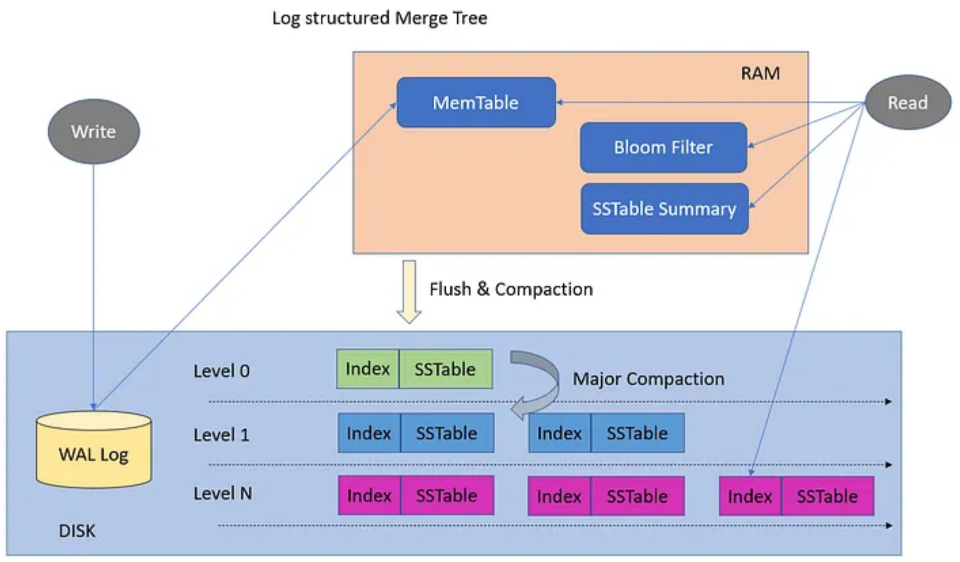
LSM Trees: the Go-To Data Structure for Databases, Search Engines, and More

Ankit Dwivedi Follow 6 min read · May 6, 2023

329 6

I dive into the fascinating world of LSM Trees and how they revolutionize the way large amounts of data are stored and retrieved. 🌳💡

What is a LSM tree?



Overview of LSM Tree design

LSM tree, short for Log-Structured-Merge Tree, is a clever algorithm design that helps us store massive amounts of data without making us wait forever to write it. It stores data in memory first, which is lightning fast. But since we can't keep everything in memory, the LSM Tree periodically flushes data to disk.

But here's where it gets even cooler! It organizes the data into layers of sorted structures, each with more and more compressed data. The top layer is the fastest to access but the least compressed. As the data piles up, it's compressed and written into a new layer called an SSTable (Sorted String Table). We can decide how many layers we need and how much compression to apply, based on the type of data we're dealing with. This trick helps us minimize disk I/O operations and make our data storage super efficient! In this blog we will go through the read and write operations in detail.

1

How write works in LSM Tree?

Let’s dive deeper into how writing works in an LSM Tree.

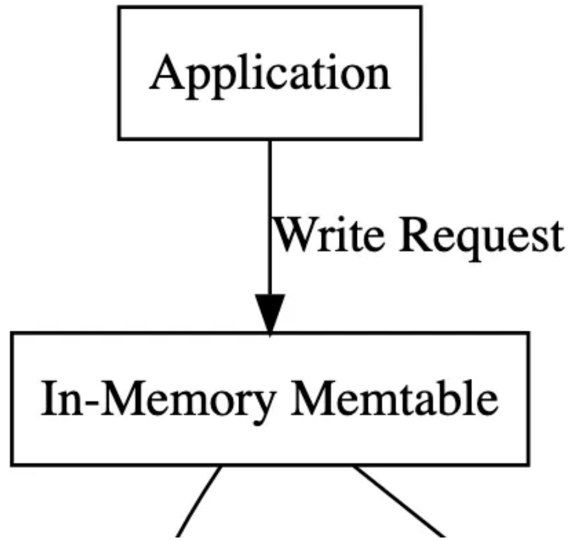
When you want to write data to the LSM Tree the first stop is the in-memory layer of the LSM Tree, which is like the top part of the tree and is super fast as well, because it’s stored in memory!

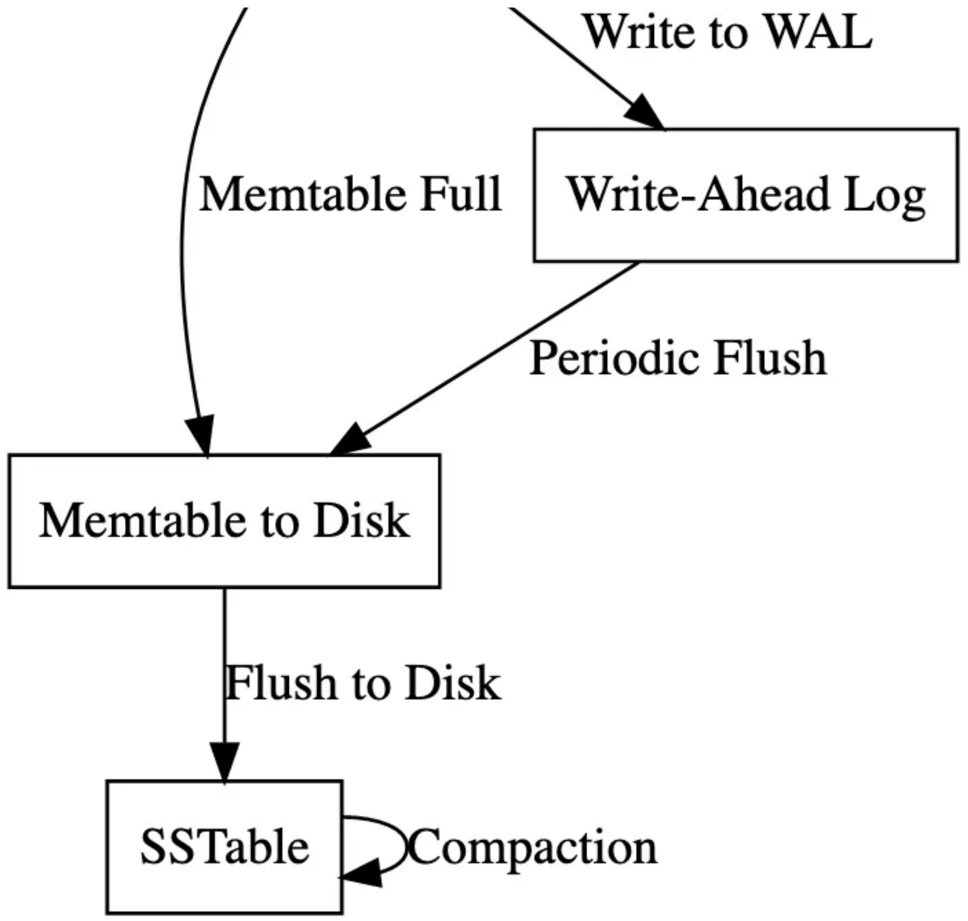
However, keeping all the data in memory indefinitely is not practical, so the LSM tree periodically flushes the data from the in-memory layer to disk.

But here’s the clever part: *before the data is flushed to disk and organized into SSTables, it is also written to the Write-Ahead Log (WAL).*

> *The Write-Ahead Log acts as a backup, ensuring the durability of the data. It serves as a log of all the changes made to the database, acting as a safety net in case of system failures or crashes.*

. . .





Write flow in LSM tree

So, when a write operation occurs,

- The data is first written to the in-memory layer for speedy performance.
- Simultaneously, *the changes are recorded in the Write-Ahead Log to ensure data integrity*.
- Then, at regular intervals or when the in-memory layer reaches a certain threshold, the data is flushed to disk and organized into SSTables.

SSTables, or Sorted String Tables, are essentially just sorted key-value pairs that are written to disk.

One of the best things about SSTables is that they're incredibly efficient for searching and reading data. As you accumulate more data, more levels of SSTables are created, with each layer being more compressed than the previous one. The number of levels and amount of compression can vary depending on the use case, but the general idea is to keep compressing the data as you go down the layers.

Now, you might be wondering ***why the LSM Tree uses this approach.***

The answer is simple: it's great at minimizing disk I/O operations. By writing to disk periodically and in larger batches, you're reducing the number of times the disk has to spin up and down to access data.

How is data read from LSM tree?

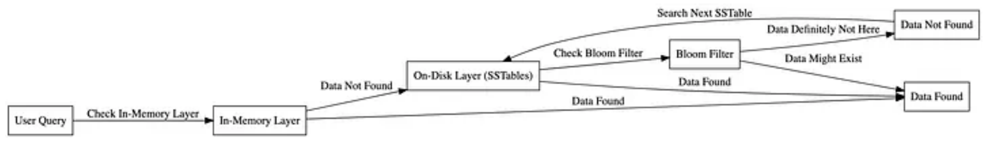
When you want to read data from an LSM tree, the process begins with a user query. You're searching for a specific piece of information, and the LSM tree jumps into action to find it for you.

The first place the LSM tree checks is the in-memory layer. Remember, this layer is super fast to access because it's stored in memory. So, if the data you're looking for happens to be in this layer, hooray! The LSM tree quickly retrieves it, and you get your desired information without any delay.

Top highlight

But, what if the data you're searching for is not in the in-memory layer? Well, don't worry, the LSM tree has a plan for that too! It moves on to the next step, which involves searching the on-disk layer, where the data is stored in what we call SSTables.

Now, this is where the bloom filter comes into play. Picture the bloom filter as a clever little assistant that helps the LSM tree narrow down its search. Before diving into each SSTable, the LSM tree consults the bloom filter to see if the data you're looking for might exist in a particular SSTable. *The bloom filter gives a probabilistic answer — it either says "the data might exist" or "the data definitely doesn't exist."*



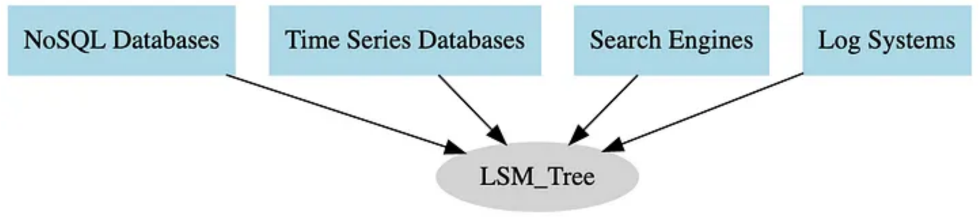
Read flow in LSM tree

If the bloom filter indicates that the data might exist in a specific SSTable, the LSM tree jumps into action again and starts searching that SSTable. It scans the sorted key-value pairs within the SSTable until it either finds the data you're looking for (yay!) or realizes it's not there.

On the other hand, if the bloom filter confidently declares that the data definitely doesn't exist in a particular SSTable, the LSM tree skips that SSTable and moves on to the next one. It's like the bloom filter acts as a reliable guide, showing the LSM tree which SSTables are worth exploring and which can be skipped, saving time and effort.

And that's how reading works in an LSM tree! The combination of checking the in-memory layer, leveraging the bloom filter, and searching the on-disk SSTables allows for efficient and speedy retrieval of data.

Diverse range of LSM use cases.



use cases of LSM tree

1. **NoSQL databases** One of the primary use cases of LSM trees is in NoSQL databases. These databases are designed to handle large amounts of unstructured or semi-structured data, and the LSM tree architecture aligns perfectly with their requirements. LSM trees offer excellent write performance, which is crucial for managing the high data ingestion rates typically encountered in NoSQL databases. The ability to efficiently handle write-intensive workloads makes LSM trees an ideal choice for storing and managing the vast amounts of data these databases handle.
2. **Time series databases** are another area where LSM trees shine. Time series data is characterized by its timestamped nature, where data points are associated with specific time intervals. LSM trees provide efficient storage and retrieval of time series data, thanks to their sorted structure. This allows for quick access to data points based on timestamps, enabling efficient analysis and querying of time-based data.
3. **Search engines**, LSM trees play a vital role in facilitating fast and accurate search operations. Search engines need to index and retrieve vast amounts of data quickly to provide relevant search results. LSM trees, with their ability to handle large datasets and provide efficient read operations, are a natural fit for search engine architectures. They allow for speedy retrieval of indexed data, making search queries lightning-fast and providing a seamless user experience.
4. **Log systems** LSM trees also find their place in log systems, such as those

used for real-time event streaming or log processing. In log systems, data needs to be written in an append-only manner, preserving the order of events. LSM trees excel in this scenario, as they offer efficient write operations by sequentially appending new data to the in-memory layer. The write-ahead log (WAL) ensures durability and recovery in case of system failures, further enhancing the reliability of log systems.

So, whether it's handling massive amounts of data in NoSQL databases, managing time series data efficiently, powering lightning-fast search engines, or enabling reliable log systems, LSM trees have established themselves as a go-to choice in various domains. Their unique characteristics, such as efficient write performance, scalability, and data compression, make them a valuable asset in today's data-intensive applications.

> *In case you have any doubts or suggestions leave them in the comments :)*

. . .

References

- *Patrick O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth O'Neil. (1996). The Log-Structured Merge-Tree (LSM-Tree). Acta Informatica. Retrieved from https://www.cs.umb.edu/~poneil/lsmtree.pdf*
- *WiredTiger. (n.d.). LSM. WiredTiger Documentation. Retrieved from https://source.wiredtiger.com/2.5.2/lsm.html*
- *Alibaba Cloud. (2021, June 1). Starting from Zero: Build an LSM Database with 500 Lines of Code. Retrieved from https://www.alibabacloud.com/blog/starting-from-zero-build-an-lsm-database-with-500-lines-of-code_598114*

Database System Design Interview Data Structures NoSQL

Written by Ankit Dwivedi

158 followers · 5 following

Follow

Software Engineer | Building largescale systems

All your favorite parts of Medium are now in one sidebar for easy access.

Responses (6)

Ishu

What are your thoughts?

Rachit Saxena
Aug 18, 2024 (edited)

Howz the read can be faster, provided that they have to search in memory first followed by different SS tables in worst case. Won't reads be faster in B-trees applied in SQl DBs ?

3 1 reply Reply

Akash Agarwal
Sep 5, 2023 (edited)

> periodically flushes the data from the in-memory layer to disk.

what's the frequency? is it configurable since it's an algo?

3 2 replies Reply

Rohansshetty
Oct 13, 2024

Does it internally uses BTree for indexing data?

What is a typical example of key in case of search engine implementations?

How is the read faster here compare to BTree?

What are the major LSM databases and which companies use them ?

4 2 replies Reply

See all responses

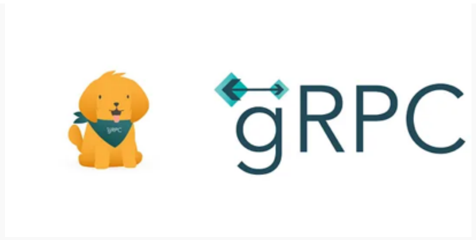
More from Ankit Dwivedi

Ankit Dwivedi

The Debugger: A Behind-the-Scenes Look at How It Works

Let's try to understand how debuggers work at a technical level and their inner workings

Sep 2, 2023 48



Ankit Dwivedi

gRPC Explained: Part 1 Introduction

Welcome to the first part of the blog series on gRPC. In this series, we will explore the ins...

Sep 24, 2023 109 2

Ankit Dwivedi

Rate Limiting and Load Shedding: Keeping Distributed Systems...

Ever wonder how your favorite website or app stays up and running, even when millions of...

May 14, 2023 34

Ankit Dwivedi

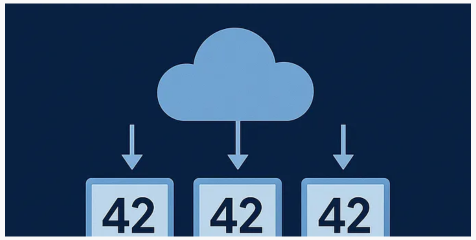
gRPC Explained: Part 2 Protobuf

In the previous blog we got a comprehensive introduction of gRPC, in this installment we...

Oct 8, 2023 31

See all from Ankit Dwivedi

Recommended from Medium



Ankit Kumar Srivastava

Design a Distributed Counter

System requirements

Jul 16 8

In Acing the Software Engineer Int... by LiveRunG...

Trie

Read full article for free: https://liverungrow.medium.com/trie-...

Jun 9 3

In JavaScript in Plain English by Debarshi Banerjee

Algorithm at Scale: HyperLogLog

# Demystifying HyperLogLog: A Probabilistic Cardinality Estimator

Jun 10 2

In Nerd For Tech by Nikhil Bhatnagar

System Design of a Ride Booking System (Uber/Ola/Rapido)

This article deals with the Hld and Lld of a ride booking system where we cover the ap...

Jun 15 6 1

In ITNEXT by Animesh Gaitonde

Solving Double Booking at Scale: System Design Patterns from Top...

Learn how Airbnb, Ticketmaster, and booking platforms handle millions of concurrent...

Oct 8 1.8K 22

Tech Guide

How I Built a Lightning-Fast Redis Cache—And What Broke at Scale

It started out perfect.

Aug 4 10

See more recommendations