Anatomy of Cassandra's Write Operation

Let’s dig deeper into the components involved in Cassandra’s write path.

We'll cover the following

- Commit log
- MemTable
- SStable

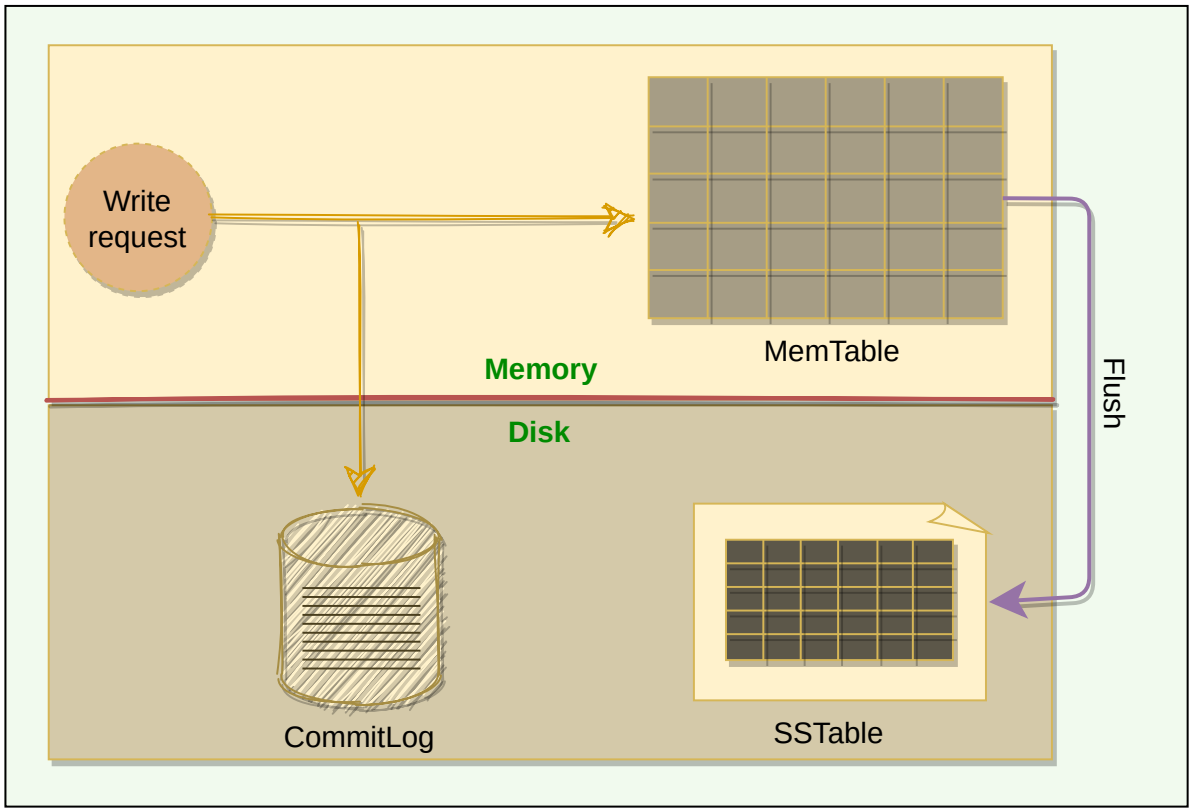
Cassandra stores data both in memory and on disk to provide both high performance and durability. Every write includes a timestamp. Write path involves a lot of components, here is the summary of Cassandra’s write path:

1. Each write is appended to a **commit log**, which is stored on disk.
2. Then it is written to **MemTable** in memory.
3. Periodically, MemTables are flushed to **SSTables** on the disk.
4. Periodically, compaction runs to merge SSTables.

Let’s dig deeper into these parts.

Commit log#

When a node receives a write request, it immediately writes the data to a commit log. The commit log is a write-ahead log and is stored on disk. It is used as a crash-recovery mechanism to support Cassandra's durability goals. A write will not be considered successful on the node until it's written to the commit log; this ensures that if a write operation does not make it to the in-memory store (*the MemTable, discussed in a moment*), it will still be possible to recover the data. If we shut down the node or it crashes unexpectedly, the commit log can ensure that data is not lost. That's because if the node restarts, the commit log gets replayed.



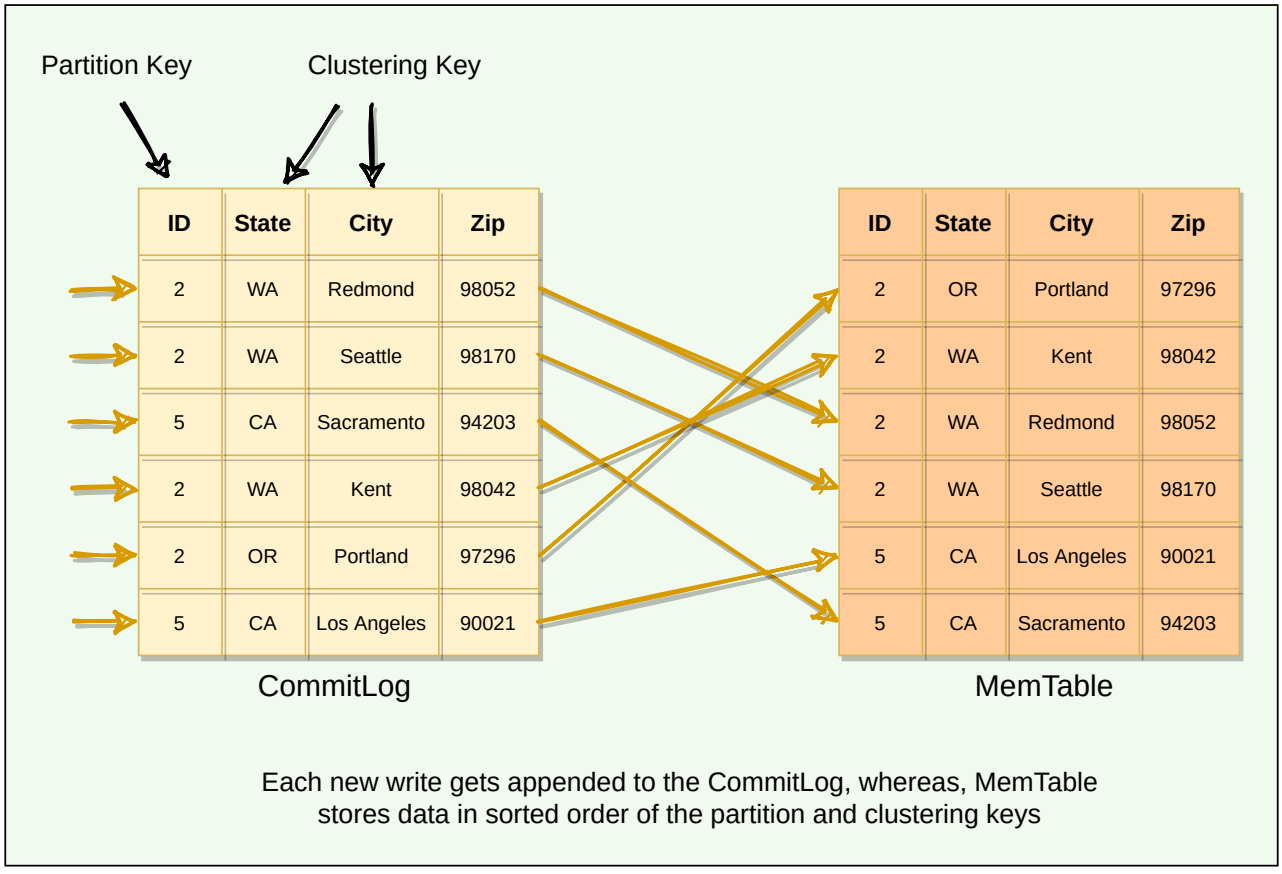
Cassandra's write path

MemTable#

After it is written to the commit log, the data is written to a memory-resident data structure called the MemTable.

- Each node has a MemTable in memory for each Cassandra table.

- Each MemTable contains data for a specific Cassandra table, and it resembles that table in memory.
- Each MemTable accrues writes and provides reads for data not yet flushed to disk.
- Commit log stores all the writes in sequential order, with each new write appended to the end, whereas MemTable stores data in the sorted order of partition key and clustering columns.
- After writing data to the Commit Log and MemTable, the node sends an acknowledgment to the coordinator that the data has been successfully written.



Storing data to commit log and MemTable

SStable#

When the number of objects stored in the MemTable reaches a threshold, the contents of the MemTable are flushed to disk in a file called SSTable. At this point, a new MemTable is created to store subsequent data. This flushing is a non-blocking operation; multiple MemTables may exist for a single table, one current, and the rest waiting to be flushed. Each SStable contains data for a specific table.

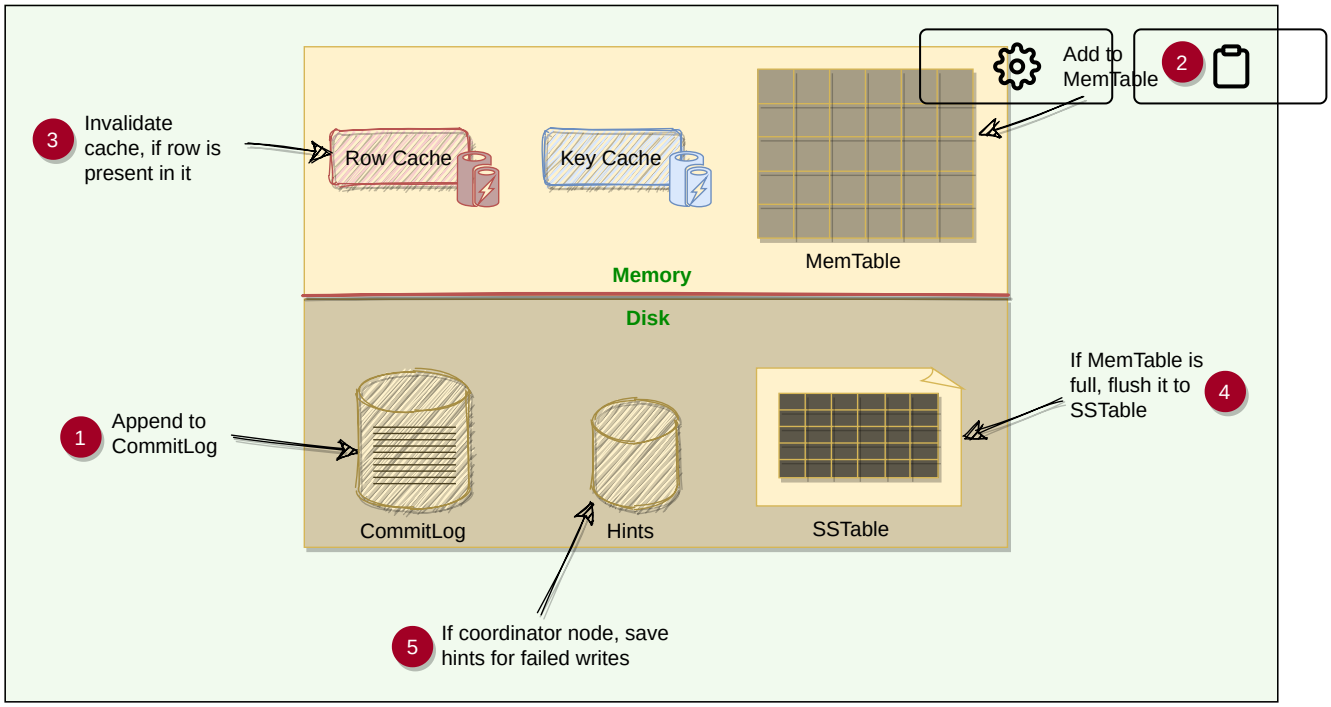
When the MemTable is flushed to SStables, corresponding entries in the Commit Log are removed.

Why are they called 'SSTables'? The term 'SSTables' is short for 'Sorted String Table' and first appeared in Google's Bigtable which is also a storage system. Cassandra borrowed this term even though it does not store data as strings on the disk.

Once a MemTable is flushed to disk as an SSTable, it is immutable and cannot be changed by the application. If we are not allowed to update SSTables, how do we delete or update a column? In Cassandra, each delete or update is considered a new write operation. We will look into this in detail while discussing Tombstones.

The current data state of a Cassandra table consists of its MemTables in memory and SSTables on the disk. Therefore, on reads, Cassandra will read both SSTables and MemTables to find data values, as the MemTable may contain values that have not yet been flushed to the disk. The MemTable works like a write-back cache that Cassandra looks up by key.

Generation number is an index number that is incremented every time a new SSTable is created for a table and is used to uniquely identify SSTables. Here is the summary of Cassandra's write path:



Anatomy of Cassandra's write path

← Back

Next →

Gossiper

Anatomy of Cassandra's Read Operati...

Mark as Completed

Report an Issue