DENKV: Addressing Design Trade-offs of Key-value Stores for Scientific Applications

Safdar Jamil, Awais Khan, Kihyun Kim, Jae-Kook Lee, Dosik An, Taeyoung Hong, Sarp Oral, Youngjae Kim

7th International Parallel Data Systems Workshop (PDSW’22)
DENKV: Deduplication-extended Node-local LSM-tree-based Key-value Store

- HPC applications generate huge amount of redundant data
- Distributed key-value stores gained attention for HPC systems
- A node-local LSM-tree-based key-value store for HPC systems
- Integrate data deduplication to overcome write and space amplification problems
- Introduced asynchronous partly inline deduplication (APID)
  - Leverages background thread pool
- Maintained performance while reducing $4x$ write and $8x$ space amplification
Outline

- Background
  - Distributed Key-Value stores in HPC
  - Log-Structured Merge (LSM) Tree-based KV stores
  - Deduplication 101

- Deduplication in HPC

- Proposed Architecture
  - DENKV: Design goals
  - Write and Read operation flow

- Evaluation

- Conclusion and Q&A
Background
Emerging storage technologies have opened new opportunities for the use of KV stores in HPC
- The use-case includes storing intermediate results
Background

Distributed Key-Value Stores in HPC

- Emerging storage technologies have opened new opportunities for the use of KV stores in HPC
  - The use-case includes storing intermediate results

- A variety of distributed KV stores have been developed.

HPC Application
HPC applications

- Compute and data intensive → Solve complex problems
- Execution time in weeks → Simulate world-class scenarios
- Generate huge amount of data
  - In terms of terabytes to petabytes
  - 4 petabytes of data generated for single image
- High IO bandwidth demand

Photo credit: https://eventhorizontelescope.org/blog/astronomers-reveal-first-image-black-hole-heart-our-galaxy
Log-Structured Merge Tree-based Key-Value Stores

- Log-Structured merge (LSM) tree-based KV stores
  - Highly write-optimized
  - Suitable candidates for node-local NVMe SSDs or burst buffers in HPC environment

Diagram:
- Put Op
- KV Pair
- State Change (MT → IMT → SST)
- Background Thread Pool
- DRAM
- SSD
- Flush/Compact
- Write Ahead Log (WAL)
- L0
- Ln
- MT
- IMT
- SST
Log-Structured Merge Tree-based Key-Value Stores

- Log-Structured merge (LSM) tree-based KV stores
  - Highly write-optimized
  - Suitable candidates for node-local NVMe SSDs or burst buffers in HPC environment

Limitations of LSM-tree
- High write amplification (WA) – more writes than application intended
- High space amplification (SA) – more space utilization than application required
Background

Log-Structured Merge Tree-based Key-Value Stores

- Log-Structured merge (LSM) tree-based KV stores
  - Write and Space amplification problems

![Diagram of KV Pair and State Change (MT → IMT → SST)]

- KV Pair
- State Change (MT → IMT → SST)

![Diagram of DRAM, SSD, Flush/Compact, and SSTs]
- DRAM
- SSD
- Flush/Compact
- SST 00
- SST 01

- Updated key-value pair
- Invalid key-value pair
Background

Log-Structured Merge Tree-based Key-Value Stores

- Log-Structured merge (LSM) tree-based KV stores
  - Write and Space amplification problems

- Unclaimed invalid key-value pairs lead to space amplification

![Diagram showing state change (MT → IMT → SST) and key-value pairs]

- Updated key-value pair
- Invalid key-value pair
Log-Structured Merge Tree-based Key-Value Stores

- Log-Structured merge (LSM) tree-based KV stores
- Write and Space amplification problems

![Diagram of LSM tree-based KV stores]

- DRAM
- SSD
- Write Ahead Log (WAL)
- Flush/Compact
- Merge-sort

State Change (MT → IMT → SST)
Background

Log-Structured Merge Tree-based Key-Value Stores

- Log-Structured merge (LSM) tree-based KV stores
  - Write and Space amplification problems

![Diagram showing the components and state change of Log-Structured Merge Tree-based Key-Value Stores]

- KV Pair
- State Change (MT → IMT → SST)
- Merge-sort
- DRAM
- SSD
- Write Ahead Log (WAL)

<table>
<thead>
<tr>
<th>SST 00</th>
<th>SST 01</th>
<th>SST 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1, 12</td>
<td>K1, 21</td>
<td>K1, 21</td>
</tr>
<tr>
<td>K2, 34</td>
<td>K3, 45</td>
<td>K2, 34</td>
</tr>
<tr>
<td>K3, 56</td>
<td>K7, 23</td>
<td>K3, 45</td>
</tr>
<tr>
<td>K4, 56</td>
<td>K8, 67</td>
<td>K4, 56</td>
</tr>
<tr>
<td>K5, 34</td>
<td>K9, 45</td>
<td>K5, 34</td>
</tr>
<tr>
<td>K6, 56</td>
<td>Ka, 90</td>
<td>K6, 56</td>
</tr>
</tbody>
</table>

13
Background

Log-Structured Merge Tree-based Key-Value Stores

- Log-Structured merge (LSM) tree-based KV stores
  - Write and Space amplification problems

- This Merge-Sort operation lead to high number of internal writes

- Storage optimization technique, data deduplication, can be adopted to reduce WA and SA.
Deduplication in HPC

Deduplication 101

0. User Data
Deduplication in HPC

Deduplication 101

0. User Data
1. Chunking
Deduplication in HPC

Deduplication 101

0. User Data
1. Chunking
2. Fingerprinting
Deduplication in HPC

Deduplication 101

1. Chunking
2. Fingerprinting
3. Duplicate Lookup

0. User Data

Deduplication Management Metadata
Deduplication in HPC

Deduplication 101

0. User Data
1. Chunking
2. Fingerprinting
3. Duplicate Lookup
4. Update Deduplication Metadata
Deduplication in HPC

Classification of Deduplication

☑ Inline Deduplication
   ☐ Performs deduplication during the write process (within critical section)
   ☐ Normally increased write latency
   ☐ Helps improve write endurance problem
   ☐ Immediate improvement of storage

☑ Offline Deduplication
   ☐ Performs deduplication after the write process finishes (outside of critical section)
   ☐ Lowers write latency compared to inline deduplication
   ☐ Requires temporal storage space to acquire the duplicate data
Deduplication in HPC
Deduplication in HPC applications datasets

- Korean Institute of Science and Technology Information (KISTI) host 5th Supercomputer, Nurion
- A petaflop machine ranked 11th in 2018 by Top500
- Peak performance of 25.3 petaflops
- Cray CS500 with 8,305 compute nodes
- 21 Petabytes of Storage
- Lustre File system
Deduplication in HPC applications datasets

- Collected Top 10 applications dataset at Nurion supercomputer[^1]
- Sample of data is collected for only 10 minutes copying
- Implemented in-house deduplication analysis tool
- Analyzed the deduplication ratio
  - Deduplication ratio – amount of data that can be removed

<table>
<thead>
<tr>
<th>Application</th>
<th>Total Size</th>
<th>Dedup. Ratio</th>
<th>Application</th>
<th>Total Size</th>
<th>Dedup. Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abacus</td>
<td>386 GB</td>
<td>41.8 %</td>
<td>CESM</td>
<td>273 GB</td>
<td>25.7 %</td>
</tr>
<tr>
<td>Charmm</td>
<td>382 GB</td>
<td>23.1 %</td>
<td>Gaussian</td>
<td>293 GB</td>
<td>20.4 %</td>
</tr>
<tr>
<td>Lammps</td>
<td>24 GB</td>
<td>42.5 %</td>
<td>MOM</td>
<td>323 GB</td>
<td>53.9 %</td>
</tr>
<tr>
<td><strong>MPAS</strong></td>
<td><strong>197 GB</strong></td>
<td><strong>81.7 %</strong></td>
<td>Siesta</td>
<td>566 GB</td>
<td>52.1 %</td>
</tr>
<tr>
<td>VASP</td>
<td>1 TB</td>
<td>27.3 %</td>
<td>ANSYS</td>
<td>544 GB</td>
<td>23.8 %</td>
</tr>
</tbody>
</table>

[^1]: https://www.ksc.re.kr/eng/resource/nurion
Deduplication in HPC applications datasets

- Collected Top 10 applications dataset at Nurion supercomputer[^]
- Sample of data is collected for only 10 minutes copying
- Implemented in-house deduplication analysis tool
- Analyzed the deduplication ratio

HPC applications generate highly redundant data [SC’12].

<table>
<thead>
<tr>
<th>Application</th>
<th>Size</th>
<th>Deduplication Ratio</th>
<th>Application</th>
<th>Size</th>
<th>Deduplication Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abacus</td>
<td>386 GB</td>
<td>41.8 %</td>
<td>CESM</td>
<td>273 GB</td>
<td>25.7 %</td>
</tr>
<tr>
<td>Charmm</td>
<td>382 GB</td>
<td>23.1 %</td>
<td>Gaussian</td>
<td>293 GB</td>
<td>20.4 %</td>
</tr>
<tr>
<td>Lammps</td>
<td>24 GB</td>
<td>42.5 %</td>
<td>MOM</td>
<td>323 GB</td>
<td>53.9 %</td>
</tr>
<tr>
<td>MPAS</td>
<td>197 GB</td>
<td>81.7 %</td>
<td>Siesta</td>
<td>566 GB</td>
<td>52.1 %</td>
</tr>
<tr>
<td>VASP</td>
<td>1 TB</td>
<td>27.3 %</td>
<td>ANSYS</td>
<td>544 GB</td>
<td>23.8 %</td>
</tr>
</tbody>
</table>

[^]: https://www.ksc.re.kr/eng/resource/nurion
Deduplication in HPC

Deduplication in LSM-tree

- Novel way to minimize WA and SA
- Incorporating value-based deduplication
  - Can help reduce the actual size of KV store
- Adopting deduplication at tradition LSM-tree

- Performance overhead of inline dedup at MemTable
- Breaks structural constraints at SSTables
  (Single instance of valid KV Pairs)
- Complex compaction operation
Deduplication in HPC

Deduplication in LSM-tree

- Adopting deduplication at tradition LSM-tree

Performance overhead of inline dedup at MemTable

- **Performance overhead of inline dedup at MemTable**

<table>
<thead>
<tr>
<th>FP (Value)</th>
<th>Key list</th>
<th>Ref. Count</th>
<th>Chunk Location</th>
<th>Mem. Addr</th>
<th>VPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP(12)</td>
<td>K1</td>
<td>0x00</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP(34)</td>
<td>K2, K5</td>
<td>0x04</td>
<td>0x04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP(56)</td>
<td>K3, K4, K6</td>
<td>0x08</td>
<td>0x08</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance results:

- **YCSB Benchmark** | Workload A: 100% Write | Workload B: 50% write & 50% read

- **RocksDB** vs **Mem_Dedup**

<table>
<thead>
<tr>
<th>Workload A</th>
<th>Workload B</th>
</tr>
</thead>
<tbody>
<tr>
<td>100K</td>
<td>500K</td>
</tr>
<tr>
<td>RocksDB</td>
<td>Mem_Dedup</td>
</tr>
</tbody>
</table>
Deduplication in HPC

Deduplication in LSM-tree

- Adopting deduplication at tradition LSM-tree

Breaks structural constraints at SSTables (Single instance of valid KV Pairs)

Complex compaction operation
Proposed Architecture
Proposed Architecture

DENKV: Deduplication-extended Node-local LSM-tree-based Key-value Store

- Design Goals
  - Maintain performance characteristics of LSM-tree
  - Minimum deduplication overhead for client operations
  - Reduce write and space amplification
  - Maintain the structural constraint of LSM-tree
DENKV: Design Overview

- **KV Pair**
- **State Change (MT → IMT → SST)**
- **Chunk Values**

**Proposed Architecture**

- **Asynchronous**
  - Background thread pool
- **Partly inline**
  - Out of critical section

**DENKV**

- **Background Thread Pool**
- **DRAM SSD**
- **SSD**
- **KV Store**
- **NVMe SSD**

**Writing value chunks in UVL**

**Flush IMemtable to SSTable**

**APID**

**Meta-SSTs**

**Fixed-size Value Chunking**

**SHA1-based FingerPrinting**

**Duplicate Lookups**

**Dedup Metadata (CIT)**
DENKV: Write Operation Flow

Proposed Architecture

KV Pair
State Change
(MT —> IMT —> SST)
Chunk Values

Put Op

MT

Flush IMemtable to SSTable

APID

Background Thread Pool

DRAM

SSD

Meta-SST

Write value chunks in UVL

FP (Value) | Ref. Count | Offset
---|---|---
FP(12) | 1 | 0x00
FP(34) | 2 | 0x04
FP(56) | 3 | 0x08

Fixed-size Value Chunking
SHA1-based Fingerprinting
Duplicate Lookups
Dedup Metadata (CIT)
DENKV: Read Operation Flow

Proposed Architecture

GET Op

MT

IMT

GET Op

SSD

DRAM

KV Pair

State Change

(MT —> IMT —> SST)

Chunk Values

KV Pair

State Change

(MT —> IMT —> SST)

Chunk Values

Meta-SST

Unique Value Log (UVL)

K1, 0x00  K2, 0x04
K3, 0x08  K4, 0x08
K5, 0x04  K6, 0x08

0x00 0x04 0x08 0x12
Proposed Architecture

DENKV: Read Operation Flow

GET Op

IMT

MT

Refer Manuscript
- Garbage Collection
- Crash Consistency of Chunk Information Table

KV Pair
State Change
(MT —> IMT —> SST)

Chunk Values

Meta-SST

Unique Value Log (UVL)

K5, 0x04
K6, 0x08

0x00 0x04 0x08 0x12

12 34 56
Evaluation
Evaluation

System configuration

❑ System Setup

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel(R) Xeon(R) CPU E5-4640 v2 @ 2.20GHz 4 CPU nodes (10 cores per node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>256 GB DDR3 DRAM</td>
</tr>
<tr>
<td>Storage</td>
<td>Samsung SSD 970 EVO 1TB</td>
</tr>
</tbody>
</table>

❑ Benchmark

❑ In-house simulation of dedup patterns of HPC application
❑ Varying value sizes: 4KB and 1MB
❑ Fixed size keys 16 bytes
❑ 1 Million KV pairs for 4KB
❑ 100 thousand KV pairs for 1MB
Evaluation

Compared systems

- RocksDB
  - Vanilla LSM-Tree based KV Store
  - Follows the traditional LSM-Tree structure

- BlobDB
  - KV separation design atop of RocksDB
  - Optimized for write and read operations

- DENKV
  - Our proposed deduplication incorporated KV Store
Evaluation

Questions to be answered

❑ How much deduplication influence the performance in general?

❑ How much write amplification is reduced?

❑ How much space amplification is reduced?

❑ What are the bottlenecks?
Evaluation

Performance analysis

- 4 KB KV Pairs

#### THROUGHPUT (KIOPS)

<table>
<thead>
<tr>
<th>DEDUP RATIO</th>
<th>RocksDB</th>
<th>BlobDB</th>
<th>DENKV</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>50</td>
<td>55</td>
<td>52</td>
</tr>
<tr>
<td>30%</td>
<td>51</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>60%</td>
<td>51</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>90%</td>
<td>51</td>
<td>53</td>
<td>53</td>
</tr>
</tbody>
</table>
Evaluation

Performance analysis

- 4 KB KV Pairs

![Graph showing performance comparison between RocksDB, BlobDB, and DENKV with throughput and dedup ratio.]

- Performance drops due to extra deduplication steps
- With increasing dedup ratio, performance improves
Evaluation

Write and space amplification analysis

- 4 KB KV Pairs
Evaluation

Write and space amplification analysis

- 4 KB KV Pairs
  - 4x WA reduced with small KV pairs
  - 4.6x SA reduced with small KV pairs
Performance analysis

1 MB KV Pairs

![Graph showing throughput vs. dedup ratio for RocksDB, BlobDB, and DenKV.]

- RocksDB
- BlobDB
- DenKV

Throughput (KIOPS) vs. Dedup Ratio

- 0% Dedup Ratio
- 30% Dedup Ratio
- 60% Dedup Ratio
- 90% Dedup Ratio
Performance analysis

- 1 MB KV Pairs

- Performance drops due to extra deduplication steps
- Outperforms all with highest dedup ratio
Evaluation

Write and space amplification analysis

- 1 MB KV Pairs

![Write Amplification Graph](image1)

![Space Amplification Graph](image2)
Write and space amplification analysis

- 1 MB KV Pairs

- 8x WA reduced with small KV pairs
- 8.9x SA reduced with large KV pairs
Evaluation

Questions to be answered

❑ How much deduplication influence the performance in general?
  - With small keys, performance is comparable.
  - There is a performance drop with large KV pairs.

❑ How much write amplification is reduced?
  - With 50% deduplication ratio, around 43% write amplification is reduced on average

❑ How much space amplification is reduced?
  - With 50% deduplication ratio, on average 45% less amount of space is utilized

❑ What are the bottlenecks?
  - Deduplication operation interfere the foreground IOs results in write stalls.
Conclusion
Conclusion

- HPC applications generate significant amount of redundant data

- Distributed KV stores are gaining significant attention in HPC
  - Distributed KV stores rely on monolithic KV stores
  - LSM-tree-based KV stores suffer from high WA and SA

- DENKV introduced APID (asynchronous partly inline deduplication) module
  - Reduces WA and SA while maintaining the performance
Thank you

safdar@sogang.ac.kr
https://sites.google.com/view/safdarjamil95