Outback: Fast and Communication-efficient Index for Key-Value Store on Disaggregated Memory

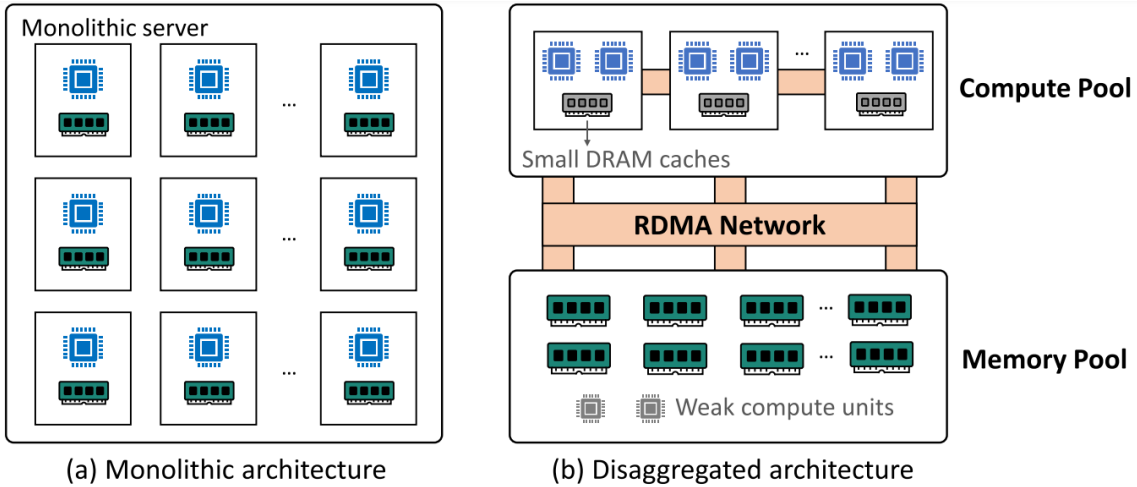
Yi Liu, Minghao Xie
Prof. Chen Qian, Prof. Heiner Litz, Prof. Yuanchao Xu
Center for Research in Systems and Storage
University of California, Santa Cruz

UC SANTA CRUZ BaskinEngineering

Center for Research in Systems and Storage

NSF

Disaggregated KV Store with Far Memory

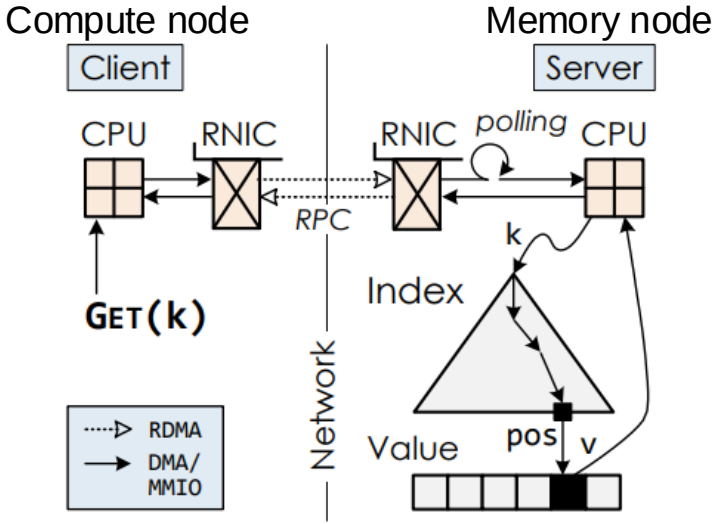


(a) Monolithic architecture (b) Disaggregated architecture

- Disaggregated architecture decouples the compute and memory resources into independent and distributed resource pools connected by RDMA/CXL, etc,.

[1] FORD: Fast One-sided RDMA-based Distributed Transactions for Disaggregated Persistent Memory

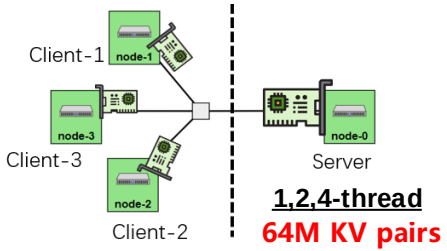
RDMA RPC based KV store



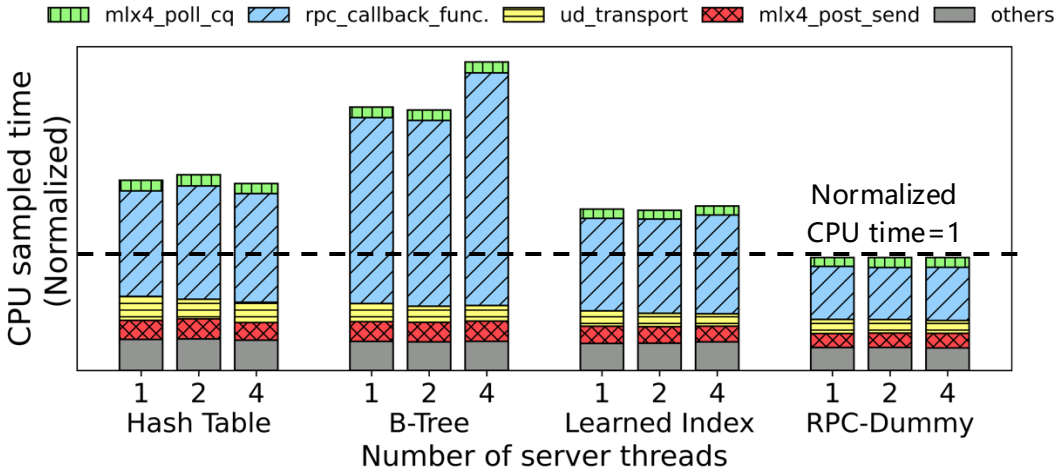
Cell (ATC'16, B tree)

[1] Fast {RDMA-based} Ordered {Key-Value} Store using Remote Learned Cache.

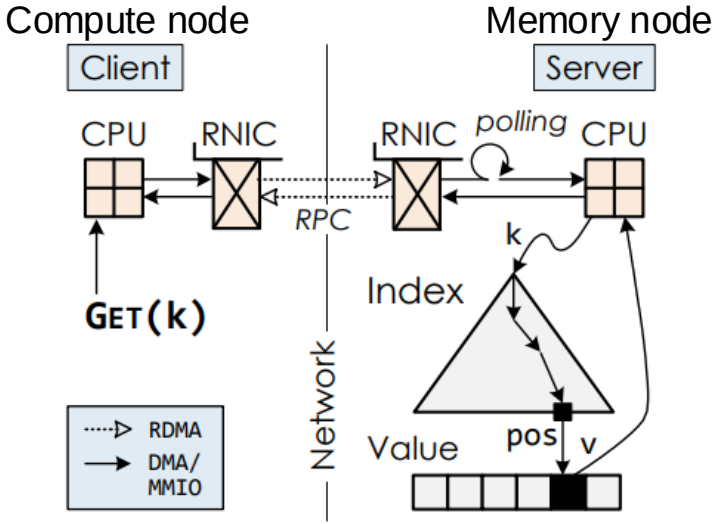
Microbenchmark



- Normalized CPU time for data query with:
  - Hash table/B-tree/Learned index/Dummy



RDMA RPC based KV store



Cell (ATC'16, B tree)

FaSST (OSDI'16, Hash table)

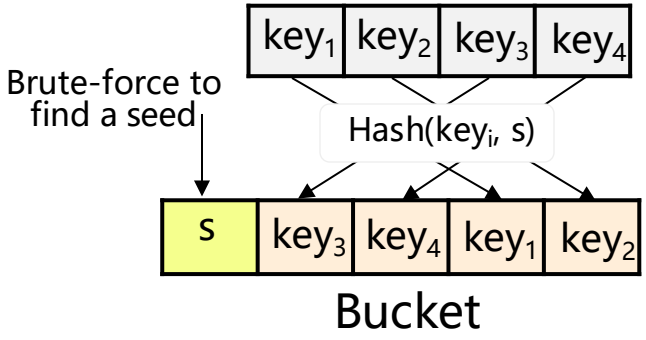
☹

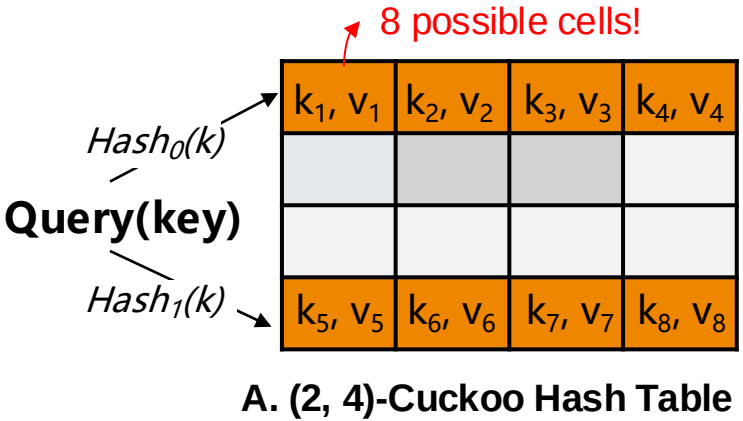
Computation on server side->Latency

[1] Fast {RDMA-based} Ordered {Key-Value} Store using Remote Learned Cache.

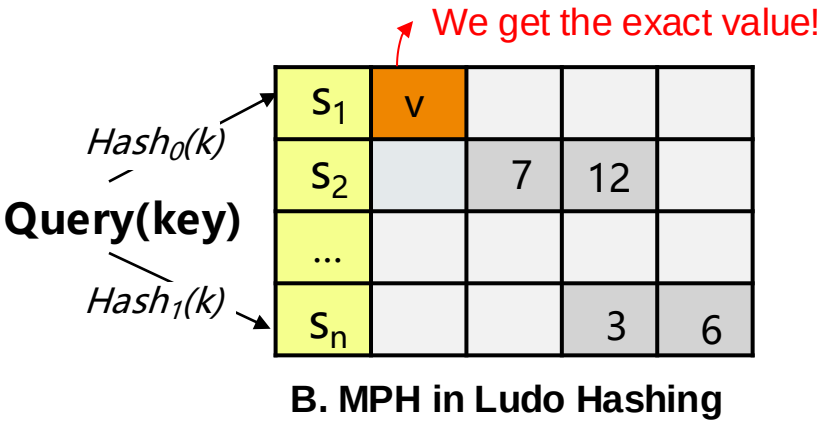
Range Lookup v.s. Point Lookup

- Cuckoo Hashing / Learned Index only returns a range of possible locations
- Minimal perfect hashing (MPH) returns the exact location



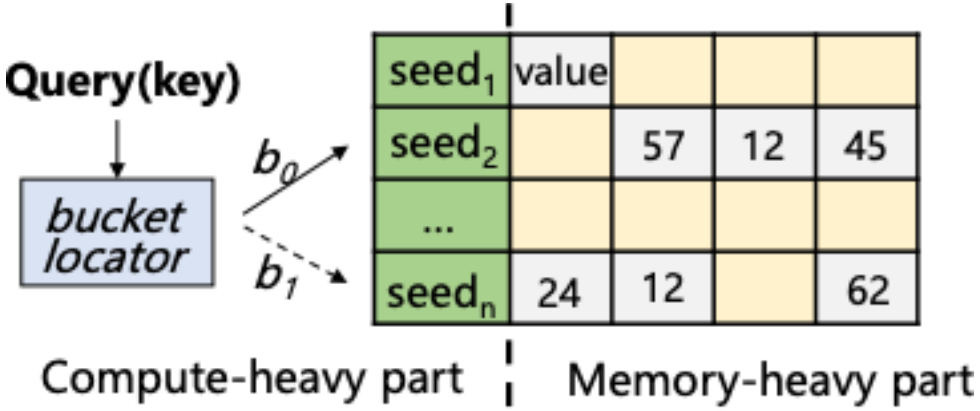


A. (2, 4)-Cuckoo Hash Table



B. MPH in Ludo Hashing

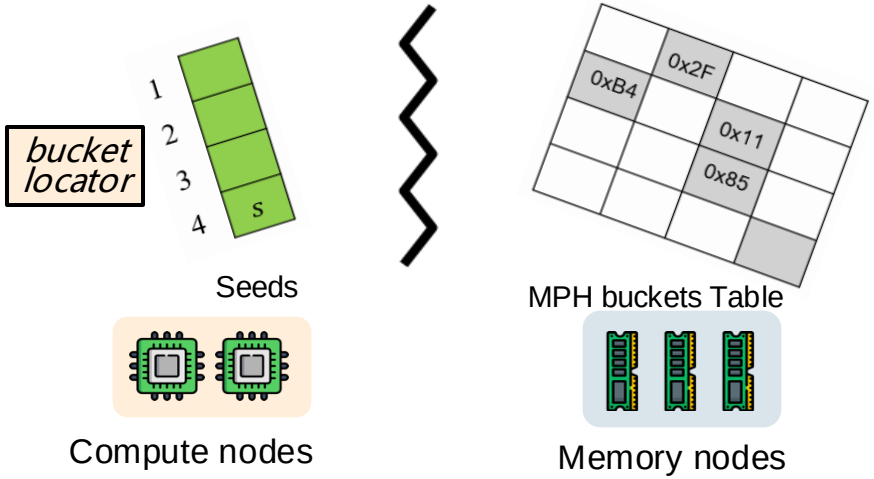
Ludo Hashing: a disaggregation fit!



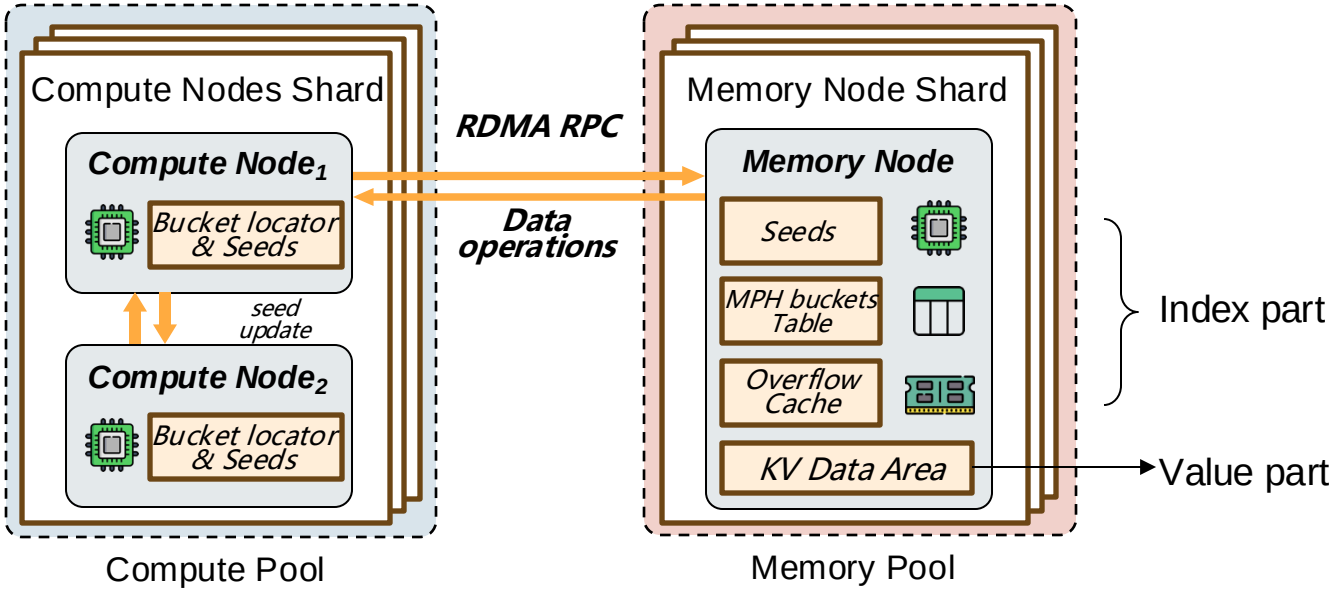
Input: The Ludo lookup structure and the key k
Output: The lookup result v of k
begin
`// Step I: compute bucket location`
1 $b \leftarrow$ Othello lookup result of k
2 $B \leftarrow h_b(k)$-th bucket of the table
`// Step II: compute slot location`
3 $s \leftarrow$ seed stored in B
// $\mathcal{H}_s(k) \in \{0, 1, 2, 3\}$
4 $v \leftarrow B.slot[\mathcal{H}_s(k)]$
end

Ludo Hashing Lookup Algorithm

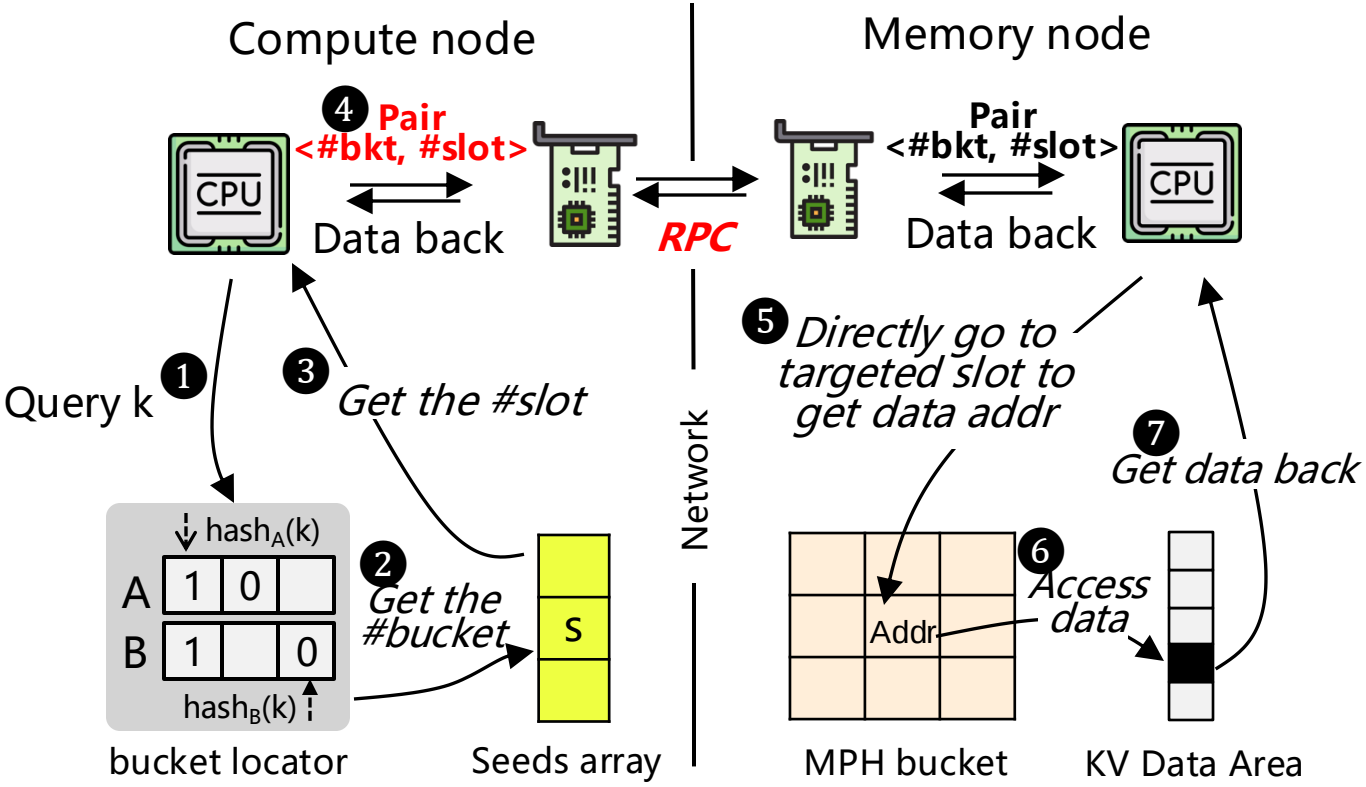
DESIGN


bucket locator
1
2
3
4
s
Seeds
Compute nodes
0xB4
0x2F
0x11
0x85
MPH buckets Table
Memory nodes

DESIGN

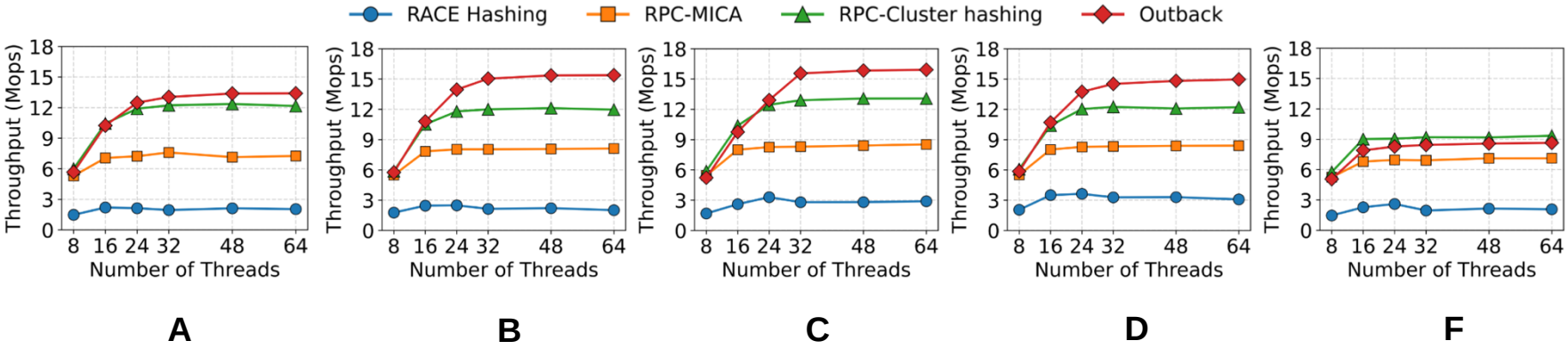

Compute Nodes Shard
Compute Node1
Bucket locator & Seeds
seed update
Compute Node2
Bucket locator & Seeds
Compute Pool
RDMA RPC
Data operations
Memory Node Shard
Memory Node
Seeds
MPH buckets Table
Overflow Cache
KV Data Area
Index part
Value part
Memory Pool

Lookup Operation



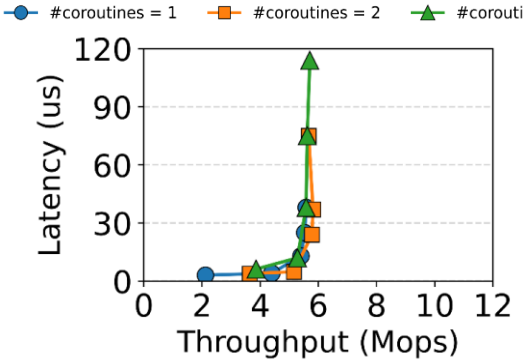
Performance Evaluation

- YCSB A/B/C/D/F workloads, E is not included because this work is not optimized for data scan.
- Client threads number: 8->64, and keeps #server threads=4 and running on only 4 cores.

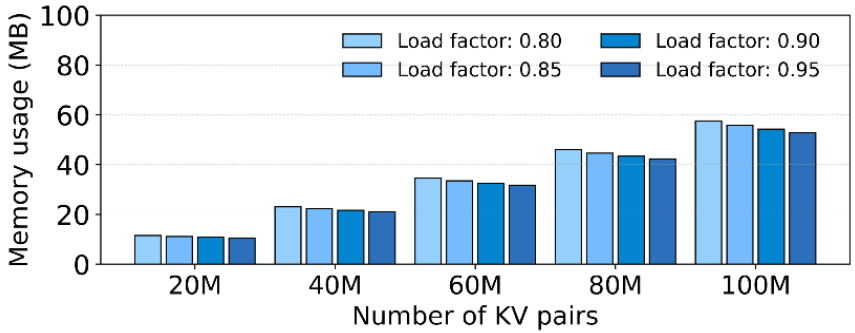


Overhead Evaluation

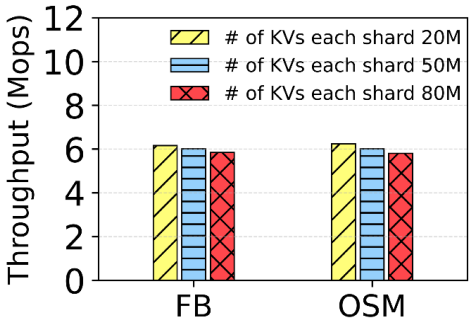
- Increased #Coroutines/thread does not increase the performance unlike increased #threads
- Memory cost: one-sided RDMA solutions cost 5x of MBs or more on each compute node for index caching
- Capacity impact is trivial on dataset of Facebook and Open Street Map



Coroutine Impact



Memory Cost



Capacity Impact

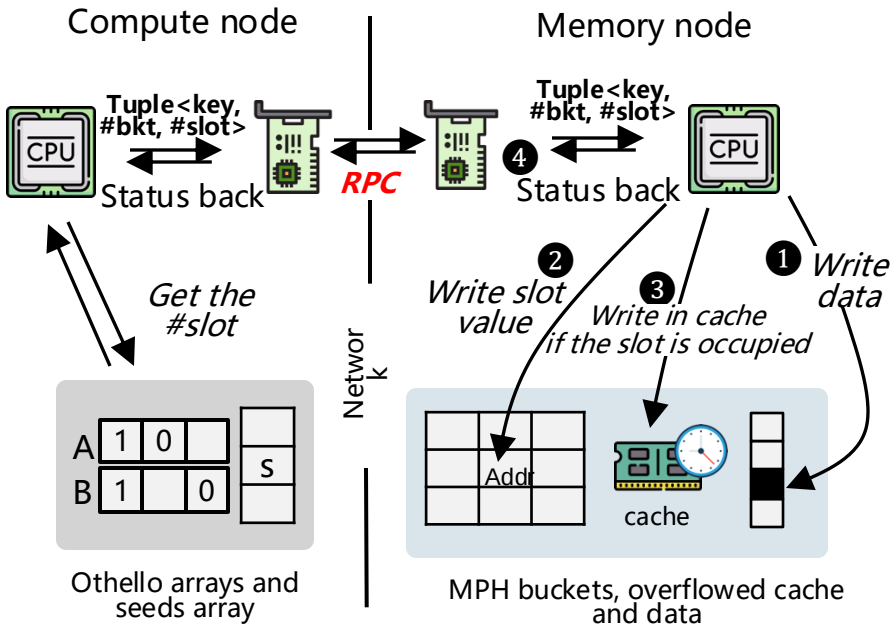
Project Status

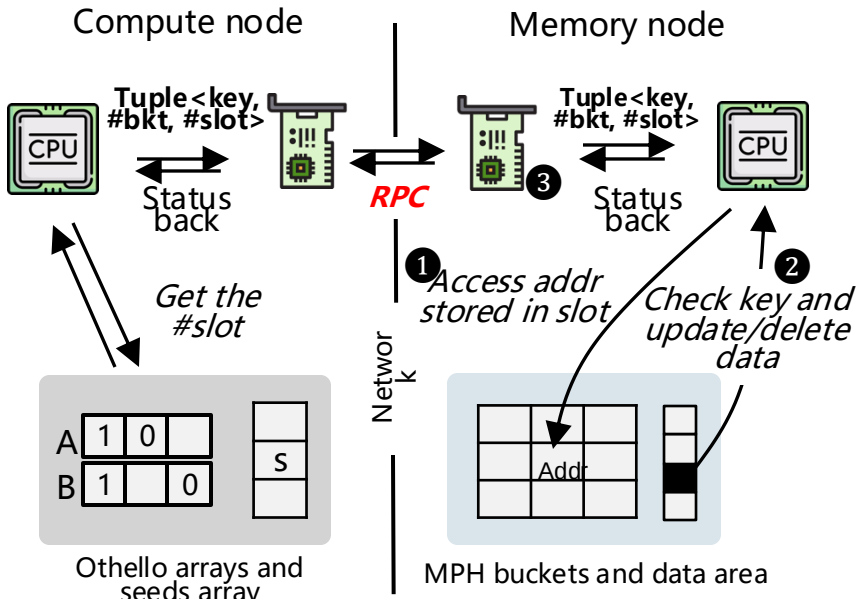
- Update since last IAB Meeting
  - **Enhanced Experiments**: Re-ran experiments on high-performance hardware, showing Outback's scalability and throughput across varied workloads.
  - **Broadened Applicability**: Demonstrated Outback's use with other data structures, supporting range queries and offloading compute tasks for devices
  - **Improved Concurrency**: Implemented bucket-level locking, clarified extendible hashing resizing, ensuring efficient load balancing and shard distribution.
  - Paper accepted by VLDB '25, open-sourced at https://github.com/yliu634/outback

Thanks for Listening

liuyi@ucsc.edu

Backup


Compute node
Memory node
CPU
Tuple<key, #bkt, #slot>
Status back
RPC
Network
Get the #slot
Write slot value
Write in cache if the slot is occupied
Write data
Addr
cache
A
B
S
Othello arrays and seeds array
MPH buckets, overflowed cache and data


Compute node
Memory node
CPU
Tuple<key, #bkt, #slot>
Status back
RPC
Network
Get the #slot
Access addr stored in slot
Check key and update/delete data
Addr
A
B
S
Othello arrays and seeds array
MPH buckets and data area

MPH Resizing

- How to enlarge the MPH table and without stop serving data requests on the server side?