

LOG-STRUCTURED MERGE-TREE

EFFECTIVE DATA STRUCTURE FOR MEMORY STORAGE

Mohammad Rahim Faihaj Alam Topu Jarif Islam

Department of CSE
Bangladesh University of Engineering & Technology

CSE300 Presentation
February 25, 2023

PRESENTATION OUTLINE

1 INTRODUCTION

- First Intro
- Concepts
 - Memtable
 - SSTables
 - Compaction
- Simple two levels LS tree

2 ALGORITHM

- Figure
- Write

- Read
- Complexity

3 ADVANTAGES & DISADVANTAGES

- Comparison with B-Tree
- Advantages
- Disadvantages

4 CONCLUSION

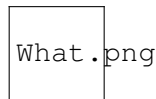
5 BACKUP

A BRIEF INTRO

Log-Structured Merge-Tree (Shortly known as LSM Tree) is a data structure with performance data structure that makes it very attractive to store data with high insert and update rates.

It comprises of tree-like data structure with 2 or more levels:

- 1 Memtable completely resided in memory
- 2 SSTables stored in disk



LIST OF CONCEPTS

LSM tree is based on 3 important concepts to optimize read and writes:

- SSTables
- Memtable
- Compaction

MEMTABLE

In-Memory DS

Write Cache

Self-Balancing Tree

Memtable.png

SSTABLES

Sorted String Table

Immutable

Key-Pair DS

SSTables.png

COMPACTION

Removes Redundant
Keys

Creates Compacted
Merge Tree

Occurs Continuously
Subsequently

Compaction.png

TWO-LEVEL LSM TREE

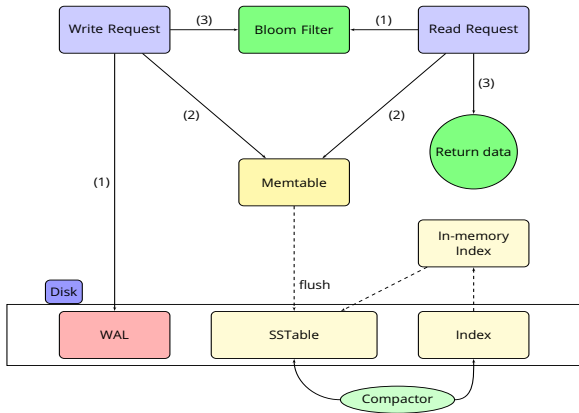
A simpler version of the LSM tree, a two-level LSM tree comprises two tree-like structures, called C_0 and C_1 . C_0 is smaller and entirely resident in memory, whereas C_1 is resident on disk.

New records are inserted into the memory resident C_0 component. If the insertion causes the C_0 component to exceed a certain size threshold, a contiguous segment of entries is removed from C_0 and merged into C_1 on disk.

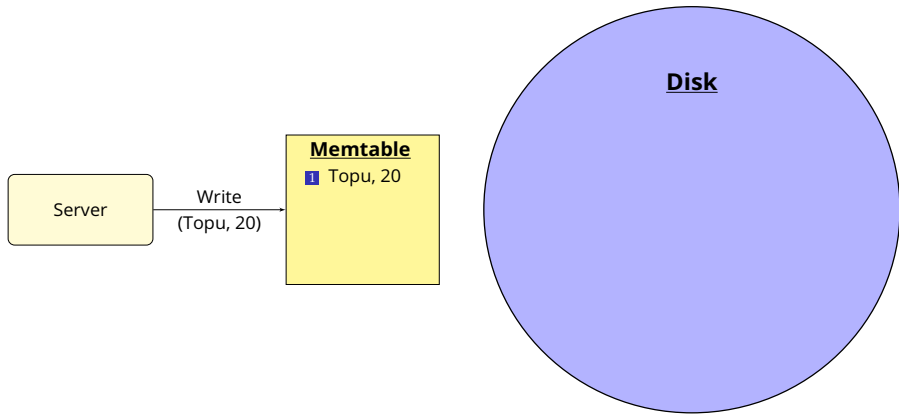
TWO-LEVEL LSM TREE

TwoLevelLSM.png

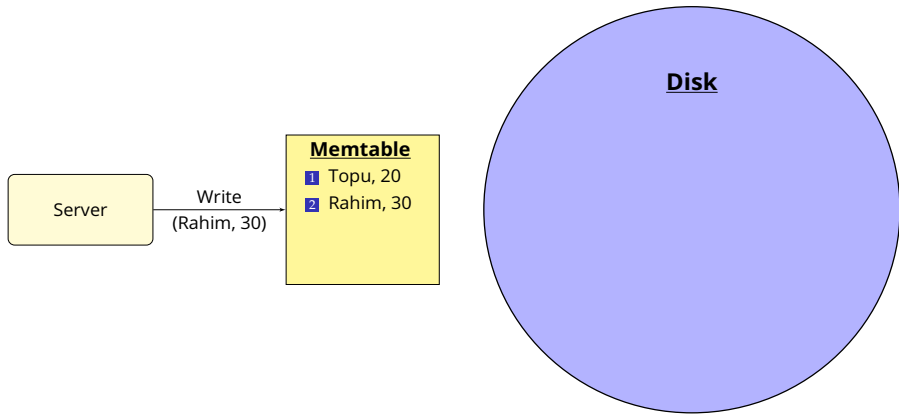
PROCESS



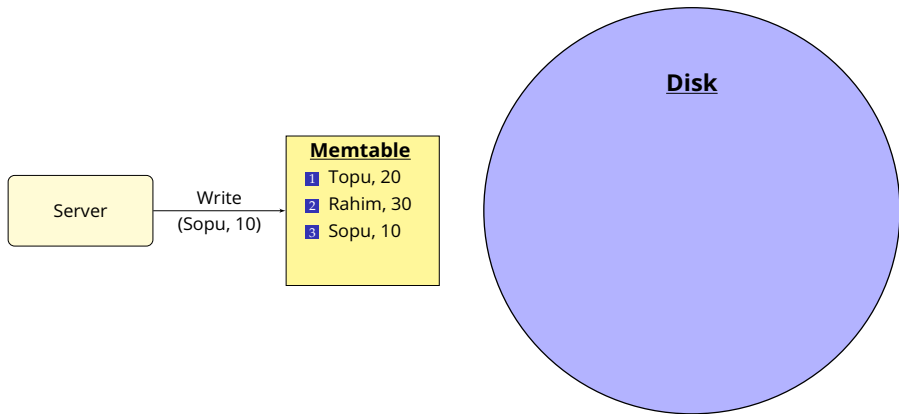
WRITE



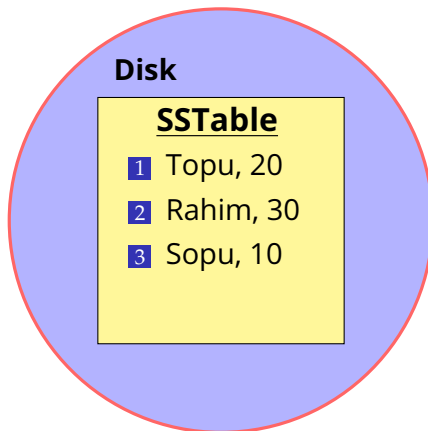
WRITE



WRITE

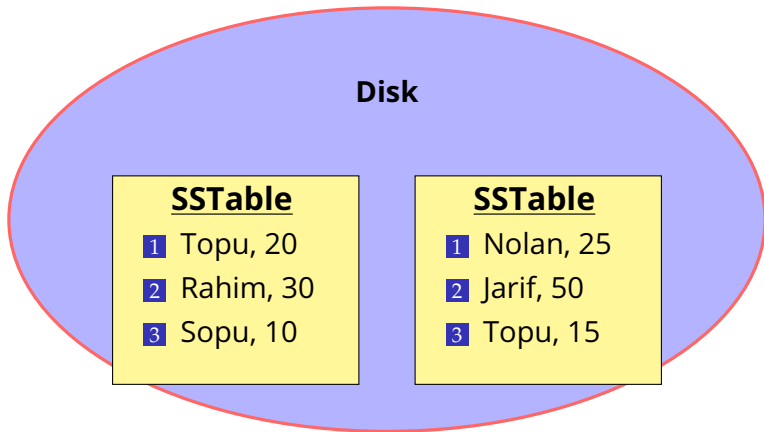


WRITE



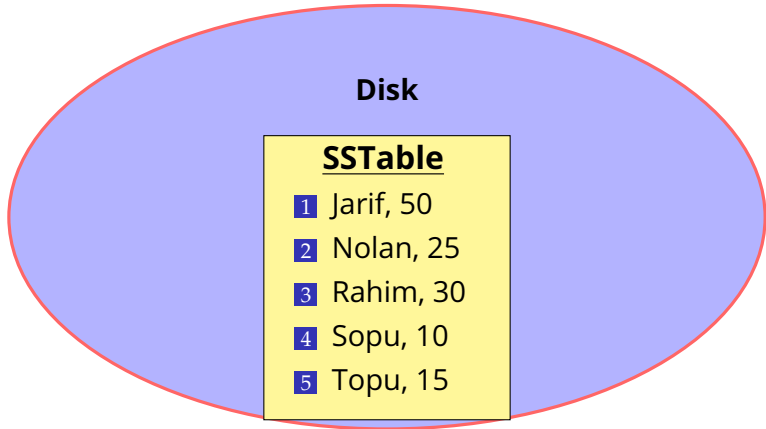
After reaching threshold value, memtable is flushed as SSTable into the disk

WRITE



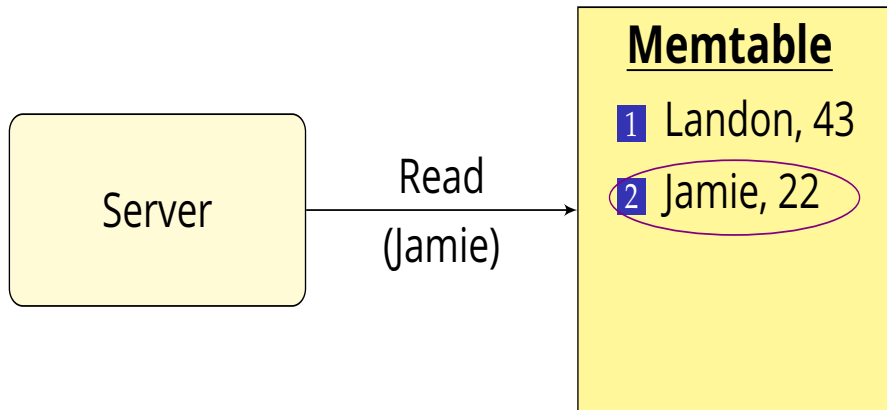
SSTable is immutable. And so the value of key "Topu" is not changed

COMPACTOR



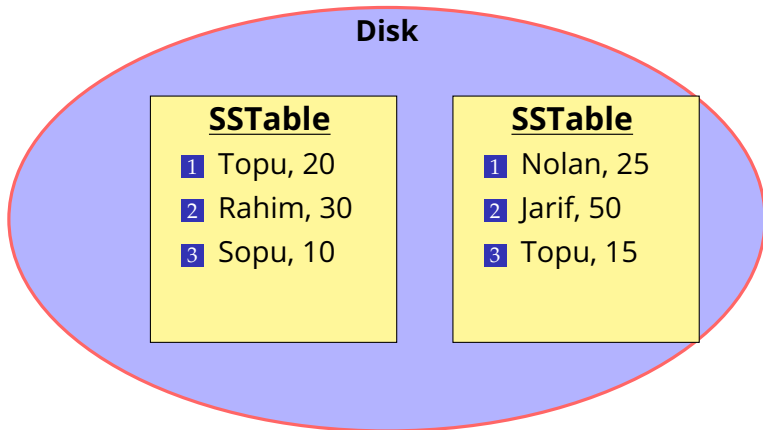
Compactor compacts SSTables, removes redundancy

READ



It first checks the memtable. If found, then returns the data.

READ




After checking memtable, it checks all the SSTables

CHECK ALL THE SSTABLES? ($N \log N$)

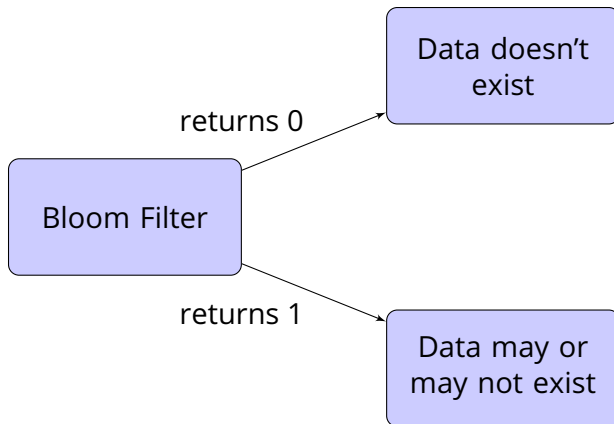
If the data we are searching for is not there in the memtable nor in the SSTables, then? Time for searching all the SSTables go in vain!

Bloom filter is there for us:

 surprise.jpg

- 1 Probabilistic data structure
- 2 Return values to tell us whether the data exists or not

BLOOM FILTER



$$\text{Complexity} = O(c) + pO(n)$$

COMPLEXITY

Action	Best Case	Worst Case
Write	$O(1)$	$O(1)$
Read	$O(1)$	$O(\log n)$

TABLE: Time Complexities of LSM Tree

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.
- Efficient use of storage

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.
- Efficient use of storage
 - Data is merged and compacted in batches, reducing the overall number of disk writes and improving storage utilization.

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.
- Efficient use of storage
 - Data is merged and compacted in batches, reducing the overall number of disk writes and improving storage utilization.
- Scalability

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.
- Efficient use of storage
 - Data is merged and compacted in batches, reducing the overall number of disk writes and improving storage utilization.
- Scalability
 - As the size of the dataset grows, LSM trees can automatically adjust the size of their memory buffer to maintain optimal performance.

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.
- Efficient use of storage
 - Data is merged and compacted in batches, reducing the overall number of disk writes and improving storage utilization.
- Scalability
 - As the size of the dataset grows, LSM trees can automatically adjust the size of their memory buffer to maintain optimal performance.
- Support for high-concurrency workloads

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.
- Efficient use of storage
 - Data is merged and compacted in batches, reducing the overall number of disk writes and improving storage utilization.
- Scalability
 - As the size of the dataset grows, LSM trees can automatically adjust the size of their memory buffer to maintain optimal performance.
- Support for high-concurrency workloads
 - LSM trees support high-concurrency workloads, allowing multiple threads or processes to write to the tree at the same time.

ADVANTAGES OF LSM TREES FROM CHAT GPT LOL

- Fast write performance
 - Writes are first written to a memory buffer and then periodically flushed to disk in a batch, reducing disk I/O and improving performance.
- Efficient use of storage
 - Data is merged and compacted in batches, reducing the overall number of disk writes and improving storage utilization.
- Scalability
 - As the size of the dataset grows, LSM trees can automatically adjust the size of their memory buffer to maintain optimal performance.
- Support for high-concurrency workloads
 - LSM trees support high-concurrency workloads, allowing multiple threads or processes to write to the tree at the same time.


DISADVANTAGES

- Higher read latency

DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels


DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity?  emoji_thinks.png

DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity? `emoji_thinks.png`
- size of the datasets


DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity?  emoji_thinks.png
- size of the datasets
 - large datasets
 - smaller datasets that can fit entirely in memory.


DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity? `emoji_thinks.png`
- size of the datasets
 - large datasets
 - smaller datasets that can fit entirely in memory.
- More expensive memory requirements

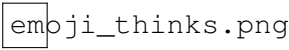
DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity?  emoji_thinks.png
- size of the datasets
 - large datasets
 - smaller datasets that can fit entirely in memory.
- More expensive memory requirements
 - LSM trees require more memory than simpler data structures like B-trees due to their use of a memory buffer and multiple levels.

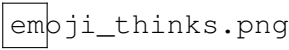
DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity?  emoji_thinks.png
- size of the datasets
 - large datasets
 - smaller datasets that can fit entirely in memory.
- More expensive memory requirements
 - LSM trees require more memory than simpler data structures like B-trees due to their use of a memory buffer and multiple levels.
- Not ideal for high-precision queries

DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity? 
- size of the datasets
 - large datasets
 - smaller datasets that can fit entirely in memory.
- More expensive memory requirements
 - LSM trees require more memory than simpler data structures like B-trees due to their use of a memory buffer and multiple levels.
- Not ideal for high-precision queries
 - LSM trees are optimized for range queries rather than individual point lookups, so they may not be the best choice for high-precision queries.

DISADVANTAGES

- Higher read latency
 - data is stored across multiple levels
- complexity? 
- size of the datasets
 - large datasets
 - smaller datasets that can fit entirely in memory.
- More expensive memory requirements
 - LSM trees require more memory than simpler data structures like B-trees due to their use of a memory buffer and multiple levels.
- Not ideal for high-precision queries
 - LSM trees are optimized for range queries rather than individual point lookups, so they may not be the best choice for high-precision queries.

LSM-TREE VS B-TREE UPDATE KOR CHAT GPT

Memtable



SSTable 1



SSTable 2



SSTable 3



Disk

Root



Node 1



Node 2



Leaf

LSM Tree

B-tree

Efficient for writes
Good for large datasets
Slow for random reads
Merge overhead

Efficient for random reads
Good for small to medium datasets
Slow for updates and inserts
Overhead from rebalancing

WHEN TO USE, WHEN NOT TO EMOJI_THINKS.PNG

SUMMARY

- Step-1: Group the minterm according to the number of 1s.

SUMMARY

- Step-1: Group the minterm according to the number of 1s.
- Step-2: Compare and merge the min terms present in successive groups.

SUMMARY

- Step-1: Group the minterm according to the number of 1's.
- Step-2: Compare and merge the min terms present in successive groups.
- Step-3: Repeat step-2 with newly formed terms till we get all prime implicants.

SUMMARY

- Step-1: Group the minterm according to the number of 1's.
- Step-2: Compare and merge the min terms present in successive groups.
- Step-3: Repeat step-2 with newly formed terms till we get all prime implicants.
- Step-4: Formulate prime implicants table(cover table) and reduce it removing the row of each essential prime implicant and the columns corresponding to the min terms.

REFERENCES

- Tutorials point
- <https://en.wikipedia.org/wiki/Implicant>

HOW BLOOM FILTERS WORK IN LSM TREES

- Bloom filters use a set of hash functions to create a bit vector that represents the presence or absence of a set of elements.
- When a new element is added to the Bloom filter, it is hashed by each of the hash functions, and the corresponding bits in the bit vector are set to 1.
- When a query is made, the query element is hashed by the same hash functions, and the corresponding bits in the bit vector are checked. If all of the corresponding bits are set to 1, it is highly likely that the element is in the Bloom filter. However, if any of the bits are not set to 1, the element is definitely not in the Bloom filter.

WHY BLOOM FILTERS ARE USED IN LSM TREES

- LSM trees store data in multiple levels, with each level using progressively larger and slower storage devices.
- Bloom filters are used to improve query performance by quickly determining which levels of the tree may contain the relevant data.
- In an LSM tree, a Bloom filter is typically used at each level of the tree to represent the set of keys stored in that level.
- When a query is made, the query key is checked against the Bloom filter at the highest level of the tree. If the Bloom filter indicates that the key is likely to be in the level, the level is searched for the key.
- Overall, using a Bloom filter in an LSM tree can significantly reduce the number of levels that need to be searched for a given query, resulting in faster query performance.

WHERE BLOOM FILTERS ARE USED IN LSM TREES

- Bloom filters are used at each level of the LSM tree to represent the set of keys stored in that level.
- The Bloom filter at the highest level of the tree is used to quickly determine if the query key may be in the lower levels of the tree.
- By using Bloom filters to eliminate irrelevant levels, the LSM tree can be searched more efficiently and with fewer disk reads, resulting in faster query performance.

WHATS A BLOOM FILTER?

- A Bloom filter is essentially a bit array of fixed size, which is initialized with all bits set to zero. To add an element to the set, the element is hashed multiple times using different hash functions, and the resulting hash values are used to set the corresponding bits in the bit array to 1.
- To test whether an element is in the set, the same hash functions are applied to the element, and the corresponding bits in the bit array are checked. If all of the bits are set to 1, the element is considered to be "possibly in the set". However, if any of the bits are set to 0, then the element is definitely not in the set.