# A Light-weight Compaction Tree to Reduce I/O Amplification toward Efficient Key-Value Stores

Ting Yao[1], Jiguang Wan[1], Ping Huang[2], Xubin He[2], Qingxin Gui[1], Fei Wu[1], and Changsheng Xie[1]

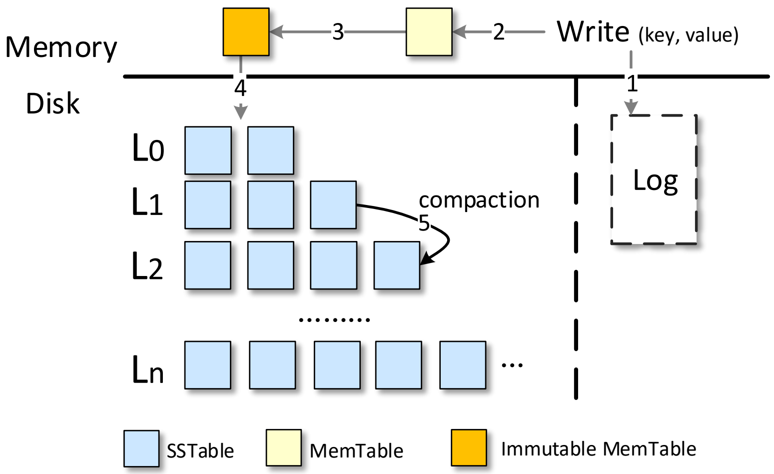
[1]Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
[2]Temple University

# Outline

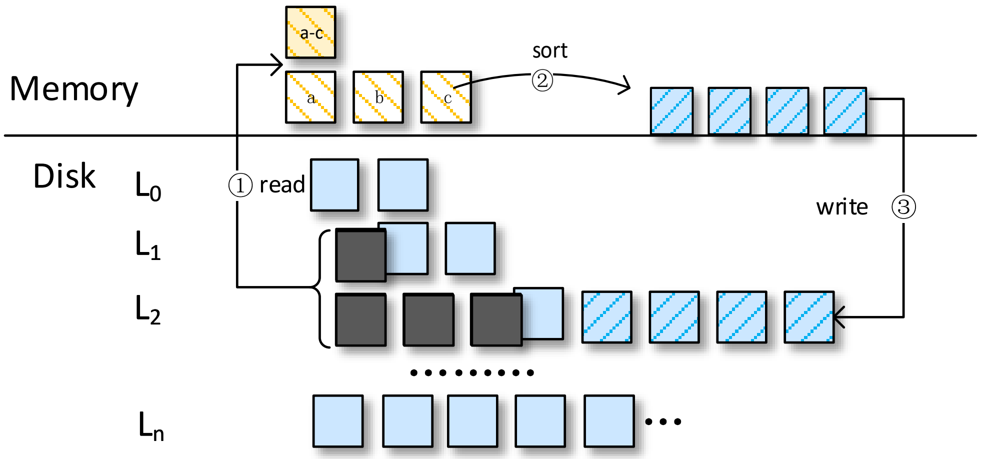
- **Background**
- LWC-tree (Light-Weight Compaction tree)
- LWC-store on SMR drives
- Experiment Result
- Summary

# Background

- Key-value stores are widespread in modern data centers.
  - Better service quality
  - Responsive user experience
- The log-structured merge tree (LSM-tree) is widely deployed.
  - RocksDB, Cassandra, Hbase, PNUTs, and **LevelDB**
  - High throughput write
  - Fast read performance

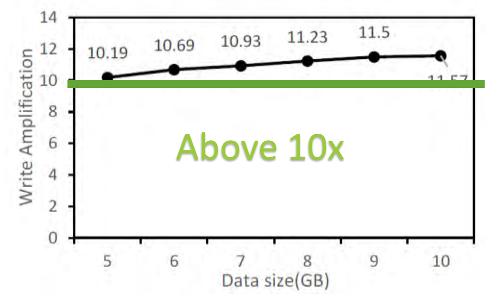


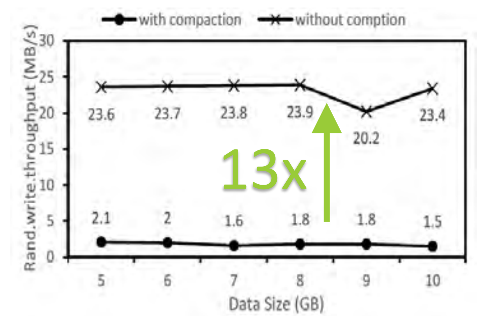
# Background · LSM-tree




The overall read and write data size for a compaction:
8 tables

- **This serious I/O amplifications of compactions motivate our design !**



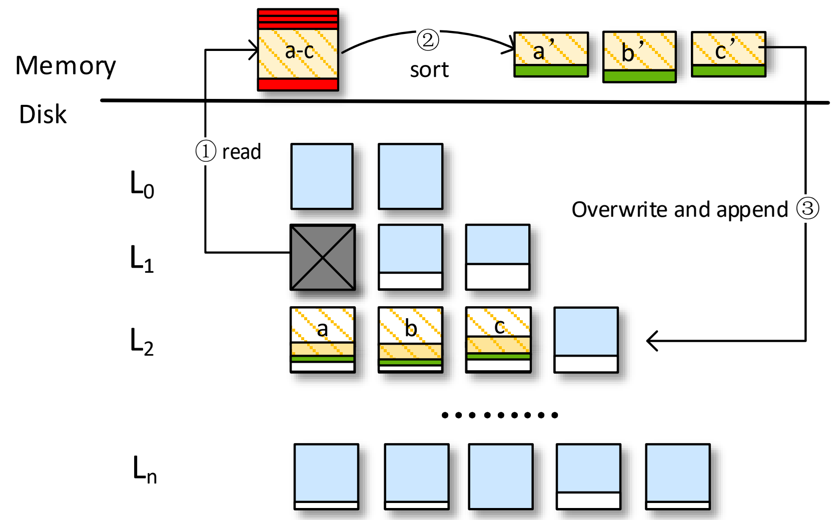
- Background
- **LWC-tree** (Light-Weight Compaction tree)
- LWC-store on SMR drives
- Experiment Result
- Summary

# LWC-tree

- Aim
  - Alleviate the I/O amplification
  - Achieve high write throughput
  - No sacrifice to read performance
- How
  - Keep the basic component of LSM-tree
  - Keep tables sorted
  - Keep the multilevel structure
  - **Light-weight compaction** – reduce I/O amplification
  - **Metadata aggregation** – reduce random read in a compaction
  - **New table structure, DTable** – improve lookup efficient within a table
  - **Workload balance** – keep the balance of LWC-tree

# LWC-tree · Light-weight compaction

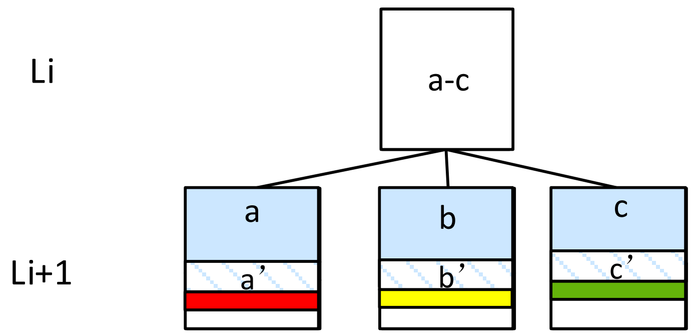
- Aim
  - Reduce I/O amplification
- How
  - append the data and only merge the metadata
    - Read the victim table
    - Sort and divide the data, merge the metadata
    - Overwrite and append the segment
- Reduce 10 x amplification theoretically (AF=10)




The overall read and write data size for a light-weight compaction: **2 tables.** (In LSM-tree, the overall data size for a conventional compaction: **8 tables.**)

# LWC-tree · Metadata Aggregation

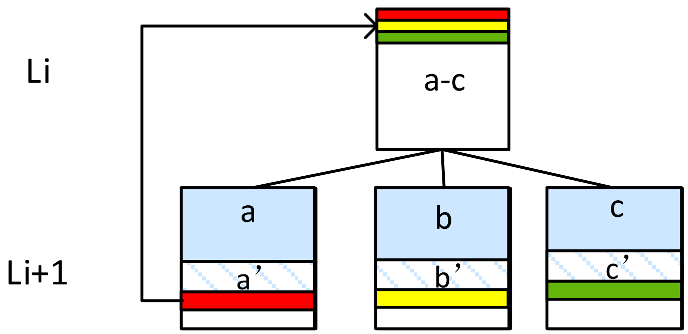
- Aim
  - Reduce random read in a compaction
  - Efficiently obtain the metadata form overlapped Dtables
- How
  - Cluster the metadata of overlapped DTables to its corresponding victim Dtable after each compaction




A light-weight compaction

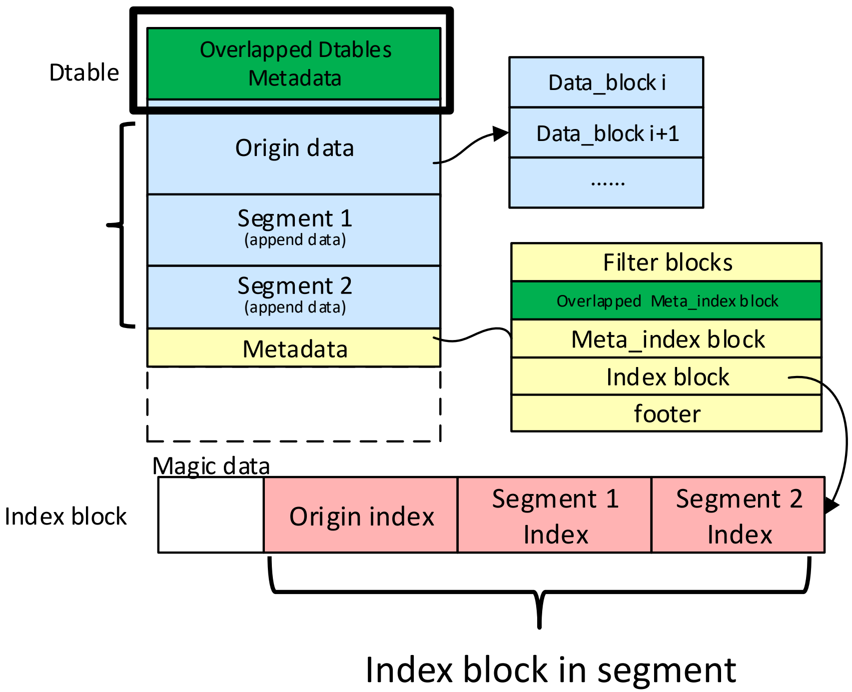
# LWC-tree · Metadata Aggregation

- Aim
  - Reduce random read in a compaction
  - Efficiently obtain the metadata form overlapped Dtables
- How
  - Cluster the metadata of overlapped DTables to its corresponding victim Dtable after each compaction




Metadata aggregation
after light-weight compaction

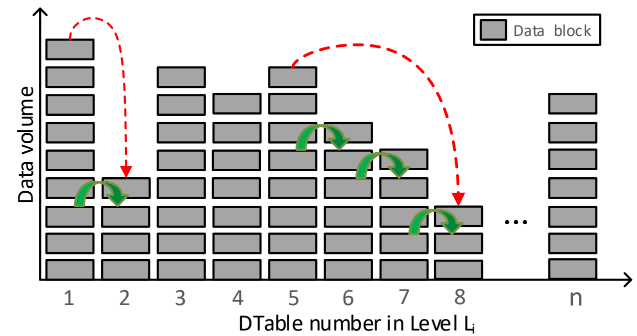
# LWC-Tree · DTable

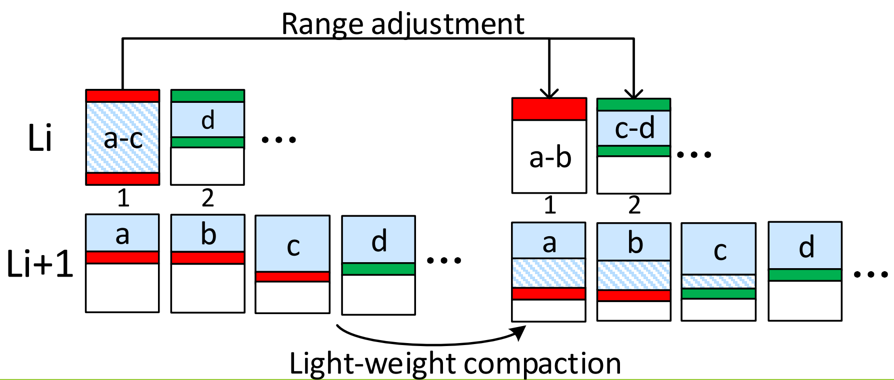



- Aim
  - Support Light-weight compaction
  - Keep the lookup efficiency within a DTable
- How
  - Store the metadata of its corresponding overlapped Dtables
  - Manage the data and block index in segment

# LWC-Tree · Workload Balance

- Aim
  - Keep the balance of LWC-tree
  - Improve the operation efficiency
- How
  - Deliver the key range of the overly-full table to its siblings after light-weight compaction
- Advantage
  - no data movement and no extra overhead



- Background
- LWC-tree (Light-Weight Compaction tree)
- **LWC-store on SMR drives**
- Experiment Result
- Summary

# LevelDB on SMR Drives

- SMR(Shingled Magnetic Recording)
  - Overlapped tracks
  - Band & Guard Regions
  - Random write constrain
- LevelDB on SMR
  - Multiplicative I/O amplification

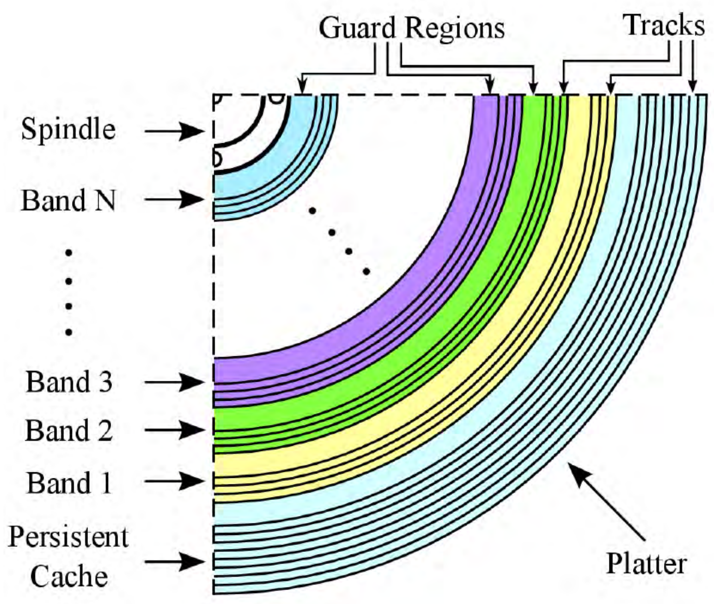
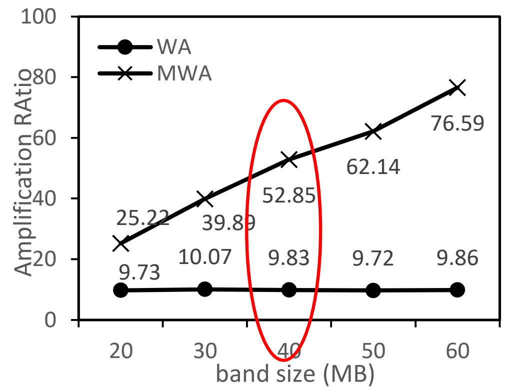



Figure from Fast 2015 “Skylight – A Window on Shingled Disk Operation”

# LevelDB on SMR Drives

- SMR(Shingled Magnetic Recording)
  - Overlapped tracks
  - Band & Guard Regions
  - Random write constrain
- LevelDB on SMR
  - Multiplicative I/O amplification

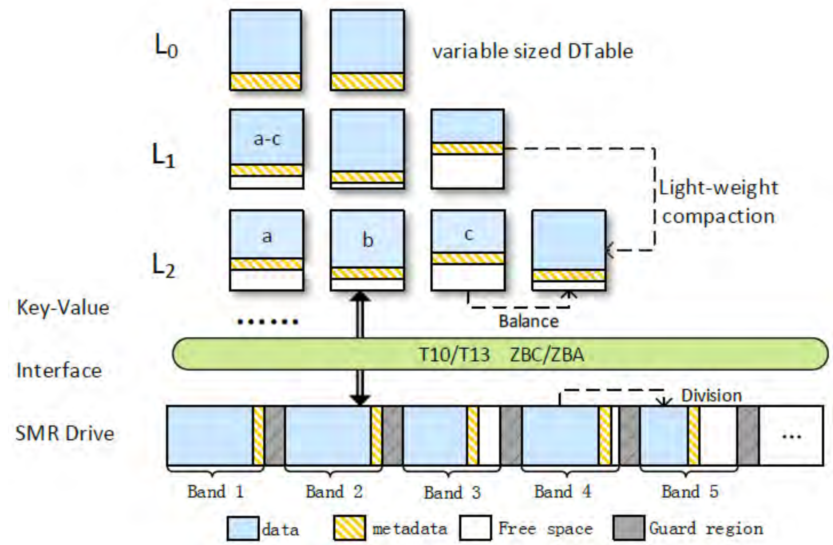



| Band size | 40 MB |
|---|---|
| WA(Write amplification of LevelDB) | 9.83x |
| AWA (Auxiliary write amplification of SMR) | 5.39x |
| MWA (Multiplicative write amplification of LevelDB on SMR ) | 52.58x |

- **This auxiliary I/O amplifications of SMR motivate our implementation!**

# LWC-store on SMR drive

- Aim
  - Eliminate the auxiliary I/O amplification of SMR
  - Improve the overall performance
- How
  - A DTable is mapped to a band in SMR drive
  - Segment appends to the band and overlaps the out-of-date metadata
  - Equal division: Divide the DTable overflows a band into several sub-tables in the same level



- Background
- LWC-tree (Light-Weight Compaction tree)
- LWC-store on SMR drives
- **Experiment Result**
- Summary

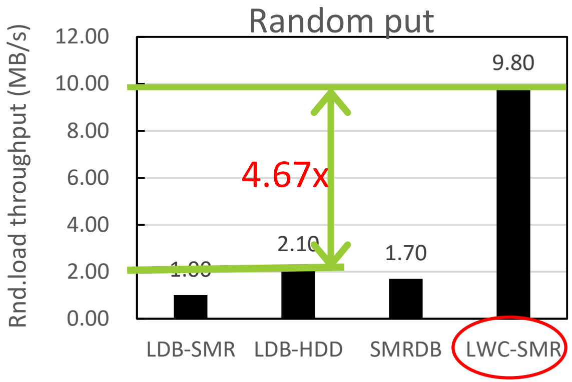
# Configuration

| Experiment Perimeter | |
|---|---|
| Test Machine | 16 Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz processors |
| SMR Drive | Seagate ST5000AS0011 |
| CMR HDD | Seagate ST1000DM003 |
| SSD | Intel P3700 |

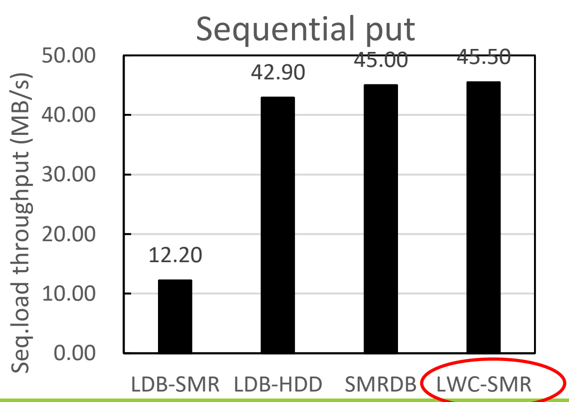
1. LevelDB on HDDs (LDB-hdd)
2. LevelDB on SMR drives (LDB-smr)
3. SMRDB*
   - An SMR drive optimized key-value store
   - reduce the LSM-tree levels to only two levels (i.e., L0 and L1)
   - the key range of the tables at the same level overlapped
   - match the SSTable size with the band size
4. LWC-store on SMR drives (LWC-smr)

# Experiment · Load (100GB data)

- Random load
  - Large amount of compactions
  - LWC-store 9.80x better than LDB-SMR
  - LWC-store 4.67x better than LDB-HDD
  - LWC-store 5.76x better than SMRDB

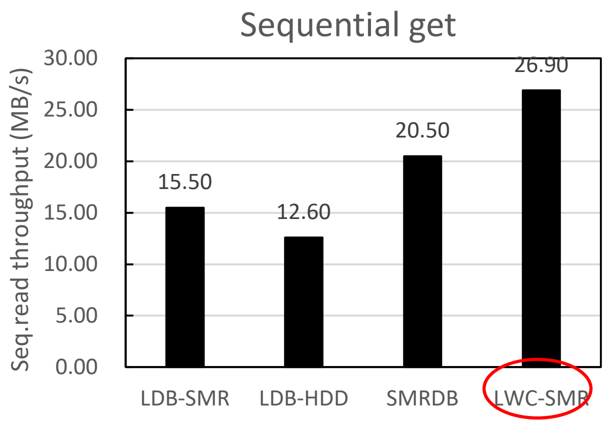


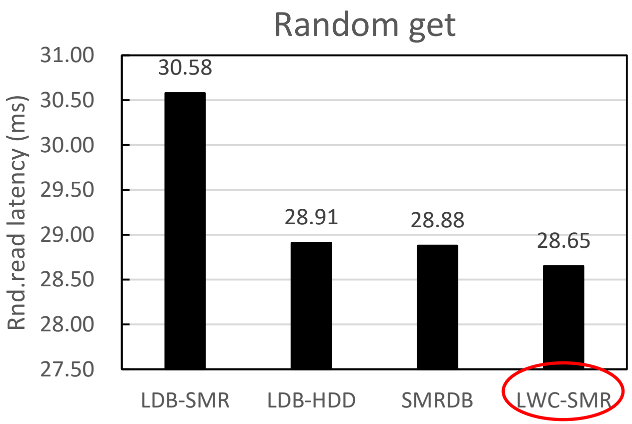

- Sequential load
  - No compaction
  - Similar Sequential load performance



# Experiment · read (100K entries)

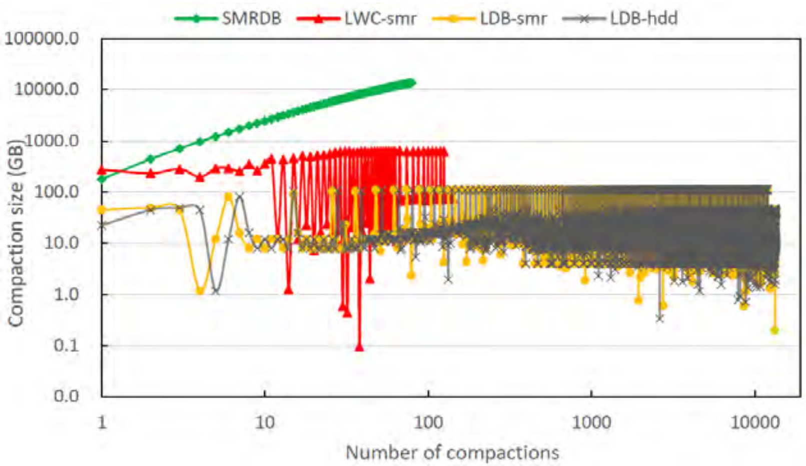
- Look-up 100K KV entries against a 100GB random load database



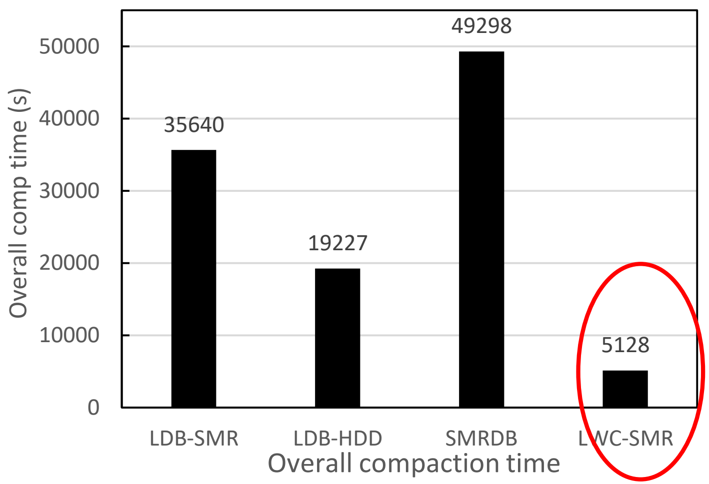
# Experiment · compaction (randomly load 40GB data)

- Compaction performance in microscope
  - LevelDB: number of compactions is large
  - SMRDB: data size of each compaction is large
  - LWC-tree: small number of compactions and small data size
- Overall compaction time
  - LWC-smr gets the highest efficiency



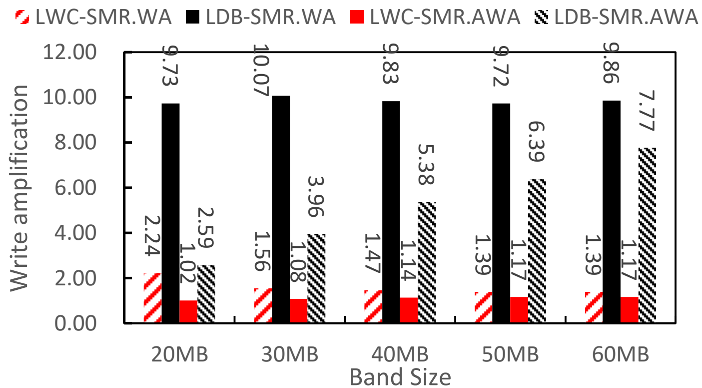
# Experiment · compaction (randomly load 40GB data)

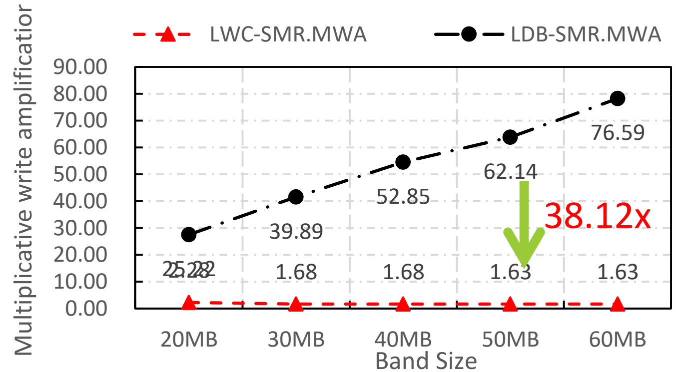
- Compaction performance in microscope
  - LevelDB: number of compactions is large
  - SMRDB: data size of each compaction is large
  - LWC-tree: small number of compactions and small data size
- Overall compaction time
  - LWC-smr gets the highest efficiency



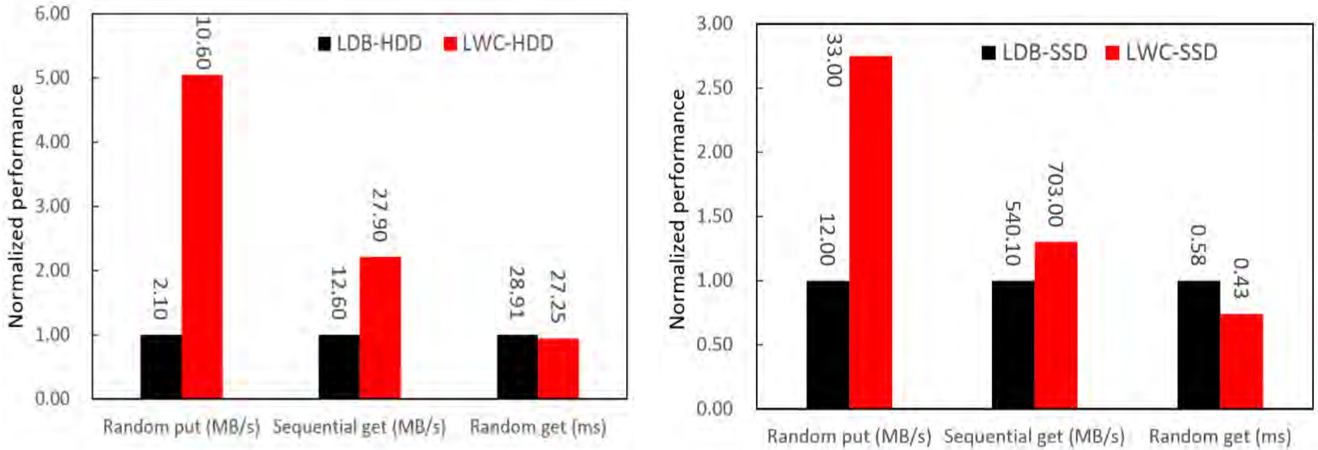
# Experiment · Write amplification

- **Competitors**
  - LWC-SMR
  - LDB-SMR
- **Write amplification (WA)**
  - Write amplification of KV-store
- **Auxiliary write amplification (ARA)**
  - Auxiliary write amplification of SMR
- **multiplicative write amplification (MWA)**
  - Multiplicative write amplification of KV stores on SMR



# Experiment · LWC-store on HDD and SSD




(a) HDD

(b) SSD

- Background
- LWC-tree (Light-Weight Compaction tree)
- LWC-store on SMR drives
- Experiment Result
- **Summary**

# Summary

- LWC-tree: A variant of LSM-tree
  - Light-weight compaction – Significantly reduce the I/O amplification of compaction
- LWC-store on SMR drive
  - Data management in SMR drive – eliminate the auxiliary I/O amplification from SMR drive
- Experiment result
  - high compaction efficiency
  - high write efficiency
  - Fast read performance same as LSM-tree
  - Wide applicability

# Thank you!

QUESTIONS?

Email: tingyao@hust.edu.cn