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

\[
\begin{align*}
  k1 & \rightarrow v1 \\
  k2 & \rightarrow v2 \\
  k3 & \rightarrow v3 \\
  k4 & \rightarrow v4 \\
  k5 & \rightarrow v5 \\
  k6 & \rightarrow v6 \\
  k7 & \rightarrow v7 \\
  k8 & \rightarrow v8 \\
  k9 & \rightarrow v9 \\
\end{align*}
\]
Key-value stores

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

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>k1</td>
<td>v1</td>
</tr>
<tr>
<td>k2</td>
<td>v2</td>
</tr>
<tr>
<td>k3</td>
<td>v3</td>
</tr>
<tr>
<td>k4</td>
<td>v4</td>
</tr>
<tr>
<td>k5</td>
<td>v5</td>
</tr>
<tr>
<td>k6</td>
<td>v6</td>
</tr>
<tr>
<td>k7</td>
<td>v7</td>
</tr>
<tr>
<td>k8</td>
<td>v8</td>
</tr>
<tr>
<td>k9</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

```
<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

![Graph showing mobile apps events distribution with a log scale for both axes, illustrating the probability density and app popularity ranking.](image-url)
LSM-trees

Memory

Disk

MemTable

L0

L1

L2
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(...):

MemTable
L0
L1
L2

Memory
Disk
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.
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 flush munk to SST
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

Space amp.: DB size during ingestion

- RocksDB
- EvenDB
- Log space
- Input size

Execution time, minutes

RocksDB throughput drops during compaction

EvenDB runs much smoother
Real dataset scans

Scan throughput dynamics, 256GB

RocksDB faster here due to long compactions

~38 minutes stall after DB creation

EvenDB 1.2x faster than RocksDB
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
  - 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