The Multi-streamed Solid-State Drive

Jeong-Uk Kang Jeeseok Hyun Hyunjoo Maeng Sangyeun Cho

Memory Solutions Lab.
Memory Division, Samsung Electronics Co.
Hwasung, Korea
E-mail: ju.kang@samsung.com

Abstract

This paper makes a case for the *multi-streamed solid-state drive (SSD)*. It offers an intuitive storage interface for the host system to inform the SSD about the expected lifetime of data being written. We show through experimentation with a real multi-streamed SSD prototype that the worst-case update throughput of a Cassandra NoSQL DB system can be improved by nearly 56%. We discuss powerful use cases of the proposed SSD interface.

1 Introduction

NAND flash based solid-state drives (SSDs) are widely used for main storage, from mobile devices to servers to supercomputers, due to its low power consumption and high performance. Most SSD users do not (have to) realize that the underlying NAND flash medium disallows in-place update; the illusion of random data access is offered by the SSD-internal software, commonly referred to as flash translation layer or FTL. The block device abstraction paved the way for wide adoption of SSDs because one can conveniently replace a HDD with an SSD without compatibility issues.

Unfortunately, maintaining the illusion of random data access through the block device interface comes at costs. For example, as the SSD is continuously written, the underlying NAND flash medium can become fragmented. When the FTL tries to reclaim free space to absorb further write traffic, internal data movement operations are incurred between NAND flash locations (i.e., garbage collection or GC) [6], leaving the device busy and sometimes unable to properly process user requests. The resultant changing performance behavior of a given SSD is hard to predict or reason about, and remains an impediment to full-system optimization [1].

In order to address the problem from the root, we propose and explore *multi-streaming*, an interface mechanism that helps close the semantic gap between the host system and the SSD. With the multi-streamed SSD, the host system can explicitly open "streams" in the SSD and send write requests to different streams according to their expected lifetime. The multi-streamed SSD then ensures that the data in a stream are not only written together to a physically related NAND flash space (e.g., a NAND flash block or "erase unit"), but also separated from data in other streams. Ideally, we hope the GC process would find the NAND capacity unfragmented and proceed with no costly data movements.

In the remainder of this paper, we will delve first into the problem of SSD aging and data fragmentation in Section 2, along with previously proposed remedies in the literature. Section 3 will explain our approach in detail. Experimental evaluation with a prototype SSD will be presented in Section 4. Our evaluation looks at Cassandra [7], a popular open-source key-value store, and how an intuitive data mapping to streams can significantly improve the worst-case throughput of the system. We will conclude in Section 5.

2 Background

2.1 Aging effects of SSD

SSD aging [16] explains why the SSD performance may gradually degrade over time; GC is executed more frequently as the SSD is filled with more data and fragmented. Aging effects start to manifest when the "clean" NAND flash capacity is consumed, and in this case, the FTL must proactively recover a sufficient amount of new capacity by "erasing" NAND flash blocks before it can digest new write data. The required erase operations are often preceded by costly GC; to make matters worse, a NAND block, a unit of erase operation, is fairly large in modern NAND flash memory with 128 or more pages in it [15]. When the SSD is filled up with more and more data, statistically, the FTL would need to copy more valid pages for GC before each NAND flash erase operation. This phenomenon is analogous to "segment cleaning" of a log-structured file system [13] and is well studied.

Figure 3 (in Section 4) clearly depicts the effects of SSD aging: The performance of Cassandra throughput (in the case of "Normal" SSD) is seriously degraded as the data set in the SSD is continuously updated—by as much as $\sim$56%. This problem will be exacerbated in systems with many threads (or virtual machines) that perform I/O concurrently. Moreover, for density improvement, a NAND block size will only grow in the future, adding to the cost of GC in general.

2.2 Prior work to mitigate SSD aging

Prior proposals reduce GC overheads by classifying access patterns and adapting to workloads inside a storage device [8, 12]. Since the effectiveness of these proposals depends on the accuracy of workload classification—random or sequential, they are especially vulnerable to workloads that have frequently changing behavior.

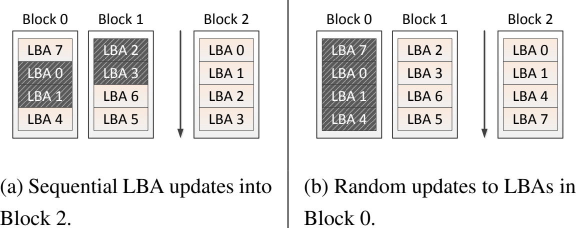
In another approach, the device detects and separates hot data from cold data [2, 4]. These techniques determine "hotness" of data based on access history of locations and require sizable resources to bookkeep the history information. Chiang et al. [2] use "time-stamps", to indicate how old given data are, and Hsieh et al. [4] employ multi-hashing to reduce the size of history information. The accuracy and benefits of hot data identification decrease when the access pattern of specific locations is changed, e.g., as in a log-structured file system.

In a practical sense, robustly deriving accurate information about data hotness and future access patterns is hard. Accordingly, enterprise SSDs where consistent access latency is of paramount importance, tend to set aside ("overprovision" or sacrifice) a generous amount of flash capacity to increase the efficiency of GC [17].

Lastly, "TRIM" is a standardized SSD command (not applicable to HDDs in general), with which the host system can pass information about what LBAs have been "unmapped" by the upper-layer software (typically the file system). This information is useful for the SSD's GC efficiency, because without the information the SSD has to assume conservatively that all NAND flash pages mapped to previously written LBAs (not overwritten) are valid and their content must be copied for preservation when the NAND flash blocks having those pages are vacated. Figure 3 shows that TRIM improves Cassandra's update throughput significantly; however, it does not alone match the full benefits of multi-streaming.

3 The Multi-streamed SSD

Before describing the proposed multi-stream approach, let's consider revealing examples that explain why traditional write pattern optimizations like append-only logging do not fully address the SSD aging problem. Figure 1 gives the examples, where two NAND flash blocks,



(a) Sequential LBA updates into Block 2.

(b) Random updates to LBAs in Block 0.

Figure 1: *Relationship between data placement and updates.* For simplicity, we assume that there are four pages per block.

Block 0 and Block 1, have been filled up and new data are written to fill Block 2. In the first example (left), a sequential write pattern was applied, and as the result, some data become invalid in Block 0 and 1. On the other hand, in the second example, a random write pattern was applied, invalidating all data in Block 0 but none in Block 1. Clearly, future GC will proceed more efficiently in this example, because an empty NAND flash block (Block 0) can be reclaimed quickly without copying data around. These examples demonstrate that an SSD's GC overheads depend not only on the current write pattern but on how data have been already placed in the SSD.

Naturally, one might argue that re-scheduling of future write requests to the SSD might solve the aging problem (like in Figure 1(b)). However, it is next to impossible for an application (and the host system in general) to know where exactly previously written data have been placed within the SSD, because the algorithms in the FTL vary from SSD to SSD and the written data are moved around by the internal GC process. Moreover, modern OS has a layered I/O subsystem comprised of file system, buffer cache, I/O scheduler and volume manager. So, perfectly controlling the order and target of writes would be extremely challenging.

3.1 Our approach

At the heart of the SSD aging problem are the issues of how to predict the lifetime of data written to the SSD and how to ensure that data with similar lifetime are placed in the same erase unit. This work proposes *multi-streaming*, an interface that directly guides data placement within the SSD, separating the two issues. We argue that the host system should (and can) provide adequate information about data lifetime to the SSD. It is the responsibility of the SSD, then, to place data with similar lifetime (as dictated by the host system) into the same erase unit.

Our design introduces the concept of *stream*. A stream is an abstraction of SSD capacity allocation that stores a set of data with the same lifetime expectancy. An SSD that implements the proposed multi-stream interface allows the host system to open or close streams and write to one of them. Before writing data, the host system opens streams (through special SSD commands) as needed.

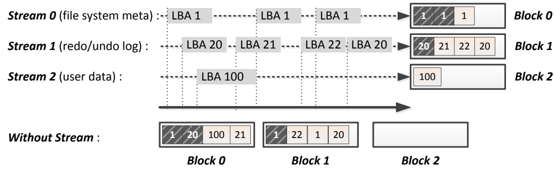


Figure 2: *The multi-streamed SSD writes data into a related NAND flash block according to stream ID regardless of LBA.* In this example, three streams are introduced to store different types of host system data.

Both the host system and the SSD share a unique stream ID for each open stream, and the host system augments each write with a proper stream ID. A multi-streamed SSD allocates physical capacity carefully, to place data in a stream together and not to mix data from different streams. Figure 2 illustrates how this can be achieved.

We believe that the multi-stream interface is abstract enough for the host system to be able to tap, with convincing use cases and results (as discussed in Section 4). Furthermore, the level of information delivered through the interface is concrete enough for the SSD to optimize its behavior with. There are other proposals to specify write data attributes, like access frequency [11]. However, it is not straightforward for the SSD to derive data lifetime from the expected frequency of data updates.

3.2 Implementation

We implemented the proposed multi-stream interface on the currently marketed Samsung 840 Pro SSD [14]. Because 840 Pro is based on the SATA III interface, we piggyback stream ID on a reserved field of both regular and queued write commands as specified in the AT attached (ATA) command set [5]. Our multi-streamed SSD prototype currently supports four streams (Stream 1 to 4) on top of the default stream (Stream 0).

We modified the Linux kernel (3.13.3) to have a conduit between an application and the SSD, through the file system and the layers below. More specifically, an application passes a stream ID to the file system through the `fadvise` system call, which, in turn, stores the stream ID in the `inode` of the virtual file system. When dirty pages are flushed into the SSD, or the application directly requests a write operation with the direct I/O facility, we send along the write request the stream ID (that can be retrieved from the associated `inode`).

4 Evaluation

4.1 Experimental setup

To evaluate the multi-streamed SSD, we conduct experiments that run Cassandra [7] (version 1.2.10), a widely deployed open-source key value store. All experiments were performed on a commodity machine with a quad-core Intel i7-3770 3.4GHz processor. We turned off power management for reliable measurements.

Table 1: Stream ID Assignment

	system	Commit-Log	flushed data	compaction data
Normal	0	0	0	0
Single	0	1	1	1
Multi-Log	0	1	2	2
Multi-Data	0	1	2	3∼4
Ratio of written data (%)	1.0	48.6	31.3	4.4, 14.7

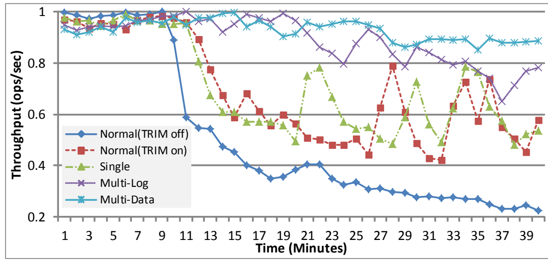
Cassandra optimizes I/O traffic by organizing its data set in or append-only "sorted strings tables" (SSTables) in disk. New data are first written to a commit log (CommitLog) and are put in a table in the main memory (MemTable) as they are inserted. Contents in the MemTable are flushed to a SSTable once they accumulate to a certain size. Since SSTables are immutable, several of them are "compacted" periodically to form a new (large) SSTable to reduce the space and time overheads of maintaining many (fragmented) SSTables. As the compaction process repeats, valid data gradually move from a (small) SSTable to another in a different size tier. We take into account how data are created and destroyed in Cassandra when we map writes to streams.

Table 1 lists four different mappings that we examine. Normal implies that all data are mapped to the default stream (Stream 0), equivalent to a conventional SSD with no multi-streaming support and is the baseline configuration. In Single, we separate all data from Cassandra into a stream (Stream 1). System data, not created by the workload itself, include the `ext4` file system meta and journal data and still go to Stream 0. Multi-Log carves out the CommitLog traffic to a separate stream, making the total stream count three (including the default stream). Finally, Multi-Data further separates SSTables in different tiers to three independent streams. Intuitively, SSTables in the same tier would have similar lifetime while SSTables from different tiers would have disparate lifetime.

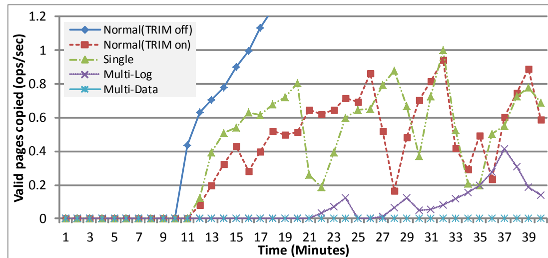
For workloads, we employ the Yahoo! Cloud Serving Benchmark (YCSB) [3] (0.1.4). We run both YCSB and Cassandra on the same machine, not to be limited by the 1Gb Ethernet. In addition, we limit the RAM size to 2GB to accelerate SSD aging by increasing Cassandra's flush frequency. The compaction throughput parameter of Cassandra was modified from 16 MB/s to 32 MB/s, as recommended by the community for SSD users.

4.2 Results

Figure 3(a) plots the normalized update throughput of all mapping configurations studied. We introduce a Normal configuration with the TRIM facility *turned off*, to gain insight about the impact of TRIM. We make the follow-



(a) Cassandra's normalized update throughput.



(b) The number of valid pages copied during GC execution.

Figure 3: *Cassandra update throughput and GC overheads, normalized.* The update throughput is shown to depend heavily on the GC overheads. We estimate GC overheads with the number of valid NAND flash pages that must be copied during GC. Trends of throughput and GC activities are similar after the 40-min. period captured in the plots.

ing key observations: (1) TRIM is shown to be critically important for the sustained performance; (2) GC overheads (Figure 3(b)) correlate very well with the throughput; and (3) Multi-Data outperforms all other configurations and sustains the throughput.

Without TRIM, Normal's performance approaches a dismal level—20% of the peak performance—, shown only briefly at the beginning of the experiment (when the SSD was relatively fresh). To put it in a different way, TRIM is very effective for Cassandra because it organizes data in a log-structured manner, writes a large file of data and deletes an entire file at a time. However, even with the benefit of TRIM, Normal still performs poorly—its performance drops from the peak by up to 53%! We do not consider Normal without TRIM any further.

The (poor) performance of Normal and others can be attributed well to the GC overheads: Valleys in plot (b) match with peaks in plot (a), and vice versa. GC overheads can be approximated by the number of valid pages that must be copied (to vacate a NAND flash block before erasing it). Because copying of valid pages involves programming (and hence consuming the bandwidth of) NAND flash memory, the ability of the SSD to serve user requests is hurt in direct proportion.

In general, our result shows that the use of multi-streaming cuts down GC overheads and in turn increases throughput (by nearly 56% when Multi-Data is com-

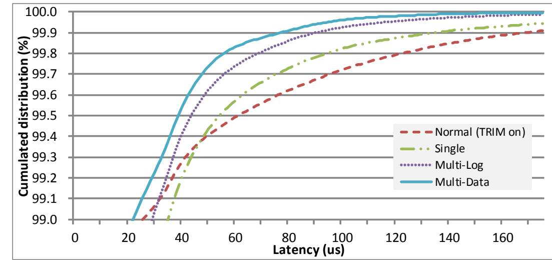


Figure 4: *Cassandra's cumulated latency distribution.* The long tail of latency distribution is also shown to be subject to GC overheads.

pared to Normal). Also shown is that how streams are allocated and mapped to application data makes a critical impact. Single bears little difference to Normal—mainly because the traffic separated with multi-streaming corresponds to only 1% of the total traffic. Multi-Log improves GC efficiency and throughput much more noticeably. The throughput now lies in between 65% and 85% of the peak. Lastly, Multi-Data, our best mapping, hits roughly 90% of the peak performance sustainably. As we intended, few GC activities are incurred, if any. We also experimented with a 15K-RPM enterprise HDD (not shown); it showed fairly consistent throughput that peaks at less than one third the performance of Multi-Data.

Figure 4 presents the latency profile obtained from the previous experiment. The plot shows that multi-streaming improves latency as well: At the 99.9th percentile, Multi-Data lowers the latency by 54%, compared with Normal and at the 99.99th percentile, by 61%.

In another experiment, we design and evaluate an in-SSD mechanism that detects multiple sequential access patterns on the fly and assigns an adequate stream ID. The goal was to gain insights into how much improvement we can obtain through automatic stream detection. Note that write patterns in Cassandra are mostly sequential. In our implementation, we detect maximum four concurrent sequential patterns and assign a stream ID to each of them. If there are more than four sequential patterns, we handed the stream ID of the least recently used sequential pattern to a newly detected one. With this mechanism, the SSD classified 71% of all data to be sequential. However, the resultant performance gain was rather marginal. This counter-intuitive result is due to how LBAs are allocated by the `ext4` file system: A large file may not always get sequential LBAs due to fragmentation. Moreover, successively created files may not get consecutive LBAs as they expand. Accordingly, in Cassandra, the chances that SSTables in the same size tier are detected as a sequential stream decrease. In the end, SSTables from different tiers and even CommitLog start to mix up across streams. Our result underscores that concrete semantic information passed by the host system

through the multi-stream interface is much more *relevant* and *robust* than automatically learned access patterns.

Finally, we examine how multi-streaming can extend the lifetime of SSDs. We have iterated a few times already that proper use of multi-streaming can improve GC efficiency. Higher GC efficiency implies that the multi-streamed SSD lowers the number of required NAND flash erase operations. Indeed, we found that Multi-Data would extend the SSD lifetime by 23%.

4.3 Discussion

Our study with Cassandra showed that an intuitive data to stream mapping can lead to large benefits in throughput, consistent latency and NAND flash lifetime on the multi-streamed SSD. We further believe that many applications and use cases will enjoy similarly large gains from the multi-streamed SSD, if reasonably good mapping is done. Other database management systems that use log-structured merge trees (like Cassandra) include HBase, LevelDB, SQLite4 and RocksDB. These applications explicitly manage data streams and orient their I/O to be sequential. In another example, consider commit (transaction) log, undo (roll-back) log and temporary table data in OLTP applications [9]. They map nicely into separate streams on the multi-streamed SSD. Lastly, some multi-head log-structured file systems and flash storage OS could relatively effortlessly steer their data writes into streams for higher, consistent performance and better media lifetime.

There are several avenues for further research. It will be interesting to develop a systematic data to stream mapping strategy that can handle multiple applications (virtual machines) running concurrently. It is also worthwhile to look at if and how multi-streaming could provide performance and fault isolation [10]. How to effectively support multi-streaming without application-level changes remains a challenge and research question. How to organize and utilize streams on multiple SSDs would be a practical, rewarding topic to explore.

5 Conclusions

We made a case for the multi-streamed SSD in this paper. We found the proposed multi-streaming concept powerful and the interface expressive; by mapping application and system data with different lifetimes to SSD streams, we demonstrated that the SSD throughput and latency QoS are significantly improved. The data mapping used in our Cassandra case study is intuitive, and similar benefits are expected from other applications with proper data to stream mapping. Our prototype SSD proves that multi-streaming can be supported on a state-of-the-art SSD and can co-exist with the traditional block interface.

6 Acknowledgments

We thank the anonymous reviewers for their constructive comments. Jaegeuk Kim and other MSL members gave helpful comments on earlier drafts of this paper. Donggun Kim contributed to the development of the prototype multi-stream firmware on the Samsung 840 Pro SSD.

References

[1] CHEN, F., KOUFATY, D. A., AND ZHANG, X. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *ACM SIGMETRICS Performance Evaluation Review* (2009), vol. 37, ACM, pp. 181–192.

[2] CHIANG, M.-L., LEE, P. C., AND CHANG, R.-C. Managing flash memory in personal communication devices. In *Consumer Electronics, 1997. ISCE'97., Proceedings of 1997 IEEE International Symposium on* (1997), IEEE, pp. 177–182.

[3] COOPER, B. F., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 143–154.

[4] HSIEH, J.-W., KUO, T.-W., AND CHANG, L.-P. Efficient identification of hot data for flash memory storage systems. *ACM Transactions on Storage (TOS) 2*, 1 (2006), 22–40.

[5] INTERNATIONAL COMMITTEE FOR INFORMATION TECHNOLOGY STANDARDS. Technical Committee T13 AT Attachment, March 2011.

[6] KAWAGUCHI, A., NISHIOKA, S., AND MOTODA, H. A flash-memory based file system. In *Usenix Winter* (1995), pp. 155–164.

[7] LAKSHMAN, A., AND MALIK, P. Cassandra, July 2008.

[8] LEE, S., SHIN, D., KIM, Y.-J., AND KIM, J. Last: locality-aware sector translation for nand flash memory-based storage systems. *ACM SIGOPS Operating Systems Review 42*, 6 (2008), 36–42.

[9] LEE, S.-W., MOON, B., PARK, C., KIM, J.-M., AND KIM, S.-W. A case for flash memory ssd in enterprise database applications. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), ACM, pp. 1075–1086.

[10] LU, L., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Fault isolation and quick recovery in isolation file systems. In *Proceedings of the 5th USENIX Conference on Hot Topics in Storage and File Systems* (Berkeley, CA, USA, 2013), HotStorage'13, USENIX Association, pp. 1–1.

[11] NVM EXPRESS CONSORTIUM. NVM Express 1.1a Specification, September 2013.

[12] PARK, S.-H., PARK, J.-W., KIM, S.-D., AND WEEMS, C. C. A pattern adaptive nand flash memory storage structure. *Computers, IEEE Transactions on 61*, 1 (2012), 134–138.

[13] ROSENBLUM, M., AND OUSTERHOUT, J. K. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS) 10*, 1 (1992), 26–52.

[14] SAMSUNG. SAMSUNG SSD 840 PRO, September 2012.

[15] SHIMPI, A. L. Micron Announces 16nm 128Gb MLC NAND, SSDs in 2014, July 2013.

[16] SHIMPI, A. L. The ssd anthology: Understanding ssds and new drivers from ocz, February 2014.

[17] SMITH, K. Understanding ssd over-provisioning, January 2013.