Demonstration

Don't mix pages with different lifetimes in one stream

Soyee Choi
Sungkyunkwan University
Suwon, Kyouggi, Republic of Korea
ithdli@skku.edu

Hyun-Woo Park
Sungkyunkwan University
Suwon, Kyouggi, Republic of Korea
music2eye@skku.edu

Sang-Won Lee
Sungkyunkwan University
Suwon, Kyouggi, Republic of Korea
swlee@skku.edu

ABSTRACT

We present a demonstration about optimizing two database storage engines by leveraging multi-streamed SSDs (MS-SSD in short). By storing data pages with similar lifetime together in the same physical flash blocks, MS-SSD can effectively reduce the overhead of garbage collection, improving the write performance and prolonging the lifespan. Thus, in order to benefit from MS-SSD, it is very crucial for database storage engines to precisely classify logical data pages according to their update intervals and to effectively map those logical data streams to physical streams in MS-SSD. Given that numerous new interfaces between host and flash memory SSD for better performance are emerging, this demonstration will provide a model case of physical database tuning on flash memory SSDs.

We have successfully multi-streamed two database engines, MySQL/InnoDB and ForestDB, by identifying several logical data streams with distinct update intervals in each engine and by taking the *stream-per-object* policy, instead of the naïve *stream-per-file* one. By running both vanilla and multi-streamed versions of two storage engines on real MS-SSD, we showcase that multi-streamed versions consistently outperform vanilla ones. In addition, we propose a set of guidelines on how to group logical streams with different update intervals into smaller number of physical streams with minimal performance degradation.

1 INTRODUCTION

During the last decade, we are witnessing that flash memory SSDs have relentlessly been replacing harddisks as the main storage because of several advantages such as high IOPS/$ and low power consumption [9]. However, to prevent data loss due to electrical interference, flash memory chips do not allow overwrite. Hence, a costly erase operation against a block is necessary prior to overwriting the existing data pages in the block [5]. For this reason, most contemporary flash storage device takes the log-structured copy-on-write approach and, among many FTL schemes, the page-mapping FTL approach is most popular [10]. In FTLs, when no more clean block is available, a costly but inevitable garbage collection (GC) operation has to be triggered so as to secure new blocks to write new incoming page writes. During GC, valid pages from the victim block has to be copybacked to a clean block. It is well known that the excessive copybacks of valid pages during GC negatively affects the performance and lifetime of flash memory SSDs. Hence, one of the key challenges in modern flash memory SSDs is to reduce the GC overhead.

Meanwhile, every modern flash memory SSD has abundant computing resource which is affordable to support other new interfaces than the existing dummy read and write block interface. In fact, numerous new interfaces between host and flash memory SSD have been actively proposed for various purposes including better performance. Among them, one notable interface is Multi-Streamed SSD (MS-SSD in short) [8]. The interface is recently standardized in the SCSI interface [11] and the commercial SSDs which support it exists. The goal of the MS-SSD is to minimize the GC overhead. That is, by storing data pages with different lifetimes in different physical flash memory blocks (i.e., different write streams), MS-SSD expects that it can reduce, compared to the existing non-multi-streamed SSD, the number of pages to be copybacked in victim blocks, and thus can improve both the write performance and the lifespan. In short, instead of writing all data pages in one stream regardless of lifetime, by explicitly allowing to cluster data pages with different lifetime into different write streams, the MS-SSD interface is intended to reduce write amplification due to GC.

However, the performance benefit of MS-SSD depends heavily on the accuracy of classifying logical data streams according to update interval [7]. Therefore, it is very crucial for storage engines to precisely classify logical data pages base on their update intervals and to effectively map those logical data streams to physical streams in MS-SSD. For this reason, the first step in making any database engine multi-streamed is to understand its write patterns and then to figure out all logical data streams distinguishable from each other in terms of update intervals.

In this demonstration, we will show how to make two database storage engines multi-streamed, MySQL/InnoDB and ForestDB, and present the benefit of each multi-streamed version over its vanilla engine in terms of transaction throughput and WAF. The contributions of this demonstration can be summarized as follows. First, we show that each database storage engine has several logical data streams with distinct update intervals. Second and more importantly, we show that, in database engines, the logical data streams with different update intervals can be found when the write patterns are analyzed *per-object*. Unlike the existing work on multi-streaming LSM-based NoSQL engines such as Cassandra and RocksDB using the *per-file* policy [6, 8], we found out that those two database engines used in this demonstration can not be effectively multi-streamed with the *per-file* policy. Given that numerous interfaces between host and flash memory SSDs for better performance are emerging, this demonstration will provide a model case of physical database tuning on flash memory SSDs. Our demonstration will proceed following the steps below:

- Using the logical data streams identified according to the *per-object* approach in each storage engine, we explain how to classify each of streams into *Hot*, *Cold* and *Warm* and the rationale behind it. (Section 2)
- Based on the classification obtained from the above step, we will show that there are numerous other combinations in mapping logical streams into physical streams than the naïve one-to-one mapping, run representative benchmark in each storage engine by changing the combinations, explain the results, and discuss its implications. (Section 4.2)

- While running benchmarks on both multi-streamed and vanilla version of each storage engine, we will show, using a GUI program, how the key metrics including CPU utilization, IOPS, TPS (transaction per second) and WAF dynamically change over time. (Section 4)
- Based on the performance results from several combinations in mapping logical streams to physical streams, we suggest a set of practical guidelines for making storage engines multi-streamed effectively. (Section 4.2)

2 BACKGROUND

2.1 MultiStream SSD

The goal of MS-SSD is to reduce the GC overhead by separating logical data pages with different lifetimes into different physical *streams* of flash blocks inside SSD [8]. Multiple *physical streams* will divide physical space in flash SSDs into several smaller spaces. Applications in the host are responsible for distinguishing data pages by explicitly attaching `stream-id` when making a write request to MS-SSD. Upon receiving write request for data page(s) with stream-id, SSD places the page(s) in the flash block belonging to the corresponding physical stream id. All the blocks belonging to each physical stream will be managed by the flash translation layer (FTL) separately from other blocks of other physical streams. Consequently, compared to the non-multi-streamed SSDs, MS-SSD expects that most pages in victim blocks upon GCs is invalidated for GC, thus minimizing the number of pages to be copybacked for GCs.

Although MS-SSD looks promising, there are at least two practical issues to be addressed when making any database engine multi-streamed. As noted above, the `stream-id` of data pages is not determined automatically by MS-SSD itself, but instead should be explicitly hinted by applications. Thus, the performance benefit of any multi-streamed database engine will be highly dependent on the accuracy of logical data stream classification. Next, because the number of physical streams available in an MS-SSD is limited in practice (e.g., 16 in the case of PM953), applications should be able to get best performance with the limited number of physical streams. For this, when the number of logical data streams from the applications is larger than that of physical streams supported by MS-SSD, a set of guidelines on how to group multiple logical streams into smaller number of physical streams with minimal performance degradation.

2.2 Logical Streams in Database Engines

As discussed above, the crux in leveraging the opportunities from MS-SSD is to accurately separate logical stream with different lifetime. In this section, we illustrate how logical streams from each of two real database engines, MySQL/InnoDB and ForestDB, are derived. From a set of separate experiments, where two storage engines were, likewise as in existing work [8] multi-streamed according to the *per-file* approach, any meaningful performance improvement was not observed. In some cases, the performance of multi-streamed versions was even worse than that of non-streamed vanilla ones. This is because the write patterns from those database engines do not reveal any distinguishable lifetime among different database files.

MySQL/InnoDB is a popular open source relational database engine, which takes the traditional in-place update policy. On the other hand, ForestDB, a storage engine for Couchbase NoSQL database [3, 4], is taking the out-of-place update approach, likewise other popular NoSQL engines such as Cassandra and RocksDB. But unlike these LSM-based NoSQL engines used in the previous work on MS-SSD [8], ForestDB is a B-tree-based storage engine, which appends new key-value versions at the end of files (that is, copy-on-write). It periodically reuses the space occupied by the invalidated old versions of key-value documents and, when the size of a database file becomes larger, the compaction operation has to be carried out. In this section, although these two database engines of MySQL/InnoDB and ForestDB take different approaches in updating data, they are common in that each engine has several object types and in turn each object type exhibits distinct update intervals. This observation clearly confirms that there exist opportunities for improving database performance by making those engines multi-streamed using the *stream-per-object* approach.

Table 1: Characteristics of MySQL TPC-C's Data Type

	avg update interval	total write (MB)	write ratio(%)
DWB (H1)	2	1330556	50
new_orders(H2)	226410	88349	3.32
order_line(C1)	26425310	202777	7.62
customer(C2)	7741410	157006	5.54
orders (W)	2861170	108706	4.085
stock (W)	2873060	771190	28.98

2.2.1 Mysql. In order to derive logical data streams, which are suitable to the purpose of MS-SSD, from MySQL/InnoDB engine we collected the write trace while running TPC-C benchmark [1]) with 200GB database size for one day. Using the trace, we calculated the average update interval, total write amount, and the relative write ratio for major object types in the database. Between the time point when data is written and updated in each LBA, write commands are issued to another LBAs. Average update interval represents the average of the numbers of intervening writes issued for all LBAs belonging to each object type. Therefore, the larger average update interval is, the less frequently the pages in the object type is updated. The results for major object types (with write ratio greater than 1%) are summarized in Table 1.

The most outstanding object from the table is *double-write buffer* (`DWB`), to which every dirty page evicted from the buffer cache has to be redundantly journaled to guarantee the page write atomicity. It shows very low value of average update interval and also occupies half of total writes in MySQL/InnoDB. For this reason, `DWB` is definitely a *hot* data object with very short lifetime and is thus denoted as `H1` in Table 1. In addition, we see from the table that the `new_orders` table has relatively low average update interval and thus it is also regarded as *hot object*. Similarly, two tables, `order_line` and `customers` are treated as *cold objects* and all other object are as *warm objects*.

2.2.2 ForestDB. A ForestDB database consists of multiple files and each database file is comprised of four data types: database header, super block, index node, and document. As mentioned above, there was no performance gain when multi-streamed using the *stream-per-file* approach. Therefore, as in the case of MySQL/InnoDB, in order to verify that those four object

Table 2: Characteristics of ForestDB's Data Type

	avg update interval	total write (MB)	write ratio(%)
DB Header (W1)	764571	6069	10.4
Index Node (W2)	848531	1048	1.8
Data Page (C)	1457589	44858	77.1
Super Block (H)	752	6222	10.7

types are suitable as logical data streams for MS-SSD, we collected the write trace while running ForestDB-Benchmark workload [2] with 7.5GB database for four hours. Using the trace, we calculated the average update interval, total write amount, and the relative write ratio for those four object types. The results are given in Table 2. From Table 2, it is obvious that `Super Block` is hot (denoted as `H`), `DB Header` and `Index Node` warm (denoted as `W1` and `W2`, respectively, and `Data Page` cold (denoted as `C`. In addition, we verified that all data pages of each object type tend to have uniform update intensity.

3 SYSTEM OVERVIEW

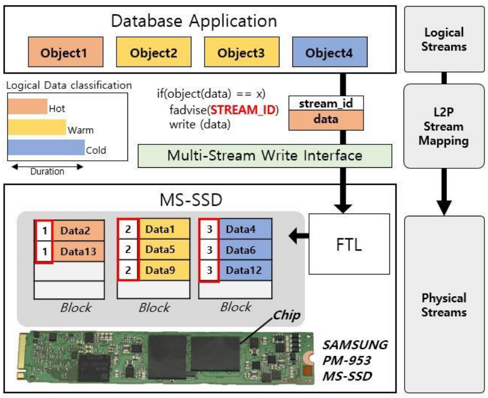


Figure 1: Multi-Streamed Database Engine: Architecture

Figure 1 shows the architectural overview of how a multi-streamed database engine interacts with MS-SSD. First, as shown in the bottom of Figure 1, a commercial MS-SSD from Samsung with NVMe interface (PM953 model) is used. The MS-SSD currently provides 16 *physical streams*. Next, as shown at the top of Figure 1, a multi-streamed database engine will, for each data page to be written, identify the *logical data stream* where the page belongs according to the *stream-per-object* policy explained in Section 2. Finally, as illustrated in the middle of Figure 1, the database engine is responsible for mapping its each logical stream to a specific physical stream in MS-SSD using the `posix_fadvise` system call. For example, the database engine can assign different physical stream to its each logical stream (that is, one-to-one mapping between logical and physical stream). As another example, as exemplified in Figure 1, four logical data streams in database engine can be combined to map to three different physical streams in MS-SSD(that is, many-to-one mapping between logical and physical stream).

The database engine will, before writing a data page, first check its logical stream, then assign the appropriate physical `stream_id` to the page according to the mapping between logical and physical streams, and finally call the multi-streamed write interface using the ioctl command of `posix_fadvise(fd,stream_id,0,POSIX_FADV_STREAMID)`. Upon receiving the command, FTL will place the data page in the physical stream corresponding to the given `stream_id`.

4 DEMONSTRATION DETAIL

4.1 Demonstration Scenario

The main goals of this demonstration are two-folds. First, we will show that real database engine can significantly benefit by accurately classifying its logical streams according to the *stream-per-object* policy and then by calling the multi-stream interface. Second, given the limited number of physical streams available in real MS-SSDs, we will show that it is possible to achieve nearly optimal performance by effectively using physical streams less than logical streams.

Table 3: Stream Combinations (MySQL/InnoDB)

5 streams	2 streams	
(H1, H2, C1, C2, W)	(H1, else)	(C1, else)
	(H1+H2, else)	(H1+C1, else)

Table 4: Stream Combinations (ForestDB)

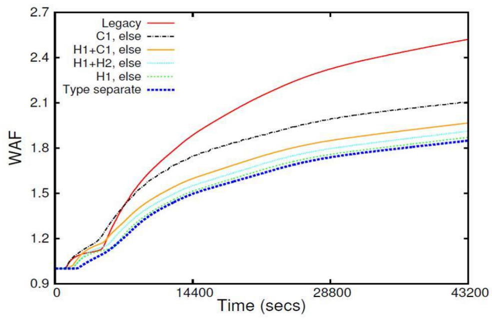
4 streams	3 streams	2 streams	
H, C , W1, W2	W1+W2, H, C	(H, else)	(C,else)
	H+C, W1, W2	(W1, else)	(W2, else)

For this, by running TPC-C an ForestDB-Benchmark on MySQL/InnoDB and ForestDB, respectively, this demonstration will present the performances of vanilla version of each storage engine. In addition, we will present, as the baseline performance, the performance of its multi-streamed version when run by assigning one physical stream to each logical stream. As shown in the first column of Table 3 and Table 4, respectively, a dedicated physical stream is assigned to each logical stream in Table 1 and in Table 2, respectively. Then, for each database engine, we will present the performance of multi-streamed version when run by grouping logical data streams into smaller number of physical streams in several meaningful combinations. In the case of MySQL/InnoDB, we tested all four combinations shown in the second column of Table 3. For example, the combination `(H1+H2, else)` in the table represents that two hot logical streams of `H1` and `H2` share one physical stream while all other three logical streams do other physical stream. Similarly, in the case of ForestDB, we tested all the six combinations shown in the second and third columns of Table 4.

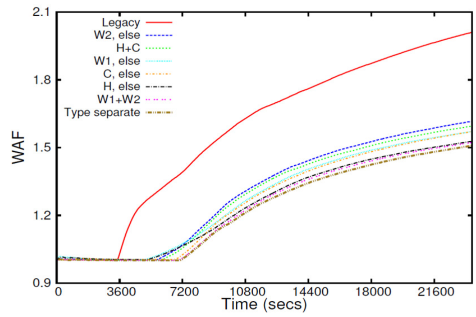
In order to show the effect of multi-streamed database visually, we made a GUI system to monitor status of computer resources utilization, which is illustrated in Figure 3. Using the GUI we will compare the effect of multistream SSD.

4.2 Preliminary Performance Evaluation

For each of MySQL/InnoDB and ForestDB engines, we have evaluated the performance of its multi-streamed version as well as its vanilla version. In the case of MySQL/InnoDB, we measured the write amplification factors over time while running



(a) TPC-C on MySQL/InnoDB



(b) ForestDB-Benchmark on ForestDB

Figure 2: Preliminary Experimental Results: Original vs. Multi-Streamed Database Engine

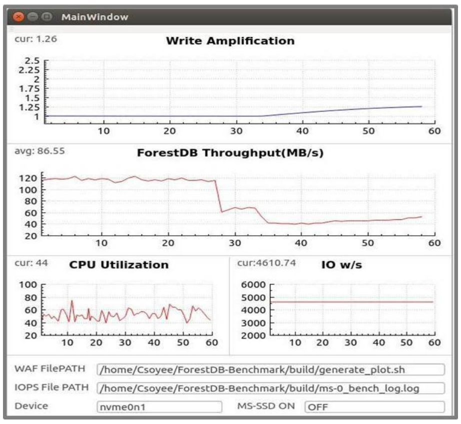


Figure 3: GUI used in Demonstration

TPC-C benchmark for twelve hours on its multi-streamed version for every five combinations in Table 1 as well as on its non-multi-streamed version. The results are presented in Figure 2(a). Similarly, in the case of ForestDB, we measured the write amplification factors over time while running ForestDB-Benchmark for six hours on its multi-streamed version for every seven combinations in Table 2 as well as the original ForestDB version. The results are presented in Figure 2(b).

From Figure 2(a) and Figure 2(b), we can make several common observations on the performance implications of combining logical streams into physical streams. First, since every multi-streamed versions always outperforms the vanilla version for both database engines, it is, obviously, always beneficial to separate at least one logical stream. Second, the best performance is achievable by one-to-one mapping between logical and physical streams. Third, it is better to combine data objects having similar lifetime rather than different lifetime. For example, when comparing the `(H1+H2, else)` case with the `(H1+C1, else)` one in the case of MySQL/InnoDB, the former case shows lower WAF value. Also, the `(W1+W2,else)` combination in ForestDB outperforms all other combinations except for one-to-one mapping logical and physical mapping case, in terms of WAF value. Fourth, it is always desirable to separate logical streams with extremely low update interval, such as `DWB (H1)` in MySQL/InnoDB and `Superblock(H)` in ForestDB. Lastly, though obvious, it is less effective to separate any logical stream with very low write ratio than to separate one with high write ratio, as exemplified by two streams of `W1` and `W2` in ForestDB.

5 CONCLUSION AND FUTURE WORK

In this demonstration, we have shown that database engines can significantly benefit from MS-SSD by appropriately identifying logical streams according to the *stream-per-object* policy. In addition, given that the number of physical streams available in real MS-SSDs is limited, we have derived a set of guidelines on effectively grouping logical streams into the fewest physical streams with minimal performance degradation.

One promising future research direction is to automatically identify logical streams out of any write-intensive application, which are suitable to MS-SSD, considering that we found out a set of logical streams from each of two database engines manually. Another challenging future work is to automatically group logical streams into minimum number of physical streams.

ACKNOWLEDGEMENTS

This research was supported in part by IITP under the “SW Starlab” (IITP-2015-0-00314) and in part by Samsung Electronics.

REFERENCES

[1] 2008. tpcc-mysql benchmark. https://github.com/Percona-Lab/tpcc-mysql. (2008).
[2] 2014. Forest Databse System Benchmark. https://github.com/couchbaselabs/ForestDB-Benchmark. (2014).
[3] 2014. ForestDB. https://github.com/couchbase/forestdb. (2014).
[4] Jung-Sang Ahn, Chiyoung Seo, Ravi Mayuram, Rahim Yaseen, J.W Kim, and Seungryoul Maeng. 2016. ForestDB: A Fast Key-Value Storage System for Variable-Length String Keys. *IEEE Trans. Comput.* 65 (March 2016), 902–915.
[5] Shingo Nishioka Atsuo Kawaguchi and Hiroshi Motoda. 1995. A Flash-Memory Based File System. In *UniSex Winter*. 155–164.
[6] Yang Fei, Dou Kun, Chen Siyu, Hou Mengwei, Kang Jeong-Uk, and Cho Sangyeon. 2015. Optimizing NoSQL DB on Flash: A Case Study of RocksDB. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing*.
[7] Yang Jingpei, Pandurangan Rajinikanth, Choi Changho, and Balakrishna Vijay. 2017. AutoStream: automatic stream management for multi-streamed SSDs. (May 2017).
[8] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-streamed Solid-State Drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*.
[9] Sang-Won Lee, Bongki Moon, and Chanik Park. 2009. Advances in Flash Memory SSD Technology for Enterprise Database Applications. In *Proceedings of the 35th SIGMOD international conference on Management of data*. 863–870.
[10] Ma, Dongzhe and Feng, Jianhua and Li, Guoliang. 2014. A Survey of Address Translation Technologies for Flash Memories. *ACM Computing Survey* 46, 3, Article 36 (Jan. 2014), 36:1–36:39 pages.
[11] William Martin(T10 Technical Editor). 2015. SCSI Block Commands - 4 (SBC-4) (Working Draft Revision 9): 4.34 Stream Control. (November 2015), 110–112 pages. http://www.t10.org/cgi-bin/ac.pl?t=f&f=sbc4r09.pdf