Building PebblesDB using FLSM

First, we can see the first black line above the memory and below the storage device.

The blue block in the memory is called Immutable Table.

Each blue block in the storage device is an SSTable.

The left side indicates which level the sstable is currently at.

When we are short of memory space.

Then it will become SSTable to write to the storage device.

Then when we level 0 want to do Compaction.

The sstable will be cut according to the guared.

So we can see that the sstable 10 and 210 is cut by guared 101 into two sstables, 10 and 210.

And 10 this sstable does not merge with 1, 100 this sstable

The 210 sstable does not combine the 200 and 400 sstable as well.

Then do the sstable for level 0 250 and 500

Same as before

Split by guard 401 into 250 and 500 two sstable

250 will be placed in the guard101 to guard 401 area.

550 will be placed in the area after guard 401

When you finally finish Compaction, it will look like this

YCSB Performance. The figure shows the throughput (bigger is better except for Total-IO bars) of different key-value stores on the YCSB Benchmark suite run with four threads. PebblesDb gets higher throughput than RocksDB on almost all workloads, while performing 2 times lower IO than RocksDB.

Load-A and Load-E do 50M operations each, all other workloads do 10M operations each.

Figure presents the results: PebblesDb outperforms both RocksDB and HyperLevelDB on write workloads, while obtaining nearly equal performance on all other workloads.

Overall, PebblesDb writes 50% less IO than RocksDB.

On write-dominated workloads like Load A and Load E, PebblesDb achieves 1.5 to 2 better throughput due to the faster writes offered by the underlying FLSM data structure.

For the read-only Workload C, PebblesDb read performance is better than other key-value stores due to the larger sstables of PebblesDb

For the range-query-dominated Workload E, PebblesDb surprisingly has performance close (6% overhead) to the other key-value stores.

InWorkload F, all writes are read-modify-writes: the workload does a get() before every put() operation.

As a result,the full write throughput of PebblesDb is not utilized, resulting in performance similar to that of other key-value stores.

------

Figure presents the results.

We find that both RocksDB and PebblesDb significantly outperform

Wired Tiger on all workloads, demonstrating why LSM-based stores are so popular. While RocksDB performs 40% more IO than Wired Tiger, PebblesDb writes 4% lesser IO than Wired Tiger.

We investigated why PebblesDb write throughput is not 2× higher than RocksDB as in the YCSB benchmark.

As inHyperDex, MongoDB itself adds a lot of latency to each write (PebblesDb write constitutes only 28% of latency of MongoDB write) and provides reques.

---

This paper presents PebblesDb, a high-performance keyvalue store that achieves low write amplification, high write throughput, and high read throughput simultaneously.

PebblesDb outperforms widely-used stores such as RocksDB on several workloads.

PebblesDb is built on top of a novel data structure, Fragmented Log-Structured Merge Trees, that combines ideas from skip lists and log-structured merge trees.

The design of the novel FLSM data structure combines ideas from skip lists and log-structured merge tree.

Experimental results demonstrate that PebblesDB dominates LSM-based stores in many workloads.

PebblsesDB is a high-performance key-value store that achieves low write amplification, high write throughput, and high read throughput simultaneously.