FloDB: Unlocking Memory in Persistent Key-Value Stores

O. Balmau\(^1\), R. Guerraoui\(^1\), V. Trigonakis\(^2\), I. Zablotchi\(^1\)

\(^1\)EPFL, \(^2\)Oracle

EuroSys ‘17, Belgrade
Outline

Motivation

- FloDB
- Evaluation
- Conclusion
KV Stores

Very **simple** data stores.
KV pairs.

In-memory vs. **persistent**
KV Stores

Very **simple** data stores.
KV pairs.

**In-memory vs. persistent**

Log-Structured Merge (LSM)
LSM Overview
LSM Overview
LSM Overview

- RAM
- Disk
  - HDD/SSD
- skiplist/hash table
LSM Overview

- On-disk levels
- Sorted files (SST)
- Many SSTs/Level

RAM

Disk HDD/SSD

skiplist/hash table

$L_0$

$L_n$
LSM Overview

- put, remove sequential
- skiplist/hash table

RAM

Disk HDD/SSD
LSM Overview

- **put, remove**
- **sequential**
- **get**
- **concurrent**

RAM

Disk
HDD/SSD

- \(L_0\)
- \(L_n\)
Current LSM Limitations

1. Scalability with **threads**.
   Due to global locking synchronization.

2. Scalability with **memory size**.
   Need to keep elements sorted (expensive).
Current LSM Limitations

1. Scalability with **threads**.

   This talk: focus on **memory component**.

2. Scalability with **memory size**.

   Need to keep elements sorted (expensive).
Memory Scalability

Throughput (Mops/s)

Memory component size

16 threads; write-only workload.
Memory Scalability

Throughput (Mops/s)

Average persistence throughput

Memory component size

16 threads; write-only workload.

RocksDB
HyperLevelDB
LevelDB
Outline

- Motivation
- FloDB
- Evaluation
- Conclusion
FloDB

New design for LSM memory component
1-Level Mem. Component

Classic LSM

Skiplist/Hash table

Disk
1-Level Mem. Component

**Skiplist**
- ✓ Already sorted, flush to disk as is.
- ✗ O(log n) time to insert elements.

**Hash Table**
- ✗ Sort before writing to disk.
- ✓ O(1) updates
FloDB Structure

- fast unsorted
- sorted

Disk
FloDB Structure

hash table

fast unsorted

sorted

Disk
FloDB Structure

hash table

fast unsorted

skiplist

sorted

Disk
FloDB Concurrency

put, remove

concurrent

concurrent hash table (HT)

concurrent skiplist (SL)

Disk

fast unsorted

sorted
FloDB Main Challenge

Ensure efficient **data flow**:

```plaintext
hashtable → skiplist → disk

~100M ops/s  ~10M ops/s  ~1M ops/s
```
FloDB Data Flow

Draining

Goal: Keep HT empty
Continuous bg. op

SL Multi-insert
Novel operation

Insert multiple elements in SL at a time
Skiplist Multi-insert

Intuition: **Path sharing.**
Skiplist Multi-insert

Intuition: Path sharing.
Skiplist Multi-insert

Intuition: Path sharing.
Skiplist Multi-insert

Intuition: Path sharing.
Skiplist Multi-insert

Intuition: Path sharing.
FloDB Tradeoffs

Scans + In-place updates

Drain HT completely
FloDB Tradeoffs

Scans + In-place updates

Scan only SL + disk
Outline

- Motivation
- FloDB

Evaluation

- Conclusion
Evaluation Setup

**Code:**
FloDB – open source, implemented on top of LevelDB.
http://lpd.epfl.ch/site/flodb

**Workloads:**
Focus on write-intensive workloads.
Evaluation Setup

Compare FloDB with state-of-the-art LSM KV stores:

- FloDB
- RocksDB
- HyperLevelDB
- LevelDB
Steady-state Throughput
Write-only workload

Throughput (Mops/s)

Number of threads

Mem. component size 128MB

(higher is better)

FloDB
RocksDB
HyperLevelDB
LevelDB

(higher is better)
Steady-state Throughput
Write-only workload

Throughput (Mops/s)

Number of threads

FloDB
RocksDB
HyperLevelDB
LevelDB

Average persistence throughput

(higher is better)

Mem. component size 128MB
Steady-state Throughput
Write-only workload

higher is better

→ FloDB reaches SSD persistence throughput.

Number of threads
Mem. component size 128MB

Throughput

FloDB
RocksDB
HyperLevelDB
LevelDB

0
0.2
0.4
0.6
0.8
1.0
1.2

10
20
40
80

0
1
2
4
8

Steady-state Throughput
Write-only workload

(higher is better)

→ FloDB reaches SSD persistence throughput.
Scalability with Memory Size
Bursts of Writes

(higher is better)

Throughput (Mops/s)

16 threads; FloDB mem. split: ¼ HT, ¾ SL

Memory Component Size

FloDB
RocksDB
HyperLevelDB
LevelDB

128MB 256MB 512MB 1GB 2GB 4GB
Scalability with Memory Size
Bursts of Writes

Throughput (Mops/s)

FloDB an order of magnitude better

FloDB
RocksDB
HyperLevelDB
LevelDB

Memory Component Size

16 threads; FloDB mem. split: ¼ HT, ¾ SL

(higher is better)
Scalability with Memory Size
Bursts of Writes

(higher is better)

→ Can sustain larger burst of writes.

Through

Memory Component Size

16 threads; FloDB mem. split: ¼ HT, ¾ SL
Related work

- RocksDB, HyperLevelDB.
  - Better concurrency by reducing size and number of critical sections.

- cLSM (EuroSys ‘15)
  - based on LevelDB. Design goal to increase thread scalability.

- LSM disk-component:
  - bLSM (SIGMOD/PODS ‘12), HyperLevelDB, LSM-trie (USENIX ATC ‘15), VT-tree (FAST ‘13), WiscKey (FAST ‘16)

- In-memory KV stores:
  - KiWi (PODC ‘16), Masstree (EuroSys ‘12), MemC3 (NSDI ‘13), Memcache (NSDI ‘13), MICA (NSDI ‘14)
More in the paper

- Operations implementation
  - Range scan consistency.
  - In-place updates.
- Experiments
  - Thread scalability.
  - Skewed workloads.
  - Scans.
  - Multi-insert.
  - And more...
Key Takeaways

✅ FloDB – novel two-level memory component for LSM.
Key Takeaways

✅ FloDB – novel two-level memory component for LSM.

✅ Novel multi-insert operation for concurrent skiplists.
Key Takeaways

- FloDB – novel two-level memory component for LSM.
- Novel multi-insert operation for concurrent skiplists.
- Scales with memory size and with threads.

Thank you! Questions?