Breaking the LSM Tail-latency Barrier by Revisiting Growth Factors

Giorgos Xanthakis
ICS-FORTH & University of Crete
Heraklion, Greece
gxanth@ics.forth.gr

Antonis Katsarakis
Huawei
Edinburgh, Scotland
antonios.katsarakis@huawei.com

Giorgos Saloustros
ICS-FORTH & University of Crete
Heraklion, Greece
gesalous@ics.forth.gr

Angelso Bilas
ICS-FORTH & University of Crete
Heraklion, Greece
bilas@ics.forth.gr

Abstract

The datacenter storage stack is able to handle a wide range of workloads, including both read and write operations. The foundation to manage these workloads is the log-structured merge tree (LSM-tree). LSM-trees support high write throughput and acceptable read performance but suffer from long write stalls and high tail latency. To provide acceptable user-facing tail latency, current LSM-tree implementations use incremental compaction to stall for a limited time, usually in the order of a few seconds. To mitigate the long write stalls in production environments, companies overprovision their infrastructure, to prevent a large number of requests reaching a single server [1].

In this work, we analyze the causes of tail latency in LSM KV stores employing incremental compaction. Due to the nature of the LSM-tree structure, the size of each level increases by a factor of f (growth factor) from the previous level. This results in N levels with the size of the n-th level being 90% of the total storage capacity and the $N - 1$ levels being the 10% of the total capacity. Additionally, because incremental compaction partially frees space from each level, it always operates on levels that are (nearly) full, except for the last level. As a result, the amount of work incremental compaction performs is $N - 1$ compactions for every L_0 compaction, resulting in the common path in long compaction chains with multiple GBs of work.

Current LSM-based systems create not only long compaction chains but also fat compaction chains. With the typical Sorted String Tables (SST) size (64 MB), and the growth factor for each compaction, the average amount of work for each SST is f times the SST size. Traditionally, LSM-based systems require compacting lower levels to create space for higher levels, forming long chains of compactions in this process. Thus, systems using incremental compaction still result in high tail latency and long write stalls to free memory in L_0.

LSM based systems are optimized for devices with a few hundreds IOPs (Hard Disk Drives) that was the right choice when HDDs were the dominant storage technology. However, the storage landscape has changed, and new storage technologies have emerged, such as Non-Volatile Memory Express (NVMe) SSDs. These devices are designed for millions of IOPs and provide high performance with small I/O sizes. LSM systems are not designed to exploit the capabilities of fast storage devices, such as NVMe SSDs, and fully utilize their characteristics, such as high performance with smaller I/O sizes and millions of IOPs.

Today, the state-of-the-art LSM KV store that uses incremental compaction is RocksDB [3]. However, according to engineers in Meta [1, 2], the tail latency of RocksDB is not a concern for them for two reasons. Firstly, Meta overprovisions each RocksDB instance, maintaining a low request rate per second (10000 queries per second) on the average case. Secondly, on average, Meta utilizes 40% of the storage capacity [2], thus not observing long write stalls due to overprovisioning.

In this paper, we present *sLSM*, a new LSM-tree design that reduces write stalls and tail latency by exploiting the characteristics of NVMe SSDs. *sLSM* achieves the following design goals: (1) decrease the amount of work per compaction with smaller SSTs, (2) maintain the same number of levels with state of the art approaches using a variable growth factor between levels, and (3) use variable size SSTs to control I/O amplification for levels that employ variable growth factor.

We implement *sLSM* as an extension in RocksDB and evaluate its efficiency examining and measuring tail latency with different workloads. Preliminary results show that *sLSM* when compared to RocksDB reduces 99th tail-latency by 4.5×, for YCSB Load A. Finally, for mixed workloads (YCSB Run A), it reduces write tail latency by 4.8× and read tail latency by 12.5×.

We thankfully acknowledge the support of the European Commission under project DaFab (GA No.101128693).

References

[1] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H. C. Du. Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies*, FAST'20, page 209–224, USA, 2020. USENIX Association.

[2] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications. *ACM Trans. Storage*, 17(4), oct 2021.

[3] Facebook. Rocksdb. http://rocksdb.org/, 2018.