LearnedKV:
Integrating LSM and Learned Index for Superior Performance on SSD

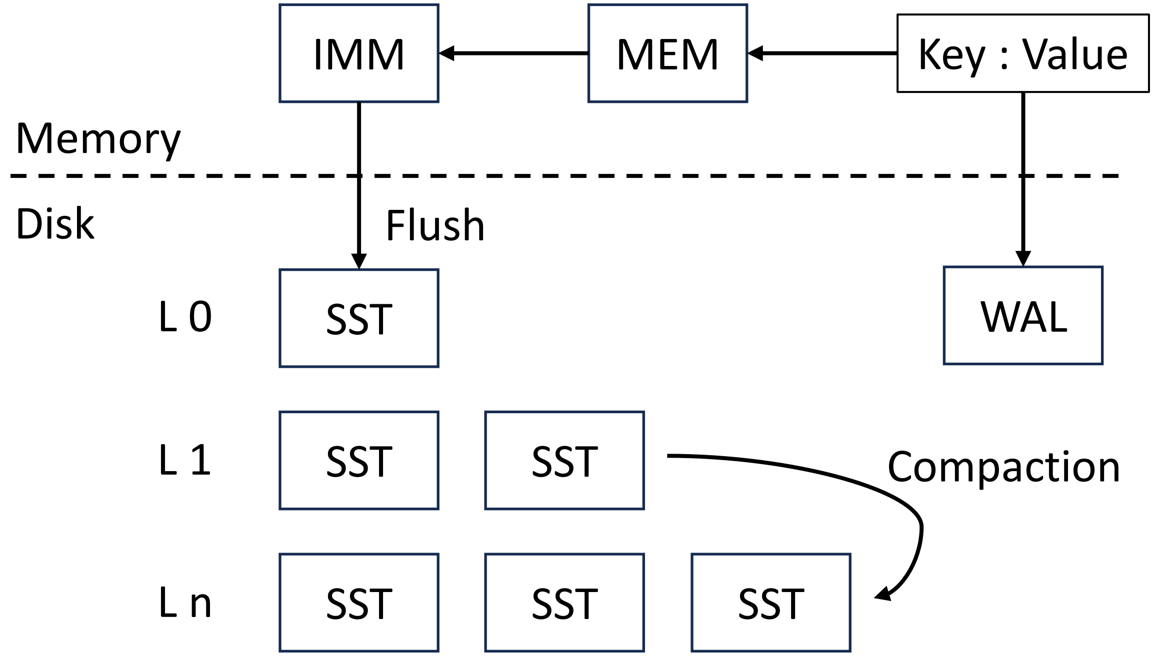
- Wang, Wenlong, and David Hung-Chang Du.
- University of Minnesota

2025.1.22

Presentation by Guangxun Zhao
GuangxunZhao@dankook.ac.kr

Introduction

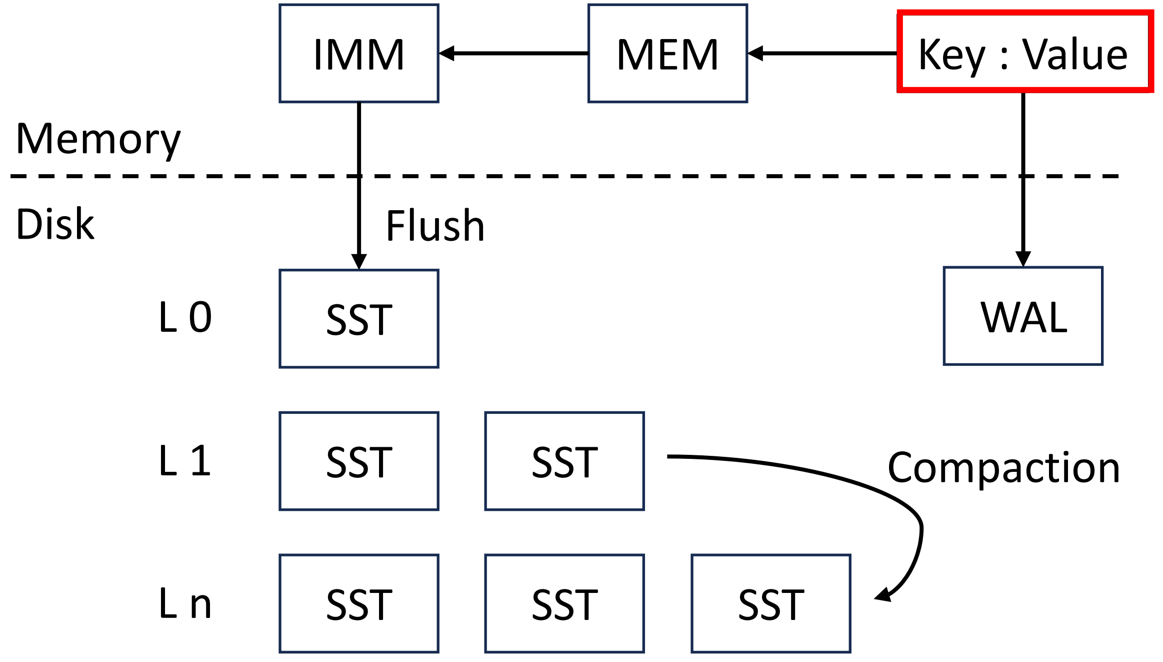
- To effectively handle bigdata use Key-Value Stores.
- LSM-tree is write optimized data structure.



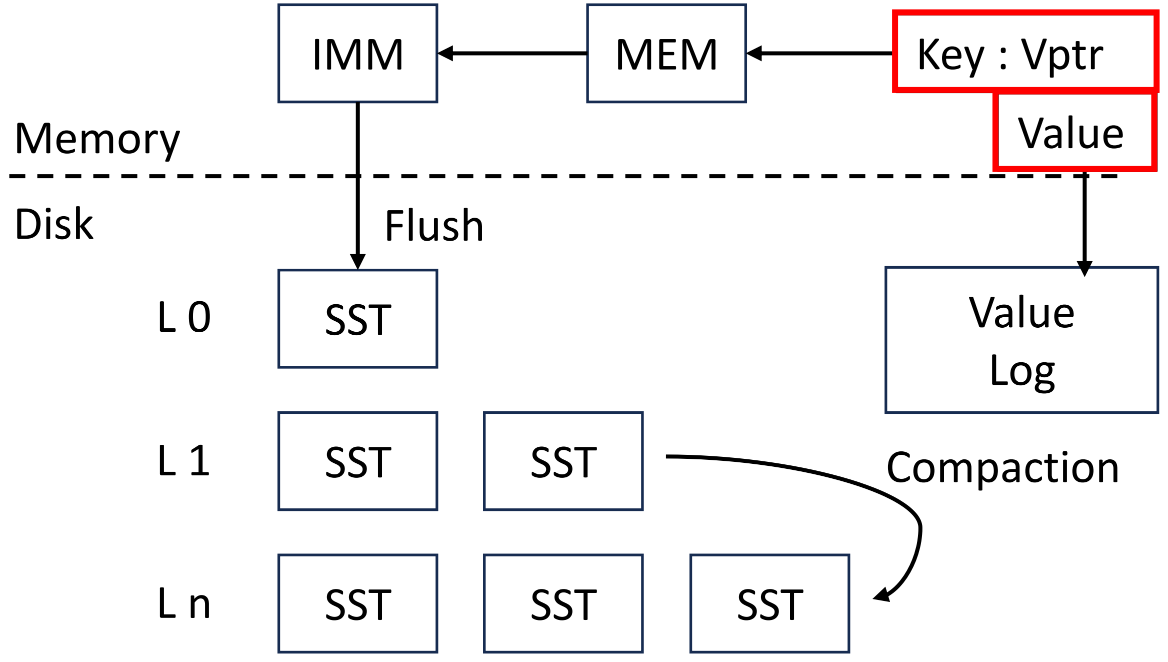
MongoDB RocksDB
LEVELDB DynamoDB

RocksDB and Wisckey Structure

- Write/Read amplification problem in LSM-tree.
- Wisckey uses Key-Value separation to reduce Write/Read amplification .



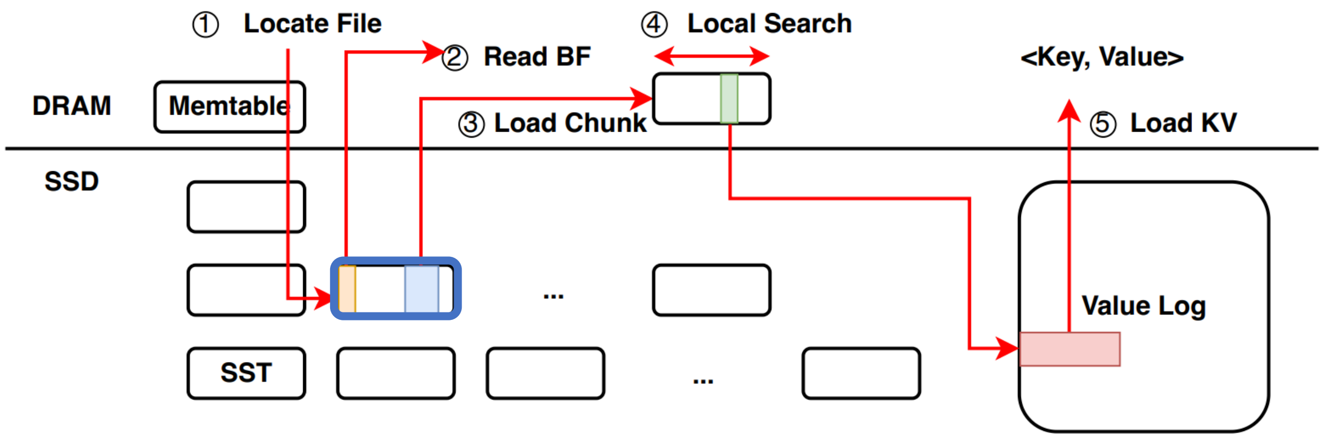
Structure of RocksDB



Structure of Wisckey

Bourbon

- Learning Algorithm: Greedy-PLR.
- Bourbon using the Learned Index to replace or bypass the index blocks of SST files.



Bourbon lookup process

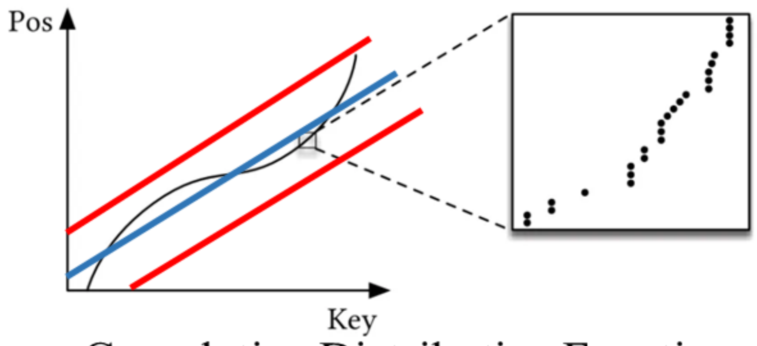
Piecewise Linear Regression

Pos = slop*key + intercept

Pos - err , Pos + err

Pos: Position where the key is stored (index in the predicted array)
slope: Slope of the CDF
key: Key value to search for
intercept: y-intercept (bias) of the CDF

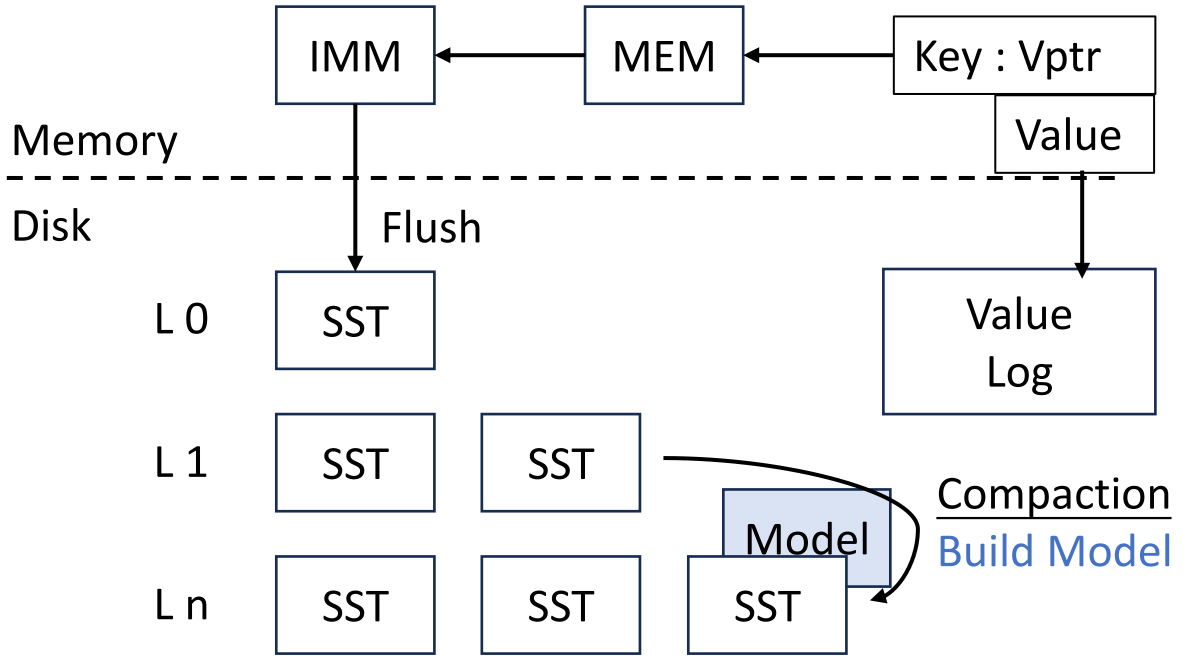
Model
Boundary



Cumulative Distribution Function

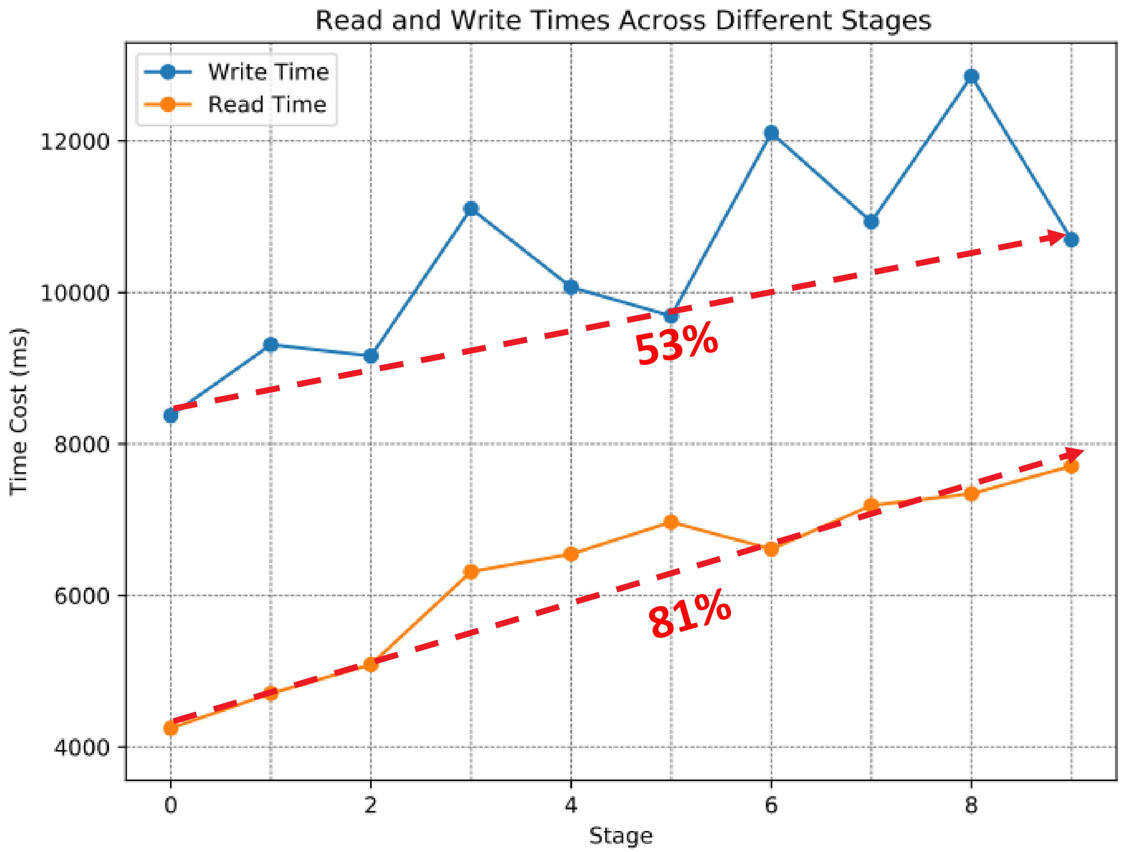
Bourbon

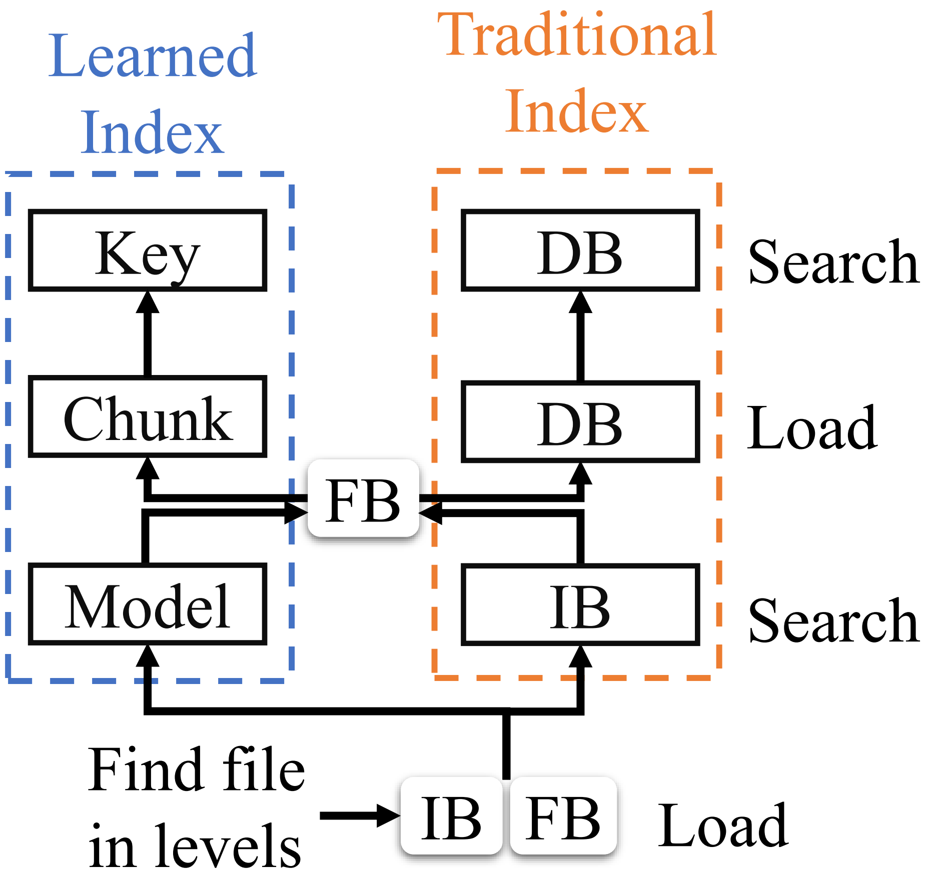
- Models need to be rebuilt during compaction.
- The design merely adds the Learned Index instead of treating it as an independent replacement to optimize the system.



Read and Write time cost of the RocksDB

- 1M workload into 10 stages in RocksDB, writing 100K new keys into it for each stage.





Tiered index

- Aim to build a tiered index where the LSM-tree is used for absorbing random writes, and the Learned Index helps accelerate the lookup process.
    1. Lookup performance
    2. Compaction cost
    3. Re-insert after Garbage Collection
    4. Update latency

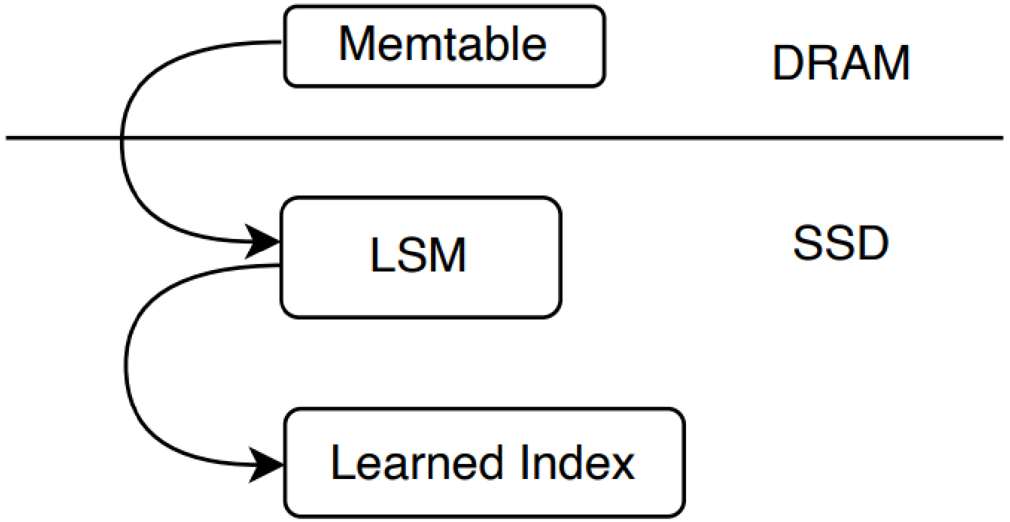


Figure 4: Abstraction of tiered storage with LSM and Learned Index

Challenges

- Challenges of integrate LSM with learned indexes
  - Challenge 1: How to efficiently convert data from the LSM-tree to the Learned Index?
  - Challenge 2: Under which conditions does the conversion provide more benefits?

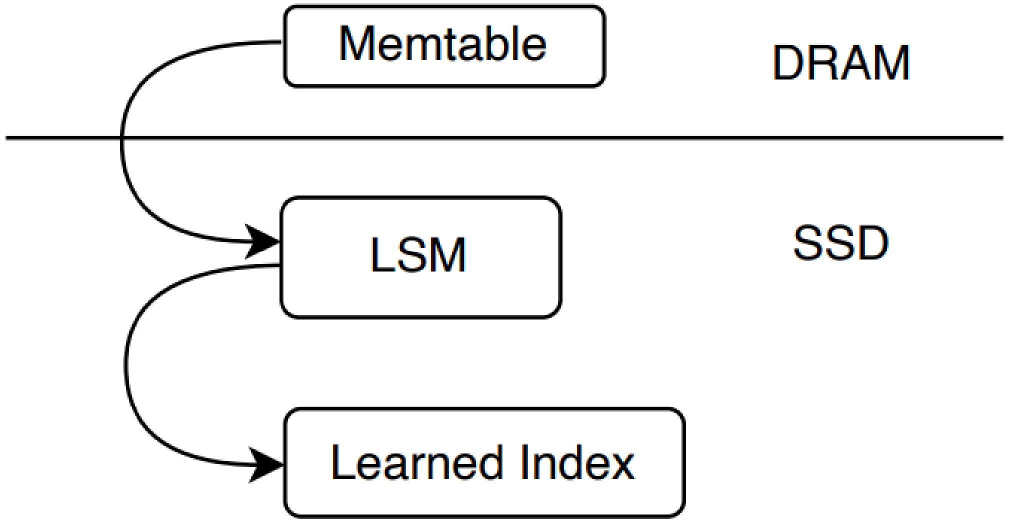
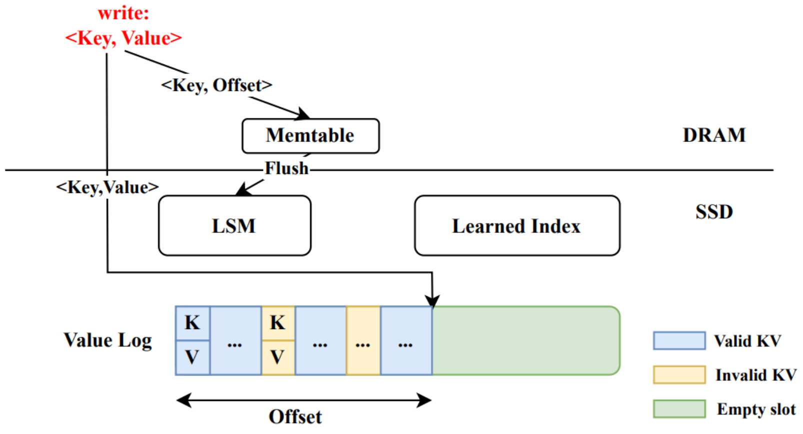


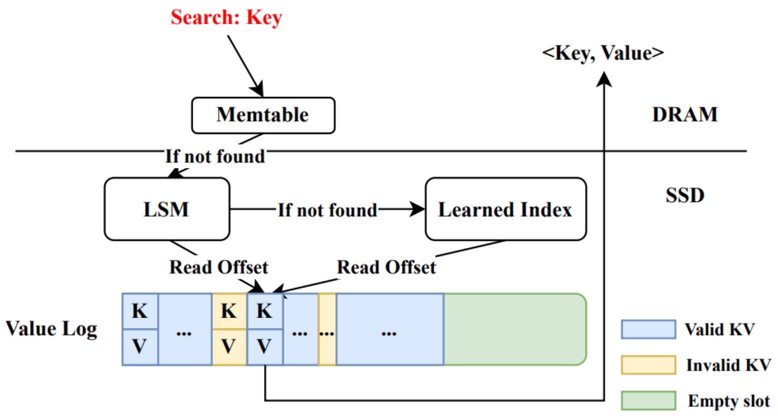
Figure 4: Abstraction of tiered storage with LSM and Learned Index

LearnedKV Structure

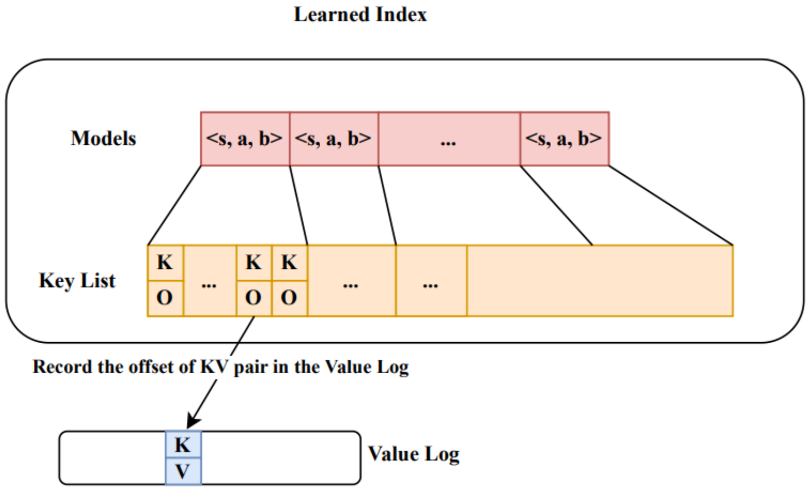
- LearnedKV consists of three main components:
  - LSM-Tree, Learned Index, VLog.



(a) Write operation in LearnedKV

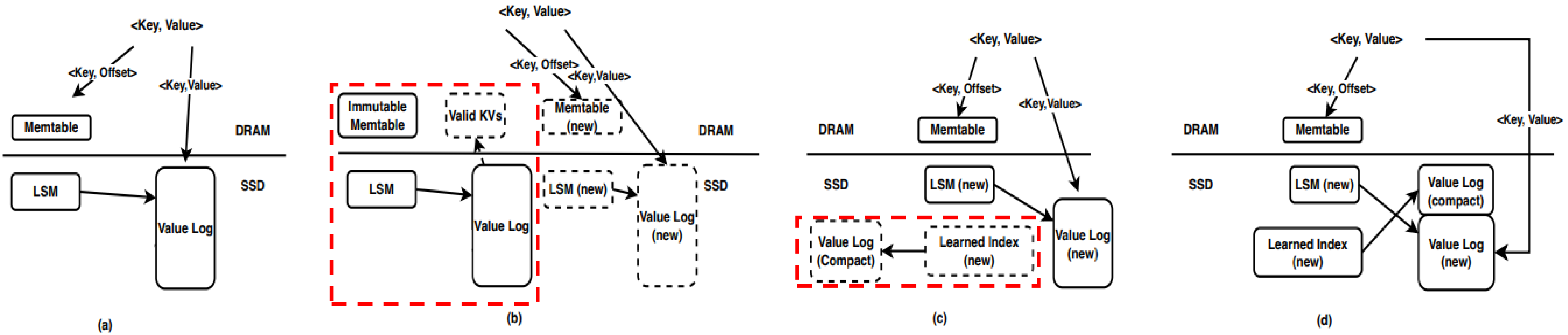


(b) Read operation in LearnedKV



LearnedKV Structure

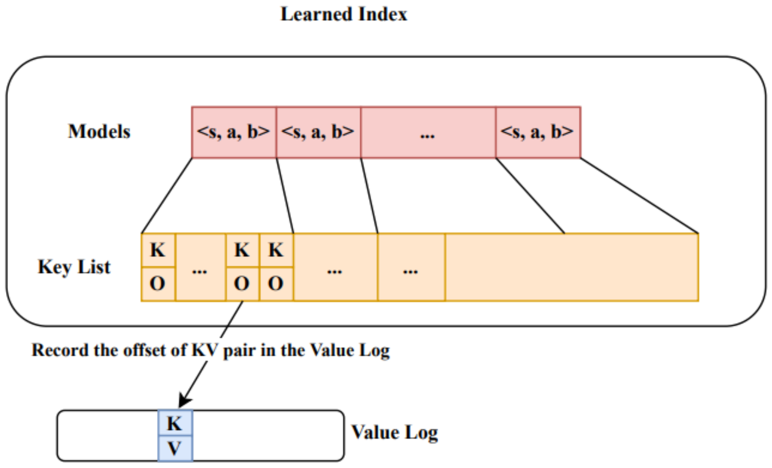
- GC-triggered Conversion from LSM-tree to Learned Index.
- When performing Vlog GC, sorting and learning are applied. After that, the old LSM-tree is deleted and replaced with a new LSM-tree.



Greedy-PLR+

- Learned index
  - Starting key (s), slope (a), and intercept (b).

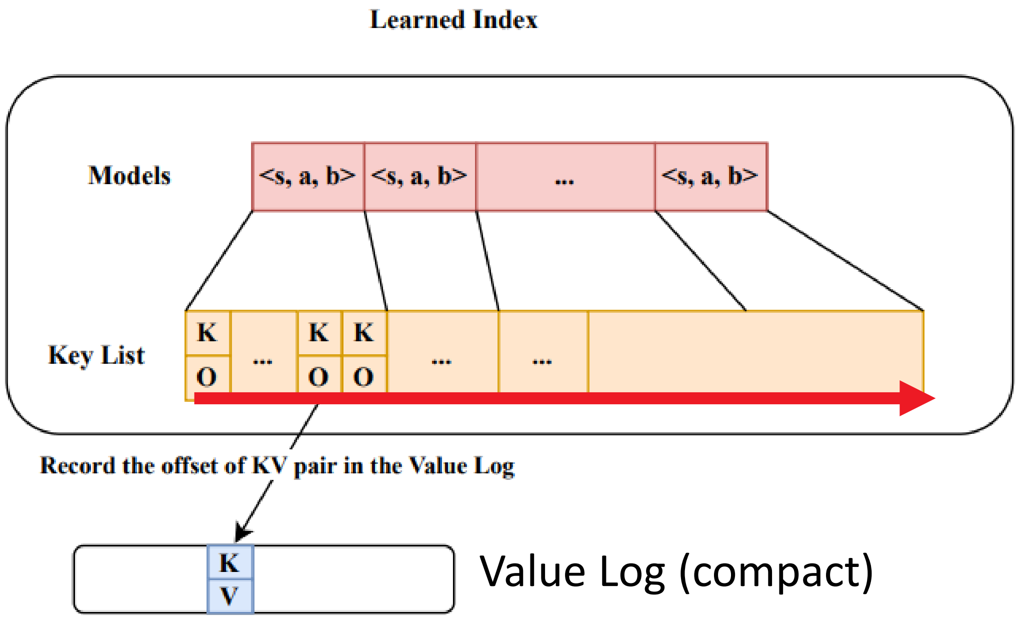
- <u>Pos=a×k+b</u>
  - Pos=8000
  - Error bound=1
  - Page number :0,8000//4096=1, 2

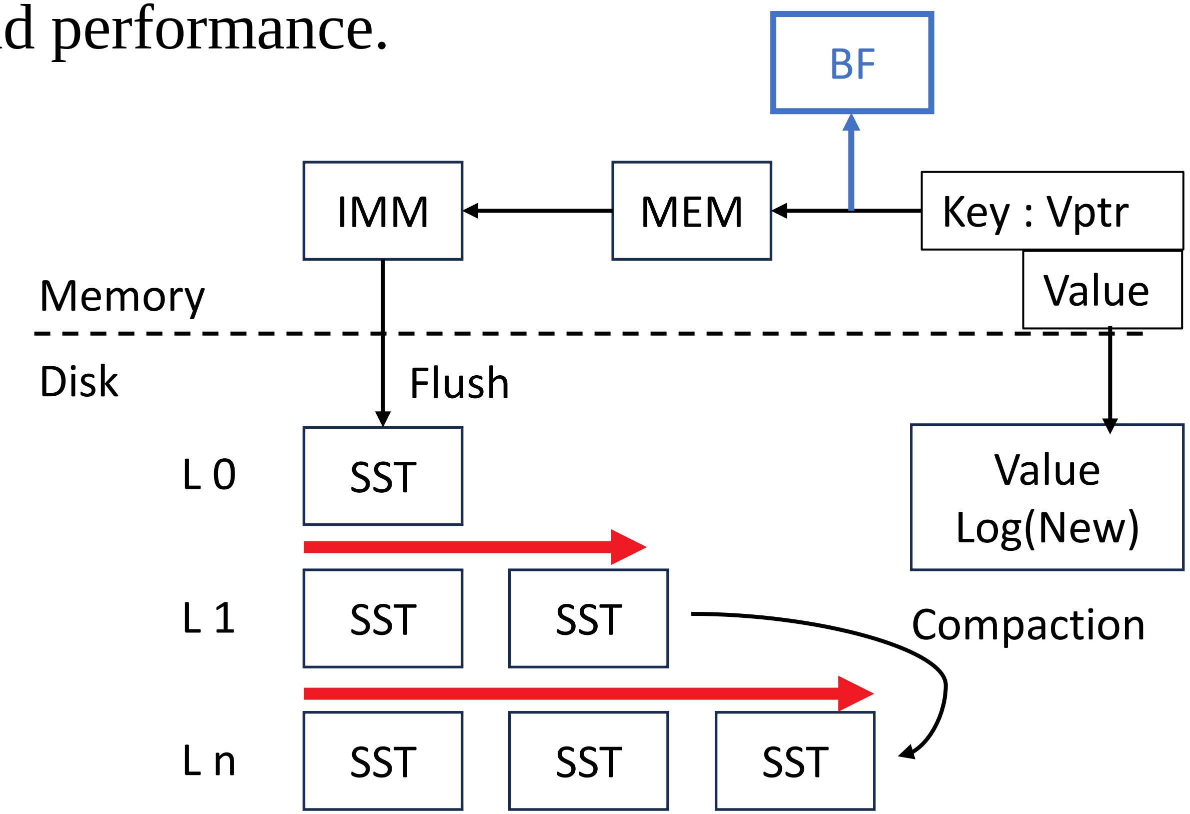


	Bourbon	LearnedKV
예측 모델 방식	바이트 오프셋을 직접 예측	페이지 단위로 예측
검색 범위	[p−e,p+e]	[p−e,p+e] (SSD 페이지 단위)
SSD 접근 방식	오프셋 기반 페이지 로드	페이지 단위에서 직접 예측 후 로드

LearnedKV structure

- Range query optimize with Learned index's Key list.
- Use In-memory Bloom filter to optimize read performance.





Evaluation

Component	Specification	Schemes Compare
Processor	12- core Intel Xeon E5-2620 v3 2.40GHz CPU	BlobDB within RocksDB
Memory	62GB	HashKV(KV separation DB)
Storage	854GB SSD	B+-Tree
key & value size	8 bytes & 1016 bytes	

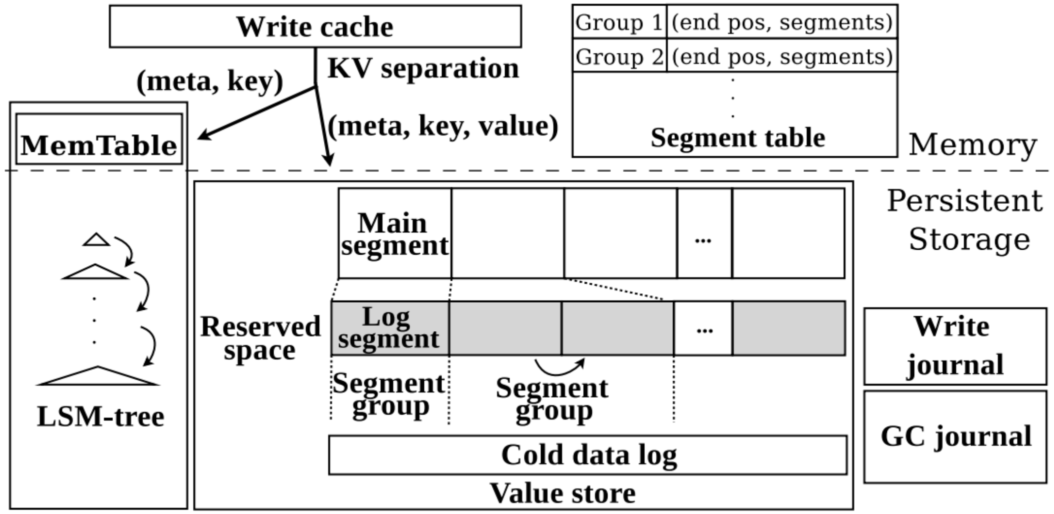


Figure 3: HashKV architecture.

Overall Performance Comparison

- P0 load 1M KV pairs.(YCSB load)
- P1,P2,P3 500,000 read and update operations. (YCSB A)
  - Zipfian distribution.

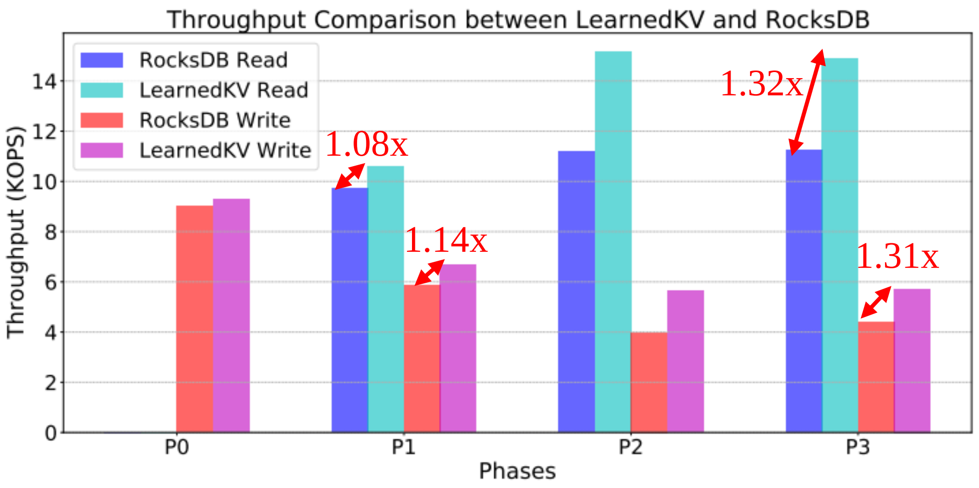


Figure 8: Throughput Comparison between LearnedKV and RocksDB. (P0: Load; P1,P2,P3: Read and Update; Learned Index is built in the middle of P1; GC happens in P1,P2,P3)

		LearnedKV	RocksDB
Key-Ptr Space			
LSM		4.2MB	23MB
key_array	Learned index	7.7MB	-
model		8.0KB	-
Total		**11.9MB**	**23MB**
Key-Value Space			
vlog		978MB	978MB

Overall Performance Comparison

- Each scan request reads about 500KB of KV pairs.
- "Set 1" contains 0.5M updates before the scan requests, while "Set 2" includes 1M update requests before the scan.

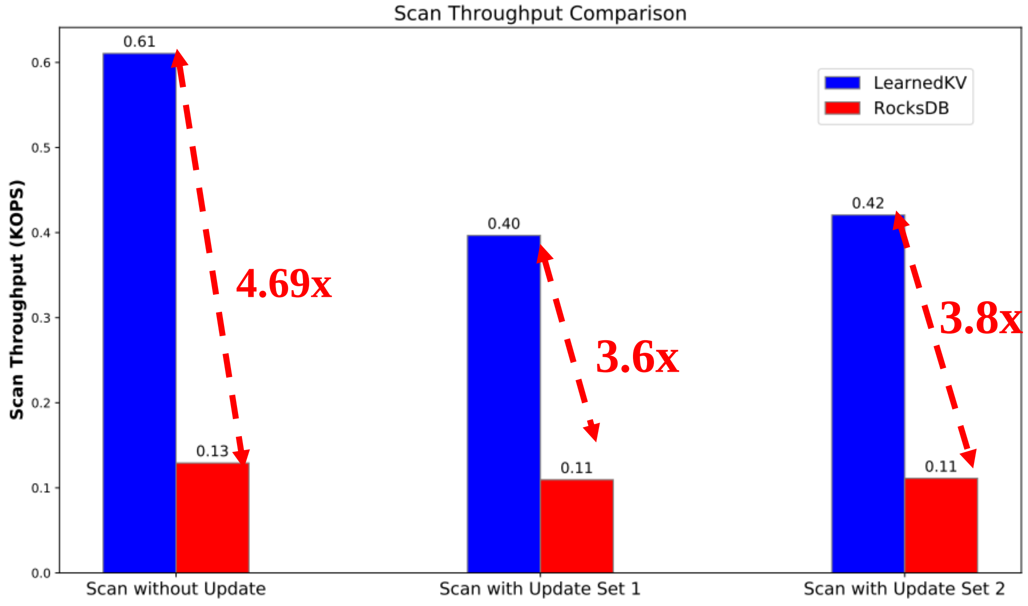


Figure 10: Performance Comparison of Range Scan

Other KV Stores

- LearnedKV can also outperform HashKV 3.59x in terms of write performance
- With an 8-byte value setup, LearnedKV outperforms the B+-Tree by more than 5 times in both read and write performance.

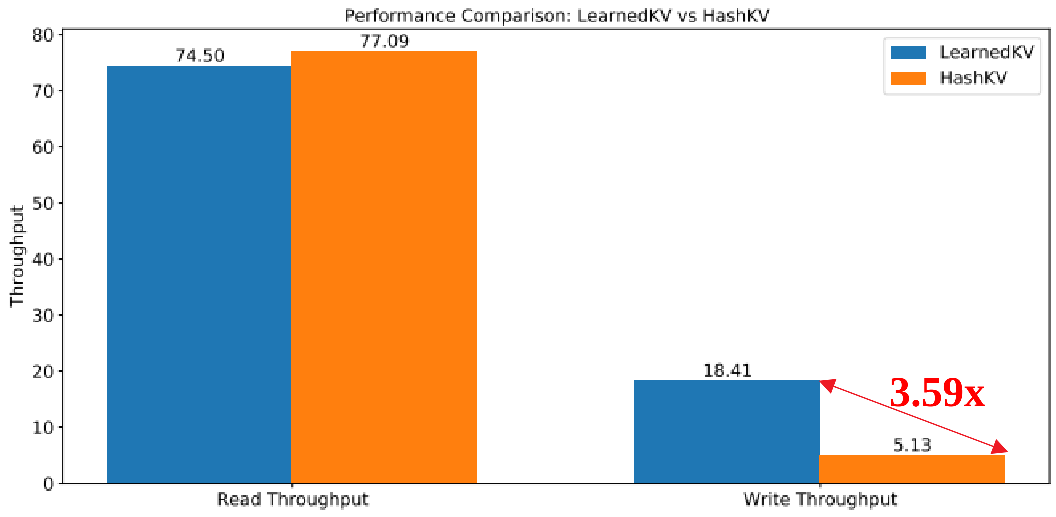


Figure 11: Comparison between LearnedKV and HashKV

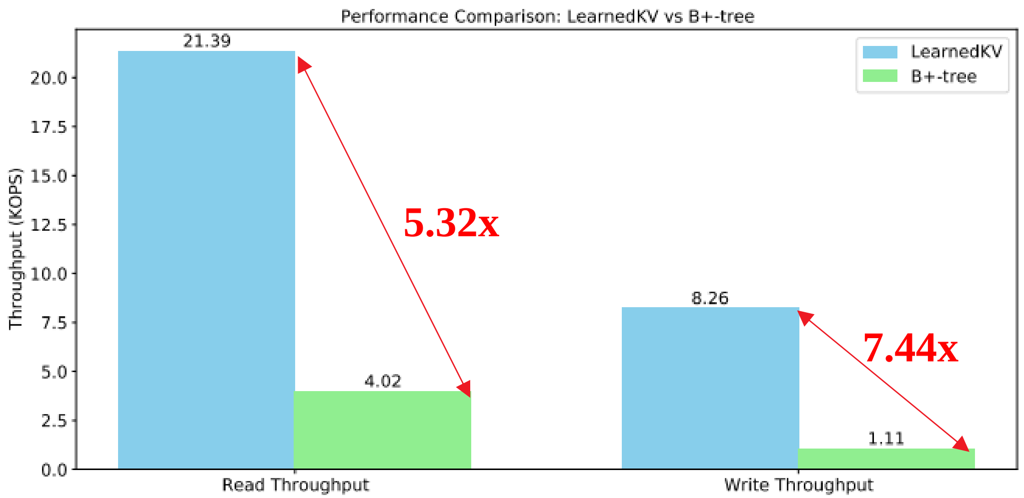


Figure 12: Comparison between LearnedKV and B+_Tree

Drill-down Analysis

- Read and write operation numbers and their average time cost in the P3.
- 500K read operations in Read time break down.

Operation	Number		Avg Time Cost (µs)
Write	500,377		165.584
Read	499,623		66.6561
Read Operation			
Read through LI	131,508	26	$\boxed{37.5138}$
Read through RD	383,256	%	$\boxed{58.0299}$ 2.14x
Load form vlog	499,623		11.6553

Table 2: Summary of operations and their time costs. (LI: Learned Index; RD: RocksDB)

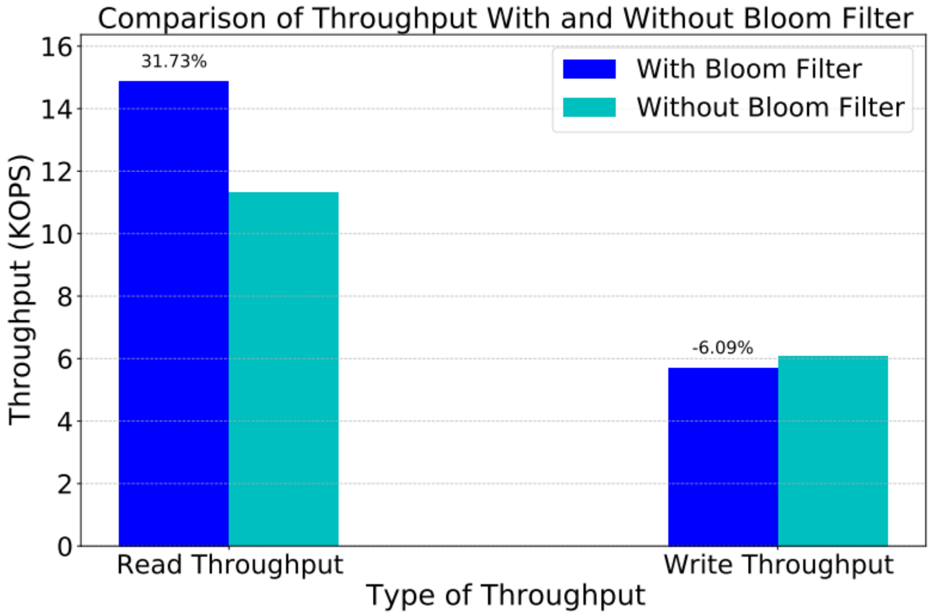
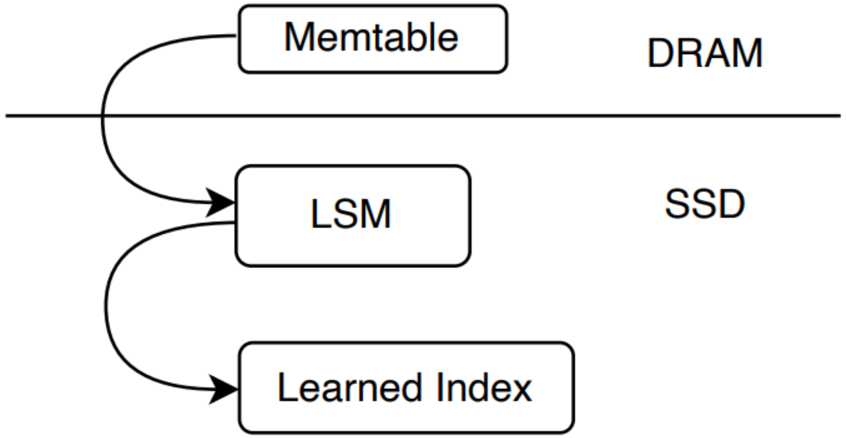


Figure 13: Comparison of Throughput With and Without Bloom Filter

Conclusion

- This paper proposes LearnedKV, an efficient tiered KV store that integrates an LSM with a Learned Index.
- LearnedKV standalone design significantly reduces the size of the LSM, further enhancing both read and write performance.
- LearnedKV outperforms the latest KV systems, achieving up to 1.32x better performance in read requests and up to 1.31x better write performance.



LearnedKV:
Integrating LSM and Learned Index for Superior Performance on SSD

Thank you

2025.1.22

Presentation by Guangxun Zhao
GuangxunZhao@dankook.ac.kr