Fault Tolerance and Compaction

Let's learn how BigTable handles fault tolerance and data compaction.

We'll cover the following

Fault tolerance and replication#

Fault tolerance in Chubby and GFS#

As discussed earlier (https://www.educative.io/collection/page/5668639101419520/5559029852536832/6338075595112448), BigTable uses two independent systems Chubby and GFS. Both of these systems adopt a replication strategy for fault tolerance and higher availability. For example, a Chubby cell usually consists of five servers, where one server becomes the master and the remaining four work as replicas. In case the master fails, one of the

replicas is elected to become the leader; thus, minimizing Chubby's downtime. Similarly, GFS stores multiple copies of data on different ChunkServers.

Fault tolerance for Tablet server#

BigTable's master is responsible for monitoring the Tablet servers. The master does this by periodically checking the status of the Chubby lock against each Tablet server. When the master finds out that a Tablet server has gone dead, it reassigns the tablets of the failing Tablet server.

Fault tolerance for the Master#

The master acquires a lock in a Chubby file and maintains a lease. If, at any time, the master's lease expires, it kills itself. When Google's Cluster Management System finds out that there is no active master, it starts one up. The new master has to acquire the lock on the Chubby file before acting as the master.

Compaction#

Mutilations in BigTable take up extra space till compaction happens. BigTable manages compaction behind the scenes. Here is the list of compactions:

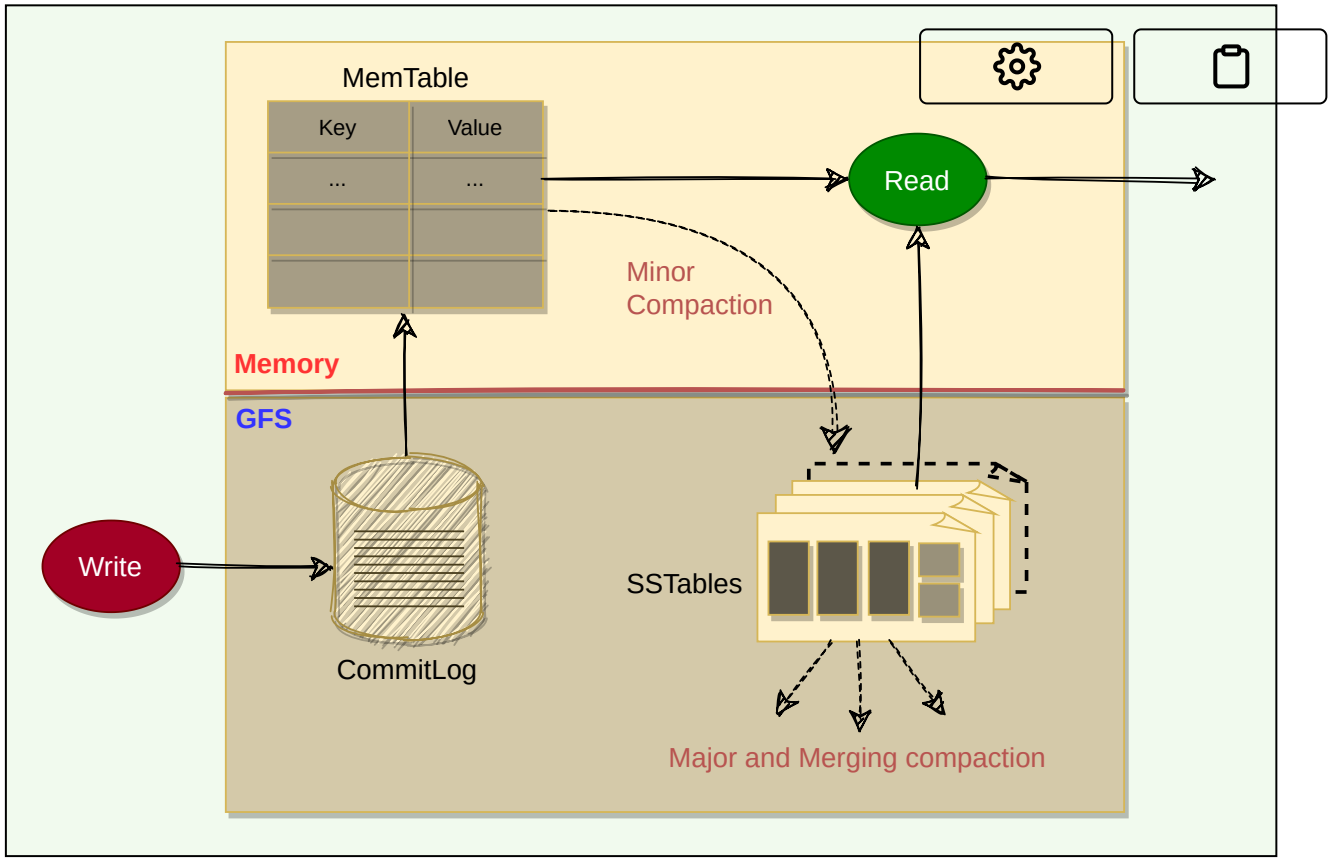
1. **Minor Compaction**: As write operations are performed, the MemTable grows in size. When the MemTable reaches a certain threshold, it is frozen, and a new MemTable is created. The frozen MemTable is converted to an SSTable and written to GFS. This process is called minor compaction. Each minor compaction creates a new SSTable and has the following two benefits:
    - It reduces the memory usage of the Tablet server, as it flushes the MemTable to GFS. Once a MemTable is written to GFS,

corresponding entries in the commit-log are also removed.

- It reduces the amount of data that has to be read from the commit log during recovery if this server dies.

2. **Merging Compaction** — Minor compaction keeps increasing the count of SSTables. This means that read operations might need to merge updates from an arbitrary number of SSTables. To reduce the number of SSTables, a merging compaction is performed which reads the contents of a few SSTables and the MemTable and writes out a new SSTable. The input SSTables and MemTable can be discarded as soon as the compaction has finished.

3. **Major Compaction** — In Major compaction, all the SSTables are written into a single SSTable. SSTables created as a result of major compaction do not contain any deletion information or deleted data, whereas SSTables created from non-major compactions may contain deleted entries. Major compaction allows BigTable to reclaim resources used by deleted data and ensures that deleted data disappears from the system quickly, which is important for services storing sensitive data.



Major, minor, and merging compaction in BigTable

← Back
The Life of BigTable's Read & Write O...

Next →
BigTable Refinements

Mark as Completed

Report an Issue