One-Sided RDMA-Conscious Extendible Hashing for Disaggregated Memory

Pengfei Zuo, Jiazhao Sun, Liu Yang, Shuangwu Zhang, Yu Hua*

Huawei Cloud
*Huazhong University of Science and Technology

USENIX Annual Technical Conference (ATC), 2021
Disaggregated Memory

Resource Utilization
Failure Isolation
Elasticity
Hashing Indexes in Local Memory

Monolithic Server

CPU

Search/Insertion/Deletion/Update

Local LOAD&STORE

Hash Table
Hashing Indexes in Disaggregated Memory

CPU

Search/Insertion/Deletion/Update

Local LOAD&STORE

Hash Table

Compute Pool

Memory Pool
Hashing Indexes in Disaggregated Memory

- **Compute Pool**
  - CPU
  - Search/Insertion/Deletion/Update
- **Memory Pool**
  - Hash Table

RDMA READ/WRITE/ATOMIC
Challenge 1: Hash Collision

Challenge 1: Many remote reads & writes for handling hash collisions
Challenge 1: Many remote reads & writes for handling hash collisions
Challenge 1: Hash Collision

Challenge 1: Many remote reads & writes for handling hash collisions
Challenge 1: Hash Collision

Challenge 1: Many remote reads&writes for handling hash collisions
Challenge 2: Concurrency Control

Challenge 2: Locking has high overhead
Challenge 3: Remote Resizing

Challenge 3: Moving items from the old table to the new table
### Challenge 3: Extendible Resizing

**Challenge of using extendible resizing in disaggregated memory**

- One extra RDMA for accessing the directory

(a) A hash table
(glocal depth = 2)

(b) Subtable resizing
(glocal depth = 2)

(c) Directory resizing
(glocal depth = 3)
Challenge Summary

1. Many remote reads&writes for handling hash collisions
   ➢ Cuckoo hashing, hopscotch hashing, chained hashing

2. Concurrency control for remote access
   ➢ One RDMA RTT for locking or unlocking

3. Tricky remote resizing of hash tables
   ➢ One extra RDMA RTT for accessing the directory
RDMA-Conscious Extendible (RACE) Hashing

1. Many remote reads&writes for handling hash collisions
   - Cuckoo hashing, hopscotch hashing, chained hashing
   - Solution: One-sided RDMA-conscious table structure

2. Concurrency control for remote access
   - One RDMA RTT for locking or unlocking
   - Solution: Lock-free remote concurrency control

3. Tricky remote resizing of hash tables
   - One extra RDMA RTT for accessing the directory
   - Solution: Extendible remote resizing with stale-read caching
Architectural Overview

Compute Pool

Client 0
Directory Cache

Client 1
Directory Cache

Memory Pool

RACE Hash Table
Directory:
Subtable:
Local Depth:
One-sided RDMA-Conscious (RAC) Subtable

- Design decisions
  - Associativity
  - Two choices
  - Overflow colocation

- Strengths
  - RDMA-search friendly
  - RDMA-IDU friendly
  - High memory efficiency

RAC Subtable:

- Sharing & colocation
- RDMA read with doorbell batching

A Bucket Group:

- Main bucket 0
- Overflow bucket
- Main bucket 1
- Combined bucket 0
- Combined bucket 1
Lock-free Remote Concurrency Control

• Bucket Structure: supporting the RDMA CAS
Lock-free Remote Concurrency Control

(a) Search

Read combined buckets

Read KV

Read combined buckets

Set the slot to null

(b) Insertion

Read combined buckets

Write the KV

Re-read combined buckets

Write KV

(c) Deletion

Read combined buckets

Read KV

(d) Update

Read the old KV

Update the KV pointer

Write the new KV

RAC Subtable:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ...... | N-1 | N |

www.huaweicloud.com
## Extendible Remote Resizing

- **Client Directory Cache with Stale Reads**

### Directory Cache

<table>
<thead>
<tr>
<th>Key</th>
<th>Bucket</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>XX00</td>
<td>2 00</td>
<td>Yes</td>
</tr>
<tr>
<td>XX01</td>
<td>2 01</td>
<td>Yes</td>
</tr>
<tr>
<td>XX11</td>
<td>2 01</td>
<td>No</td>
</tr>
</tbody>
</table>

### Different cases of stale reads:

- **Compute Pool**
- **Memory Pool**
- **Network**
- **RACE Hash Table**

**Header:**
- **Local Depth**
- **Suffix**
Extendible Remote Resizing

• Resize a directory

(a) Before the directory resizing

(b) After the directory resizing
Experimental Setup

• Testbed
  – 4 client machines + 1 memory machine

• Workloads
  – YCSB, 16B key + 32B value

• Comparisons
  – Pilaf cuckoo hashing [ATC’15]
  – FaRM hopscotch hashing [NSDI’14]
  – DrTM cluster hashing [SOSP’15]
• RACE hashing improves the insertion throughput by 1.4~16.9×
Search

![Graph showing Search Latency vs Load Factor for Cuckoo, Hopscotch, Cluster, and RACE.]

- **Search Latency (us)**
  - **Load Factor**
  - **Cuckoo**
  - **Hopscotch**
  - **Cluster**
  - **RACE**

![Graph showing Throughput (M reqs/s) vs The Number of Client Processes for Cuckoo, Hopscotch, Cluster, and RACE.]

- **Throughput (M reqs/s)**
- **The Number of Client Processes**
  - **Cuckoo**
  - **Hopscotch**
  - **Cluster**
  - **RACE**
Deletion hashing improves the deletion throughput by 1.7~2.1 ×
• RACE hashing improves the deletion throughput by 1.5~1.9×
• RACE hashing speeds up the YCSB hybrid workloads by 1.4~1.37×
The SRCD cache reduces the request latency by 23%~32%

- Insertion
- Search
- Deletion
- Update

Latency (us)

W/o SRCD Cache  W/ SRCD Cache
Conclusion

• Traditional distributed in-memory hashing indexes become inefficient in disaggregated memory
  – Many remote access, concurrency access, resizing

• We propose RACE hashing, the first hashing index designed for disaggregated memory
  – One-sided RDMA-conscious table structure
  – Lock-free remote concurrency control
  – Extendible remote resizing

• RACE Hashing outperforms state-of-the-art distributed in-memory hashing indexes by 1.4-13.7× in YCSB hybrid workloads
Thank you! Q&A