Write-Optimized Dynamic Hashing for Persistent Memory

Moohyeon Nam†, Hokeun Cha§, Young-ri Choi‡, Sam H. Noh‡, Beomseok Nam§

UNIST (Ulsan National Institute of Science and Technology)‡
Sungkyunkwan University (SKKU)§
Outline

Background
- Static Hashing
- Extendible Hashing
- Persistent Memory

Cacheline-Conscious Extendible Hashing
- Challenges and Contributions
- 3-Level Structure of CCEH
- Failure-atomic Directory Update

Evaluation

Conclusion
Background: Static Hashing

- Hash key collision → Full table rehashing
  - The most expensive operation in hash table
• To avoid full table rehashing:
  • Linear probing
  • Chaining
  • Double hashing such as Cuckoo hashing
To avoid full table rehashing:
  - Linear probing
  - Chaining
  - Double hashing such as Cuckoo hashing
Background: Static Hashing

- To avoid full table rehashing:
  - Linear probing
  - Chaining
  - Double hashing such as Cuckoo hashing
Insertion Latency CDF

Flat Horizontal Lines
Because of the expensive full-table rehashing
Background:
Disk-based Extendible Hashing

Hash Function:
H(key) = key % 2^G

Directory
(G=2)

```
00_2  01_2  10_2  11_2
```

Buckets

```
L=2

0000 &val0
1000 &val8

L=1

0001 &val1
0111 &val7
1111 &val15
0101 &val5

L=2

0010 &val2
0110 &val6
```

- Dynamically splits one bucket or merges two buckets at a time
Background: Extendible Hashing – Insertion

H(key) = key % 2^G

Key: 0011
Value: &val3

Directory (G=2)

Hash Collision

Key Value

Directory

Hash

Collision

Buckets

L=2

L=1

L=2

H(key) = key % 2^G

key % 2

0000 &val0
1000 &val8

0001 &val1
0111 &val7
1111 &val15
0101 &val5

0010 &val2
0110 &val6
Background: Extendible Hashing – Bucket Split

- Only overflowed bucket is modified
Background: Extendible Hashing – Bucket Split

- Update Directory
  - At least two pointers need to be updated
Background: Extendible Hashing – Directory doubling

- If a single pointer points to overflow bucket
  → Directory Doubling
Background: Extendible Hashing – Directory doubling

- If a single pointer points to overflow bucket → Directory Doubling
Background: Extendible Hashing – Directory Doubling

If a single pointer points to overflow bucket → Directory Doubling
Background: Extendible Hashing – Directory Doubling

- If a single pointer points to overflow bucket → Directory Doubling

H(key) = key % 2^G

Key 11011
Value &val27

Update Directory

Update Directory

L=1
L=2
L=3
L=3

Directory (G=3)

Buckets

0000 &val0
1000 &val8
0101 &val5

0001 &val1
0101 &val5

0112 1002 1012 1102 1112

L=1
L=2
L=3
L=3
Persistent Memory

Characteristics

• High performance – Comparable to DRAM
• Byte-addressability – As DRAM
• Persistence – As storage devices (HDD/SSD)

Challenges

• Atomic unit of writes → 8-bytes
• Data transfer unit between CPU cache and PM → 64 byte cacheline
• Order of memory writes is not guaranteed
Background

- Static Hashing
- Extendible Hashing
- Persistent Memory

Cacheline-Conscious Extendible Hashing

- Challenges and Contributions
- 3-Level Structure of CCEH
- Failure-atomic Directory Update

Evaluation

Conclusion
Challenge in In-Memory Extendible Hashing

**Directory (G=2)**

<table>
<thead>
<tr>
<th></th>
<th>00₂</th>
<th>01₂</th>
<th>10₂</th>
<th>11₂</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bucket</strong></td>
<td>0000 &amp;val0</td>
<td>Cacheline 1</td>
<td>Cacheline 1</td>
<td>Cacheline 1</td>
</tr>
<tr>
<td></td>
<td>0010 &amp;val2</td>
<td>Cacheline 2</td>
<td>Cacheline 2</td>
<td>Cacheline 2</td>
</tr>
<tr>
<td></td>
<td>0110 &amp;val6</td>
<td>Cacheline 3</td>
<td>Cacheline 3</td>
<td>Cacheline 3</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>1000 &amp;val8</td>
<td>Cacheline 64</td>
<td>Cacheline 64</td>
<td>Cacheline 64</td>
</tr>
</tbody>
</table>

**Problems**

- Page-sized bucket → 64 cacheline accesses per bucket
Challenge in In-Memory Extendible Hashing

Problems
Cacheline-sized small bucket $\rightarrow$ a large directory
(8 byte pointer per cacheline)
# Challenge in Extendible Hashing on PM

## Problems

Split operation updates multiple pointers $\rightarrow$ Not Failure-Atomic
Contributions

3-Level Structure
→ Introduces an intermediate level, *Segment*
→ Lookup via only *two cacheline accesses*

Failure-atomic Directory Updates
→ Introduces *the split buddy tree* to manage split history

Failure-atomic Segment Split
→ *Lazy deletion* scheme to minimize dirty writes
A group of multiple cacheline-sized buckets = Segment
Using intermediate level "Segment",
**CCEH** reduces directory size while keeping bucket size small

3-Level Structure
Directory → Segment → Cacheline-sized Bucket
Minimize Cacheline Accesses in Segment

Q: With large segments, how can we minimize cacheline accesses?
Q: With large segments, how can we minimize cacheline accesses?
Minimize Cacheline Accesses in Segment

Q: With large segments, how can we minimize cacheline accesses?
Minimize Cacheline Accesses in Segment

Use hash key as index for both directory and segment
→ No need to access irrelevant buckets
Contributions

3-Level Structure
→ Introduces an intermediate level, Segment
→ Lookup via only two cacheline accesses

Failure-atomic Directory Updates
→ Introduces the split buddy tree to manage split history

Failure-atomic Segment Split
→ Lazy deletion scheme to minimize dirty writes
Using **MSB** segment index, split segments are pointed by adjacent **directory** entries.
Suppose a system crashes while S5 splits.
Recovery: Split History Buddy Tree in CCEH

Each segment must be pointed by $2^{G-L}$ directory entries.

If not, rollback the split.

**Stride** = $2^{G-L}$

Global Depth $G = 3$
Local Depth $L = 1$

Stride = 4
Recovery: Split History Buddy Tree in CCEH

Global Depth:
Depth: 0 1 2 3

Directory (G=3)

- S1: L=1
- S1: L=1
- S1: L=1
- S1: L=1
- S2: L=2
- S2: L=2
- S2: L=2
- S3: L=3
- S3: L=3
- S4: L=3
- S5: L=1

Stride = $2^{G-L}$

Global Depth G = 3
Local Depth L = 1
Stride = 4

- Each segment must be pointed by $2^{G-L}$ directory entries
- If not, rollback the split
Contributions

3-Level Structure
→ Introduces an intermediate level, Segment
→ Lookup via only two cacheline accesses

Failure-atomic Directory Updates
→ Introduces the split buddy tree to manage split history

Failure-atomic Segment Split
→ Lazy deletion scheme to minimize dirty writes
Segment Split: Legacy CoW

Copy-on-Write Split → Lock-Free Search is enabled
Segment Split: Lazy Deletion

Lazy Deletion → Minimizes dirty writes

Directory

Segment Split: Lazy Deletion

Local Depth = 2

Segment 00

00 01000 &val8
01 00001 &val1
10 01010 &val10
11 00011 &val3

Single Cacheline-flush invalidates all the migrated data
Outline

Background

• Static Hashing
• Extendible Hashing
• Persistent Memory

Cacheline-Conscious Extendible Hashing

• Challenges and Contributions
• 3-Level Structure of CCEH
• Failure-atomic Directory Update

Evaluation

Conclusion
## Experimental Setup

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>2x Intel Xeon Haswell-Ex E7-4809 v3 → 8 cores, 2.0 GHz → 20MB L3 cache</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>64GB of DDR3 DRAM</td>
</tr>
</tbody>
</table>
| **PM** | Quartz: A DRAM-based PM latency emulator  
* To emulate write latency, we inject stall cycle after each `clflush` instructions |
| **Workload** | 160 Million random number dataset |
CCEH compared to legacy Extendible Hashing
Cons: Low utilization and more cacheline flushes due to hash collisions
Pros: Constant number of cacheline accesses with varying directory size
CCEH is 70% faster than Level Hashing (OSDI’18) on PM
Fewer # of cacheline accesses
Lazy deletion → Efficient segment split
Load Factor

**CCEH (Optimization)**
K = Linear probing distance in Segment

- **CCEH (K=4)**
- **CCEH (K=64)**
- **Level Hashing**

**Level Hashing**
Suffer from full table rehashing like the other static hashing

**CCEH**
Dynamic allocation of small segments → Smooth curves
Cacheline-Conscious Extendible Hashing (CCEH)

- 3-Level Structure
  - Introduced an intermediate level, Segment
  - Constant Lookup: Only two cacheline accesses $\rightarrow$ Write-Optimal

- Failure-Atomic Write-Optimal Lazy Deletion $\rightarrow$ Minimize I/O
- Failure-Atomic Directory Updates $\rightarrow$ Log-less directory update

Disk-based hashing needs to be modified for PM to make effective use of cachelines.

Source Codes: http://github.com/DICL/CCEH
Thank you