CS2309 Assignment 2

Tan Yee Jian (A0190190L)

September 9, 2021

1 Problem Statement

[SR12] Log-structured Merge Trees (LSM Trees) have a good write throughput, but high read amplification. This is the opposite of the commonly used update-in-place storage systems (such as B-Trees) which have great read performance, but low write throughput. This is costly because as observed at Yahoo!, the low latency workloads that emphasize on writes are increasing from 10-20% to about 50%, which is getting unsuitable for update-in-place systems. LSM Trees are a good candidate for write-intensive tasks, but modifications need to be made so it has a better real-world performance.

1.1 Solution

Therefore, the researchers have come up with *bLSM Trees*, a LSM Tree with the advantages of B-Trees. This is done by adding Bloom filters in LSM Trees (hence the *b* in *bLSM*) to increase index performance and replacing the merge scheduler in LSM Trees with the new "spring and gear" merge scheduler. When measured under a new performance metric, *read fanout* that is claimed to better characterize real-world index operations, *bLSM* outperform B-Trees in most cases.

2 Pros and Cons

2.1 Pros of *bLSM*

2.1.1 Using Bloom Filters

Using Bloom filter reduces read amplification by trimming the search tree, from N to $1 + \frac{N}{100}$. This allows *bLSM* to have near-optimal read performance.

Since LSM Trees only have a small tree in memory that holds temporary data, reading a value might require searching through the different levels of trees. Bloom filters make this faster by giving an instant way of checking if a key-value pair is in a given tree.

This drastically decreases read amplification from N to $1 + \frac{N}{100}$, although by using 10 bits to encode a key leads to about 1% false positive rate and increases memory utilization by 5%, but it solves the fundamental disadvantage of LSM Trees of having slow random reads.

2.1.2 "Spring and Gear" scheduler

The new "spring and gear" scheduler solves the problem brought about by *snowshoveling*, so application writes are not blocked for long period of times, improving write latencies.

Snowshovelling is also known as *tournament sort* or *replacement-selection* sort, which improves throughput for sequential workloads at the cost of blocking application writes for an arbitrarily long period of time. It also increases the data each merge from C_0 to C_1 consumes.

The autors modified the gear scheduler which interacts badly with *snowshovelling*, into the "spring and gear" scheduler. This new scheduler restricts the utilization of C_0, the top-level tree between a upper and lower limit, allowing space for the new scheduler to absorb merge spikes due to overfull trees.

2.2 Cons

2.2.1 Merge Threads

Merging in *bLSM* Trees is difficult to implement, solving the issues using mutex introduces additional concurrency overheads.

First, batch tree iterator operations are needed to amortize page pins and mutex acquisitions in the merge threads. Secondly, merge scheduling suffer from the cost of acquiring a coarse-grained mutex for each merged tuple of page. Even though these problems can be solved, there are unseen concurrency bottlenecks due to the complexity of these systems. The authors ran an experimental setup of the bLSM trees on *Yahoo! Cloud Serving Benchmark Tool*, and at 100% blind writes, *bLSM* got a lower expected throughput on SSDs than on HDDs, which was at around $33,000$ operations per second. Since each SSD provides 285MB/sec sequential reads and writes, compared to $110 - 130$MB/sec of HDD, the expected $2 - 3$x higher throughput on SSDs have not been achieved due to the concurrency bottlenecks.

2.2.2 Bloom filters and Recovery

Even though existing LSM Trees implement write-ahead logs for consistency upon crash, the bloom filters are too huge to write to disk and therefore complicate crash recovery.

Bloom filters are known to be compact and efficient, but the challenge lies on recovery of database systems that utilize bloom filters. The bloom filters are small, but not small enough to be synchronously written to disk by block writers. Since they are not persisted to disks, it poses a problem in data recovery during crashes.

3 Conclusion

bLSM Trees have drastically improved the performance of LSM Trees by solving its fundamental shortcoming: high read amplification by adding Bloom Filters and using a "Spring and Gear" merge scheduler. However, there are multiple implementation disadvantages, first being complicated to implement and lower-than-theoretical efficiency due to concurrency bottlenecks, as well as recovery with bloom filters still needs further work and testing the implementation, which the authors omitted in their experiments.

References

[SR12] Russell Sears and Raghu Ramakrishnan. "BLSM: A General Purpose Log Structured Merge Tree". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. SIGMOD '12. Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, pp. 217–228. ISBN: 9781450312479. DOI: 10.1145/2213836.2213862. URL: https://doi-org.libproxy1.nus.edu.sg/10.1145/2213836.2213862.