Closing the Performance Gap Between Volatile and Persistent K-V Stores

Yihe Huang, Harvard University
Matej Pavlovic, EPFL
Virendra Marathe, Oracle Labs
Margo Seltzer, Oracle Labs
Tim Harris, Oracle Labs
Steve Byan, Oracle Labs

ATC 2018, 07/13/2018, Boston MA, USA
Key-Value Stores: Persistent or Fast?

- Persistent - Slow
- Volatile - Fast

or
Key-Value Stores: Persistent or Fast?

- Persistent
- Slow

or

- Volatile
- Fast

What about both?
Non-Volatile Memory

✓ Faster than HDD/SSD

✓ Byte-addressable
Non-Volatile Memory

- Faster than HDD/SSD
- Byte-addressable
- Slower than DRAM
- Tricky to manage consistently
Non-Volatile Memory

- Faster than HDD/SSD
- Byte-addressable
- Slower than DRAM
- Tricky to manage consistently

DRAM K-V stores don’t easily translate to NVM
Outline

• Cross-Referencing Logs:
  Getting the best of DRAM and NVM

• Bullet:
  Fast and persistent key-value store
Outline

• Cross-Referencing Logs:
  Getting the best of DRAM and NVM

• Bullet:
  Fast and persistent key-value store
Caching

Volatile store (frontend) → caches → Persistent store (backend)
Caching

Volatile store (frontend)

Persistent store (backend)

Caches

NVM
Caching

Volatile store (frontend)

How?

Persistent store (backend)

Caches

NVM
Reading: Standard

Volatile store (frontend) \[\rightarrow\] Lookup on cache miss \[\rightarrow\] Persistent store (backend)

NVM
Writing: Logs

1. Persistently log op.
2. Update frontend
3. Return to client
4. Update backend in background
Wr How? ogs

1. Persistently log op.
2. Update frontend
3. Return to client
4. Update backend in background

Volatile store (frontend)

Persistent store (backend)

NVM
Writing: Logs

How?

1. Persistently log op.
2. Update frontend
3. Return to client
4. Update backend in background

Volatile store (frontend)

Persistent store (backend)

Challenges: persistence, low latency, scalability, consistency
Cross-Referencing Logs

✅ Persistence:
  Place logs in NVM

✅ Low latency
  Few NVM accesses per log append

✅ Scalability
  Multiple logs (one per thread)

✅ Consistency
  Cross-log references
Cross-Referencing Logs

- **Persistence:**
  - Place logs in NVM

- **Low latency**
  - Few NVM accesses per log append

- **Scalability**
  - Multiple logs (one per thread)

- **Consistency**
  - Cross-log references
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
</table>

append

consume
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

append

consume
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

Key 2, OP: put “B”
Not Applied, Pr: ⊥
## Cross-Referencing Logs

### Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key 1</td>
<td>put “A”&lt;br&gt;Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key 2</td>
<td>delete&lt;br&gt;Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key 2</td>
<td>put “B”&lt;br&gt;Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Append**

- Key 2, OP: delete<br>Not Applied, Pr: ⊥
- Key 1, OP: put “A”<br>Not Applied, Pr: ⊥
- Key 2, OP: put “B”<br>Not Applied, Pr: ⊥

**Consume**
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”  
Not Applied, Pr: ⊥

Key 2, OP: delete  
Not Applied, Pr: ⊥

Key 2, OP: put “B”  
Not Applied, Pr: ⊥
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

Key 2, OP: delete
Not Applied, Pr: ⊥

Key 2, OP: put “B”
Not Applied, Pr: ⊥

Key 3, OP: delete
Not Applied, Pr: ⊥

append

consume
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

Key 2, OP: delete
Not Applied, Pr: ⊥

Key 2, OP: put “B”
Not Applied, Pr: ⊥

Key 2, OP: put “C”
Not Applied, Pr: ⊥

Key 3, OP: delete
Not Applied, Pr: ⊥
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key 1, OP: put “A”</td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key 2, OP: delete</td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key 2, OP: put “B”</td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key 2, OP: put “C”</td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key 3, OP: delete</td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Key 1, OP: put “A”
  - Not Applied, Prev: ⊥

- Key 2, OP: delete
  - Not Applied, Prev: ⊥

- Key 2, OP: put “B”
  - Not Applied, Prev: ⊥

- Key 2, OP: put “C”
  - Not Applied, Prev: ⊥

- Key 3, OP: delete
  - Not Applied, Prev: ⊥
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

Key 2, OP: delete
Not Applied, Pr: ⊥

Key 2, OP: put “B”
Not Applied, Pr: ⊥

Key 3, OP: delete
Applied, Pr: ⊥
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”  
Not Applied, Pr: ⊥

Key 2, OP: delete  
Not Applied, Pr: ⊥

Key 2, OP: put “B”  
Not Applied, Pr: ⊥

Key 3, OP: delete  
Applied, Pr: ⊥

append

consume
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key 1</td>
<td>OP: put “A”</td>
<td>Not Applied</td>
<td>Pr: ⊥</td>
</tr>
<tr>
<td>Key 2</td>
<td>OP: delete</td>
<td>Not Applied</td>
<td>Pr: ⊥</td>
</tr>
<tr>
<td>Key 3</td>
<td>OP: delete</td>
<td>Applied</td>
<td>Pr: ⊥</td>
</tr>
</tbody>
</table>

append

Key 2, OP: put “B”
Not Applied, Pr: ⊥

Key 2, OP: put “C”
Not Applied, Pr: ⊥

consume
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>put &quot;A&quot;</td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>put &quot;B&quot;</td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>delete</td>
<td>Applied, Pr: ⊥</td>
<td></td>
</tr>
</tbody>
</table>

append

consume
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key 1, OP: put “A”</td>
<td></td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
</tr>
<tr>
<td>Key 2, OP: delete</td>
<td></td>
<td>Not Applied, Pr: ⊥</td>
<td></td>
</tr>
<tr>
<td>Key 2, OP: put “B”</td>
<td>Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key 3, OP: delete</td>
<td>Applied, Pr: ⊥</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
- Applied, Pr: ⊥
- Not Applied, Pr: ⊥

Key 2, OP: delete
- Applied, Pr: ⊥
- Not Applied, Pr: ⊥

Key 2, OP: put “B”
- Applied, Pr: ⊥

Key 2, OP: put “C”
- Not Applied, Pr: ⊥
- Applied, Pr: ⊥

Key 3, OP: delete
- Applied, Pr: ⊥
Cross-Referencing Logs

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Key 1, OP: put “A”**
  - Applied
  - Prev: ⊥

- **Key 2, OP: delete**
  - Applied
  - Prev: ⊥

- **Key 2, OP: put “B”**
  - Applied
  - Prev: ⊥

- **Key 2, OP: put “C”**
  - Applied
  - Prev: ⊥

- **Key 3, OP: delete**
  - Applied
  - Prev: ⊥
Outline

• Cross-Referencing Logs:
  Getting the best of DRAM and NVM

• Bullet:
  Fast and persistent key-value store
Outline

• Cross-Referencing Logs:
  Getting the best of DRAM and NVM

• Bullet:
  Fast and persistent key-value store
  • Basic design
  • Optimizations
Outline

• Cross-Referencing Logs:
  Getting the best of DRAM and NVM

• Bullet:
  Fast and persistent key-value store
  • Basic design
  • Optimizations
Bullet K-V Store

Volatile store (frontend)

Persistent store (backend)

Caches

NVM
Bullet K-V Store

Volatile store (frontend)

Persistent store (backend)

Caches

NVM
Bullet K-V Store

Volatile store
(frontend)

Persistent store
(backend)

caches
Experimental Setup

• 16 CPU cores, 512 GB DRAM

• Intel’s NVM emulation platform

• Zipfian distribution of keys (YCSB)

• Measuring 99%ile latency

• Comparing against HiKV
Raw Hash Table Performance
Bullet K-V Store

Volatile store (frontend)

Persistent store (backend)

caches
Bullet K-V Store

Volatile store
(frontend)

Persistent store
(backend)

X-Ref Logs
Bullet K-V Store

Volatile store (frontend)  Persistent store (backend)

Writers

X-Ref Logs
Bullet K-V Store

Volatile store
(frontend)

Gleaners

Persistent store
(backend)

Writers

X-Ref Logs
Bullet Performance

Read-only

Latency in ns

Throughput in ops/sec

volatile
phash
Bullet Performance

Read-only

- volatile
- phash
- hikv-ht

Latency in ns vs Throughput in ops/sec graph.
Bullet Performance

Read-only

Throughput in ops/sec

Latency in ns

- volatile
- phash
- hikv-ht
- bullet-st
Bullet Performance

15% writes

Throughput in ops/sec vs. Latency in ns for various data structures:
- volatile
- phash
- hkv-ht
- bullet-st
Outline

• Cross-Referencing Logs:
  Getting the best of DRAM and NVM

• Bullet:
  Fast and persistent key-value store
  • Basic design
  • Optimizations
Optimization: Dynamic Worker Threads

Volatile store (frontend)

Writers

Persistent store (backend)

Gleaners

X-Ref Logs
Optimization: Dynamic Worker Threads

Volatile store
(frontend)

Persistent store
(backend)

Writers

Gleaners

X-Ref Logs
Read-Heavy Workload

Volatile store (frontend)

Persistent store (backend)

Writers

Gleaners

X-Ref Logs

Volatile store

Persistent store
Write-Heavy Workload

Volatile store (frontend)

Persistent store (backend)

Writers

Gleaners

X-Ref Logs
Optimization: Overwriting

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key 1</td>
<td>put “A”</td>
<td>Not Applied</td>
<td>⊥</td>
</tr>
<tr>
<td>Key 2</td>
<td>delete</td>
<td>Not Applied</td>
<td>⊥</td>
</tr>
<tr>
<td>Key 2</td>
<td>put “B”</td>
<td>Not Applied</td>
<td>⊥</td>
</tr>
<tr>
<td>Key 2</td>
<td>put “C”</td>
<td>Not Applied</td>
<td>⊥</td>
</tr>
<tr>
<td>Key 3</td>
<td>delete</td>
<td>Applied</td>
<td>⊥</td>
</tr>
</tbody>
</table>

Append

Consume
Optimization: Overwriting

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

Key 2, OP: delete
Not Applied, Pr:

Key 2, OP: put “B”
Not Applied, Pr: ⊥

Key 2, OP: put “C”
Not Applied, Pr:

Key 3, OP: delete
Applied, Pr: ⊥
Optimization: Overwriting

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

Key 2, OP: delete
Not Applied, Pr: ⊥

Key 2, OP: put “C”
Not Applied, Pr: ⊥

Key 3, OP: delete
Applied, Pr: ⊥

append
consume
Optimization: Overwriting

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
Not Applied, Pr: ⊥

Key 2, OP: delete
Not Applied, Pr: ⊥

Key 2, OP: put “B”
Not Applied, Pr: ⊥

Key 2, OP: put “C”
Not Applied, Pr: ⊥

Key 3, OP: delete
Applied, Pr: ⊥

append

consume
Optimization: Overwriting

Log entry:

<table>
<thead>
<tr>
<th>Key</th>
<th>OP</th>
<th>Applied</th>
<th>Prev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key 1, OP: put “A”
Not Applied, Pr: \( \perp \)

Key 2, OP: delete
Not Applied, Pr: \( \perp \)

Key 2, OP: put “B”
Not Applied, Pr: \( \perp \)

Key 2, OP: put “C”
Not Applied, Pr: \( \perp \)

Key 3, OP: delete
\( \text{Applied, Pr: } \perp \)

IGNORE
Optimization: Overwriting

Volatile store (frontend)

Persistent store (backend)

Writers

Gleaners

X-Ref Logs
Bullet Performance

Read-only

- volatile
- phash
- hikv-ht
- bullet-st

Latency in ns vs Throughput in ops/sec
Bullet Performance

Read-only

Latency in ns

Throughput in ops/sec

0 5000000 10000000 15000000 20000000 25000000

Latency in ns

0 100 200 300 400 500 600 700 800

Throughput in ops/sec
Bullet Performance

15% writes

Latency in ns vs Throughput in ops/sec for different data access patterns and storage mechanisms:
- volatile
- phash
- hikv-ht
- bullet-st
- bullet-dyn-ovr

The graph shows the latency of operations in nanoseconds (ns) on the y-axis and the throughput in operations per second (ops/sec) on the x-axis for different storage systems and data access patterns.
Bullet Performance

2% writes

Throughput in ops/sec

Latency in ns

- volatile
- phash
- hikv-ht
- bullet-st
- bullet-dyn-ovr
Summary

• DRAM cache for NVM
  • DRAM frontend, NVM backend
  • Novel caching scheme: Cross-referencing logs
Summary

• DRAM cache for NVM
  • DRAM frontend, NVM backend
  • Novel caching scheme: Cross-referencing logs

• Bullet K-V store
  • Almost identical hash tables in DRAM and NVM
  • DRAM performance for reads
  • Substantial latency improvements for writes
Summary

• DRAM cache for NVM
  • DRAM frontend, NVM backend
  • Novel caching scheme: Cross-referencing logs

• Bullet K-V store
  • Almost identical hash tables in DRAM and NVM
  • DRAM performance for reads
  • Substantial latency improvements for writes

• More details in the paper!
Backup Slides
Optimization Summary

• Dynamic worker threads

• Overwrites

• Non-blocking reads (see paper)

• Backend access off critical path (see paper)
Bullet Performance

Read-only
Bullet Performance

2% writes

![Graph showing latency vs throughput for different conditions.](image)
Bullet Performance

15% writes

[Graph showing latency versus throughput with various data points and lines representing different configurations such as volatile, phash, hikv-h, bullet-st, +lfr, +opt, +dyn, +wrc(bullet-full).]
Bullet Performance

50% writes

[Graph showing latency vs. throughput for various systems including volatile, phash, hikv-ht, bullet-st, +ifr, +opt, +dyn, +wrc(bullet-full)]
Bullet Performance (end-to-end)
Bullet Performance (end-to-end)

2% writes
Bullet Performance (end-to-end)

15% writes
Bullet Performance (end-to-end)

50% writes

![Graph showing performance metrics](image-url)
Latency Distribution

Reads

![Latency Distribution Graph](image-url)
Log Size Sensitivity

![Bar graph showing throughput (Mops) vs. percent puts for different log sizes (1MB, 4MB, 16MB, 64MB).](image)
Experimental Setup

• 16 CPU cores
• 512 GB DRAM
• NVM simulated using DRAM
  • 384 GB (out of 512GB)
  • 300ns load (2x DRAM)
  • 100ns persist barrier
  • Same throughput
• DRAM cache fits all data
• Zipfian distribution of keys (YCSB)
• 50M K/V pairs, 16 B keys, 100 B values
• Measuring 99%ile latency
Dynamic Worker Thread Switching

Throughput (Mops)

Time in 30 sec intervals

# Gleaners
Cross-Referencing Logs

✅ Persistence:
  Circular buffers in NVM

✅ Low latency
  Fast appends (3 persist barriers)

✅ Scalability
  Multiple logs, stall only when all full
  Coarse-grained synchronization
  Epoch-based space reclamation

✅ Consistency
  Cross-log references
Cross-Referencing Logs Summary

- Persistent circular buffers (in NVM)
- Cross-log pointers for consistency
- Coarse-grained synchronization
- Fast appends
- Epoch-based space reclamation
Read-only

Latency in ns

Throughput in ops/sec