Summary –

Pebble DB is a high performance key-value store system that is built on a new data structure called as Fragmented Log-Structured Merge Trees (FLSM). The fragmented log structured merge trees are based on the concept of making guards which breaks keys into small fragments when performing compaction which help in improving the write throughput and reducing the write amplification that occurs in LSM – key value store. Based on the concept it is similar to LSM trie as it also breaks down keys based on hash function and compact them to new sub levels. Pebble DB also helps in performing range search which was not possible with LSM trie since it uses hash function.

Questions

1. “Figure 2 illustrates compaction in a LSM key-value store.” Please use the example to explain why compaction operation can be very expensive.

![Figure 2: LSM Compaction. The figure shows sstables being inserted and compacted over time in a LSM.](image-url)
The key-value store contains two sstables in Level 1 initially. Let us assume that Level 0 is configured to hold only one sstable at a time; when this limit is reached, compaction is triggered. At time t1, one sstable is added, and a compaction is triggered is at time t2. Similarly, sstables are added at time t3 and t5 and compactions are triggered at time t4 and t6. When compacting an sstable, all sstables in the next level whose key ranges intersect with the sstable being compacted are rewritten. Since the key ranges of all Level 0 sstables intersect with key ranges of all Level 1 sstables, the Level 1 sstables are rewritten every time a Level 0 sstable is compacted. This rewriting of sstables at each compaction leads to a high write amplification.

2. “Instead of rewriting the sstable, FLSM’s compaction simply appends a new sstable fragment to the next level.” Compared to the LSM-tree in-place rewriting, this appending is more efficient. However, what’s the tradeoff (any negative impact of the appending)?

Compared to the LSM tree which performs in-place rewriting of all the keys when compaction is done, appending a new sstable fragment is more efficient as it helps in improving the write performance and also leads to lesser write amplification.

However this leads to poor read and range performance since it will have a large number of sstables. Also multiple sstables contain the same key and can have overlapping key ranges. This leads to bad read performance.

It also leads to the false positive scenario where an element is not present but the bloom filter returns that it is present.

3. “FLSM performance is significantly impacted by how guards are selected.”
   Could you give a criterion of being good guards?

The points to be considered for a being a good guard –

- Guards are generated based on the guard probability function gp (key, i).
- Guards should be selected based on the density and spread of keys. For example consider the range of 1 – 100 where keys are more densely populated between 20 – 50 then guards should be selected in such a way that there should be more number of guards between them so that it is more evenly spaced.
- Guard intervals should have almost equal number of sstables so that it structure is more balanced.
- The number of guards increases as the level number increases. For example Level 0 would have lower number of guards and Level 5 would have a large number of guards.
4. “Guard probability $gp(\text{key, i})$ is the probability that key becomes a guard at level $i$.” Why is the probability a function of level number?

The main reason why the guard probability is a function of level number is cause the number of keys and sstables are higher as the level number is higher. For example – Level 5 would have a large number of keys and sstable fragments compared to that of Level 0.

The number of keys is higher as the level increases cause during compaction the number of keys and fragments being generated keeps increasing at each level. Hence to maintain a balanced structure and so that guard intervals being almost equal the guard probability function needs to be based on the level number.

5. Guards are continuously generated with key insertions. And “We note that in many of the workloads that were tested, guard deletion was not required.” Could you solve the contradiction? What’s the consequence of not conducting guard-deletion operations in a store keeping admitting new keys?

A large number of keys are inserted into the structure and this causes a huge number of guards to be created. But in cases keys are deleted and guards become empty this causes an imbalance in the structure. Guard deletion doesn’t really affect the read performance as the read and range queries just skip the guards that are empty. But if we would have a large number of guards compared to the number of keys this would be a major imbalance and would not maintain the number of sstables between guard intervals. Having large number of guards would also lead to large number of sstable fragments being present. Hence guard deletion should be considered to be a mandatory procedure to maintain balance.