DEPART: Replica Decoupling for Distributed Key-Value Storage

Qiang Zhang and Yongkun Li, University of Science and Technology of China; Patrick P. C. Lee, The Chinese University of Hong Kong; Yinlong Xu, Anhui Province Key Laboratory of High Performance Computing, University of Science and Technology of China; Si Wu, University of Science and Technology of China

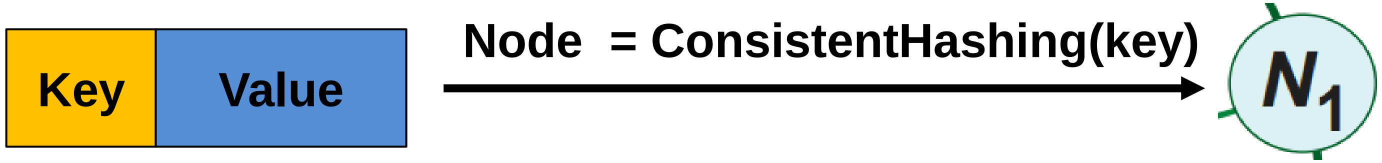
Background

➢ Distributed Storage Server

- Multiple storage nodes
- Consistent hash ring

➢ Client behavior

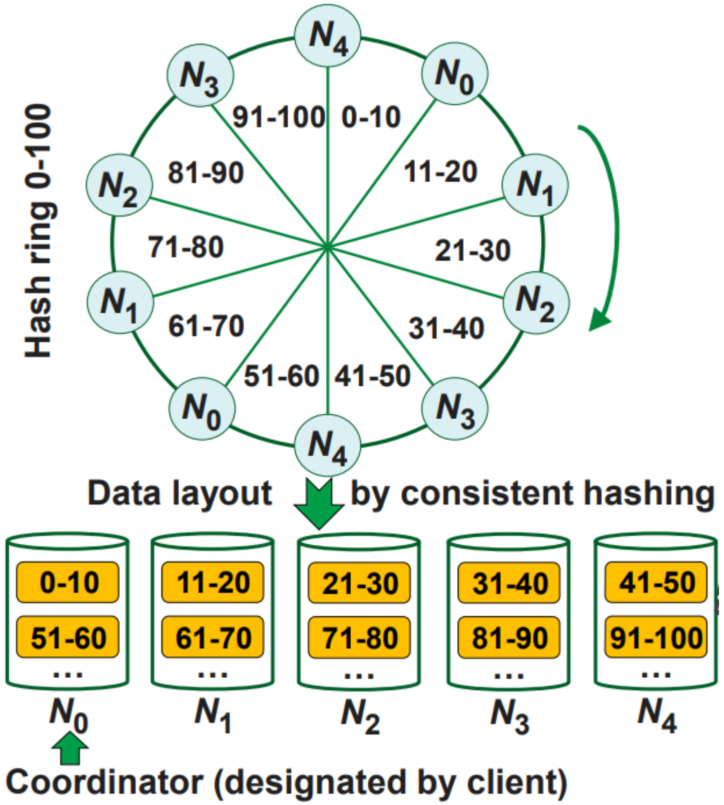
- Calculate consistent hashes and find nodes



Key Value

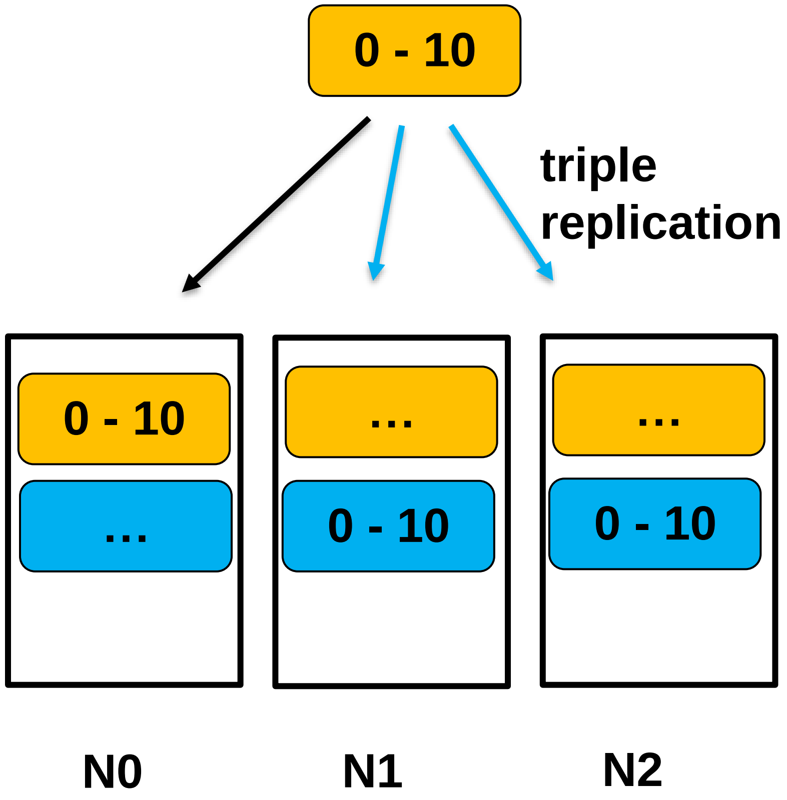
Node = ConsistentHashing(key)

N_1



Background

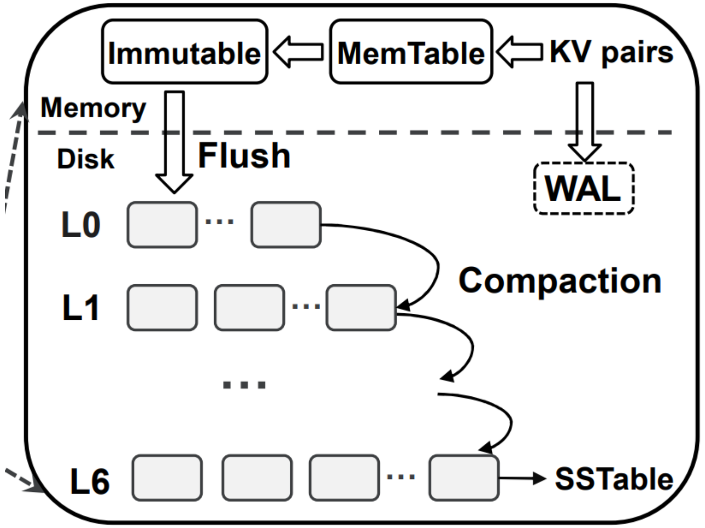
- **Fault tolerance & Reliability**
  - Large systems have the potential for error
  - Data loss or inconsistency
- **Solution: Replication**
  - Each KV-pair is stored in multiple nodes
  - Two kinds of data: primary and redundant



Background

➢ In-node Storage structure: LSM-Tree

- MEM(Tree) + Disk
- Multi-Level tree (KV-pairs is **sorted** by key in each level)
- The capacity of each level is limited;
  the higher the level, the bigger the capacity
- Write: Mem table → Immutable
  -> Level ->**Compaction**(if full)
  Write amplification
- Read: Query top level to bottom level
  Read amplification



Internal storage structure (LSM-tree)

The larger the LSM-tree, the more serious the read/write amplification

Problem

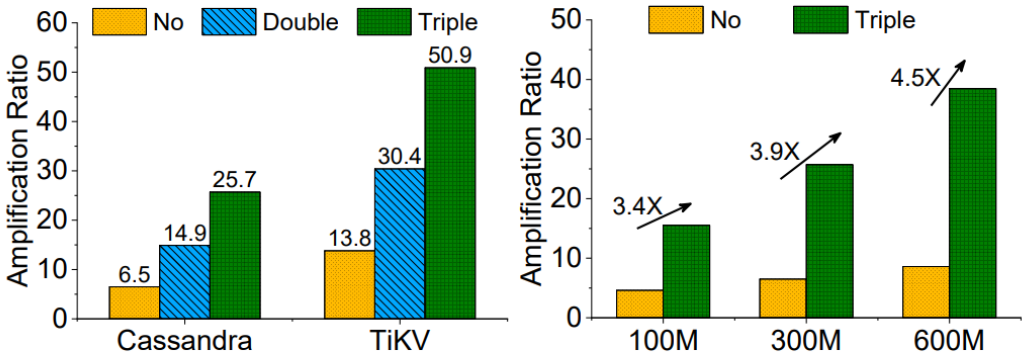
- **Uniform indexing**

Primary data and redundant data are stored in the same LSM tree(Cassandra, TiKV, etc…)

- **Problem**

Uniform indexing cause serious read amplification and write amplification

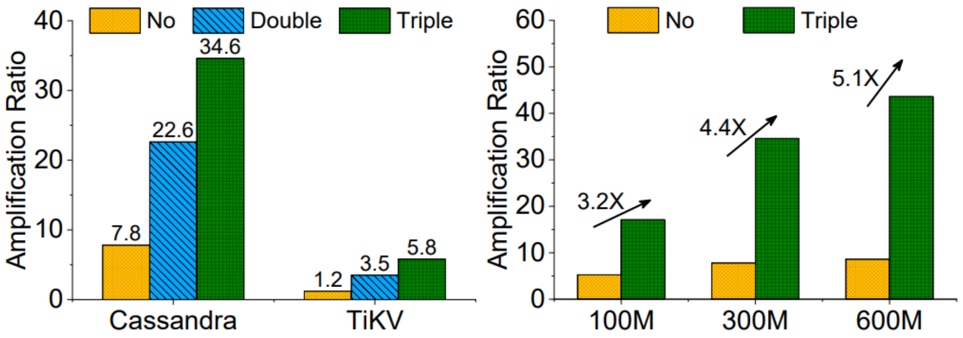
Verification



(a) Impact of replication in Cassandra and TiKV

(b) Impact of KV store size in Cassandra

Write 300 M KV-pairs



(a) Impact of replication in Cassandra and TiKV

(b) Impact of KV store size in Cassandra

Read from 300 M

Increase degree of replication

Increase KV store size

Increase LSM-tree size

read/write amplification

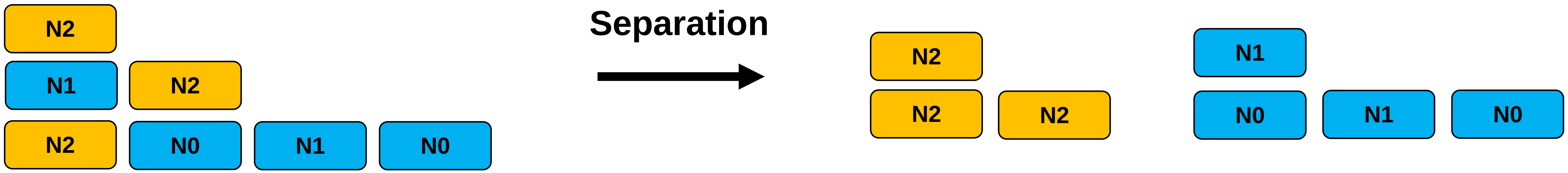
Basic Idea

- **Idea**
  - **Separate** primary and redundant data to reduce LSM-tree size

Plain Design

➢ Multiple LSM trees

- Separate uniform tree into primary data tree and redundant tree



N2

N1 N2

N2 N0 N1 N0

Separation

N2

N2 N2

N1

N0 N1 N0

Uniform LSM-Tree

Primary LSM-Tree

Redundant LSM-Tree(can be further split)

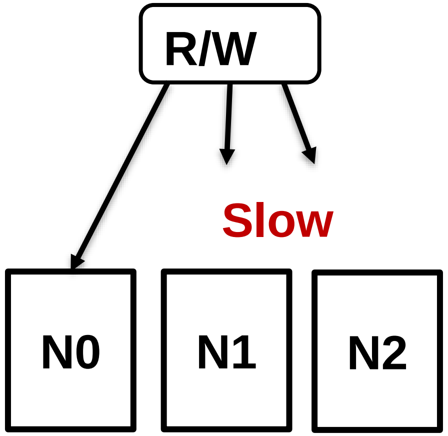
➢ Evaluation

- Advantage : Reduce read/write amplification
- Problem: More memory overhead(Same memory usage for each tree)

Idea

➢ Consistency

- Read/Write consistency level (RCL/WCL)
  - The minimum number of successful reads/writes that need to be confirmed for a successful operation
- Consistency level is a Balance on Performance and Reliability
- Quantified consistency Level : WCL + RCL



Success R/W operation in RCL/WCL = 1

Fixed consistency Level

Idea

➢ LSM-Tree

- Strong Sorted in each level(High read performance, low write performance) Not suitable for WCL > RCL

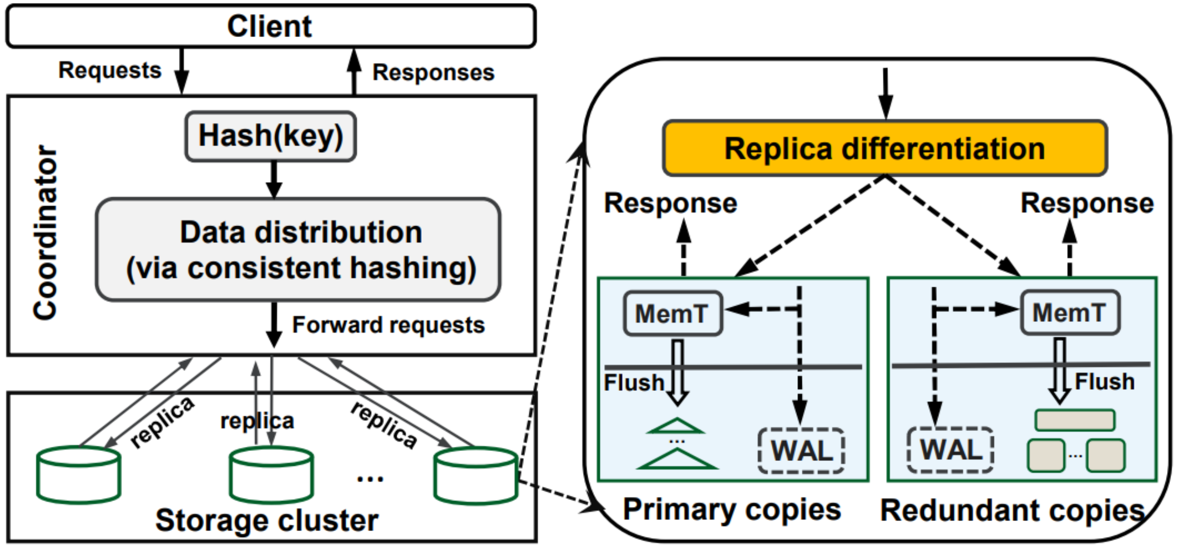
➢ Idea

- **Trick on consistency level : Balance on Read and Write**
  - **WCL > RCL -> Guarantee write performance -> weakly ordered**
  - **WCL < RCL -> Guarantee read performance -> strongly ordered**

Design

➢ Separation + Two-layer log

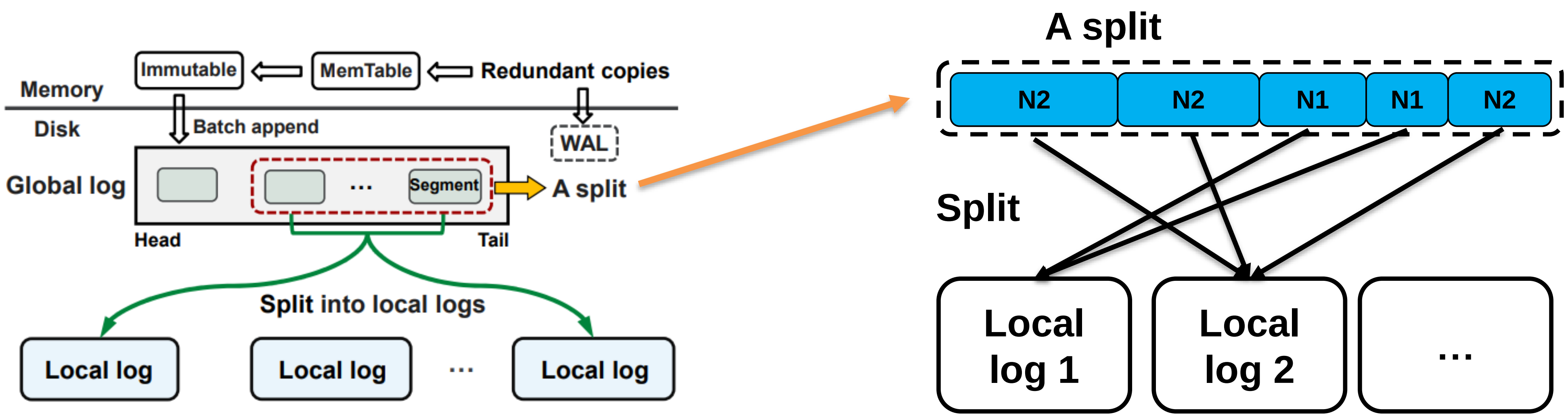
- Replica differentiation : recalculate hash to check if a KV pair is primary or redundant
- Primary data : LSM-Tree
- Redundant data : Two-layer log
    - global log
    - local log



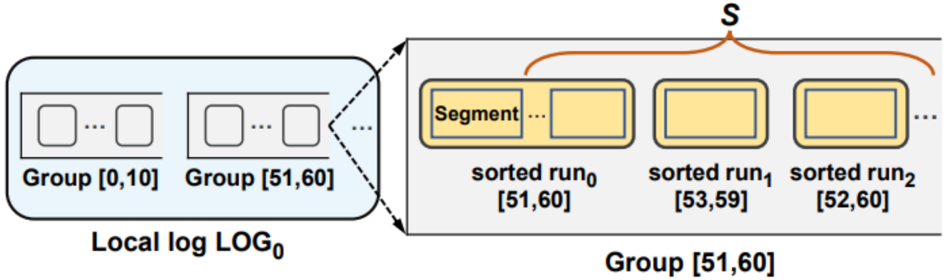
Design

➢ Two-layer log

- Global log :save all flushed KV-pairs without any extra metadata
  - **Feature Inline + Append-only → Fast write**
  - **Problem Hard to recovery(read) + garbage collection cost**
- Local log : take some log from the end of the global log and save them to local log (**background**)

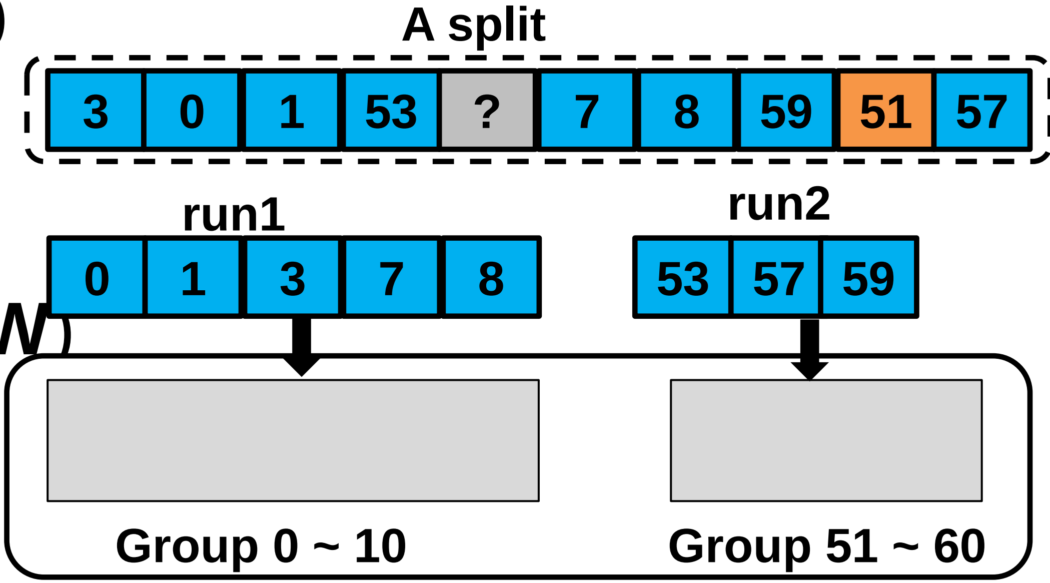


Design



➢ Local Log Structure

- Range-based grouping(virtual nodes-friendly)
- Sorted inside each run
- Each split generate **1** run for each group



➢ Tunable Ordering(balance between R/W)

- user-configurable S: maximum runs in a range group
- Merge new runs into old ones when the S reaches the limit(big overhead)
- S = 1 Read ↑ Write ↓ (Like a simple LSM-Tree, for high RCL)
- S = Inf Read ↓ Write ↑ (for high WCL)

Evaluation

- **Setup**
  - Distributed System: local cluster, 10Gb/s switch
  - Hardware: 500G SSD 12-core E5-2650 v4
- **Workloads**
  - Generated by YCSB(Zipfian constant 0.99)
  - Pair size 1KB(key size: 24B)
- **Settings**
  - Compare objects: Cassandra(traditional), mLSM, DEPART
  - Storage system: 5 nodes + triple replication
  - Different consistency level (WCL= 1,RCL=1 & WCL = 3,RCL=1)

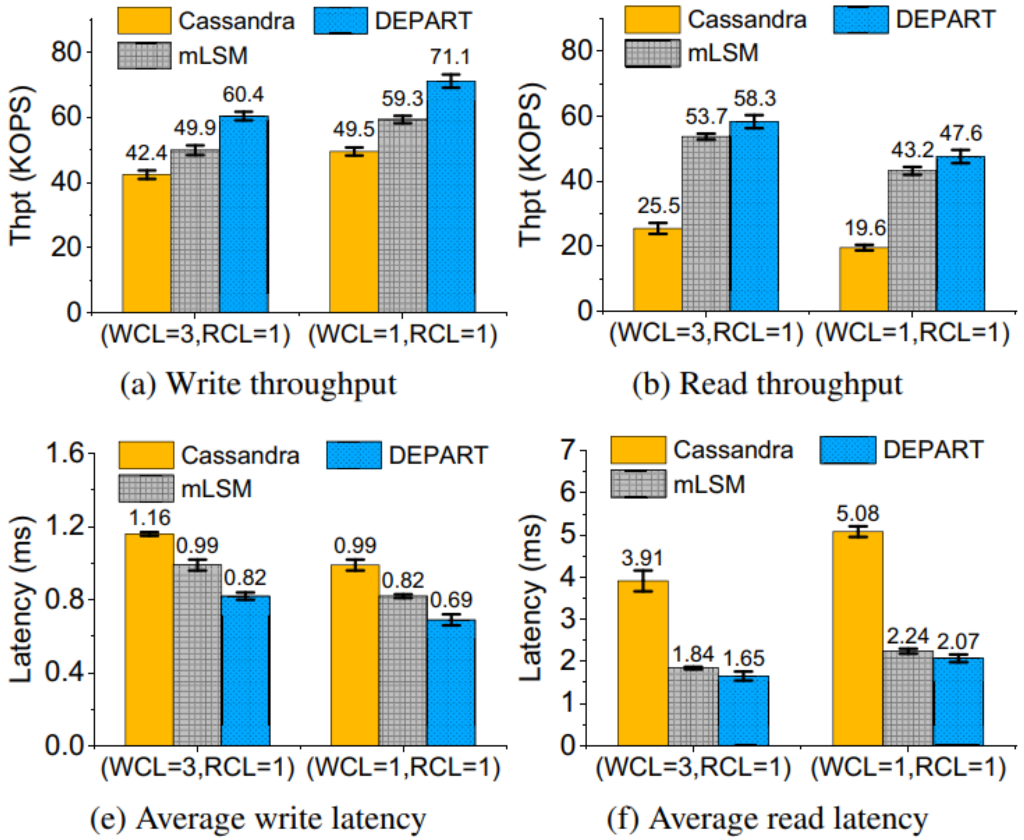
Evaluation

➢ IO Throughputs / Latency

DEPART > mLSM > Cassandra

- DEPART vs. Cassandra
  - Smaller LSM Tree
- DEPART vs. mLSM
  - Smaller gap
  - (2-layer-log > LSM?)(unknown size of runs)

- WCL = 3 vs. WCL = 1
  - (Why does WCL=3 have better read performance)



(a) Write throughput

(b) Read throughput

(e) Average write latency

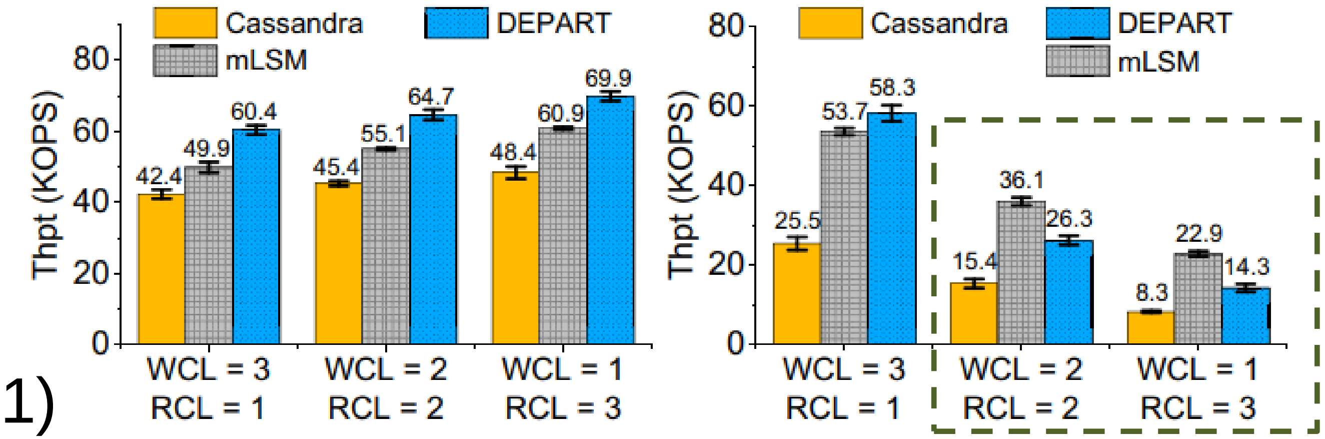
(f) Average read latency

Evaluation

➢ **Performance under different consistency configurations**

DEPART ? mLSM > Cassandra

- Write
  - DEPART > mLSM
- Read
  - DEPART < mLSM (when RCL > 1)



(a) Write throughput (b) Read throughput

DEPART prefer write performance, It is possible to configure a larger number of runs

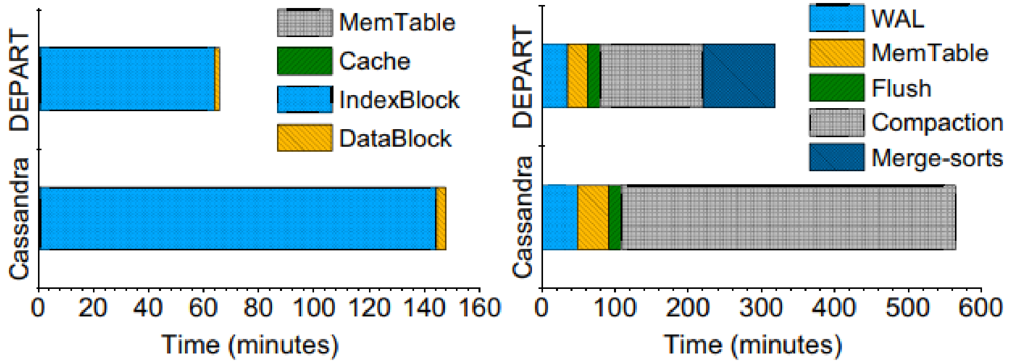
Evaluation

➢ Time breakdown

No data for mLSM

DEPART > Cassandra

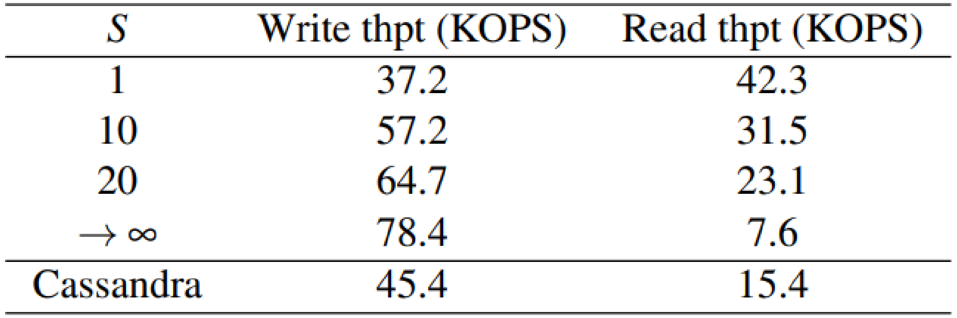
- Read
  - Less index time
- Write
  - Merge sort < compaction



(a) Read time breakdown

(b) Write time breakdown

➢ Impact of the ordering degree S



S	Write thpt (KOPS)	Read thpt (KOPS)
1	37.2	42.3
10	57.2	31.5
20	64.7	23.1
→ ∞	78.4	7.6
Cassandra	45.4	15.4

Conclusion

➢ Problem

- Big LSM-Tree cause serious read and write amplification

➢ Idea

- Separation primary data and redundant data
- Trick on consistency level : Balance on Read and Write

➢ Solution

- Store primary data in LSM-Tree
- Store redundant data in 2-layer-log
- 2-layer-log can adjust internal data organization to meet different consistency requirements