Differentiated Key-Value Storage Management for Balanced I/O Performance

Yongkun Li[1], **Zhen Liu[1]**, Patrick P. C. Lee[2], Jiayu Wu[1], Yinlong Xu[1,3]
Yi Wu[4], Liu Tang[4], Qi Liu[4], Qiu Cui[4]

[1]University of Science and Technology of China [2]The Chinese University of Hong Kong
[3]Anhui Province Key Laboratory of High Performance Computing, USTC [4]PingCAP

USENIX ATC 2021

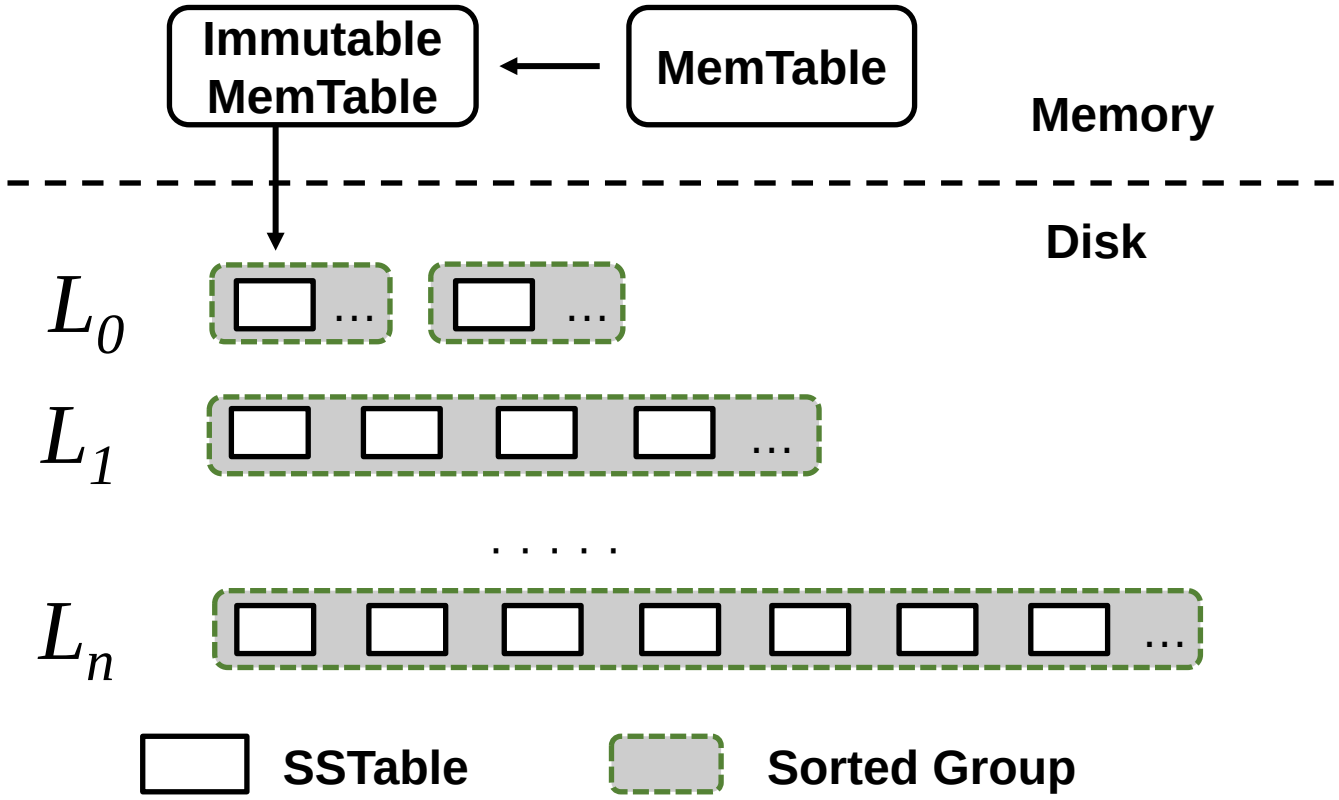
Background

- Real-world workloads are diverse and mixed
  - Value size varies in a large range
  - Writes, reads, and scans are common

- **Log-structured merge (LSM) tree**
  - Transform random writes into sequential writes
  - Support efficient reads and range scans
  - **Limitation:** high write amplifications

LSM-tree: Basics

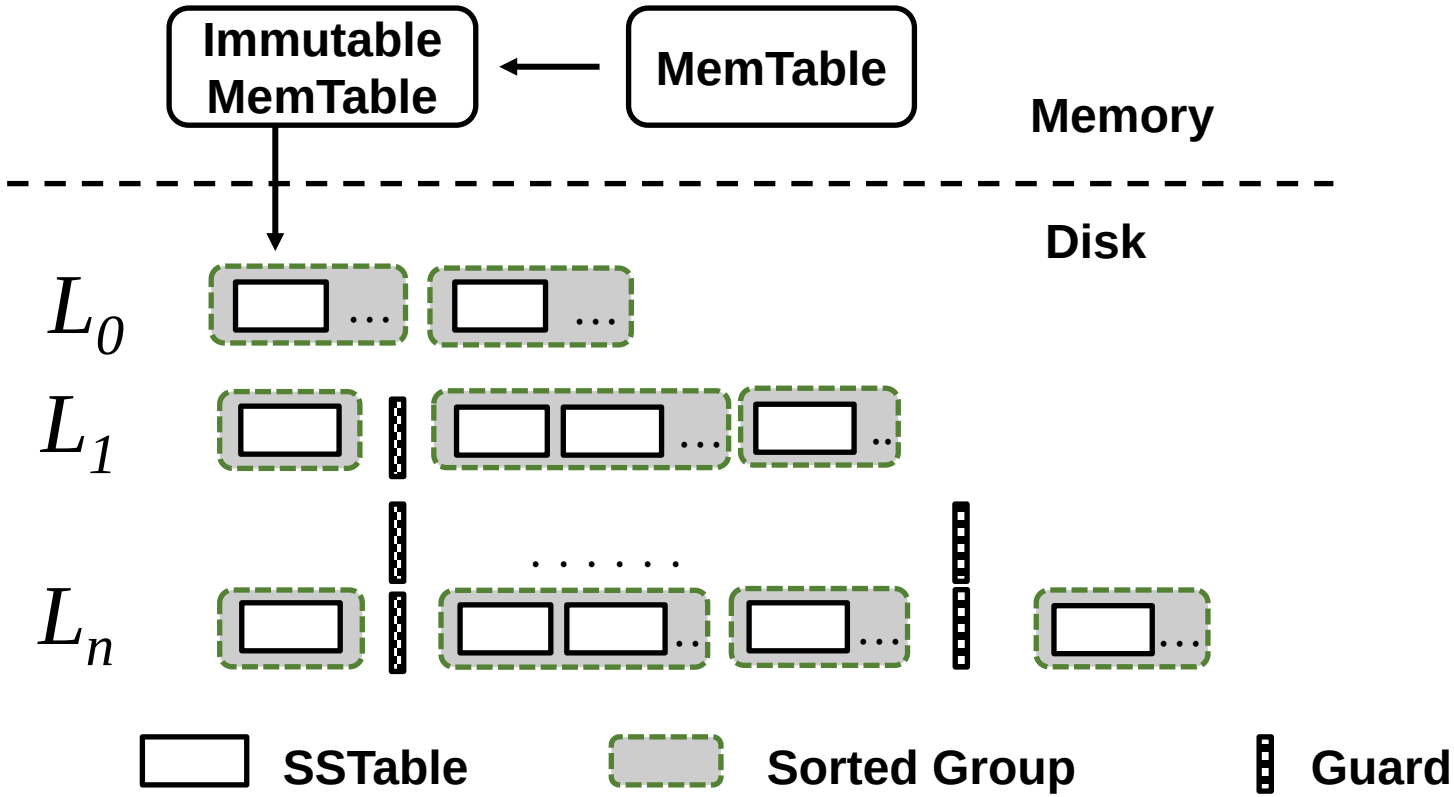
- Store keys and values together
  - Keys and values are fully sorted in each level
  - Compaction across levels → high I/O amplifications



Relaxing Fully-Sorted Ordering

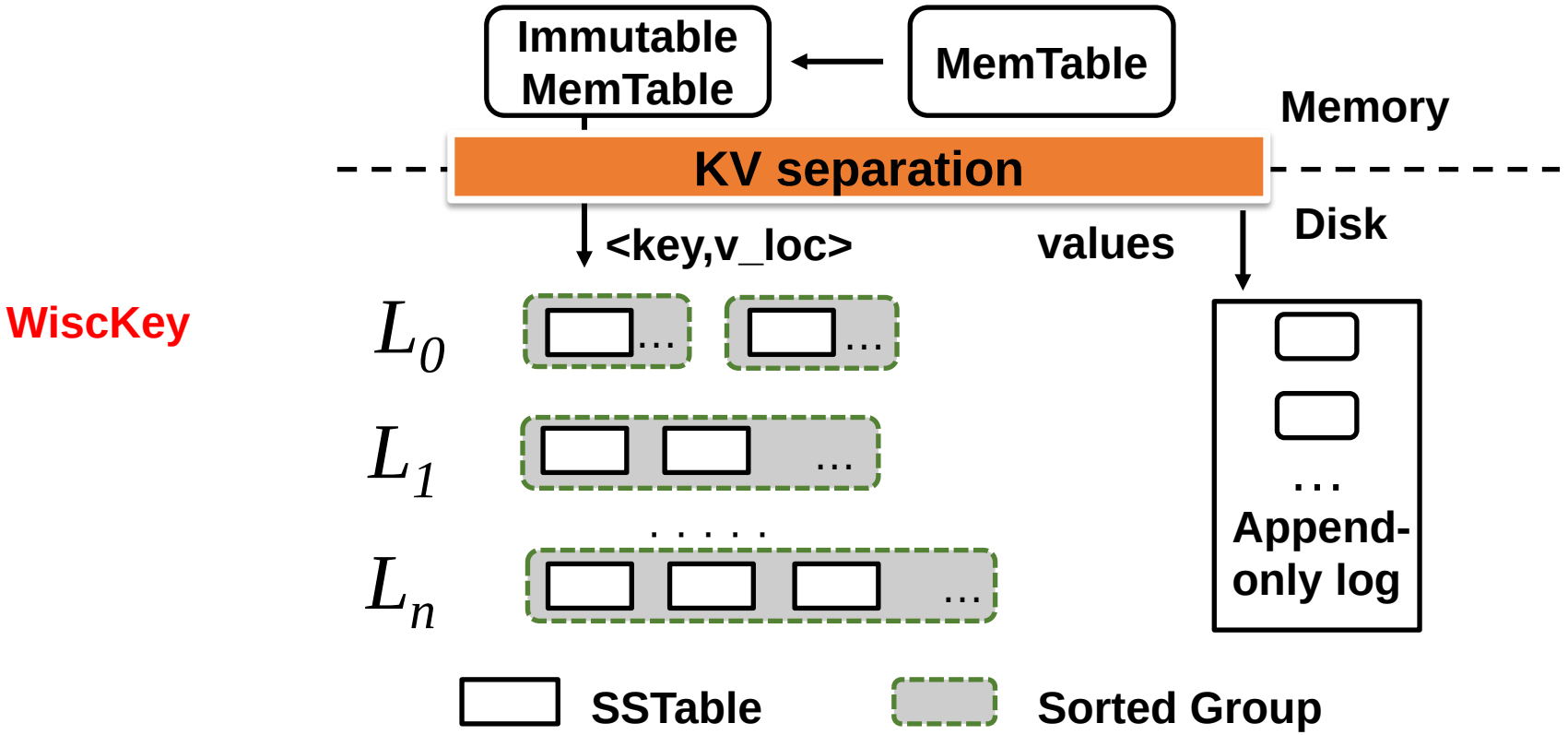
- Each level is not necessarily fully sorted by keys
  - e.g., PebblesDB [SOSP'17], Dostoevsky [SIGMOD'18], etc.
  - Support efficient writes, but sacrifice reads and scans

Fragmented LSM-tree in PebblesDB



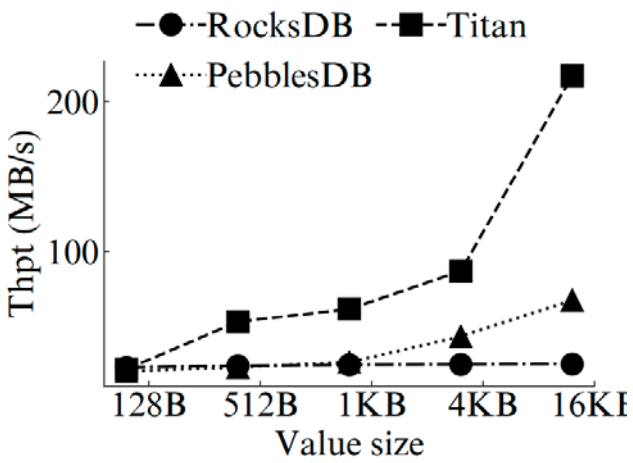
KV Separation

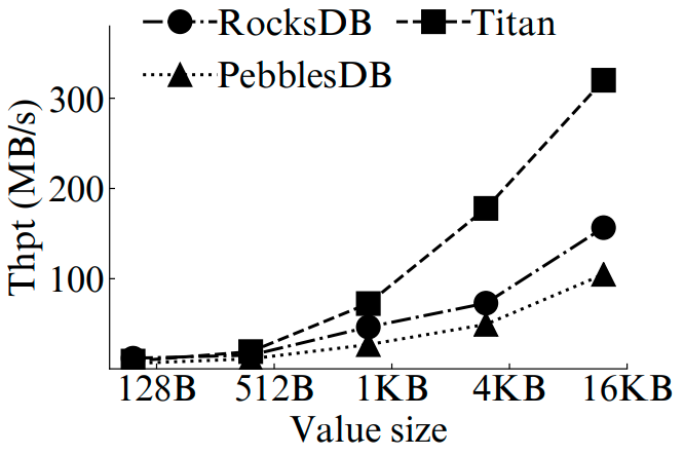
- Store keys and values separately
  - e.g., WiscKey, HashKV, Titan, Bourbon, etc.
  - Support efficient writes and reads, but have **poor** scan performance

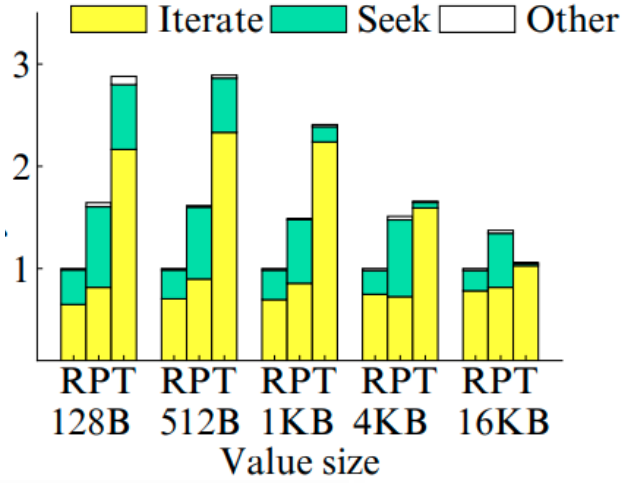


Trade-off Analysis

- Are the optimizations suitable for all conditions?
  - Relax fully-sorted ordering
    - Efficient in small-to-medium values
  - KV separation
    - Suitable for large values







Trade-offs between reads/writes and scans

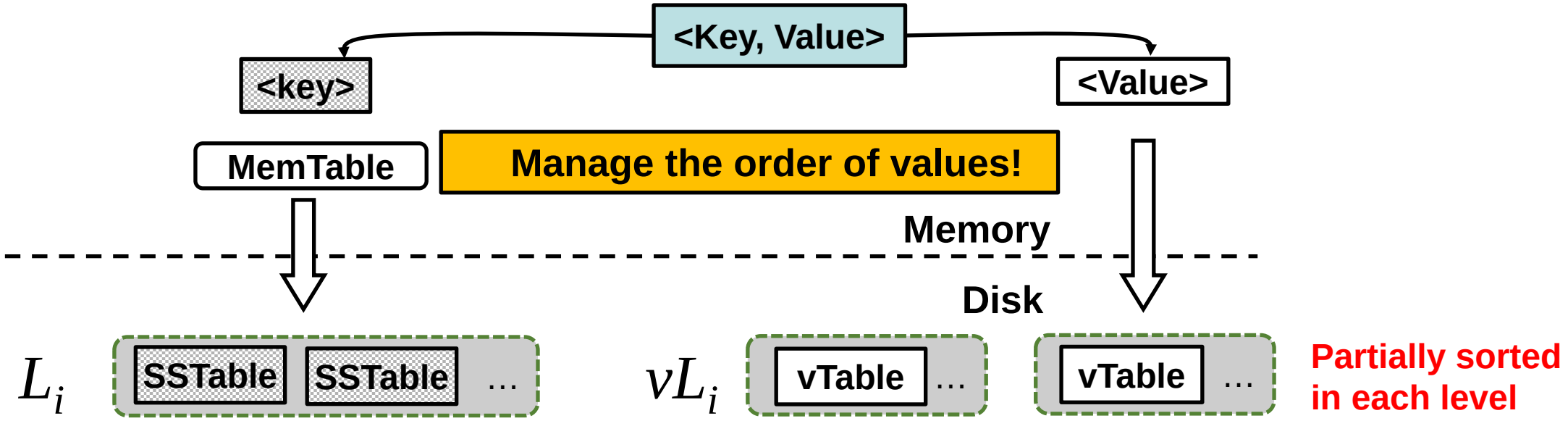
Our Contributions

- **DiffKV**, a KV store realizing balanced I/O performance via **differentiated KV management**
  - Coordinate differentiated management of ordering for keys and values
  - Manage values with partially-sorted ordering
- Merge optimization techniques
- Fine-grained KV separation
  - Differentiate **small, medium, and large** KV pairs for mixed workloads
- Implementation atop PingCAP Titan[*] and extensive evaluation

[*] https://github.com/tikv/titan

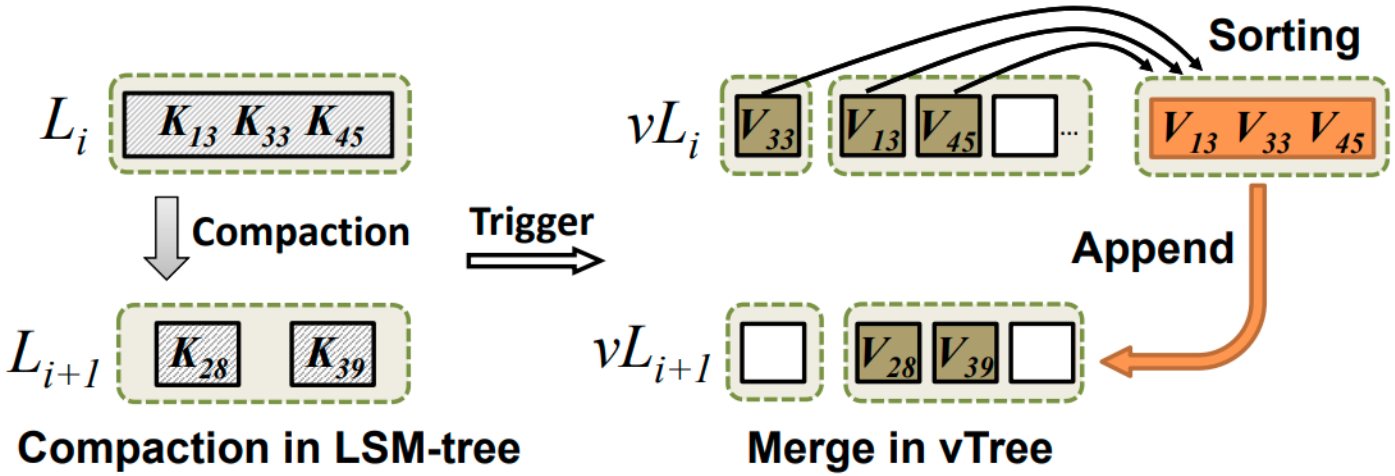
Differentiated KV Management

- Decouple keys and values during flushing
  - **vTree**: a multiple-level tree; each level has multiple sorted groups
  - Each sorted group is a collection of **vTables**
  - Values in a level are *not* fully sorted and have overlapped key ranges



Differentiated KV Management

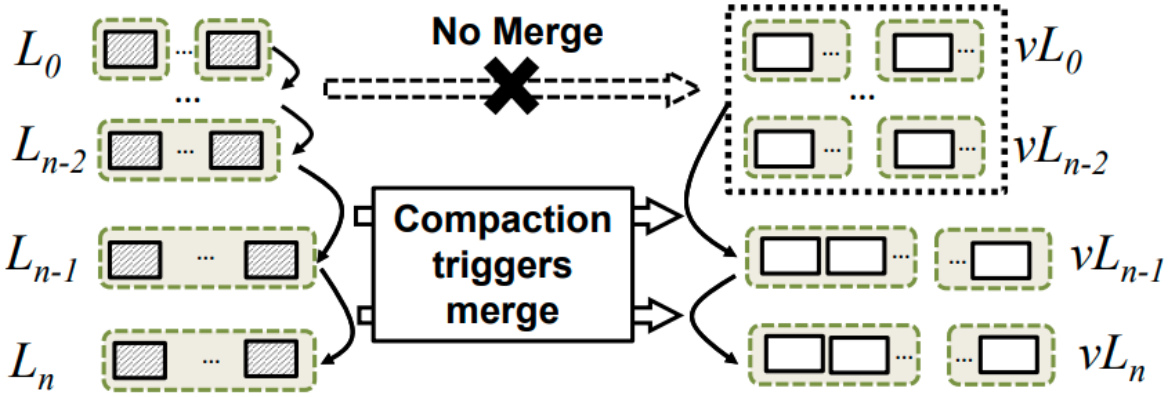
- Compaction-triggered merge
  - Involve values whose keys participate in compaction
  - Be triggered when compaction happens in LSM-tree
  - Reorganize all compaction-related values in one level, and then **append** them to next level



Overhead of updating LSM-tree can be hidden!

O1: Lazy Merge

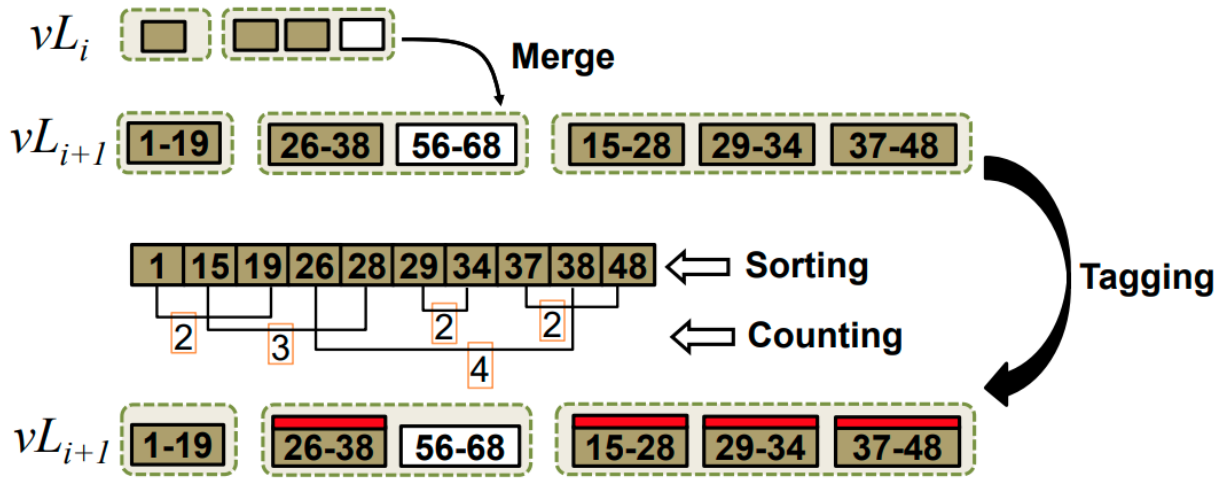
- Problem: frequent merge operations
  - Each compaction triggers a merge operation
- Idea:
  - Values are **delayed to merge** until the target level is one of the last two levels



- Lazy merge significantly reduces number of merge operations

O2: Scan-optimized Merge

- Problem: too many sorted groups within one level
  - Apply append-only merge policy
- Idea:
  - Detect number of overlapping vTables after normal merge
  - Add a tag to indicate participation in the next merge
- Carefully adjust the degree of ordering for values in vTree



Fine-grained KV Separation

- KV separation is advantageous for large KV pairs, but has marginal benefits for small KV pairs
- Selective approach:
  - Small values: stored entirely in LSM-tree
  - Medium values: stored in vTree
  - Large values: stored in vLogs
- Hotness-awareness
  - Hot-cold separation scheme
  - Greedy garbage collection

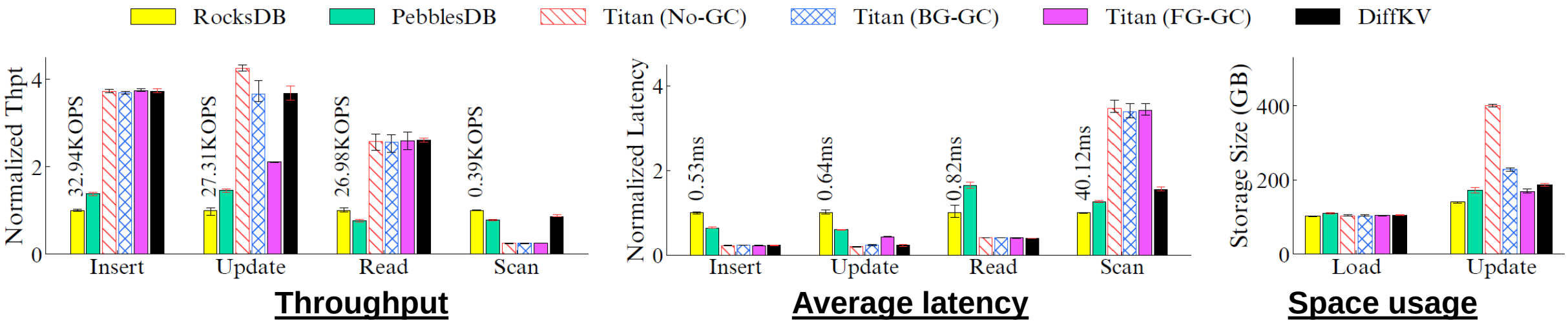
Experiments

- Testbed backed with a Samsung 860 EVO 480 GB SSD
- KV stores
  - RocksDB, PebblesDB, Titan (no GC, background GC, foreground GC)
  - DiffKV: built on Titan to reuse KV separation
- Workloads
  - Key size: 24 bytes
  - Value size: average 1KB (follow Pareto distribution)

(i) insert 10 GB KV pairs (ii) update 300 GB KV pairs

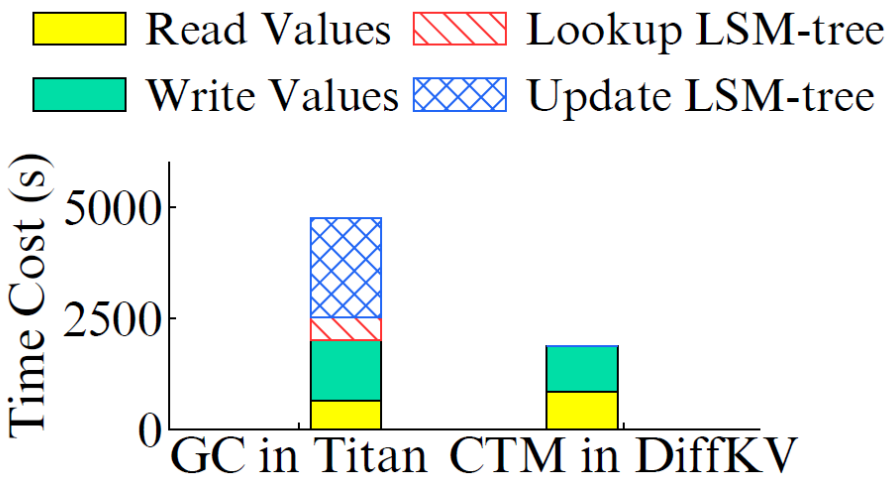
(iii) read 10 GB KV pairs (iv) scan 10 GB KV pairs

Microbenchmarks of DiffKV

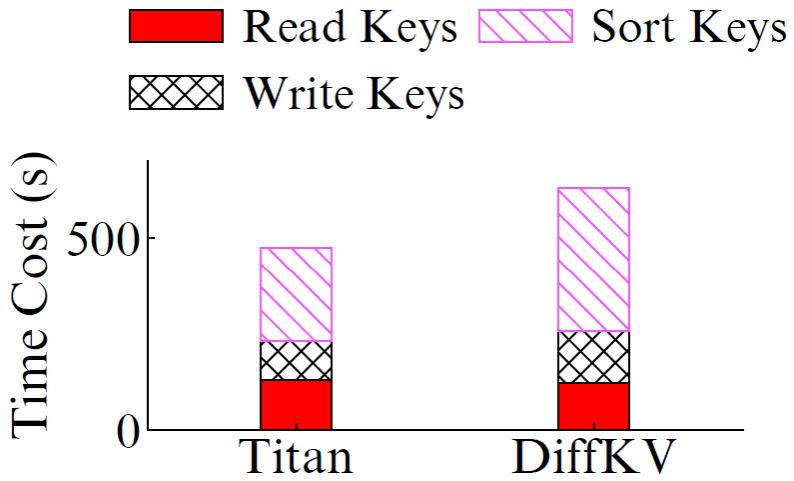


- Compared to RocksDB and PebblesDB
  - 2.7-3.8x inserts; 2.3-3.7x updates; 2.6-3.4x reads
  - Comparable scan performance
- Compared to Titan
  - 3.2x scans; up to 1.7x updates; 43.2% lower scan latency
- DiffKV has acceptable space usage

Impact of Merge Optimizations



Value GC/merge overhead



Key compaction overhead

- Coordinated merge design
  - Reduce 60.7% of time cost of value management
  - Slightly increase key compaction overhead

Conclusions

- **DiffKV**: differentiated key-value storage management for balanced I/O performance
- More evaluation results and analysis in paper
- Source code: https://github.com/ustcadsl/diffkv

Thanks for our attention!

For any questions, please feel free to contact

Prof. Yongkun Li@USTC

ykli@ustc.edu.cn

http://staff.ustc.edu.cn/~ykli/