TRIAD: Creating Synergies Between Memory, Disk and Log in Log Structured Key-Value Stores

USENIX ATC ’17, Santa Clara CA

O. Balmau
EPFL

D. Didona
EPFL

R. Guerraoui
EPFL

W. Zwaenepoel
EPFL

H. Yuan
Nutanix

A. Arora
Nutanix

K. Gupta
Nutanix

P. Konka
Nutanix
**KV Stores**

Very **simple** data stores.

**KV pairs.**

Simple operations: **update**, **read**.
KV Stores

<table>
<thead>
<tr>
<th>Persistent</th>
<th>Single machine</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-memory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Green: Persistent
- Blue: In-memory
- Light Green: Distributed
- Dark Green: Single machine
KV Stores

<table>
<thead>
<tr>
<th>Persistent</th>
<th>Single machine</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-memory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
KV Stores

- Single machine
  - Persistent
    - TRIAD
  - In-memory
- Distributed
  - Log-Structured Merge (LSM)
TRIAD in a Nutshell

TRIAD LSM KV: achieves 2x throughput on production wklds.

Methods: Reducing background I/O in LSMs.
TRIAD in a Nutshell

TRIAD LSM KV: achieves 2x throughput on production wklds.

Methods: Reducing background I/O in LSMs.

😄 No need to know workload a priori.
😄 LSM KV semantics preserved.
LSM Overview
LSM Components

**SSTables**
- sorted files
- many SSTables/Level

**Sorted** memory component
LSM Updates

Commit Log

update

Cm

L₀

Lₙ

RAM

Disk HDD/SSD
LSM Reads

- $L_0$
- $L_n$
- $C_m$
- RAM
- Disk HDD/SSD

 read
LSM Background Ops: **Flushing**

Flushing
From memory to L0.
LSM Background Ops: **Flushing**

**Mem component full**

---

**RAM**

**Disk**

**Cm**

**L0**

**Commit Log**

<table>
<thead>
<tr>
<th>K1</th>
<th>V1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>K1'</td>
<td>V1'</td>
</tr>
<tr>
<td>K3</td>
<td>V3</td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
</tr>
</tbody>
</table>
LSM Background Ops: **Flushing**
LSM Background Ops: Flushing

Commit log full

RAM

Disk

flushing

Cm

L0

Commit Log

<table>
<thead>
<tr>
<th>K1</th>
<th>V1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>K3</td>
<td>V3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K1</th>
<th>V1’</th>
</tr>
</thead>
<tbody>
<tr>
<td>K3</td>
<td>V3’</td>
</tr>
</tbody>
</table>
LSM Background Ops: **Flushing**

Cm

RAM

Disk

flushing

K1  V1'
K2  V2
K3  V3'

flush_1

Commit Log
LSM Background Ops: **Compaction**

Flush: From memory to L0.

Compaction: Levels on disk.

RAM

Disk

HDD/SSD
LSM Background Ops: **Compaction**

Disk

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

L0

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

L1

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Ln

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### LSM Background Ops: **Compaction**

#### Disk

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LSM Background Ops: Compaction

Disk

L0

L1

Ln

Key | Val
--- | ---
... | ... 
K   |   

K rewritten to disk

Key | Val
--- | ---
... | ... 
K   |   

Key | Val
--- | ---
... | ... 

Key | Val
--- | ---
... | ... 

Key | Val
--- | ---
... | ... 

Key | Val
--- | ---
... | ...
LSM Background Ops: **Compaction**

Disk

[Diagram showing LSM with levels L0, L1, and Ln, with keys and values being rewritten to disk.]
LSM Background Ops: **Compaction**

![Diagram of LSM Background Ops: Compaction](image-url)

- **Disk**
- **L0**
- **L1**
- **Ln**

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- Key | Val
- --- | ---
- ... | ...
- ... | ...

- **K rewritten n+1\text{th} time to disk!**
LSM Background Ops: **Compaction**

**Write amplification (WA)**

\[ \approx \text{amount of rewrites of data to disk} \]

K rewritten \(n+1^{\text{th}}\) time to disk!
Insight

Severe competition for compute/storage resources between LSM background ops and user ops.
Background I/O Overhead

- Long & slow bg. ops → slowdown of user ops.

![Bar chart showing Background I/O Overhead for RocksDB and RocksDB No BG I/O under Uniform and Skewed 50r-50w conditions.](chart.png)
Background I/O Overhead

- Long & slow bg. ops $\rightarrow$ slowdown of user ops.

\[ \text{K Operations/s} \]

- RocksDB
- RocksDB No BG I/O

\[ \text{up to 3x throughput gap} \] 😞
Goal

Decrease background ops overhead to increase user throughput.
TRIAD
Three techniques work together and are complementary.
TRIAD

TRIAD-MEM

Workload: Skewed workloads

Improve WA in: Flushing and Compaction

TRIAD-LOG

Workload: Uniform workloads

Improve WA in: Flushing
TRIAD-MEM

Workload

Skewed workloads

Improve WA in

Flushing and Compaction
Problem: Flushing with Skewed Workloads

RAM
Disk

RAM
Disk

flushing

Cm

LO

Commit Log
Problem: Flushing with Skewed Workloads

RAM

Disk

Cm

flushing

L0

Commit Log

K1 V1

K1 V1

Problem: Flushing with Skewed Workloads
Problem: Flushing with Skewed Workloads

RAM

Disk

flushing

Cm

L0

Commit Log

<table>
<thead>
<tr>
<th>K1</th>
<th>V1_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>V1_2</td>
</tr>
</tbody>
</table>
Problem: Flushing with Skewed Workloads
Problem: Flushing with Skewed Workloads

Commit Log

<table>
<thead>
<tr>
<th>K1</th>
<th>V1_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>V1_2</td>
</tr>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>K1</td>
<td>V1_3</td>
</tr>
</tbody>
</table>

RAM

Disk

flushing

Cm

L0
Problem: Flushing with Skewed Workloads
Problem: Flushing with Skewed Workloads
Problem: Flushing with Skewed Workloads
Problem: Flushing with Skewed Workloads

High data skew
- Flush because commit log is full
- Flush mostly empty mem comp

Commit Log

<table>
<thead>
<tr>
<th>K1</th>
<th>V1_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>K1</td>
<td>V1_2</td>
</tr>
<tr>
<td>K1</td>
<td>V1_3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>K1</td>
<td>V1_n</td>
</tr>
</tbody>
</table>

RAM

Disk

flushing

Flushing with Skewed Workloads

High data skew
- Flush because commit log is full
- Flush mostly empty mem comp

Commit Log

<table>
<thead>
<tr>
<th>K1</th>
<th>V1_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>K1</td>
<td>V1_2</td>
</tr>
<tr>
<td>K1</td>
<td>V1_3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>K1</td>
<td>V1_n</td>
</tr>
</tbody>
</table>

RAM

Disk
Problem: Compaction with Skewed Workloads

Popular key K1
Problem: Compaction with Skewed Workloads
Problem: Compaction with Skewed Workloads
Problem: Compaction with Skewed Workloads
Problem: Compaction with Skewed Workloads

Rewritten to disk
**Problem:** Compaction with Skewed Workloads

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

File on L1 **rewritten to disk twice** because of one key 😞
TRIAD-MEM: Hot-cold key separation
TRIAD-MEM: Hot-cold key separation

Idea:
- Keep hot keys in memory
- Flush only cold keys
- Keep hot keys in CL

RAM

Disk

Commit Log

Commit	Log

\begin{array}{|c|c|}
\hline
K1 & V1_1 \\
\hline
K1 & V1_2 \\
\hline
K1 & V1_3 \\
\hline
K1 & V1_4 \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
K1 & V1_n \\
\hline
\end{array}
**TRIAD-MEM: Hot-cold key separation**

**Idea:**
- Keep **hot keys** in memory
- Flush only **cold keys**
- Keep **hot keys** in CL

![Diagram of TRIAD-MEM](image)

- **RAM**
- **Disk**
- **Commit Log**

- **L0**

- **Flushing**

<table>
<thead>
<tr>
<th>K1</th>
<th>V1_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>K3</td>
<td>V3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
</tr>
</tbody>
</table>
TRIAD-MEM Summary

✓ Good for skewed workloads.

✓ Reduce flushing WA: less data written from memory to disk.

✓ Reduce compaction WA: avoid repeatedly compacting hot keys.
**TRIAD-LOG**

**Workload**
- Uniform workloads

**Improve WA in**
- Flushing
Problem: Flushing with Uniform Workloads

### Key-Value Pairs

<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></td>
<td></td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
</tr>
</tbody>
</table>

### RAM to Disk Flushing

- **RAM**
- **Disk**
- **Commit Log**

<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>K1</td>
<td>V1'</td>
</tr>
<tr>
<td>K3</td>
<td>V3</td>
</tr>
<tr>
<td>K3</td>
<td>V3'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
</tr>
</tbody>
</table>
Problem: Flushing with Uniform Workloads

RAM

Disk

flushing

Cm

Commit Log

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>V1</td>
</tr>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>V1’</td>
</tr>
<tr>
<td>K3</td>
<td>V3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
</tr>
</tbody>
</table>
Problem: Flushing with Uniform Workloads

RAM

Disk

Cm

flushing

Commit Log

L0

flush

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>V1'</td>
</tr>
<tr>
<td>K2</td>
<td>V2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
</tr>
</tbody>
</table>

| K1  | V1  |
| K2  | V2  |
| K1  | V1' |
| K3  | V3  |
| K3  | V3' |
| ... |     |
| Kn  | Vn  |
Problem: Flushing with Uniform Workloads

Insight:
Flushed data already written to commit log.

Idea:
Use commit logs as SSTables. Avoid bg I/O due to flushing.
TRIAD-LOG

Point to most recent entry in CL.
TRIAD-LOG

RAM

Disk

Commit Log

<table>
<thead>
<tr>
<th>Key</th>
<th>Val</th>
<th>CL Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>V1'</td>
<td>3</td>
</tr>
<tr>
<td>K2</td>
<td>V2</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kn</td>
<td>Vn</td>
<td>n</td>
</tr>
</tbody>
</table>

flushing

Cm

L0
**TRIAD-LOG**

**CL-SSTable**
Only flush CL Index from memory and couple it with the current Commit Log.

**Commit Log**

Only flush CL Index from memory and couple it with the current Commit Log.

Keep index in memory for further reads.
TRIAD-LOG Summary

✓ Good for uniform workloads.

✓ Reuse Commit Log as L0 SST.

✓ No more flushing of mem component to disk.
TRIAD Summary

TRIAD-MEM, TRIAD-DISK, TRIAD-LOG:

- Complementary, targeting different wklds.
- Working simultaneously.
- Transparent to the workloads; no a priori knowledge needed.
Evaluation
Evaluation

- Compare TRIAD with RocksDB
- **Workloads:** Production, Synthetic
- **Metrics:** Throughput, Write Amplification (WA)
- **Code:** https://github.com/epfl-labos/TRIAD
Write Amplification (WA)

\[ WA = \frac{\text{total data written to storage}}{\text{data written by app}} \]
Production Workloads: Throughput

higher is better

- Prod Wkld 1 ~uniform
- Prod Wkld 2 skewed

KOPS:

RocksDB
TRIAD

~uniform skewed
Production Workloads: Throughput

TRIAD: stable throughput across wklds.

higher is better

2x

Prod Wkld 1 ~uniform
Prod Wkld 2 skewed

RocksDB
TRIAD
Production Workloads: Write Amplification

- Prod Wkld 1: ~uniform
- Prod Wkld 2: skewed

lower is better

Write Amplification

- RocksDB
- TRIAD
Production Workloads: Write Amplification

TRIAD: low and uniform WA.

lower is better

Prod Wkld 1
~uniform

Prod Wkld 2
skewed
TRIAD: Throughput Breakdown Synthetic Workloads

Skew 1%-99%

Skew 1%

higher is better
TRIAD: Throughput Breakdown Synthetic Workloads

Higher is better

Complementary techniques

- TRIAD-MEM efficient for skewed workloads.
- TRIAD-LOG efficient for uniform workloads.
More in Our Paper

- More production workloads
- More synthetic workloads
- Detailed breakdown of TRIAD techniques
- TRIAD-DISK
Related work

- LevelDB: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to reduce WA.
  - LSMs: RocksDB, bLSM (SIGMOD/PODS '12), VT-tree (FAST '13), HyperLevelDB, LSM-trie (USENIX ATC '15), Cassandra, WiscKey (FAST '16).

- B-epsilon trees: Tucana (USENIX ATC '16), BetrFS (FAST '15, '16).

- No hot/cold key separation, not use the commit log as a pseudo-SST.

- Hot/cold key separation idea used in different context.

- FLASH storage systems: dual-pool algorithm (SAC '07), Application-Managed Flash (FAST '16).
Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to **reduce WA**.
  - **LSMs**: RocksDB, bLSM (SIGMOD/PODS ’12), VT-tree (FAST ’13), HyperLevelDB, LSM-trie (USENIX ATC ’15), Cassandra, WiscKey (FAST ’16).
Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to reduce WA.
  - **LSMs**: RocksDB, bLSM (SIGMOD/PODS ’12), VT-tree (FAST ’13), HyperLevelDB, LSM-trie (USENIX ATC ’15), Cassandra, WiscKey (FAST ’16).
  - **B-epsilon trees**: Tucana (USENIX ATC ’16), BetrFS (FAST ’15, ’16)
Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to reduce WA.
  - **LSMs**: RocksDB, bLSM (SIGMOD/PODS ’12), VT-tree (FAST ’13), HyperLevelDB, LSM-trie (USENIX ATC ’15), Cassandra, WiscKey (FAST ’16).
  - **B-epsilon trees**: Tucana (USENIX ATC ’16), BetrFS (FAST ’15, ’16)
  - But no hot/cold key separation, not use commit log as a pseudo-SST.
Related work

- **LevelDB**: first LSM-based KV store. No attempts to reduce WA.

- A number of systems attempt to **reduce WA**.
  - **LSMs**: RocksDB, bLSM (SIGMOD/PODS ’12), VT-tree (FAST ’13), HyperLevelDB, LSM-trie (USENIX ATC ’15), Cassandra, WiscKey (FAST ’16).
  - **B-epsilon trees**: Tucana (USENIX ATC ’16), BetrFS (FAST ’15, ’16)
    - But no hot/cold key separation, not use commit log as a pseudo-SST.

- Hot/cold key separation idea used in different context.
  - **FLASH storage systems**: dual-pool algorithm (SAC ’07), Application-Managed Flash (FAST ’16)
Take-home Messages

✓ TRIAD: LSM key-value store with high throughput and low WA.
Take-home Messages

✓ TRIAD: LSM key-value store with high throughput and low WA.

✓ Complementary techniques transparent to workload types.
Take-home Messages

✓ TRIAD: LSM key-value store with high throughput and low WA.

✓ Complementary techniques transparent to workload types.

✓ Impact of LSM I/O on user throughput reduced.