Reducing Compaction Process of LSM with LvlupDB

Jinghuan YU
Your Institution

Second Name
Second Institution

Abstract

abstract abstract

1 Introduction

With the development of big data technology, the volume and the throughput of modern application system has greatly increased. To improve writing performance of the application system, Google proposed Bigtable [5] storage system in 2008 and introduce the great advantage of LSM data structure. Log Structured Merge Tree, also known as LSM [9], is a specially designed data structure optimizing warm writing performance proposed in 1996. Due to its high writing performance and simple index structure, LSM gets widely used in big data storage systems, in-memory databases and embedding databases like Hbase [2], Cassandra [1],MongoDB [3], LevelDB and SQLite. LSM uses the in-memory buffer to achieve high writing performance, it aggregates a batch of writing operations and flashes them into the storage system once the memory space is limited. This buffer and batch technique force the writing operations into sequential access to the file system. Although LSM can take advantage of more efficient read and write on the file system, it also introduces the compaction process to keep the entire order of the persistent record. This additional software-level overhead can cause significant performance jitter and write amplification.

There are two basic resources while analyzing the performance and bottleneck of a storage system: 1) the computation resources; and 2) the data transmission resources referred to as I/O $(Input/Output)$ resources; The Figure 1 shows the

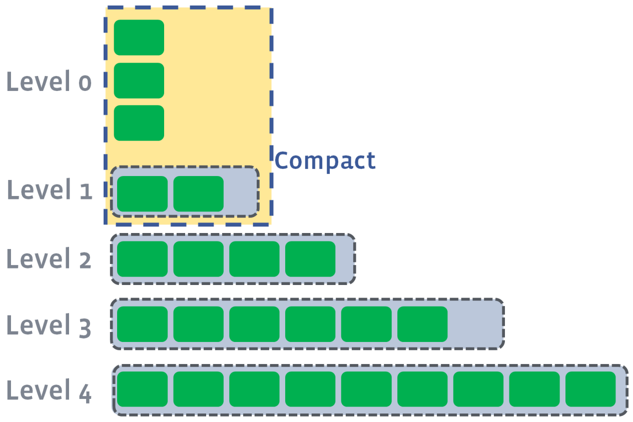

Figure 1: A typical compaction process starts with choosing the files need compaction most. Then the system will decode the persistent data into memory processable format and merge all the entries inside these files. At last, create a new file to store the processed entires and remove outdated files. It can be concluded as three main steps: 1. file choosing step; 2. entry merging step; 3. wrtie out step.

basic steps of compaction process, unlike other operations in LSM-based system, most of processing in compaction will consume both two resources, especially the entry merging step. In merging step, the system firstly read a period of data from sorted string tables (SST), decode it from the file format into in-memory format, then compare it with other entires.

There are many recent proposals focus on reducing the impact of compaction and improving throughput by using new storage materials [6, 8, 10–14]. Byte-addressed and fast nonvoltile memory such as 3d xpoint [7] can bring following superiorities: 1) hgiher random access performance; 2) lower in-place update latency; and 3) better application-level parallelism. With optimized redesigning of LSM [8], LSM structure can achieve higher throughput. But the traditional architectures of LSM [4] can not reduce the performance impact of compaction even after changing the file choosing strategy, balancing the sorted files.

2 Background

In the content above, we discuss the detail of Compaction, and to get a more intuitive impression about the proportion and difference between these different compactions. Using benchmark to simulate a high-pressure environment, and during the experiment we find the speed of memtable compaction will continue grow to a limited boundary according to both the memory throughput and IO stack throughput.Before we can detailed analyze the overhead of compaction process, we should first figure out the common architecture of LSM-based databases.

3 Motivation

4 Design

5 Evaluation

6 Related Work

7 Acknowledgements

References

[1] Apache cassandra. http://cassandra.apache.org/.

[2] Apache hbase – apache hbase^{TM} home. https://hbase.apache.org/.

[3] Mongodb download center | mongodb. https://www.mongodb.com/download-center/community.

[4] Small datum: Name that compaction algorithm. https://smalldatum.blogspot.com/2018/08/name-that-compaction-algorithm.html.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[6] Yen-Ting Chen, Ming-Chang Yang, Yuan-Hao Chang, Tseng-Yi Chen, Hsin-Wen Wei, and Wei-Kuan Shih. Kvftl: Optimization of storage space utilization for key-value-specific flash storage devices. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pages 584–590. IEEE, 2017.

[7] Rob Crooke and Mark Durcan. A revolutionary breakthrough in memory technology. *Intel 3D XPoint launch keynote*, 2015.

[8] Sudarsun Kannan, Nitish Bhat, Ada Gavrilovska, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. Redesigning lsms for nonvolatile memory with novelsm. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 993–1005, 2018.

[9] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

[10] Zhaoyan Shen, Feng Chen, Yichen Jia, and Zili Shao. Didacache: A deep integration of device and application for flash based key-value caching. In *FAST*, pages 391–405, 2017.

[11] Hui Sun, Wei Liu, Jianzhong Huang, and Weisong Shi. Co-kv: A collaborative key-value store using near-data processing to improve compaction for the lsm-tree. *arXiv preprint arXiv:1807.04151*, 2018.

[12] Sung-Ming Wu, Kai-Hsiang Lin, and Li-Pin Chang. Kvssd: Close integration of lsm trees and flash translation layer for write-efficient kv store. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*, pages 563–568. IEEE, 2018.

[13] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. Hikv: A hybrid index key-value store for dram-nvm memory systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 349–362, 2017.

[14] Jiacheng Zhang, Youyou Lu, Jiwu Shu, and Xiongjun Qin. Flashkv: Accelerating kv performance with open-channel ssds. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):139, 2017.