Parallel Multi-split Extendible Hashing for Persistent Memory

Jing Hu, Jianxi Chen, Yifeng Zhu, Qing Yang, Zhouxuan Peng, Ya Yu
Outline

- **Background**
  - Persistent Memory
  - Static Hashing
  - Dynamic Hashing

- **Design**
  - Overview
  - Lock-free parallel
  - Multi-split
  - Instant recovery

- **Evaluation**

- **Conclusion**
Background: Persistent Memory

- **Pros:**
  - non-volatility
  - high-density
  - byte-addressable
  - near-zero standby power

- **Cons:**
  - limited endurance
  - limited write bandwidth
  - 1/6 of DRAM
  - 1/3 read bandwidth of DRAM

Intel Optane DC Persistent Memory
512 GB per module at most
DIMM compatible
Background: Persistent Memory

- **inconsistency due to large data:**
  - Cpu only support 8-byte atomic write (64-bit bus)
  - Larger than 8-byte: Copy on Write (COW) or Logging
  - Prevent Reorder: mfence and clflush

![Diagram of CPU, cache, Bus, and Optane DCPMM]
Background: Hash Table

- PM is a promising replacement for DRAM.
- Hash table is widely used in main memory systems.
- Redesign hash table for PM is essential.

Optane DCPMM

replacement

DRAM
**Background: Hash Table**

- **Hash Table**: a data structure that stores or retrieves data from the position calculated by the hash function.

- **Hash collision**:
  - Linear probing, chaining, double hashing
  - Resizing:
    - (1/3) Full table: static hashing
    - Non-full table: dynamic hashing

- **Linear probing**

- **Double hashing**

- **Chaining**
Background: Static hashing

- Long resizing latency and massive extra data movement:
  - Rehash full table or 1/3 table

- Low concurrency:
  - Level hashing: slot lock for read/write (expensive lock overhead)
  - Clevel hashing: a single background thread to rehash (bottleneck)
Background: Dynamic hashing

- Extendible hashing (DRAM)
- Non-full table rehashing
- Hash collision:
  - Split a bucket directly
  - Allocate a new directory and then split a bucket (directory is small)
Background: Dynamic hashing

- **Massive data movement:**
  - Rehash 1/2 table data

- **Low concurrency**
  - **CCEH:** segment reader/writer lock (_limited concurrency_)
  - **Dash:** bucket writer lock (_limited concurrency_)

50th International Conference on Parallel Processing (ICPP)
August 9-12, 2021 in Virtual Chicago, IL
Outline

- Background
  - Persistent Memory
  - Static Hashing
  - Dynamic Hashing
  - Motivation

- Design
  - Overview
  - Lock-free
  - Multi-split
  - Instant recovery

- Evaluation

- Conclusion
Existing hashing schemes:
- **Long rehashing latency**: static hashing
- **Low concurrency**: lock overhead or single thread bottleneck
- **Massive extra data movement** in rehashing

**PMEH** (Parallel Multi-split Extendible Hashing for Persistent Memory):
- Dynamic hashing: short resizing latency
- Eliminate lock overhead
- Reduce the number of data movement in rehashing
- Guarantee data consistency
Eliminating lock overhead:

1. **Lock-free for segment:**
   - directory is divided into zones
   - one zone only is bound to one thread

2. **Lock-free for directory:**
   - introduce an extra directory array
   - multi-threads use CAS add a new directory to directory array.
   - directory entries updating: amortize into subsequent access of each thread

Lock-free parallel access in PMEH
Design: Multi-split

- **Reducing data movement:**
  - Using multi-split instead of 2-split
  - Reduce extra data movement (33%, 8-split vs 2-split)
  - Gradual split to reduce latency

Hash table expanded in 2-split

Hash table expanded in 8-split
Guarantee Data Consistency:

1. Crash when inserting a record:
   - (One-bit flag) in bucket: indicates whether finish inserting or not.

2. Crash when segment splitting:
   - (directory ID, entry ID, one-bit flag): recorded for every segment splitting.

3. Crash when directory expanding:
   - (directory pointer) for every thread:
     - null: indicates that no new directory is allocated
     - non-null: indicates that a new directory is allocated
Outline

- **Background**
  - Persistent Memory
  - Static Hashing
  - Dynamic Hashing
  - Motivation

- **Design**
  - Overview
  - Lock-free
  - Multi-split
  - Instant recovery

- **Evaluation**

- **Conclusion**
Evaluation: Experimental setup

- **Platform:**
  - Intel Optane DCPMM 512 GB(4X128GB), APPDIRECT mode
  - PMDK

- **Comparisons:**
  - DASH: default version, 256-byte bucket, two stash buckets [VLDB’20]
  - CLEVLE: store key-value instead of pointer, 128-byte bucket [ATC’20]
  - CCEH: lazy deletion version, default probing distance four buckets [FAST’19]
  - LEVEL: bucket lock, 128-byte bucket [OSDI’18]
  - PMEH: our PMEH hashing

- **Benchmark:** YCSB
Evaluation: Scalability

- PMEH is up to 1.38x and 1.10x faster than Dash for Insertion and search, respectively.
Evaluation: Scalability

- PMEH is up to 1.97x faster than Dash for deletion in 32 threads
- In conclusion, PMEH has better scalability than other hashing schemes.

Delete throughput
PMEH can reduce 52% extra data movement than Dash.
Conclusion

- Existing hashing schemes have low concurrency and massive extra data movement in rehashing.
- PMEH
  - Lock-free parallel
  - Multi-split
  - Instant recovery
- 1.38x faster for insertion, 1.97x faster for deletion, and 52% reduction for extra data movement.
Thank you!