Application-Driven Flash Management: LSM-tree based Database Optimization through Read/Write Isolation

Heerak Lim
Seoul National University
rockylim@snu.ac.kr

Abstract

The role of high-performance storage devices is becoming increasingly important in the web-scale infrastructure. In particular, next-generation storage devices such as Non Volatile Memory Express (NVMe) based solid state devices (SSDs) are being actively introduced to data centers. However, applications running on data centers do not take into account the characteristics of these high-performance storage devices. We focus on the performance degradation of the mixed workload of reads and writes when using high-performance storage devices in the log-structured merge tree (LSM-tree) based database systems. To address this problem, we propose application-driven flash management scheme to isolate read/write operation.

ACM Reference format:
Heerak Lim. 2018. Application-Driven Flash Management: LSM-tree based Database Optimization through Read/Write Isolation. In *Proceedings of Middleware'18, Rennes, France, December 2018,* 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recently, as the cloud market is growing rapidly, data centers are actively introducing next-generation high-performance storage devices. Database systems are the most typical applications that should provide a highly reliable service with high-performance storage devices in data centers. Especially, a lot of modern I/O intensive database applications including RocksDB [4] and Cassandra [1] use log-structured merge trees (LSM-tree) [9]. However, these database systems do not fully utilize the capabilities of the next-generation storage devices such as NVMe because it does not reflect the characteristics of the recent high-performance storage devices. As a result, the database system provides performance that does not take full advantage of the high bandwidth of the storage device.

The next-generation storage devices, such as NAND-based NVMe SSDs, have good performance for read operations, but they perform poorly when read requests are mixed with write operations. Such performance degradation is caused by internal flash translation layer (FTL) processes such as garbage collection (GC) within the SSD delay the I/O operations [5, 8] and different response time between reads and writes. Furthermore, with LSM-trees, compaction causes read and write requests to be intermixed very frequently to the storage devices. **The main contributions of this work** are to introduce a preliminary experimental evaluation of such performance drop and present a design that overcomes the existing problem through application-driven flash management.

We perform the following experiments on the machine with 512 GiB Samsung 960 Pro NVMe SSD. To evaluate raw device performance and database application, we use FIO benchmark and

Middleware'18, Rennes, France
2018. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

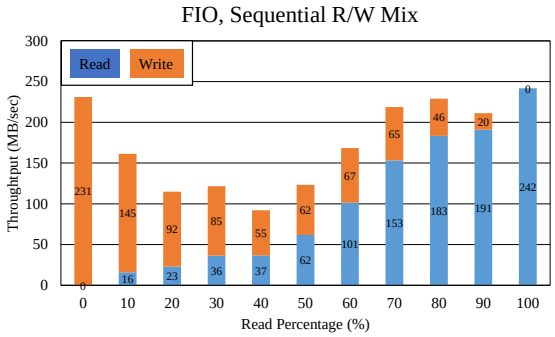

Figure 1. FIO benchmark - read and write throughput comparison based on read percentage: the read and write performance drops sharply instead of falling linearly as they are mixed.

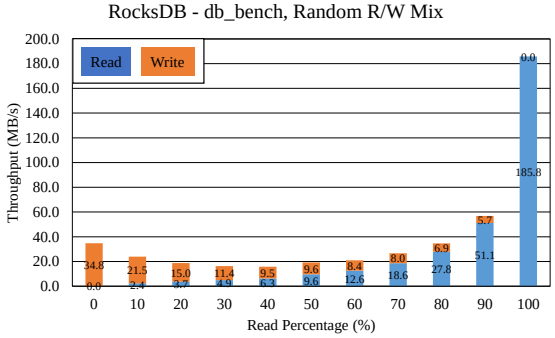

Figure 2. RocksDB's benchmark tool - read and write throughput comparison based on read percentage: the read performance drops sharply instead of falling linearly as it is mixed with writes. Note that db_bench parameters not specified are set as default values.

RocksDB benchmark tool (db_bench). Figure 1 shows that performance drops dramatically on workloads with read and write. In particular, when read percentage is 50, that is, in a workload where reads and writes are half occupied, the read performance is about 25.6% compared with read-only. Also, write only performs 26.8% compared with write-only. We have seen how this mixed workload affects the performance of the applications actually used. Figure 2 illustrates that the read performance is seriously degraded as the write request rate of the workload increases. Notably, even if write occupies only 10 percent of the workload, read performance drops to about a quarter of read-only.

2 RELATED WORK

One of the related solutions to solve the performance degradation in database systems caused by the workload of mixed reads and writes is presented in [10]. This research introduced Rails, an approach based on redundancy that physically separates reads from writes to achieve predictable read performance in the presence of writes. This study demonstrated that data replication design enables the

predictable and efficient performance of read/write workloads and evaluate it through benchmark and real workload experiments. However, this approach is limited in that the embedded FTL still manages SSD internal operations such as data placement or garbage collection (GC). Thus, the I/Os which an application issues and SSD internal operations will not operate in harmony.

Many researchers suggest the architectures that streamline the data path along with their application context. Multi-stream [7] places data together internally on the flash by channeling the data with similar aging through the same stream. However, there is a limitation that data placement and GC are controlled by the embedded FTL and it sets a ceiling on the number of streams. Recently, Open-Channel SSDs [3] has been proposed as a method to manage the internal parallelism of SSDs. They expose the geometry inside of the SSD and share control responsibilities with the host in order to implement and maintain features that typical SSDs implement strictly in the device firmware. As a consequence, host can manage data placement, I/O scheduling, and GC.

Another study [6] introduced a novel architecture using Open-Channel SSDs that an application implements their own FTL. In this study, authors consider the tendency of LSM-trees to build an application-driven FTL. They implemented RocksDB backend storage engine for Open-Channel SSDs and introduced liblightnvm, a user-space I/O library for Open-Channel SSDs [2]. With their scheme, reads are not disturbed by writes of other threads, so they show predictable performance.

In section 3, we describe the limitations of previous related studies and present a new solution to overcome them.

3 APP-DRIVEN FLASH MANAGEMENT

Recently, a lot of I/O intensive applications including RocksDB use LSM-tree as its data structure, which features fast write performance through append-only fashion writes. Because LSM-trees have internal implementations that are logically similar to FTL, user-space FTLs can be implemented without significant performance overhead. In the previous study [10], it sacrificed the capacity of the SSD for predictable performance because of the replication design. Another study [6] demonstrated the possibility of application-driven FTLs using Open-Channel SSDs and obtained the predictable performance. However, this resulted in a drop in the parallelism of the SSD, causing lower overall performance compared with the high bandwidth of the SSDs.

To address above problems we propose optimized application-driven flash management for LSM-tree based database systems. We chose RocksDB as our target database system. In RocksDB, all incoming data is buffered in the memtable and then persisted as a level 0 sorted string table (.sst file) when the buffer is flushed. When compaction is triggered, these sst files are merged with the different sst files to form a next level sst files, and the existing sst files are outdated. This process is similar to managing invalid blocks in FTL. Therefore, these outdated sst files will eventually become the target of the GC and will be erased in flash. SSDs maintain parallel units that can internally process I/Os in parallel. A parallel unit could be a channel, die, or a complex of them. If a 16-channel SSD has 8 dies per channel, there are 128 parallel I/O units in total. These parallel units are independent of each other and do not affect each other's operation. If an I/O thread gathers all the data of each sst file into parallel unit(s) in the SSD, sst files can be read or written without disturbance of other I/O threads since each of sst files has it's dedicated parallel unit(s). In addition, when compaction threads read sst files to merge them, there is no conflict with the write request of other I/O threads, so unnecessary delay due to latency difference between read and write disappears. In other words, read and write are separated, reducing unnecessary latency and achieving predictable performance.

We can simply implement the above I/O isolation by mapping one sst file to one parallel unit. Since a sst file is not distributed across multiple parallel units, data with similar age and access patterns will be collected in a single sst file. That means, there is a tendency that read and write are not mixed within the single sst file. However, through several other evaluations, we have found that this design will eventually degrade parallelism and lead to overall performance degradation. Static mapping of sst files to a parallel unit does not take into account the size or level of the sst file, which can cause applications fail to fully utilize the parallelism of the storage device.

To address this problem, we will implement LSM-tree based FTL that can fully utilize the parallelism of the storage device while accomplishing the above-mentioned read/write isolation. For example, parallel units can be dynamically allocated taking into account the size and the level of the sst file. The LSM-tree based FTL scheme will be evaluated against the design with maximized parallelism that has been considered as the best scheme for the high throughput, which show low and unpredictable performances under the mixed workload.

4 CONCLUSION

We expect the database system to be able to take advantage of high-performance storage devices as well as provide predictable performance. Additionally, we presuppose that this work would be a successful study of Software-Defined Storage solution using Open-Channel SSDs, a newly emerging storage stack.

5 ACKNOWLEDGEMENTS

This work is supported by the Samsung Electronics Co., Ltd. in Korea and is supervised by Prof. Heon Y. Yeom. I would like to express my gratitude to Prof. Heon Y. Yeom and Hwajung Kim for excellent advice.

References

[1] Apache Cassandra, http://cassandra.apache.org/.
[2] liblightnvm - User space I/O library for Open-Channel SSDs, http://lightnvm.io/liblightnvm.
[3] Open-Channel Solid State Drive Interface Specification, https://openchannelssd.readthedocs.io/en/latest/specification.
[4] RocksdB, https://rocksdb.org.
[5] Feng Chen, Rubao Lee, and Xiaodong Zhang. 2011. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. IEEE, 266–277.
[6] Javier González, Matias Bjørling, Seongno Lee, Charlie Dong, and Yiren Ronnie Huang. 2016. Application-Driven Flash Translation Layers on Open-Channel SSDs. (03 2016).
[7] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-streamed Solid-State Drive.. In *HotStorage*.
[8] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Yang-Suk Kee, and Moonwook Oh. 2014. Durable write cache in flash memory SSD for relational and NoSQL databases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 529–540.
[9] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33, 4 (1996), 351–385.
[10] Dimitris Skourtis, Dimitris Achlioptas, Noah Watkins, Carlos Maltzahn, and Scott A Brandt. 2014. Flash on Rails: Consistent Flash Performance through Redundancy.. In *USENIX Annual Technical Conference*. 463–474.