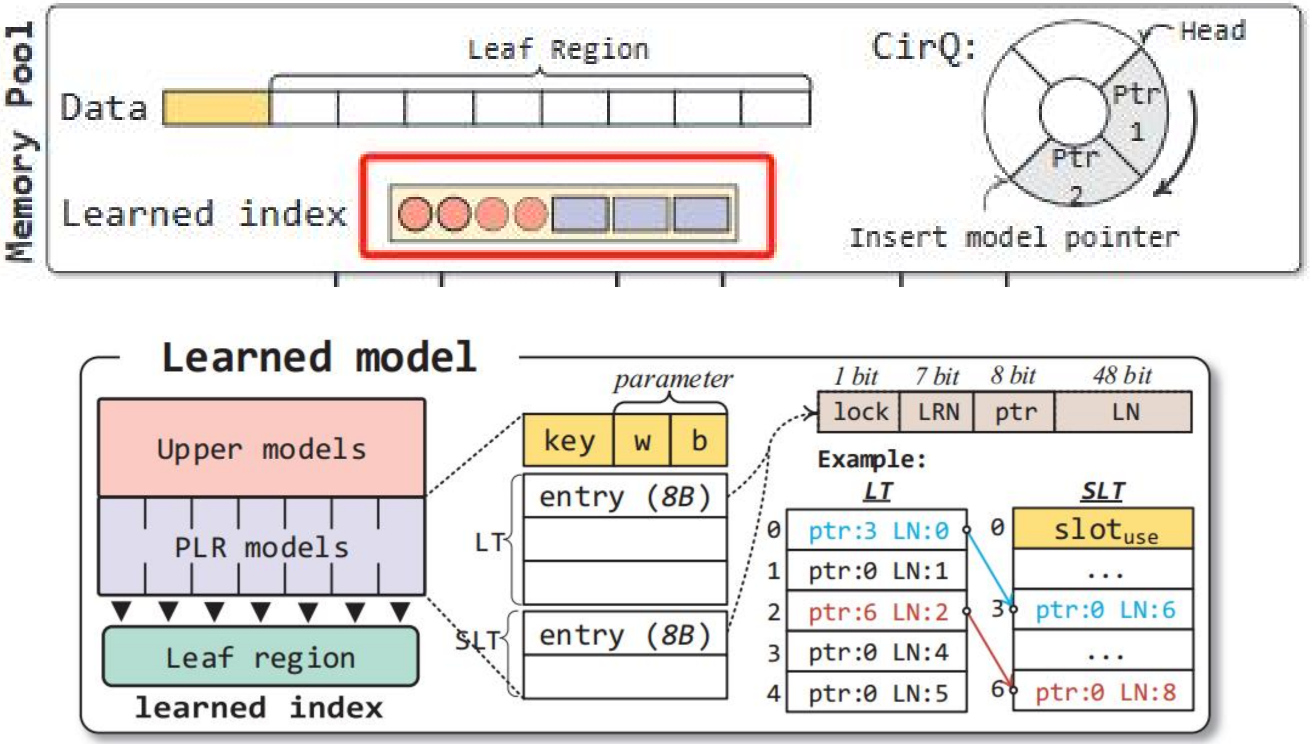
$$L_{range} = [\frac{f(X_i) - \varepsilon}{\delta}, \frac{f(X_i) + \varepsilon}{\delta}] \quad \forall i \in (0, N)$$



Contain δ data

One Sided Indexing

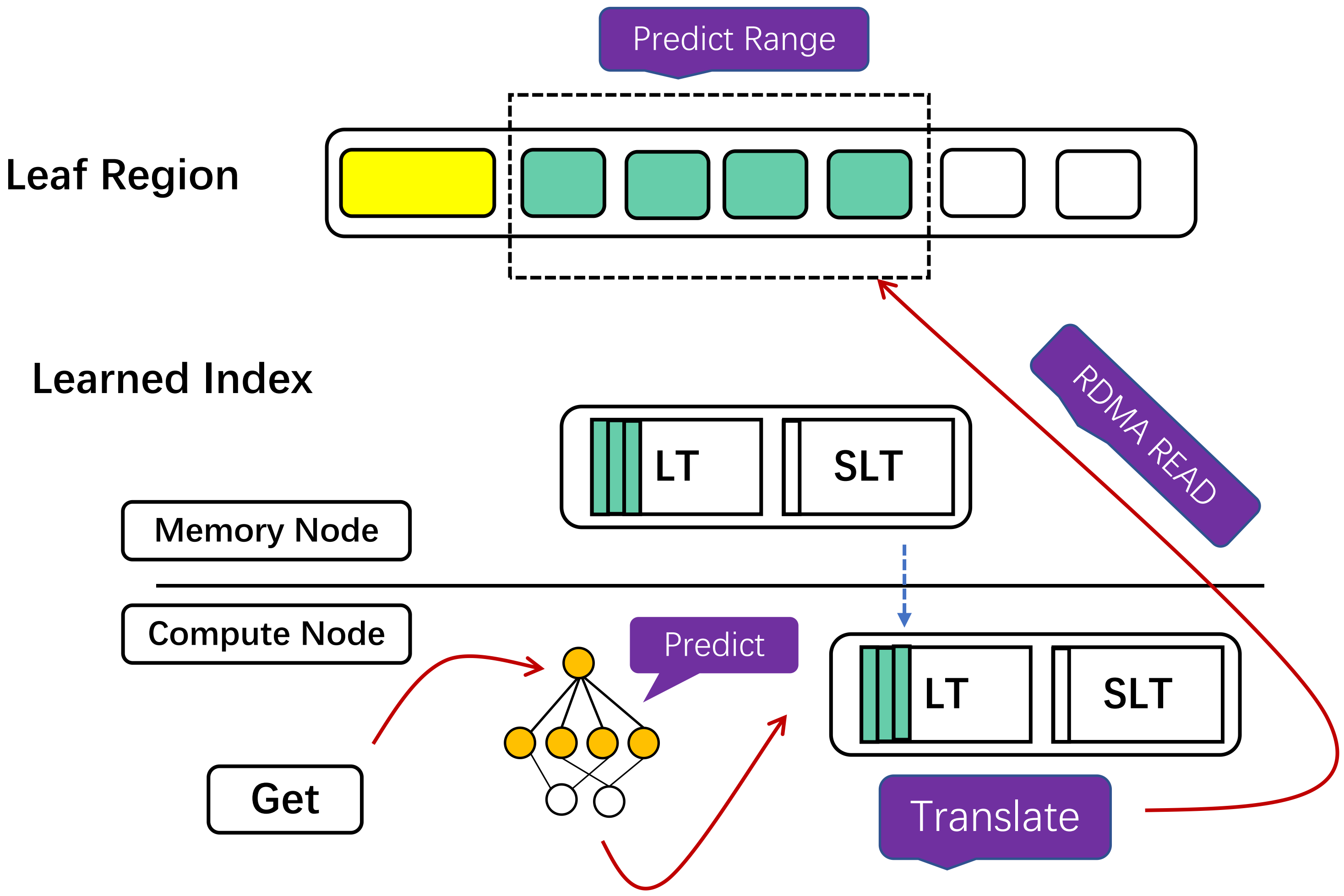
- Upper models is trained on smallest keys
- LT and SLT store the leaf numbers to access leaves
- Each leaf entry points to its corresponding Synonym-leaf entry
- Each entry has a lock to ensure atomically update leaf.



$$phy\_addr = l_{num} * l_{size} + LR_{addr}$$

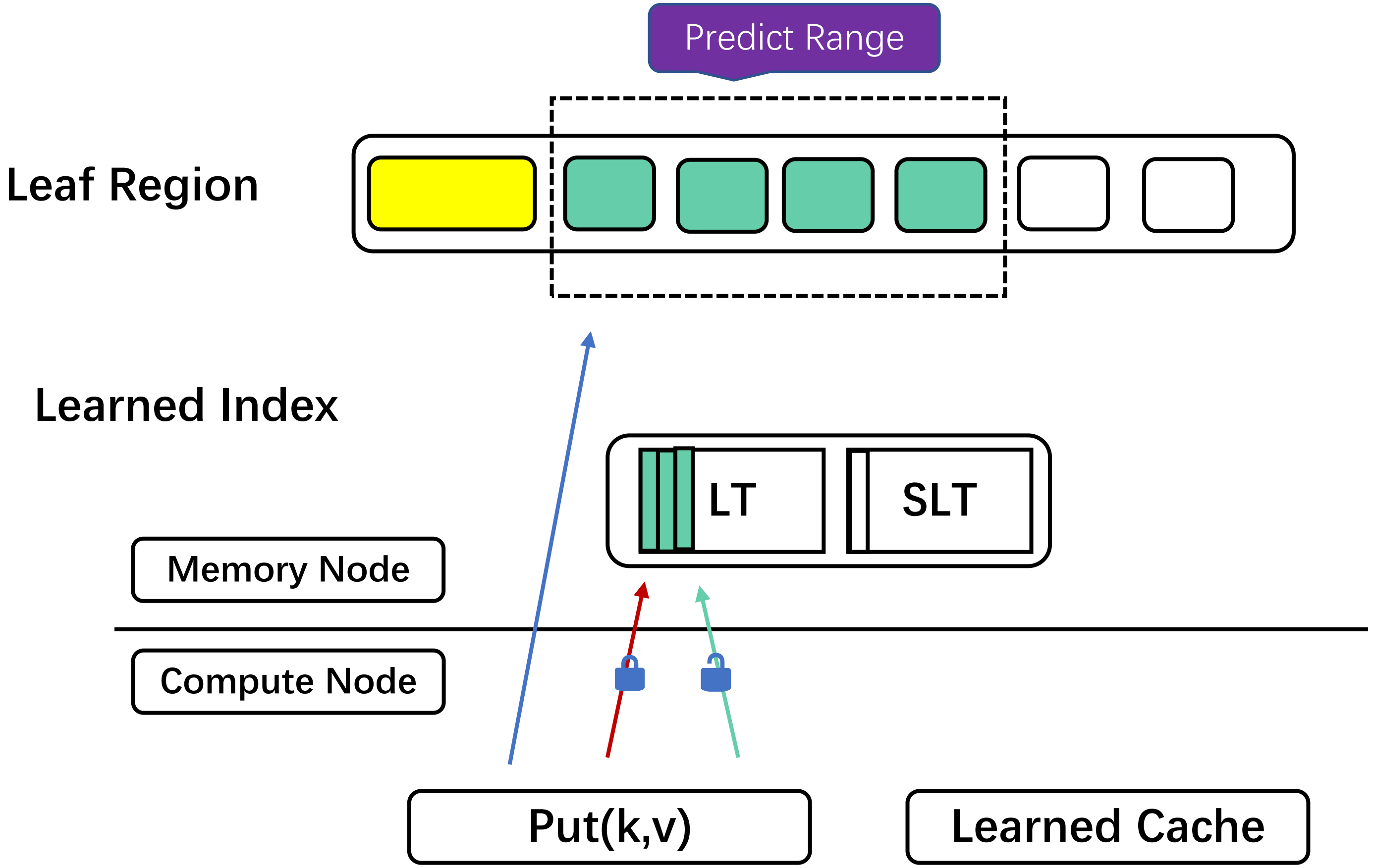
One Sided Indexing(ch1, ch3) – Get(k)

- 1. Predict a range for key based on Equation
- 2. Translate range into leaf address
- 3. One sided RDMA read value



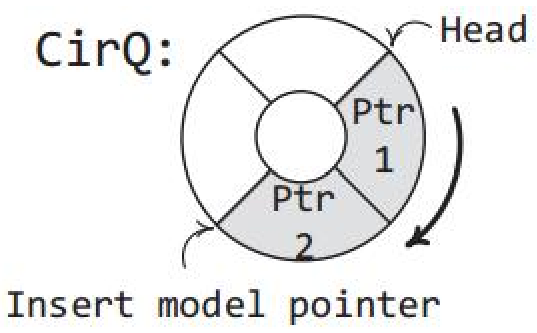
One Sided Indexing(ch1, ch3) – Put(k,v)

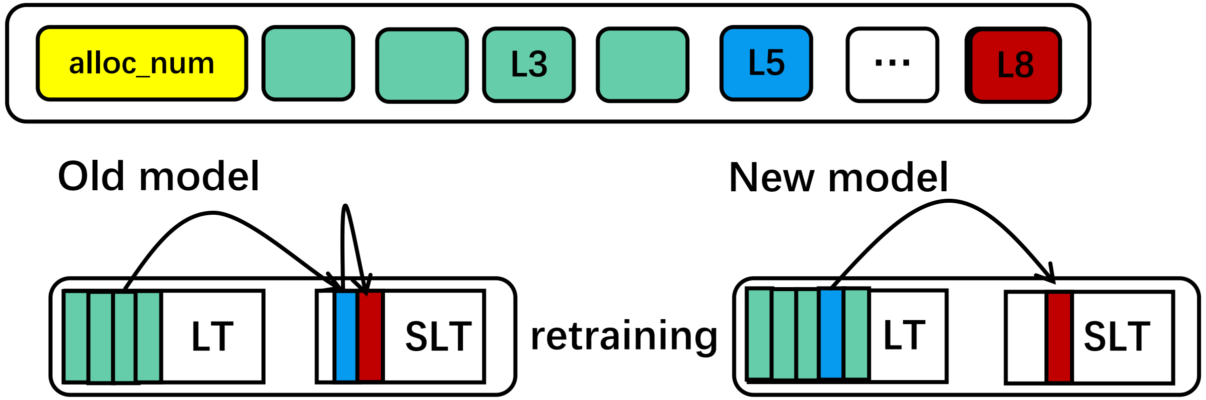
- 1. Fetching like point query without reading synonym leaves
- 2. determines the leaf to be inserted and locks leaf by changing lock bit
- 3. Read leaf and its synonym leaves to ensure data are up to date
- 4. Insert data into the fetched leaves according to data order and unlock



Asynchronous Retraining(ch2)

➢ Key idea: Use circular queue (CirQ) to identify the pending retraining models, and concurrently retrains models on background.



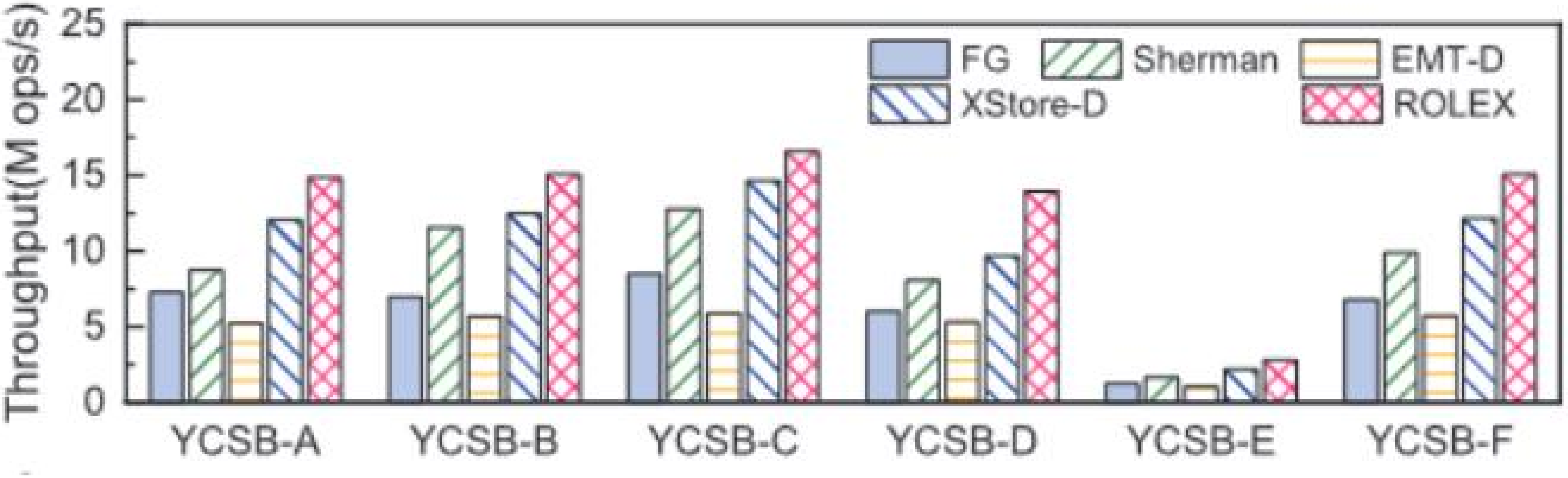


Evaluation

- Experimental Setup
  - 3 compute nodes and 3 memory nodes
  - 100Gb Mellanox ConnectX-5 IB RNIC
- Workloads
  - YCSB, Normal and Lognormal data distributions…
  - 8B keys and values
- Comparisons
  - Xstore-D(Tree + Learned Index) [OSDI'20] One Sided Read + Two Sided Write
  - Sherman(Fine-grained B-link Tree) [SIGMOD'22] One Sided
  - FG(Fine-grained B-link Tree) [SIGMOD'19] One Sided
  - EMT-D(eRPC + MassTree) [NSDI'19] Two Sided

Performance in YCSB

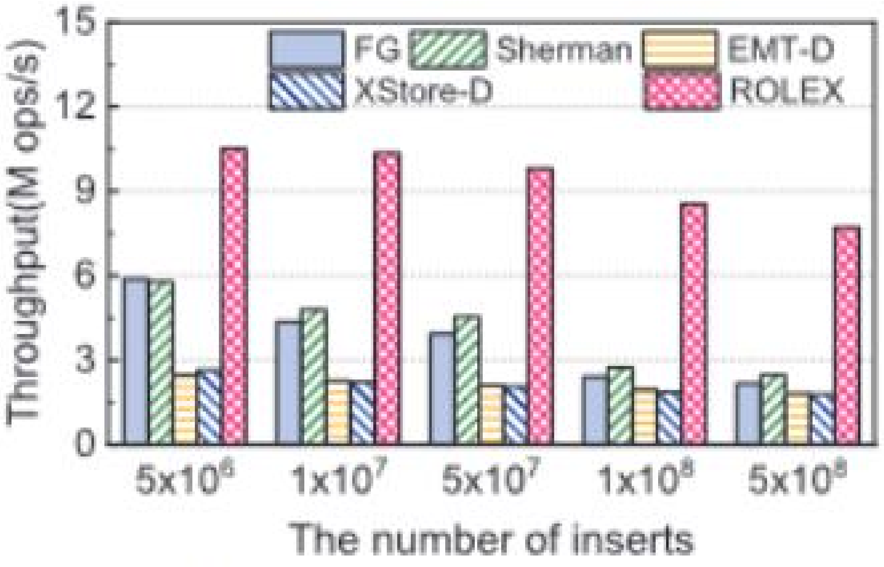
- ➢ Competitive performance on static workloads
- ➢ 1.3x~2.8x improvements on dynamic workloads



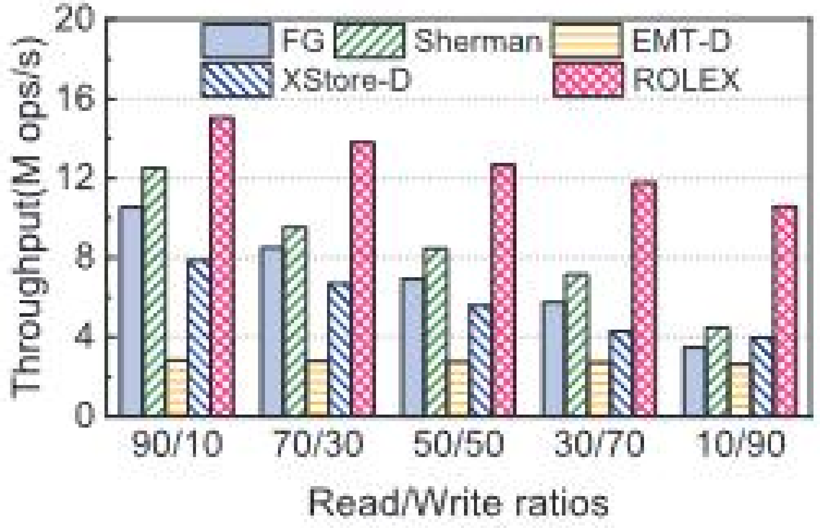
50% update 5% update 0% update 5% insert 95% range query Read modify write

Performance in Various Scenarios

- ROLEX improves insert throughput by 1.8x-4.3x
- ROLEX outperforms the other schemas in write-intensive workloads



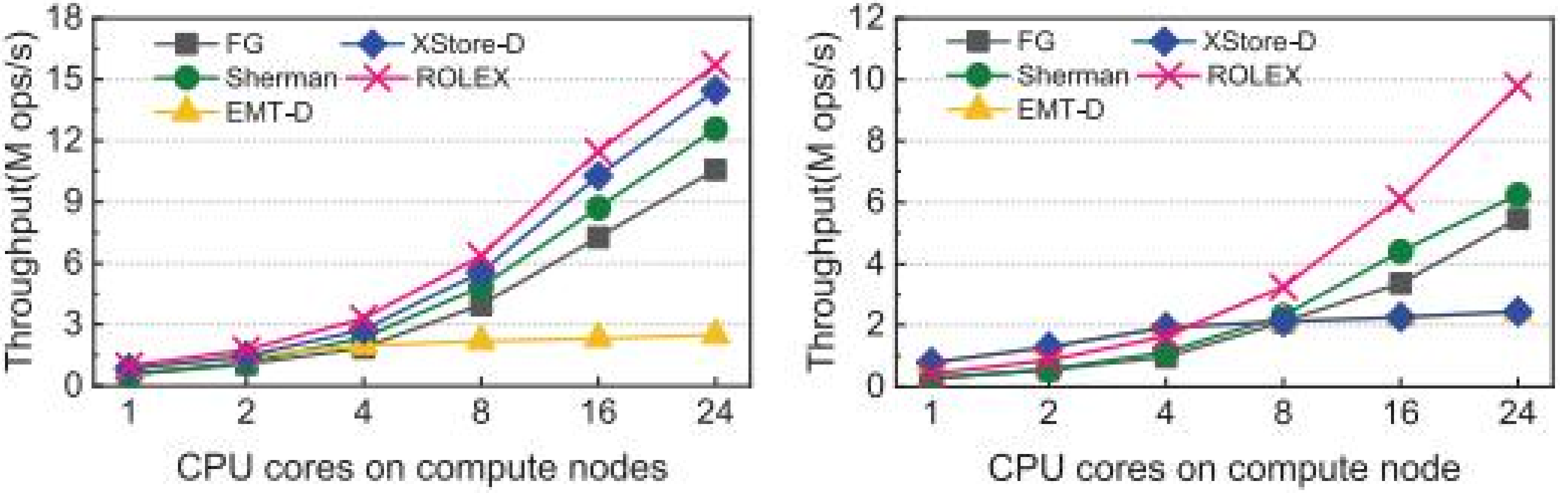
(a) Write-only throughput.



(c) Hybrid read/write throughput.

Scale with CPU cores on compute nodes

➢ ROLEX efficiently scale with computing resources



(a) Read throughput.

(b) Write throughput.

Conclusion

- Build Ordered KV Store on DMS → ROLEX
  - Limited CPU on Memory Nodes
    - Retraining-decoupled Learned indexes
    - One Sided RDMA index operations
  - Overloaded bandwidth for data transferring
    - Asynchronously retraining models on memory nodes
  - Inconsistency issue
    - Execute index operations atomically
    - Check old models after retraining