vStream: Virtual Stream Management for Multi-streamed SSDs

Hwanjin Yong[§†], Kisik Jeong[†], Joonwon Lee[†], and Jin-Soo Kim[‡]

[§]*Samsung Electronics Co., South Korea*
[†]*Sungkyunkwan University, South Korea*
[‡]*Seoul National University, South Korea*

Abstract

Recently, multi-streamed SSDs have been proposed to reduce the cost of garbage collection in NAND flash-based SSDs. In multi-stream SSDs, application developers can control data placement within the SSD by specifying a stream ID along with the WRITE command. However, commercial SSDs support only a small number of streams due to the device's limitation in hardware resources. This makes it difficult to fully utilize the benefit of the multi-streamed SSDs.

In this paper, we propose a new concept of virtual streams (vStreams) for the multi-streamed SSDs. vStreams allow application developers to manage a sufficient number of streams regardless of the physical streams supported by the device. We also present novel mechanisms to monitor the lifetime of each stream with a negligible memory overhead and to map one or more vStreams into a physical stream at run time. Our evaluations with RocksDB show that the proposed vStream SSD improves the throughput by 70% compared to the legacy SSD with no stream support.

1 Introduction

NAND flash-based solid-state drives (SSDs) are becoming increasingly popular within enterprise data centers due to high performance and low power consumption. NAND flash memory, however, does not support in-place update and has limited P/E (Program and Erase) cycles. Typically, SSDs employ a software layer called flash translation layer (FTL) that manages all the operations on the flash memory and provides the traditional block device interface by redirecting new write requests to unprogrammed pages. In addition, FTLs perform garbage collection (GC) to reclaim free space when the available free space becomes low. This is done by copying valid data in a victim block to a new flash block and then erasing the victim block. Therefore, the GC processing significantly decreases not only SSD's performance but also its lifespan.

Recently, multi-streamed SSDs [13] have been proposed to reduce the cost of GC and improve its performance by grouping the data having the same stream ID in the same flash block. The host, thereby, can control data placement in the flash media by tagging a stream ID along with each write request. The multi-stream interface presents new opportunities for system designers to place the data in an effective way to mitigate the cost of GC. Because of its benefits, multi-stream support has been standardized in the SCSI/SAS [19] and the NVMe (NVM Express) [17] specifications. Both specifications allow up to 65535 streams per device. However, the actual number of streams supported by the device (we call these *physical streams*) is restricted to $4 \sim 16$ [13, 22, 23] in commercial SSDs. This is because it is difficult for the SSD to support a large number of streams due to the device's limitation in hardware resources. Increasing the number of physical streams cannot be achieved without extra hardware cost and/or a loss in performance.

When SSDs support only a limited number of physical streams, application developers are forced to carefully assign a specific stream ID according to the *hotness* of the data. However, this is possible only when application developers have detailed knowledge about the characteristics (i.e., lifetime) of their data. It gets more complicated when the characteristics of the data dynamically change over time. Furthermore, when the underlying SSDs are either replaced with or upgraded to other SSDs having a different number of physical streams, the host applications have to be modified to match the new physical stream count. This leads to severe degradation of portability and compatibility from an application development point of view. Consequently, the limited number of physical streams, which is also different from device to device, is a big obstacle to full utilization of the benefit of multi-streamed SSDs.

To address these limitations, we propose a new concept of virtual streams (vStreams) that are independent of the physical streams available in the device. The goal of

this paper is to provide a large number of virtual streams to application developers without adding any extra hardware resource nor performance degradation.

The experimental results with RocksDB workloads show that the vStream SSD can not only improve the throughput by 70% but also reduce the cost of GC by 18% compared to the legacy SSD with no stream support. The remainder of the paper is organized as follows. Section 2 presents the background. Section 3 presents the related work. Section 4 describes the design of virtual stream management in detail, Section 5 shows the evaluation of vStream SSD, and Section 6 concludes the paper.

2 Background

NAND Flash Memory: A NAND flash memory package is composed of multiple dies and each die has multiple planes. Typically, NAND flash memories support multi-plane operation, which performs a read, program, or erase operation on multiple planes in a parallel manner to improve internal flash parallelism [12]. Each plane is composed of multiple blocks and each flash block, in turn, contains multiple rows (i.e., wordlines) of flash cells. In addition, each cell is capable of storing multiple bits of data due to the multi-level cell technology. For example, the least significant bit, the central significant bit, and the most significant bit of each cell are grouped to form an LSB, CSB, and MSB page, respectively, in TLC NAND flash memory.

Recently, the page size of NAND flash memory has continuously been increased for high capacity and high I/O bandwidth [4, 15]. Furthermore, 3D NAND (such as Samsung's VNAND) technology has been proposed to provide an even higher storage capacity, performance, and endurance. A traditional 2D NAND flash has to separate the program steps within the LSB, CSB, and MSB page in order to mitigate the program interference on neighbor cells [1, 2]. On the contrary, 3D NAND flash is capable of programming the multiple bits at once via the *one-shot programming* technique [11, 14] due to the virtually coupling-free structure which reduces the cell-to-cell program interference. Thanks to one-shot programming, the multiple pages (LSB, CSB, and MSB) can be programmed simultaneously. Thus, programming speed is enhanced and power consumption is reduced.

Multi-streamed SSD: Figure 1 shows a simplified example of how a multi-streamed SSD operates. The legacy SSD (i.e., SSD without stream support) simply places data in their original request order.

In contrast, multi-streamed SSDs place data into flash blocks according to the stream ID, instead of the request order. If the host can assign data with similar lifetime to the same stream ID, then those data will be stored in the same flash block. This helps to significantly reduce or eliminate the GC activity because all of the data belonging to those blocks are likely to be invalidated at the same time.

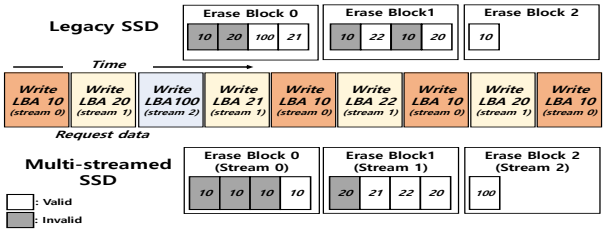

Figure 1: Comparison of data placement with or without the multi-stream support.

RocksDB: RocksDB is a popular NoSQL database based on log-structured merge trees (LSM-trees) [6, 9]. In RocksDB, new data are inserted into a sorted structure in the main memory called MemTable, which maintains the most recently updated key-value data. At the same time, the data are optionally appended to a write-ahead log (WAL) for recovery purposes. When the MemTable exceeds its size limit, the current MemTable becomes immutable, and the contents of MemTable is flushed into the storage device as a static sorted table file format, called SSTable.

Since SSTables are immutable, the compaction process needs to be conducted periodically when the total size of each level exceeds the predefined threshold. During the compaction processing, redundant or invalid key-value data are removed to minimize space amplification and valid data from the $Level_i$ SSTables are moved to the $Level_{i+1}$ SSTables. Therefore, the data lifetime on the $Level_{i+1}$ SSTables is much longer than those on the $Level_i$ SSTables because the $Level_i$ SSTables are compacted more frequently. Previous work with RocksDB [22] showed that the multi-streamed SSDs reduce the overhead of GC significantly by assigning different stream IDs to the data depending on the type of files and the level of SSTables.

3 Related Work

Much work has been done to reduce the cost of GC by classifying data according to their hotness. Specifically, DAC (Dynamic dAta Clustering) [5] maintains a number of logical regions and assigns a specific region number to each data so that data with similar temperatures are stored in the same flash block. In DAC, the region number associated with the data changes dynamically; if some data are overwritten, they are migrated to the upper region, while the data copied to other flash blocks during GC are moved down to the lower region. Yang et al. presented AutoStream [23], an automatic stream detection mechanism based on data temperature. AutoStream au-

tomatically assigns a stream ID by monitoring the update frequency of the incoming data.

All the previous researchers have worked at the *logical page* level; their goal is to find a set of logical pages that have the similar update frequency and group them together in the same flash block. However, our approach works at the stream level; we leverage the *virtual* stream ID assigned by application developers and focus on identifying a set of virtual streams that have similar lifetime. Compared with previous works, our approach is more effective with much less overhead.

4 Virtual Stream Management

4.1 vStream (Virtual Stream)

In NVMe SSDs, there is a limitation in the maximum number of concurrently open streams. If the host tries to use a new stream beyond this limitation, an arbitrary stream is released to free the stream resources associated with that stream ID for the new stream. Also, when all the stream resources are allocated for exclusive use for specific namespaces [16], the subsequent write commands are treated as normal write commands that do not specify a stream ID [17]. On the contrary, SAS SSDs return error responses when receiving an open stream request beyond the limitation [19]. Therefore, the host system should explicitly close one or more open streams in SAS SSDs when the available streams are exhausted to process new stream requests.

In any case, the small number of physical streams leads to a situation where the data which have different lifetimes are mixed together in a flash block due to frequent release and reuse of stream IDs. For this reason, application developers want SSD manufacturers to support a large number of physical streams, but it is not easy because of the device's restricted hardware resources such as limited over-provisioning area [20] and durable write buffer size [21] to process concurrent I/O requests from multiple streams.

Our approach is to provide a large number of virtual streams (vStreams) by *virtualizing* the notion of streams, instead of increasing the number of physical streams. It is motivated by the observation that if the data belonging to different stream IDs have similar lifetimes, then we can group those stream data to share the same physical stream resources because they are very likely to be invalidated together. This allows application developers to manage a sufficient number of streams regardless of the amount of device's hardware resources, making storage applications more portable and easier to be interoperable among different SSDs from different vendors.

Figure 2 depicts the overall architecture for virtual stream management. The vStream lifetime identifier calculates the lifetime of each virtual stream and the vStream clustering manager maps one or more vStreams to a physical stream according to their lifetimes. The role of each component is explained in more detail in the following subsections.

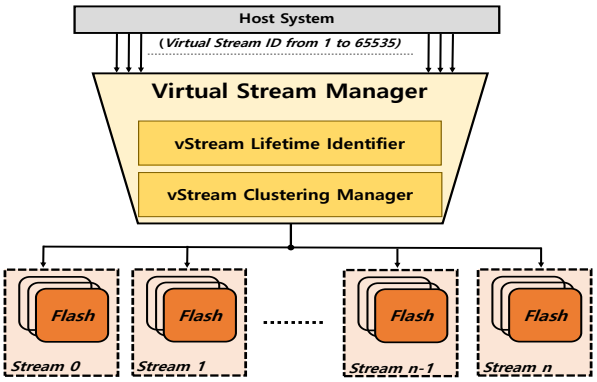

Figure 2: vStream Architecture with n physical streams

4.2 Identifying vStream Lifetime

In order to determine the lifetime of each vStream, we need to monitor all the WRITE requests to the logical pages in the vStream. However, it is impractical to collect information for every logical page belonging to a vStream. To minimize computation cost and memory overhead, we focus on dead logical pages (*dead pages*) instead of live logical pages (*live pages*) in each vStream. The lifetime of a dead page can be simply calculated by measuring the time when the page is written and the time it is overwritten by the WRITE command or invalidated by the DISCARD (or TRIM [18]) command. We define the average lifetime of dead pages as the sum of the lifetimes of all the individual dead pages divided by the total number of dead pages. To reduce the computation cost further, we calculate the average lifetime of dead pages lazily; we update the sum of the lifetimes of dead pages and the number of dead pages only when a logical page is invalidated.

To calculate the lifetime of a dead page, we need to keep track of the written time for each logical page which requires a huge amount of memory. For example, a 1TB SSD based on the full 4KB page-mapping has 256M logical pages. If we use a 4-byte timestamp per logical page, approximately 1GB of memory is required to maintain the time when a logical page is written.

In order to reduce the memory overhead, we use a novel scheme which leverages the following characteristics of modern SSD architecture. First, the basic unit of NAND flash program operation is the flash page size which is larger than the logical page size. Second, multiple flash pages are programmed simultaneously due to one-shot programming. Third, two or more planes can

be programmed at the same time by using a multi-plane operation. Finally, modern SSDs exploit channel-level, chip-level, and die-level parallelism to achieve higher throughput [3,10,12]. As a large number of logical pages are written into the NAND flash memory at once in modern SSDs, we can make them share a single timestamp for the written time.

We assume a realistic organization of a 1TB SSD based on TLC NAND flash memory where the number of channels is four ($c = 4$) and a channel is composed of 4 chips ($w = 4$). Each chip internally has two dies ($d = 2$) and flash blocks in each die are organized as two planes ($b = 2$). A flash block is composed of 256 wordlines and each wordline holds the data for LSB, CSB, and MSB pages ($p = 3$). Finally, the flash page size is 16KB that can contain the data of four 4KB logical pages ($l = 4$). In this case, we can reduce the memory overhead by a factor of 768 ($= c * w * d * b * p * l$) by sharing the same timestamp among the logical pages written together, requiring only 1.3MB of memory to maintain the written time for all logical pages. Thus, we can support a large number of vStreams with a reasonable resource overhead.

4.3 Clustering vStreams

To classify vStreams according to their lifetimes, we use the *K*-means clustering algorithm [8] which is one of the most widely used data clustering algorithms. We let *K* represent the number of physical streams supported by the SSD. Our statistical classification scheme measures the Euclidean distance between the average lifetime of each vStream and the median lifetime of a certain physical stream to map vStreams closer to one of physical streams with the similar lifetime. Moreover, we periodically (e.g., every 10 minutes in our evaluation) updates the mapping from vStreams to physical streams in order to adapt to new stream requests and/or the changes in the workloads' access patterns.

5 Evaluation

5.1 Experimental Setup

The proposed virtual stream management scheme has been implemented on an open-source SATA SSD simulator called DROLL [7] after extending it to support multi-stream interface with up to four physical streams. We configured the DROLL simulator to emulate 128Gb TLC 3D NAND flash memory [11] providing the total capacity of 128GB with a 7% over-provisioning area. The simulator runs a full page-mapping FTL where the logical page size is set to 4KB.

The block traces of RocksDB are obtained by running RocksDB on the Ext4 file system with enabling the DISCARD command. We use `db_bench` to generate overwrite workload which overwrites randomly generated key-value pairs in RocksDB. For RocksDB, compression is disabled and the size of SSTable is set to 128MB. Also, the max bytes for the compaction threshold of Level-1 is configured to 256MB and the size multiplier of the next-level compaction threshold is set to 4x, which results in the maximum of 6-level SSTables on a 128GB SSD.

In our evaluation with RocksDB, we use the total 10 virtual streams by assigning different virtual stream IDs to each level of SSTables, write-ahead log file (WAL), the manifest file, RocksDB temporary files, and system data generated by the Ext4 file system. We modified the Linux kernel (4.10.0) and RocksDB (5.8.0) source codes in the same way as the previous studies [13, 22] to pass the (virtual) stream ID down to the SSD. In addition, we modified them to pass stream ID along with each DISCARD command for calculating the lifetime of stream data.

We compare the proposed scheme (vStream-SSD) with other alternatives: Legacy-SSD, DAC-SSD, Auto-SSD, and Manual-SSD. Legacy-SSD represents the traditional page-mapping FTL with no multi-stream support. DAC-SSD implements the DAC algorithm [5] which dynamically places the data into one of four logical regions. Auto-SSD indicates the FTL which automatically assigns a stream ID to each logical page based on the AutoStream technique [23]. In Auto-SSD and DAC-SSD, when the storage device receives information that some logical pages have been invalidated via the discard command, the update frequency or the logical region number associated with those pages will be reset. Finally, Manual-SSD shows an example of what a RocksDB developer can do manually to map ten virtual streams into four physical streams. For Manual-SSD, we consider the static stream ID assignment as shown in Table 1.

Table 1: Static stream ID assignment in Manual-SSD

Stream ID	1	2	3	4
File Type or Level of SSTable	System data WAL Manifest Temporary	Level-0 Level-1	Level-2 Level-3	Level-4 Level-5

5.2 Synthetic Benchmark

To evaluate the performance of various schemes in a controlled environment, we have developed a synthetic benchmark which is similar to the one used in the previous study [23]. We divide the total 120GB of storage capacity into 8 partitions: four hot partitions (each 6GB in size), two warm partitions (each 14GB in size), and two cold partitions (each 34GB in size). We loop over the 8 partitions one by one, issuing a single 128KB write request at a time in each partition. Within a partition, each 128KB region is written sequentially to the end of the partition and then the write starts over from the be-

ginning again. Since the size of the hot partition is much smaller than that of the cold partition, logical pages in a hot partition will receive the overwrite more frequently. In this experiment, we assign a different virtual stream ID to each partition and see if vStream-SSD can cluster those hot, warm, and cold streams effectively.

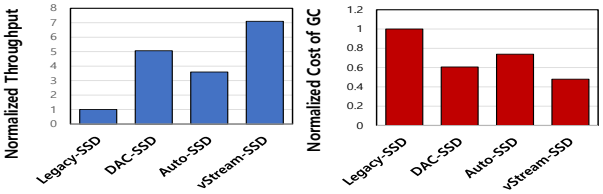

Figure 3: Comparison of normalized throughput and cost of GC with synthetic benchmark

Figure 3 illustrates the throughput and cost of GC in Legacy-SSD, DAC-SSD, Auto-SSD, and vStream-SSD. We measure the cost of GC as the number of valid pages copied during GC. All the results are normalized to those of Legacy-SSD for comparison. First, we observe that the multi-stream support is very effective; it improves the performance by 3x∼7x compared to Legacy-SSD which has no streams. Second, Auto-SSD achieves much lower performance than DAC-SSD. This is because Auto-SSD maintains the update frequency based on the large chunk unit (1MB in size by default) to reduce the monitoring cost and classifies logical pages according to their update frequencies in logarithmic scale. In comparison, DAC-SSD utilizes much more fine-grained information at each logical page level. Finally, Figure 3 shows that vStream-SSD performs best among the evaluated schemes. When comparing to Legacy-SSD, vStream-SSD improves the throughput by 7x and reduces the cost of GC by 51%. The reason is that the lifetime of vStream does not change dynamically in this workload. Therefore, vStream-SSD can continuously separate the data according to the lifetime of vStream after the initial clustering is completed.

5.3 RocksDB

Figure 4 compares the throughput and the cost of GC in Legacy-SSD, DAC-SSD, Auto-SSD, Manual-SSD, and vStream-SSD for our experiment with RocksDB. As in Figure 3, the results are normalized to those of Legacy-SSD. Overall, we can see that the proposed vStream-SSD outperforms other approaches. When comparing to Legacy-SSD, vStream improves the throughput by 70% while reducing the cost of GC by 18%.

Note that vStream-SSD achieves a significantly higher throughput than Manual-SSD which models the situation where application developers are forced to combine several streams statically due to the limited number of physical streams. This means that the manual and static approach used in Manual-SSD cannot cope well with the dynamic changes in application's characteristics over time. On the contrary, vStream-SSD is capable of updating the virtual-to-physical stream mapping at run time according to the changes in the lifetimes of virtual streams.

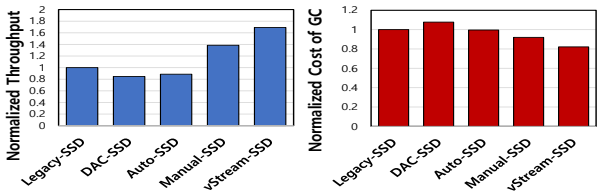

Figure 4: Comparison of normalized throughput and cost of GC with RocksDB

Another interesting result is that DAC-SSD and Auto-SSD perform worse even than Legacy-SSD which does not support multiple streams. The main reason is that both DAC and AutoStream techniques estimate the hotness of a logical page based on the frequency of *overwrite* to the logical page. However, this approach has a limitation in modern data-intensive applications such as RocksDB, where SSTables are sequentially written and then deleted after compaction. The deleted logical pages are eventually allocated to a new file, but the new file may have a different lifetime characteristic from the old file. Hence, the accumulated information on those logical pages becomes useless for new files. Figure 4 shows that vStream-SSD can handle this situation better as it gathers lifetime information not on the logical page level, but on the stream level.

6 Conclusion

We present a virtual stream management scheme to remove the restriction on the number of physical streams in the existing multi-streamed SSDs. Our vStream provides the host system with a sufficient number of virtual streams so that they can take full advantage of multi-streamed SSDs regardless of the amount of device's hardware resources. Experimental results with RocksDB show that our vStream improves the overall throughput by 70% and reduces the cost of GC by 18% compared to a legacy SSD. As future work, we plan to develop a more dynamic vStream clustering algorithm and implement the proposed scheme on a real multi-streamed SSD.

7 ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIP) (No. NRF2016R1A2A1A05005494). The ICT at Seoul National University provides research facilities for this study.

References

[1] CAI, Y., GHOSE, S., HARATSCH, E. F., LUO, Y., AND MUTLU, O. Errors in flash-memory-based solid-state drives: Analysis, mitigation, and recovery. *arXiv preprint arXiv:1711.11427* (2017).

[2] CAI, Y., GHOSE, S., LUO, Y., MAI, K., MUTLU, O., AND HARATSCH, E. F. Vulnerabilities in mlc nand flash memory programming: experimental analysis, exploits, and mitigation techniques. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on* (2017), IEEE, pp. 49–60.

[3] CHEN, F., LEE, R., AND ZHANG, X. 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* (2011), IEEE, pp. 266–277.

[4] CHEN, T.-Y., CHANG, Y.-H., HO, C.-C., AND CHEN, S.-H. Enabling sub-blocks erase management to boost the performance of 3d nand flash memory. In *Proceedings of the 53rd Annual Design Automation Conference* (2016), ACM, p. 92.

[5] CHIANG, M.-L., LEE, P. C., CHANG, R.-C., ET AL. Using data clustering to improve cleaning performance for flash memory. *Software-Practice & Experience 29*, 3 (1999), 267–290.

[6] DONG, S., CALLAGHAN, M., GALANIS, L., BORTHAKUR, D., SAVOR, T., AND STRUM, M. Optimizing space amplification in rocksdb. In *CIDR* (2017).

[7] DROLL. Open Source SATA SSD Simulator. `https://github.com/essencloud/droll`.

[8] HARTIGAN, J. A., AND WONG, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28*, 1 (1979), 100–108.

[9] HE, J., KANNAN, S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. The unwritten contract of solid state drives. In *EuroSys* (2017), pp. 127–144.

[10] HU, Y., JIANG, H., FENG, D., TIAN, L., LUO, H., AND ZHANG, S. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the international conference on Supercomputing* (2011), ACM, pp. 96–107.

[11] IM, J.-W., JEONG, W.-P., KIM, D.-H., NAM, S.-W., SHIM, D.-K., CHOI, M.-H., YOON, H.-J., KIM, D.-H., KIM, Y.-S., PARK, H.-W., ET AL. 7.2 a 128gb 3b/cell v-nand flash memory with 1gb/s i/o rate. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International* (2015), IEEE, pp. 1–3.

[12] JUNG, M. Exploring parallel data access methods in emerging non-volatile memory systems. *IEEE Transactions on Parallel and Distributed Systems 28*, 3 (2017), 746–759.

[13] KANG, J.-U., HYUN, J., MAENG, H., AND CHO, S. The multi-streamed solid-state drive. In *HotStorage* (2014).

[14] KIM, C., CHO, J.-H., JEONG, W., PARK, I.-H., PARK, H.-W., KIM, D.-H., KANG, D., LEE, S., LEE, J.-S., KIM, W., ET AL. 11.4 a 512gb 3b/cell 64-stacked wl 3d v-nand flash memory. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International* (2017), IEEE, pp. 202–203.

[15] KIM, M., LEE, J., LEE, S., PARK, J., AND KIM, J. Improving performance and lifetime of large-page nand storages using erase-free subpage programming. In *Proceedings of the 54th Annual Design Automation Conference 2017* (2017), ACM, p. 24.

[16] MARKS, K. An nvm express tutorial. *Flash Memory Summit, Santa Clara, CA 92* (2013).

[17] NVM EXPRESS WORKGROUP. NVM Express Revision 1.3. `http://nvmexpress.org/wp-content/uploads/NVM_Express_Revision_1.3.pdf`, 2017.

[18] SMITH, K. Garbage collection. *SandForce, Flash Memory Summit, Santa Clara, CA* (2011), 1–9.

[19] T10. SCSI Block Commands-4(SBC-4). `http://www.t10.org/cgi-bin/ac.pl?t=f&f=sbc4r15.pdf`, 2017.

[20] YADGAR, G., AND GABEL, M. Avoiding the streetlight effect: I/o workload analysis with ssds in mind. In *HotStorage* (2016).

[21] YAN, S., LI, H., HAO, M., TONG, M. H., SUNDARARAMAN, S., CHIEN, A. A., AND GUNAWI, H. S. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in nand ssds. In *FAST* (2017), pp. 15–28.

[22] YANG, F., DOU, K., CHEN, S., HOU, M., KANG, J.-U., AND CHO, S. Optimizing nosql db on flash: A case study of rocksdb. In *Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015 IEEE 12th Intl Conf on* (2015), IEEE, pp. 1062–1069.

[23] YANG, J., PANDURANGAN, R., CHOI, C., AND BALAKRISHNAN, V. Autostream: automatic stream management for multi-streamed ssds. In *Proceedings of the 10th ACM International Systems and Storage Conference* (2017), ACM, p. 3.