LSM-tree Compaction Acceleration Using In-storage Processing

Minje Lim, Jeeyoon Jung, Dongkun Shin
Department of Electrical and Computer Engineering,
Sungkyunkwan University, Suwon, Korea
scvhero9607@gmail.com, jiyun710@gmail.com, dongkun@skku.edu

Abstract

Log structured merge tree (LSM-tree) is widely used to implement key-value stores. To maintain the structure of the LSM-tree, it is necessary to perform compaction to remove duplicated entries. Compaction can delay the user's write operation and affects performance. If In-storage processing is applied to compaction, data traffic between Host-Storage can be reduced and tasks can be processed quickly to improve system performance. If compaction is efficiently processed by applying in-storage processing, overall DB performance can be improved. In this paper, we designed an IP that can accelerate the compaction and integrated it into the Cosmos+ OpenSSD.

Keywords: LSM-Tree, Key-Value Store, Compaction, FPGA, OpenSSD, In-storage processing

1. Introduction

Log-structured Merge Tree (LSM-tree) [1] based key-value store is widely used to replace existing RDBMS in fields such as Big Data Processing and there are examples such as LevelDB [2], Cassandra [3], and RocksDB [4]. Records inserted by the user are stored in the memtable, an in-memroy structure, and when the size of the memtable reaches a threshold, the memtable flush operation is performed to convert the memtable to an SSTable and store it in storage. SSTable is managed in several levels, and compaction is performed to create a new SSTable by merging existing SSTables to remove duplicate records between SSTables. Compaction may delay the user's put operation, which is a major cause of write performance degradation. In-storage processing is a technique that uses the CPU or FPGA inside the storage device to process the host's work instead. In-storage processing can improve application performance by reducing data traffic and increasing processing speed. In this paper, the following work was performed to solve the write delay problem due to compaction.

- We Designed and implemented IP that can accelerate the Compaction of LSM-Tree
- Integrate Compaction IP into Cosmos+ OpenSSD[5] and LevelDB to offload compaction.
- Increased compaction performance by up to 63% and end-to-end performance by up to 65%

2. Background

2.1 LSM-tree SSTable and compaction

SSTable is a format used to save records as files in storage in LSM-tree. It consists of a data block in which actual records are recorded, an index block in which the size of each data block and location information in the file are recorded, and a footer containing metadata information. When an SSTable in DRAM is written to level 0 of storage through *memtable flush*, records with duplicate keys may occur between the newly written SSTable and the existing SSTable. As the number of SSTables with overlapping key ranges increases, the number of SSTables that need to be checked to find a specific key increases, and *compaction* must be performed to solve this problem. Compaction is performed when the number of SSTables existing in level i reaches a threshold value, and by merging SSTables with overlapping key ranges existing in level i and level i+1, a new SSTable is recorded in level i+1.

2.2 In-Storage processing

Biscuit [6] proposed a framework for ISP development so that data processing applications are performed in the form of distributed processing in the host and SSD. YourSQL [7] applies Biscuit to the SQL database to offload query processing to the SSD and performs filtering using dedicated hardware. Catalina [8] used a Computational Storage Device (CSD) equipped with a high-performance ARM CPU to show the effectiveness of ISP's distributed processing workload.

2.2 LSM-tree offloading

Teng Zhang et al. offloaded the scan operation of LSM-Tree based KV store to computational storage.[9] A separate header is added to the data block. A header is added to each data block to remove IP metadata access. On the other hand, we used original LevelDB SSTable format.

Wei Cao et al. and X. Sun et al. offloaded the LSM-tree compaction to the FPGA PCIe accelerator. [10,11] To offload compaction using an external accelerator, additional host DRAM copy is required between the storage and the accelerator, but we integrated the FPGA accelerator and storage to remove the additional host copy overhead.

3. Design

3.1 IP Design

Due to the characteristic of level 0 where key ranges in SSTables can overlap, Level 0 - Level 1 compaction offloading requires multiple SSTable decoders for simultaneous comparison of Level 0 SSTables. While this causes an increase in FPGA resource and memory usage, level 0 SSTables have a high probability of remaining in the host page cach, so there is relatively little benefit from offloading. So we designed an IP capable of offloading level 1 – level 2 and higher compaction that requires only two decoders.

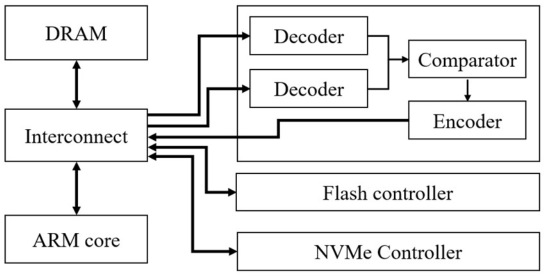

Fig. 1. Overall In-storage processing system architecture including compaction IP

Compaction IP consists of decoder, comparator, and encoder. The decoder converts the SSTable into a key-value stream format, the comparator compares the keys and transmits the smaller key-value pair first, and the encoder converts the key-value stream into the SSTable. Each component is connected through the AXI4-Stream interface and designed to operate asynchronously with each other, and the bus width connecting Compaction IP and DRAM was set to 64bit, the maximum value provided by the ARM core of Cosmos+.

Decoder is composed of SSTable Index decoder, data decoder, and Input/Output queue. Index decoders extract information necessary to process data blocks by parsing internal metadata. The data decoder converts the data block into a key-value stream and sends it to the comparator. When SSTable is loaded from NAND to DRAM, the address of the DRAM buffer and the length of the SSTable are enqueued in the input queue, and the decoder sequentially processes the SSTables inserted into the input queue. When SSTable decoding is finished, the buffer address is enqueued in the output queue so that the buffer can be reused. The data decoder issues a burst transaction using DMA engine for efficient transmission because it accesses DRAM buffer sequentially, but the index decoder directly issues a transaction in units of bytes because it accesses DRAM randomly. Since the index decoder and data decoder operate asynchronously, the time it takes to parse metadata overlaps with data-block parsing.

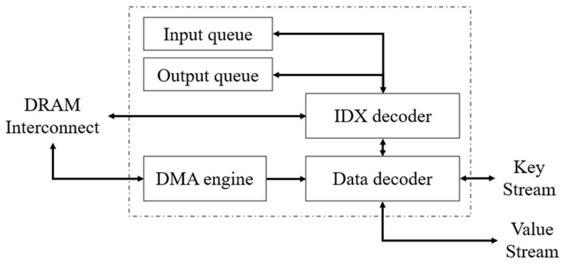

Fig. 2. Decoder structure

Comparator compares the keys received from each decoder to select a smaller key, and if the key is a non-overlapping key, the comparator delivers the record to the encoder.

Encoder converts the record received from Comparator into SSTable and writes the result to DRAM. SST Encoder receives information from data encoder, writes metadata and temporarily stores it in BRAM. When the size of SSTable exceeds a threshold and files need to be separated, the metadata stored in BRAM is flushed to DRAM and the length of the created SSTable After enqueue to the result queue, it starts writing a new SSTable.

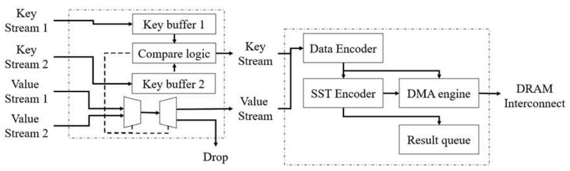

Fig. 3. Comparator, Encoder structure

3.2 SW Design

We modified LevelDB to save the SSTable in the continuous logical address without a file system.

When compaction is offloaded, the LBA and size of the target SSTable is transmitted to the SSD by NVMe IO command. Compaction IP is controlled by FTL (Flash Translation Layer) of firmware, and performs NAND-DRAM data transfer, Compaction IP monitoring, and result metadata management.

4. Experiment

Host PC is i7 4790 CPU, 12GB DRAM. Cosmos+ uses 4ch 4way setup, and Compaction IP operates at 200Mhz. Table 1 shows the FPGA resource utilizatio n consumed by Compaction IP. For the experiment, t he fillrandom workload of db_bench was used, and th e size of the SSTable was set to 32MB and the numb er of records was set to 100 million.

Table 1: Resource Utilization

	LUT	FF	BRAM
Decoder	6484 x 2	2390 x 2	0
Comparator	1010	583	0
Encoder	32192	11212	49
etc	6199	1725	5
Total (%)	21.0	3.6	9.9

4.1 Compaction Throughput

Table 2 shows the difference between compaction in the host CPU and offloading in the SSD. When the host processes compaction, the throughput is constant, but when offloading, the performance slightly increases as the value size increases.

Table 2: Compaction Throughput

Value size	128	256	512	1024
Host-only	106.7	106.3	106.0	106.3
Offloading	159.7	171.4	172.7	175.5

4.1 End-to-end Performance

Figure n shows the difference in compaction performance according to the value size. As the size of the value increases, the effect of compaction increases due to the increase in the size of the entire data set. Also, due to parallel execution of compaction and memtable flush, higher performance improvement than pure compaction performance improvement was obtained.

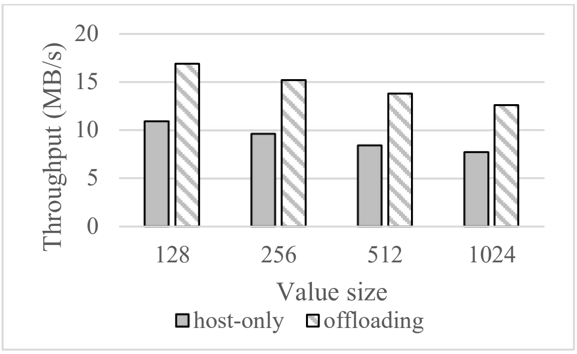

Fig. 4. End-to-end performance with different value size

5. Conclusion

In this paper, we designed and implemented FPGA IP that can accelerate the compaction of LSM-Tree, and implemented In-storage processing system through integration with Comsos+ OpenSSD platform. Compared to the host-only version, the compaction performance is improved by up to 64% and the end-to-end performance by up to 65%.

Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.IITP-2017-0-00914, Software Framework for Intelligent IoT Devices)

References

[1] P. O'Neil et al, "The log-structured merge-tree (LSM-tree)". Acta Informatica, 33(4):351–385, 1996.
[2] Google, LevelDB, https://github.com/google/leveldb
[3] Apache, Cassandra, cassandra.apache.org
[4] Facebook, RocksDB, https://github.com/facebook/rocksdb
[5] Y. H. Song et al., "Cosmos+ openssd: A nvme-based open source ssd platform," Flash Memory Summit, 2016.
[6] B. Gu et al., "Biscuit: a framework for near-data processing of big data workloads", Proceedings of the International Symposium on Computer Architecture, 2016.
[7] I. Jo et al., "YourSQL: a high-performance database system leveraging in-storage computing", Proceedings of the VLDB Endowment, 2016.
[8] M. Torabzadehkashi, et al., "Computational storage: an efficient and scalable platform for big data and hpc applications," Journal of Big Data, 2019.
[9] Wei Cao et al, "POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database", In: 18th USENIX Conference on File and Storage Technologies (FAST20). 2020, pp. 29-41
[10] Teng Zhang et al, "FPGA-Accelerated Compactions for LSM-based Key-Value Store", In: 18th USENIX Conference on File and Storage Technologies (FAST20). 2020, pp. 225-237
[11] X. Sun et al, "FPGA-based Compaction Engine for Accelerating LSM-tree Key-Value Stores", In: 2020 IEE 36th International Conference on Data Engineering (ICDE). 2020, pp. 1261-1272