EvenDB: Optimizing Key-Value Storage for Spatial Locality

Eran Gilad, Edward Bortnikov, Anastasia Braginsky, Yonatan Gottesman, Eshcar Hillel (Yahoo Research), Idit Keidar (Technion), Nurit Moscovici (Outbrain), Rana Shahout (Technion)
Key-value stores

- key -> value mapping

\[ k1 \rightarrow v1, k2 \rightarrow v2, k3 \rightarrow v3, k4 \rightarrow v4, k5 \rightarrow v5, k6 \rightarrow v6, k7 \rightarrow v7, k8 \rightarrow v8, k9 \rightarrow v9 \]

put, get, scan
Key-value stores

- key -> value mapping
- skewed workload: some keys are hotter

```
+ + + + + + +

| k1 → v1 |
| k2 → v2 |
| k3 → v3 |
| k4 → v4 |
| k5 → v5 |
| k6 → v6 |
| k7 → v7 |
| k8 → v8 |
| k9 → v9 |
```

put, get, scan
Key-value stores

- key -> value mapping
- skewed workload: some keys are hotter
- spatial locality: some ranges are hotter
  - e.g., complex keys

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>k1_l1</td>
<td>v1</td>
</tr>
<tr>
<td>k1_l2</td>
<td>v2</td>
</tr>
<tr>
<td>k1_l3</td>
<td>v3</td>
</tr>
<tr>
<td>k2_l1</td>
<td>v4</td>
</tr>
<tr>
<td>k2_l2</td>
<td>v5</td>
</tr>
<tr>
<td>k3_l1</td>
<td>v6</td>
</tr>
<tr>
<td>k3_l2</td>
<td>v7</td>
</tr>
<tr>
<td>k3_l3</td>
<td>v8</td>
</tr>
<tr>
<td>k3_l4</td>
<td>v9</td>
</tr>
</tbody>
</table>
```

put, get, scan
Key-value stores

- key -> value mapping
- skewed workload: some keys are hotter
- spatial locality: some ranges are hotter
  - e.g., complex keys
- Sample production trace:
  - appname_timestamp
  - 1% of apps ⇒ 1% key prefixes ⇒ 94% of events
LSM-trees

Ranges overlap

MemTable

More capacity (e.g., 10x)

Memory

Disk
LSM-trees are designed for temporal locality

Update time

Memory

Disk

MemTable

L0

L1

L2

Compactions merge hot and cold ranges
LSM-trees are less suited for spatial locality

Ranges are fragmented

scan(...):

Memory

Disk

MemTable

L0

L1

L2
EvenDB

- Ordered key-value store
- Optimized for spatial locality
- Low write amplification
- Persistent, fast recovery
- Atomic operations, including scan
Chunk-based organization

- Dynamically partitioned key space into *chunks*
  - Much smaller than shards
  - Much larger than blocks

- **Chunks are the basic unit for**
  - Disk I/O
  - Compaction
  - Memory caching
  - Concurrency control
Chunks metadata

Linked list of chunks

Chunk objects hold metadata - versions, sync. mechanisms, file handles, stats etc.

RAM

disk
Chunks index

Quickly locate the chunk whose range includes the given key

RAM
disk
Disk storage - updates

Immediately store in log; Occasionally merge log into SST
Disk storage - lookups

- **#1 - search row cache**
- **#2 - search log**
- **#3 - search SST**

Scans always search SST and log.
Memory cache - updates

#1 - Store in log

#2 - Store in munk

#3 - Occasionally rebalance munk

#4 - Rarely create SST from munk
Memory cache - lookups

- RAM
- disk
- Bloom filters
- row cache
- chunk
- index
- munk
- munk
- munk cache
- Search/scan munk
- funk
- SSTable
- log
- funk
- SSTable
- log
- funk
- SSTable
- log
Evaluation

- **3 benchmark suites**
  - Traces from internal production system, 256GB DB - some presented next
  - Standard and extended YCSB benchmarks - results in paper

- **State-of-the-art LSM: RocksDB**
Real dataset ingestion

Throughput dynamics - 256GB DB creation

EvenDB 4.4x faster, write amp. 4x lower (better)
Compactions impact

Throughput dynamics - 256GB DB creation

- EvenDB
- RocksDB

Execution time, minutes

RocksDB throughput drops during compaction

EvenDB runs much smoother

Space amp.: DB size during ingestion

- RocksDB
- EvenDB
- Log space
- Input size

% of total insertions
Real dataset scans

Scan throughput dynamics, 256GB

- EvenDB
- RocksDB

RocksDB faster after storage optimized

EvenDB 1.2x faster than RocksDB

~38 minutes stall after DB creation
Summary

- **EvenDB introduces a novel key-value store architecture**
- **Chunk arrangement better suited for spatially-local workloads than LSM:**
  - Lower write amplification
  - Single level of storage, no overlapping
  - Memory serves reads and writes
- **EvenDB outperforms RocksDB when:**
  - Workload is spatially-local or most working set fits in RAM
  - In par otherwise
  - Demonstrated in real workload and synthetic YCSB benchmarks