Building High-performance Tree Indexes on Disaggregated Memory

Qing Wang, Youyou Lu, and Jiwu Shu
Tsinghua University

1 Motivation

Traditional datacenters pack CPU and memory into the same hardware units (i.e., monolithic servers), leading to low memory utilization (< 65%) [3]. To attack this problem, academia and industry are exploring a new hardware architecture called *memory disaggregation*, where CPU and memory are physically separated into two different hardware units: compute servers (CSs) and memory servers (MSs). With memory disaggregation, CPU and memory can scale independently and applications can pack resources in a more flexible manner, boosting resource utilization. In this paper, we explore how to build a performant tree index on disaggregated memory.

Assumptions. Similar to prior work, we make two assumptions on the memory disaggregation architecture. First, CSs leverage high-speed RDMA networks to directly access the disaggregated memory in MSs. Second, MSs have near-zero computation power, 1 or 2 wimpy CPU cores, to support lightweight management tasks, such as network connection management and disaggregated memory allocation.

Problems. We revisit existing RDMA-based tree indexes and examine their applicability on disaggregated memory. Several RDMA-based tree indexes rely on remote procedure calls (RPCs) to handle write operations [2,5]; they are ill-suited for disaggregated memory due to near-zero computation power of MSs. For tree indexes that can be deployed on disaggregated memory, they also have some critical limitations: ① Some indexes using RDMA one-sided verbs for all index operation [6] (we call it *one-sided approach*); they can deliver high performance for read operations, but suffer from low throughput and high latency in terms of write operations, especially in high-contention scenarios (< 0.4 Mops with ~20ms tail latency). ② Other indexes bake write operations into SmartNICs or customized hardware [1], which brings high TCO and is not immediately deployable.

Our Goal. Our goal is designing a tree index on disaggregated memory that can *deliver high performance for both read and write operations with commodity RDMA NICs*.

Analysis. The following three reasons contribute to the inefficiency of one-sided approaches.

1. Excessive round trips. Due to limited semantics of one-sided RDMA verbs, modifying an index node (e.g., tree node in B+Tree) always requires multiple round trips (i.e., lock, read, write, and unlock), inducing high latency and further making conflicting requests more likely to be blocked.

2. Slow synchronization primitives. The RDMA locks used for resolving write-write conflicts are slow and experience performance collapse under high-contention scenarios.

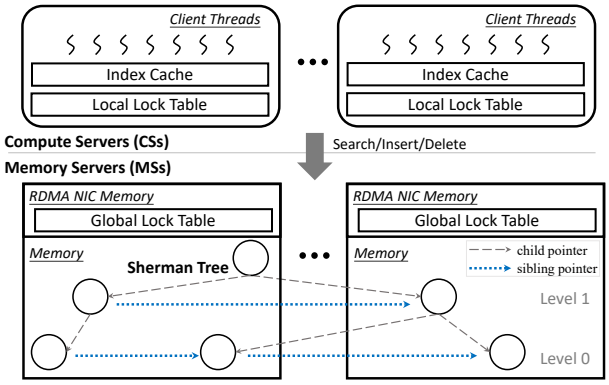

Figure 1: SHERMAN's architecture and interactions.

This is because that, to guarantee correct semantic between conflicting RDMA atomic commands, RDMA NICs adopt an internal locking scheme to serialize conflicting atomic commands. Unfortunately, an atomic command has a long critical path: two PCIe transactions at receiver side. Moreover, at the software level, such locks always trigger unnecessary retries, which consumes RDMA IOPS, and do not provide fairness, which leads to high tail latency.

3. Write amplification. In order to read tree nodes in a lock-free way and detect incomplete data due to ongoing writes, two main consistency check mechanisms are proposed. In the first mechanism, each node includes a checksum covering the whole node area (except the checksum itself) [6]; the checksum is re-calculated when modifying the associated node, and is verified when reading the node. The other mechanism, namely version-based consistency check, stores a version number at the start and end of each node [2]; when modifying a node via `RDMA_WRITE`, the corresponding two version are incremented; a node's content obtained via `RDMA_READ` is consistent only when the two versions are the same. Since the granularity of the above two mechanisms is tree node, any modification to part of the node area requires to write back the whole node (include the metadata, e.g., checksum and version), leading to severe write amplification.

2 SHERMAN Design

We design SHERMAN[1], a high-performance B$^+$Tree on disaggregated memory. Figure 1 shows its overall architecture. CSs run client threads that manipulate SHERMAN (residing in MSs) via one-sided RDMA commands.

B$^+$Tree Structure. In SHERMAN, we record a sibling pointer for every leaf node and internal node as in the B-link tree.

[1]Move details of SHERMAN can be found in the original paper [4]. SHERMAN is open-source at: https://github.com/thustorage/Sherman.

Client threads can always reach a targeted node by following these sibling pointers in the presence of node split/merging, thus supporting concurrent operations efficiently. To reduce remote accesses in the tree traversal, each CS maintains an *index cache*, which makes internal nodes' copies.

Concurrency Control. SHERMAN uses *node-grained* exclusive locks to resolve write-write conflicts: before modifying a tree node, the client thread must acquire the associated exclusive lock. SHERMAN supports lock-free search, which leverages `RDMA_READ` to fetch data residing in MSs without holding any lock. Moreover, SHERMAN uses versions to detect inconsistent data caused by concurrent writes.

2.1 Hierarchical On-Chip Lock

To accelerate lock operations, we design a *hierarchical on-chip lock (HOCL)* for SHERMAN. HOCL is structured into two parts: global lock tables on MSs, and local lock tables on CSs. Global lock tables and local lock tables coordinate conflicting lock requests between CSs and within a CS, respectively. Global lock tables are stored in the on-chip SRAM of RDMA NICs, thus eliminating PCIe transactions of MSs and further delivering extremely high throughput for RDMA atomic commands (~110 Mops). Within a CS, before trying to acquire a global lock on MSs, a thread must acquire the associated local lock, so as to avoid a large amount of unnecessary remote retries. Moreover, by adopting wait queues, local lock tables improve fairness between conflicting lock requests. Based on local lock tables, a thread can hand its acquired lock over to another thread directly, reducing at least one round trip for acquiring global locks.

2.2 Command Combination

To reduce round trips, we introduce a command combination technique. Based on the observation that RDMA NICs already provide in-order delivery property, this technique allows client threads to issue dependent RDMA commands (e.g., write-back and lock release) simultaneously, letting NICs at MSs reflect them into disaggregated memory in order.

2.3 Two-Level Version

To mitigate write amplification, SHERMAN tailors the leaf node layout of B$^+$Tree. First, entries in leaf node are *unsorted*, so as to eschew shift operations upon insertion/deletion. Second, to support lock-free search while avoiding write amplification, we introduce a *two-level version mechanism*. In addition to using a pair of *node-level versions* to detect the inconsistency of the whole leaf node, we embed a pair of *entry-level versions* into each entry, which ensures entry-level integrity. For insertion/deletion operations without split/merging events, only entry-sized data is written back, thus saving network bandwidth and making the most of the extremely high IOPS of small RDMA messages.

3 Evaluation

Our cluster consists of 8 servers, each of which is equipped with 128GB DRAM, two 2.2GHz Intel Xeon E5-2650 v4 CPUs (24 cores in total), and one 100Gbps Mellanox ConnectX-5 NIC. We emulate each server as one MS and one CS. Each MS owns 64GB DRAM and 2 CPU cores, and each CS owns 1GB DRAM and 22 CPU cores.

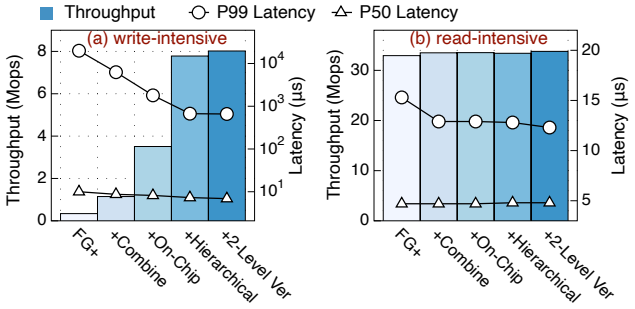

Figure 2: Overall performance.

We compare SHERMAN with FG [6], which is the only distributed B$^+$Tree that supports disaggregated memory. For fair comparison, we add index cache for FG (called FG+). Each CS owns 500MB index cache, and launches 22 client threads (176 in total in our cluster). For each experiment, we bulkload the tree with 1 billion entries (8-byte key, 8-byte value) 80% full, then perform specified workloads. The size of a tree node (i.e., internal node and leaf node) is 1KB.

To analyze SHERMAN's performance, we break down the performance gap between FG+ and SHERMAN through applying each technique one by one. Figure 2 shows the results under skewed workloads (0.99 Zipfian parameter). In write-intensive workloads (50% insert and 50% lookup), SHERMAN achieves 23.6× higher throughput with 1.4×/ 30.2× lower 50p/99p latency. All techniques contribute to the high write efficiency of SHERMAN. In read-intensive workloads (5% insert and 95% lookup), SHERMAN does not present considerable performance improvement, as expected, since all techniques we propose aim to boost write performance.

References

[1] M. K. Aguilera, K. Keeton, S. Novakovic, and S. Singhal. Designing far memory data structures: Think outside the box. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, page 120–126, New York, NY, USA, 2019. ACM.

[2] C. Mitchell, K. Montgomery, L. Nelson, S. Sen, and J. Li. Balancing cpu and network in the cell distributed b-tree store. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '16, page 451–464, USA, 2016. USENIX Association.

[3] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. Legoos: A disseminated, distributed os for hardware resource disaggregation. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 69–87, USA, 2018. USENIX Association.

[4] Q. Wang, Y. Lu, and J. Shu. Sherman: A write-optimized distributed b+tree index on disaggregated memory. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1033–1048, New York, NY, USA, 2022. ACM.

[5] X. Wei, R. Chen, and H. Chen. Fast rdma-based ordered key-value store using remote learned cache. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 117–135. USENIX Association, Nov. 2020.

[6] T. Ziegler, S. Tumkur Vani, C. Binnig, R. Fonseca, and T. Kraska. Designing distributed tree-based index structures for fast rdma-capable networks. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 741–758. ACM, 2019.