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

21st USENIX Conference on File and Storage Technologies

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

Paper Notes

By JeongHa Lee

Abstract

Disaggregated memory systems separate monolithic servers into different components, including compute and memory nodes, to enjoy the benefits of high resource utilization, flexible hardware scalability, and efficient data sharing. By exploiting the high-performance RDMA (Remote Direct Memory Access), the compute nodes directly access the remote memory pool without involving remote CPUs. Hence, the ordered key-value (KV) stores (e.g., B-trees and learned indexes) keep all data sorted to provide rang query service via the highperformance network. However, existing ordered KVs fail to work well on the disaggregated memory systems, due to either consuming multiple network roundtrips to search the remote data or heavily relying on the memory nodes equipped with insufficient computing resources to process data modifications. In this paper, we propose a scalable RDMA-oriented KV store with learned indexes, called ROLEX, to coalesce the ordered KV store in the disaggregated systems for efficient data storage and retrieval. ROLEX leverages a retraining-decoupled learned index scheme to dissociate the model retraining from data modification operations via adding a bias and some datamovement constraints to learned models. Based on the operation decoupling, data modifications are directly executed in compute nodes via one-sided RDMA verbs with high scalability. The model retraining is hence removed from the critical path of data modification and asynchronously executed in memory nodes by using dedicated computing resources. Our experimental results on YCSB and real-world workloads demonstrate that ROLEX achieves competitive performance on the static workloads, as well as significantly improving the performance on dynamic workloads by up to 2.2× than state-of-the-art schemes on the disaggregated memory systems. We have released the open-source codes for public use in GitHub.

Problem Statement and Research Objectives

- **Limited computing resources on memory nodes**: the memory nodes in the disaggregated systems fail to meet the requirements of computing-intensive operations, e.g., modifying the large B-tree and frequently retraining models.
  - Simply adding more CPUs to the memory pool ➔ decreases the scalability (the memory and computing resources fail to be independently scaled out.)
- **Overloaded bandwidth for data transferring**: Offloading data modifications to the compute nodes meets the computing requirements, which however rapidly fills up the entire bandwidth due to transferring massive data.
  - The network bandwidth becomes insufficient to enable high performance for various data requests.
- **Inconsistency issue among different nodes**: The inconsistent states occur when different compute nodes fail to atomically complete the data and model modification operations, e.g., multiple compute nodes compete for the same space to insert data and the local cache becomes stale when the models are updated.
  - The main reason is that the atomic granularity of an RDMA operation is 8B, which is much smaller than the size of each index operation. ➔ The compute nodes require multiple network round-trips.

Proposed Method

- To address the aforementioned challenges, this work propose a **scalable RDMA-oriented key-value store using learned indexes**, called ROLEX, for the disaggregated memory systems, which processes data requests on the compute nodes via oneside RDMA operations.
  - The context of **"scalable"** means that ROLEX **efficiently supports dynamic workloads** and **scales out to multiple disaggregated nodes**.
  - It shares a similar idea with **XStore** in the static (i.e., read-only) case, but differs completely from XStore in application scope, dynamic (i.e., data modification) operations, and the index structure on memory nodes.

Proposed Method

- **Retraining-decoupled Learned Indexes** : The challenges come from the high overheads of keeping all data sorted and avoiding data loss from the learned models during insertion.

When some new data are inserted, point a moves backward to a', which is out of the prediction range.

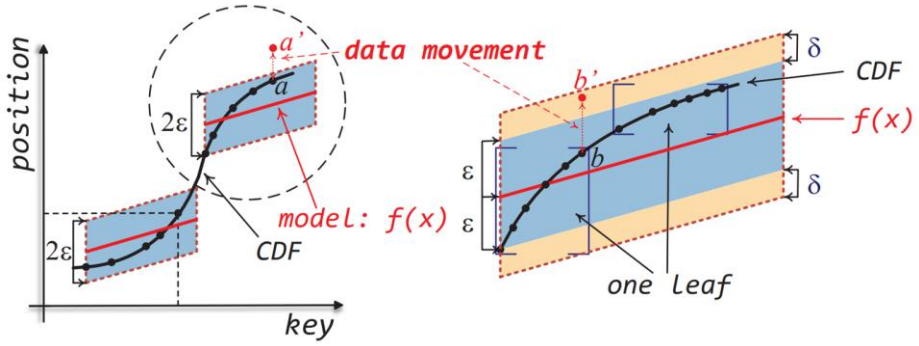


Figure 3: The retraining-decoupled learned indexes.

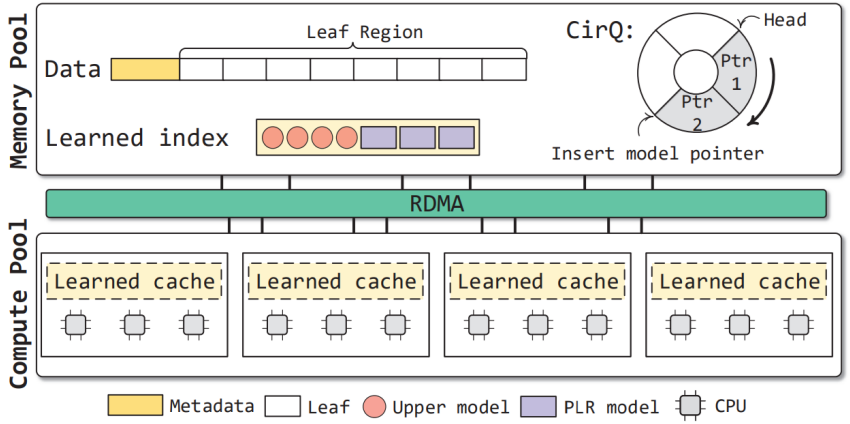


Figure 2: The design overview of ROLEX.

- **Training Algorithm** : An ***improved OptimalPLR*** algorithm
- **Data-movement constraints**
  - **Moving data within fixed-size leaves** : Data is stored into fixed-size arrays (termed as leaves) in the training phase, and each leaf contains at most δ data. All data are only allowed to be moved within their assigned leaves.
  - **Synonym-leaf sharing** : A new leaf (nl) is allocated to accommodate more data when an existing leaf (l) has insufficient slots, where nl shares the same positions (i.e., the labels used for training) with l . The nl is defined as a synonym leaf of l , which is linked via a pointer.

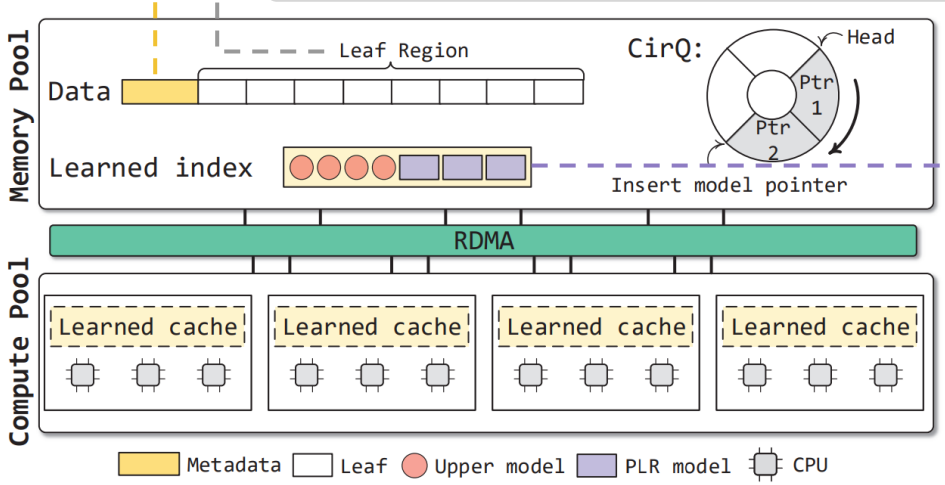
Proposed Method

ROLEX structure

- **Memory pool stores data** : we store all data into fixed-size leaves and train a learned index on these data.

(the first two 8B data) :
# of leaves currently allocated & maximum # that can be allocated

each leaf contains δ pairs of keys and values



train multiple PLR models on the stored leaves

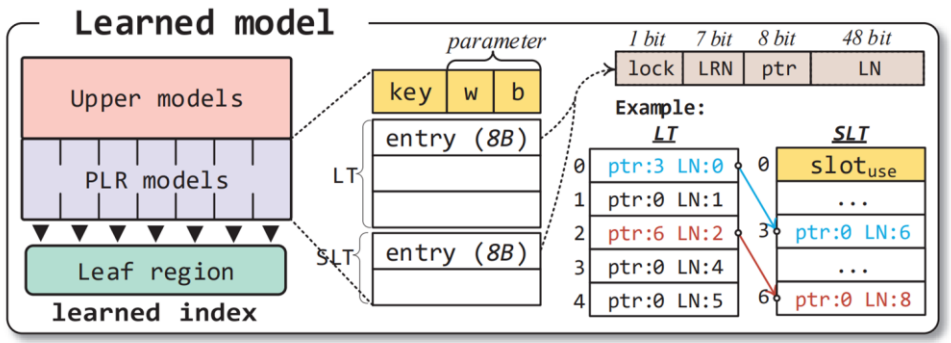


Figure 4: The structure of the learned models.

each model consists of four parts
- the covered smallest key
- the model parameters
- a leaf table (LT)
- a synonym leaf table (SLT)

- **Compute pool caches indexes** : After reading the learned models from the model region, the new compute nodes efficiently access the remote data according to the prediction range of the learned models, where the entry in the prediction range contains the leaf region number and the leaf number, thus indicating the locations of the required data in the memory pool.

Proposed Method

- One-sided Index Operations
  - The structures of LT and SLT ➔ Point query ➔ Range query ➔ Insert ➔ Update ➔ Delete
- **Asynchronous Retraining** : ROLEX maintains a circular queue (CirQ) to identify the pending retraining models, and concurrently retrains models using the shadow redirection scheme without blocking the systems.

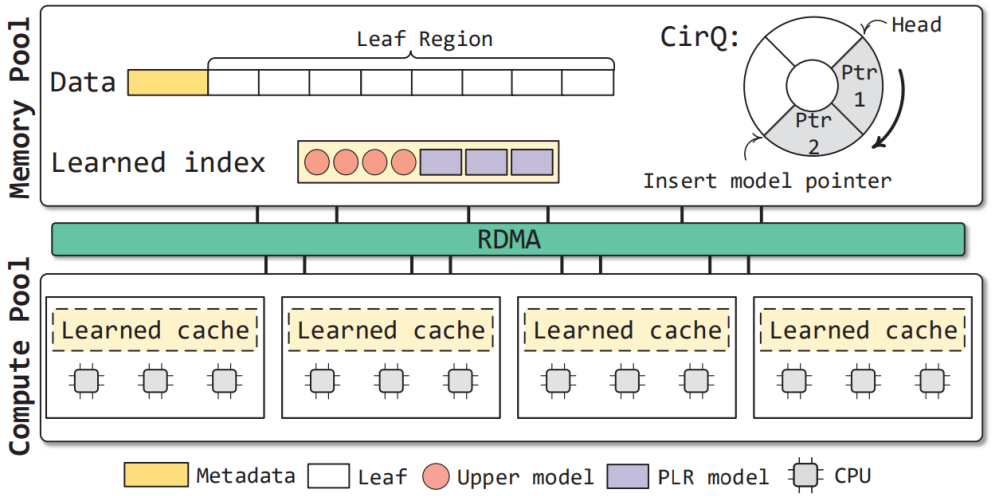


Figure 2: The design overview of ROLEX.

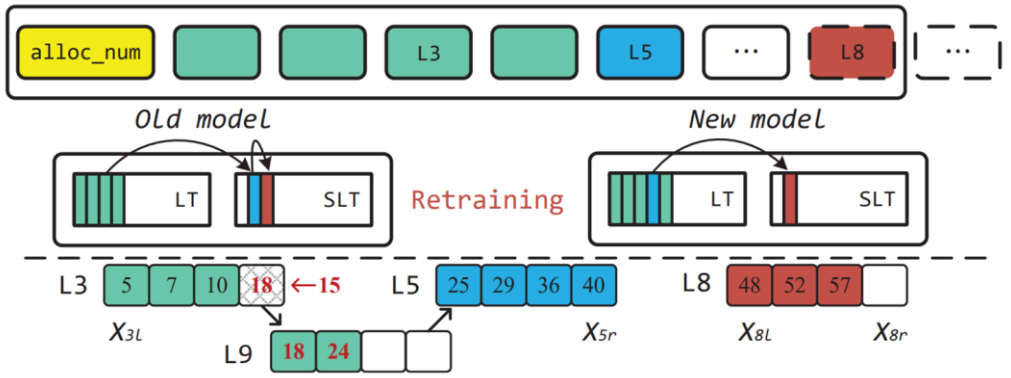


Figure 6: The consistency guarantee of concurrent retraining.

Evaluation and Results

* Static workload (YCSB C)
* Read-write workloads (YCSB A, B, D, F)
* Range-query workload (YCSB E)

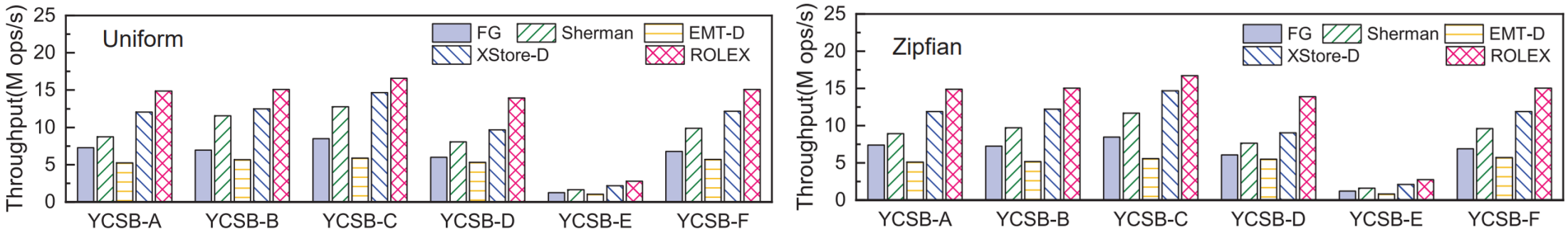
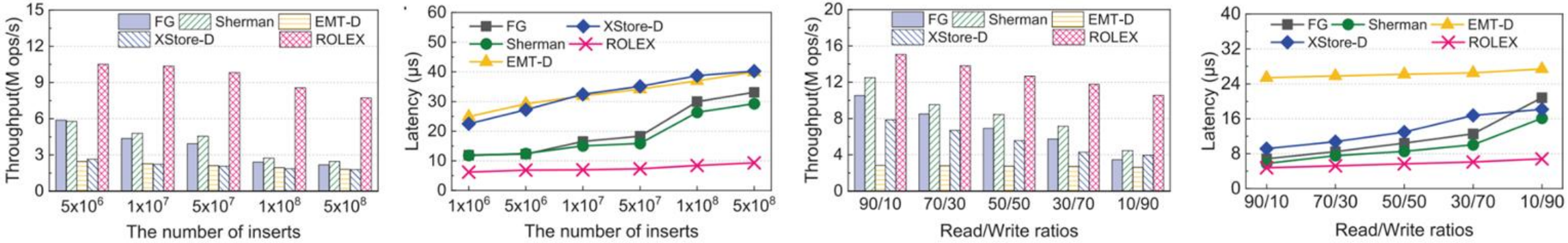


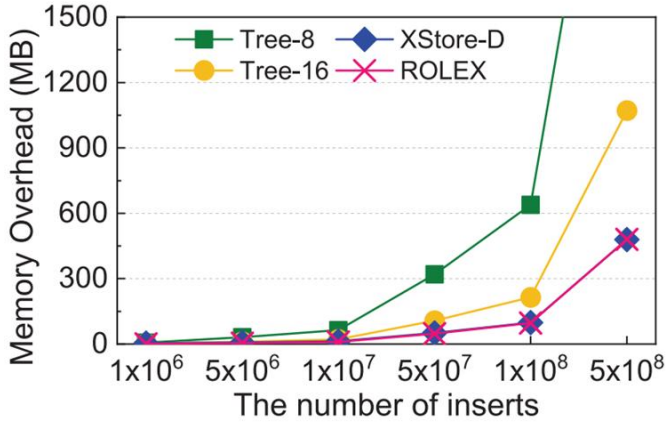
Figure 7: The throughputs on various YCSB workloads.



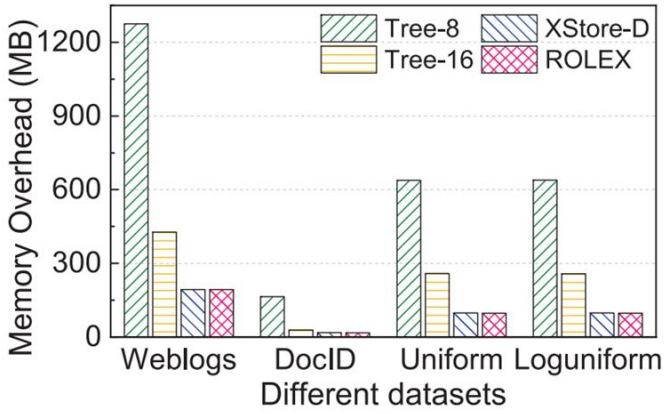
(a) Write-only throughput. (b) Write-only latency. (c) Hybrid read/write throughput. (d) Hybrid read/write latency.

Figure 8: The performance with various read/write scenarios.

Evaluation and Results



(a) Different numbers of data.



(b) Different datasets.

Figure 12: The memory footprints of the metadata. *Tree-# represents that an inner node contains # keys.*

Table 1: The metadata analysis for ROLEX.

Number of Data	$5 * 10^6$	$1 * 10^7$	$5 * 10^7$	$1 * 10^8$	$5 * 10^8$
Number of Models	5,153	10,283	51,111	101,936	526,236
Size of Models (MB)	0.0798	0.157	0.779	1.555	8.03
Size of LT (MB)	4.768	9.537	47.683	95.367	476.837

Notes

fig-left : https://www.epfl.ch/labs/lcav/statistical-analysis-of-sparse-and-piecewise-linear-regression/
fig-right : https://arxiv.org/pdf/1712.01208.pdf

- The disaggregated memory systems become important infrastructures for various applications. Among them, tree-based and learned indexes are two widely used structures for the ordered key-value stores, which provide efficient range query performance via identifying items in a given range.
- Network-attached ordered KV store
  - Tree-based structures: Tree-based structures (e.g., B+-tree) store data in the leaf nodes and construct multilevel inner nodes to search the leaves. However, the tree-based structures become inefficient to leverage one-sided RDMA for accessing remote data [44], since the local machine fails to cache the whole index structure and has to consume multiple RTTs (i.e., the network roundtrip time) for searching the inner nodes.
  - Learned Indexes. Learned indexes show significant advantages over tree-based structures in terms of searching speed and memory consumption, due to the easy-to-use and small-sized learned models. Specifically, the learned indexes view the process of searching data as a regression model, which record the positions of all data by approximating the cumulative distribution function (CDF) of the sorted keys.
- Piecewise Linear Regression