Measuring Solid-State Drive Behavior

Daniel Myers Matt Sinclair
CS 736 Final Project Report Fall 2010
The University of Wisconsin-Madison
{dsmyers, sinclair}@cs.wisc.edu

Abstract

Solid-state drives (SSDs) are a potentially revolutionary new storage technology, but it is not clear that current generation drives are inherently superior to hard disks. In order to understand the performance of these new technologies, we measure two commercial SSDs, one from Intel, the other from Kingston, on a workload of 4K writes to randomized addresses across the entire address space of each drive. Our results reveal that there is a strong difference between a fresh drive's early performance and its steady-state behavior, with the drive entering steady-state after fewer than 100000 writes. To gain insight into the effect of the flash translation layer (FTL) on drive performance, we use simulation and analytic modeling to investigate FAST, a representative hybrid FTL algorithm.

1 Introduction

Solid-state disks are a potentially revolutionary storage technology, but currently they are not obviously superior to disks. While SSDs can be faster, more reliable, and more energy efficient than disks [4], they have higher cost per byte and a limited number of write-erase cycles before they permanently wear out. Additionally, the internal behavior of SSDs, while it has been studied before [1, 3, 4], is not well understood. Performance varies widely from vendor to vendor, and little exists in the way of understanding why, mostly due to SSD internals being intensely guarded intellectual property of SSD manufacturers.

The goal of our project is to collect data on SSD performance, then try to understand the factors influencing write performance on the SSD. We focus on SSD write performance instead of read performance because random writes provide the greatest challenge to the drive's internal management algorithms. To perform this study, we created a microbenchmark that writes to randomized locations throughout the SSD and measures the latency of each write. To try and understand the factors influencing performance, we used an SSD extension to the DiskSim simulator [6]. We also created an analytical model of merge times to help understand the internal behavior of the SSD. We believe that our results will help the storage community understand and model the behavior of SSDs.

2 Background

2.1 *NAND Flash*

There are two types of flash memories – NOR and NAND flash. Since all of the drives we used in this project are NAND flash drives, we omit details on NOR flash drives, which can be found elsewhere [8]. NAND flash memory has two categories: Single-Level Cell (SLC) and Multi-Level Cell (MLC). SLC memory cells store only one bit, while MLC memory cells store two or more. This makes latency of access in MLC much longer than in SLC. Additionally, SLC tends to last significantly longer, which is why they are usually used in high-end SSDs.

Both SLC and MLC flash memories are composed of multiple chips, each of which is segmented into multiple planes. Each plane usually contains several thousand blocks. Blocks usually contain 64 or 128 pages, each of which is can typically have 2KB or 4KB of data written to it as the smallest writeable unit (writing is done at page granularity).

Flash memory has a limited number of erases that can be performed on a memory cell before that cell is worn out, and can no longer store data or be used. For MLC drives, each cell usually support approximately 10000 erase cycles, while SLC cells usually support approximately 100000 erase cycles. Flash memory cells must be erased before they are written to, so this effectively limits

the number of writes that can be performed to a cell.

2.2 *Flash Translation Layer*

The Flash Translation Layer (FTL) is an important component of an SSD. The FTL is an intermediate software layer that is responsible for translating logical addresses supplied by the OS into physical addresses within the SSD. It is also responsible for hiding the erase operations and erase-before-write operations from the OS. Internally, the FTL maps each write request from the host to an empty location (free page) in flash memory and manages the mapping information to allow future accesses to that data.

There are two general ways of mapping writes to the flash memory: page mapped and blocked mapped. Page mapped FTLs can map new data to any page within an SSD. Block mapped FTLs map new data to any block within the SSD, but only to one specific page within that block. In general, page mapped FTLs have greater flexibility in positioning data, but require extremely large mapping tables. Block mapped FTLs use smaller, simpler mapping tables, but have problems with workloads that frequently overwrite the same page. To obtain the benefits of both schemes, most commercial SSDs use *hybrid* FTL algorithms. In these schemes, most of the drive is block mapped, but a small number of *log blocks* are page mapped. The log blocks provide flexibility for dealing with overwrites, but the mapping tables remain small and simple. When we run out of space in the log blocks, we must merge data blocks and log blocks together to free up space. [5] provides a good overview of FTL, the different FTL mapping styles, and how different merging algorithms affect performance.

Garbage collection (also known as cleaning) and wear-leveling are other important tasks of the FTL. Garbage collection is needed because blocks must be erased before they are used. The garbage collector works by scanning the SSD blocks for invalid pages, then reclaiming those invalid pages. This process is similar to that of the Log-Structured File System [9]. Wear-leveling is necessary because most workloads write to a subset of blocks frequently, while rarely writing to other blocks. Because each block of flash memory only has a limited number of write-erases before it is worn out, without wear leveling, the frequently written to blocks would easily wear out well before the other blocks. Wear leveling helps solve this problem by shuffling cold (unused/less frequently used) blocks with hot (frequently used) blocks to balance out the number of writes over all of the flash memory blocks.

2.3 *Write Cliff Phenomenon*

Since we are focusing on writing to SSDs, we omit background on reading from SSDs. More information on this can be found elsewhere [1, 5]. An interesting phenomenon in consumer-grade SSDs, that does not appear to occur in high-end SSDs, is the ”write cliff” phenomenon [4]. In consumer-grade SSDs, write performance degrades significantly after the drive is filled the first time and the drive’s internal garbage collection and wear-leveling routines must begin to run. We are interested in seeing if this phenomenon holds for our experiments.

3 Related Work

There has been other work done in this area previously. Our work is most similar to [3], who conducted a series of experiments and measurements on multiple SSDs of different qualities for sequential and random workloads. They found a strong correlation between performance and randomness of data accesses for both reads and writes. Our work differs from theirs in that we focus on write performance in randomized workloads where we are continuously writing to the disk to prevent the cleaner from operating in the background, whereas their work is looking more at trends when multiple operations are going on simultaneously.

[1] implements an SSD add-on to the DiskSim simulator to explore design tradeoffs and their effect on SSD performance. Specifically, data placement, parallelism, write ordering, and workload management are presented as key design choices to considering when structuring a workload. Their work is focused on global design tradeoffs, whereas we are focusing specifically on how writes to random locations on the disk perform. [6] also implemented an SSD add-on to the DiskSim simulator that simulates the FAST FTL algorithm.

4 Implementation

To measure the effect of writes in SSDs, we created a microbenchmark that performs random 4K (i.e. single page) writes over the entire range of the drive. We perform these writes continuously to prevent the garbage collector from running in the background and artificially improving performance, because we are more interested in the behavior of the disk when cleaning can not be performed.

We synchronize after each write to force the write to take place and avoid the skewing effect on timing of buffering. The microbenchmark writes directly to the raw disk through the use of the `O_DIRECT`, `O_RDWR`, `O_SYNC`, and `O_NOATIME` flags of Linux's `open()` function. We define the latency of access for a single write to the SSD to be the time it takes to perform the write *and* flush that write to the disk. We do not include the negligible seek time in this calculation.

To reduce the volume of data we collected, we do not record exact times for writes with latencies less than 1.5 ms. Instead, we use a compression scheme that bins small access latencies at 300 μs intervals. Small access latencies are the common case, requiring only basic writing operations to the SSD, but no cleaning or other FTL operations. Access latencies larger than 1.5 ms require some sort of extra work on the part of the SSD such as cleaning or merging, so we store these times exactly.

To prevent the garbage collector from recovering and reclaiming blocks in the background while we are writing out the access latency, we perform 100000 writes before writing out data to an output file. We then repeat this process five times in a single run of our microbenchmark. While it is possible for the garbage collector to reclaim blocks while we are writing out the access latency results to the output file, we found 100000 to be a good compromise point between having too much data written out at once, which gave the garbage collector enough time to reclaim blocks, and have too little data written out at once, which would prevent us from observing SSD behavior under continuous writes.

5 SSD Results

We collected results for our microbenchmark on two different consumer-grade SSDs. The first drive is a Kingston SNV125. It has 30 GB of storage and costs $78.99 retail.[1] The second drive is an Intel X25-M. It has 80 GB of storage, has a rated throughput random write throughput of 27.3 MB/s, and costs $179.49 retail. Our data collection reveals that both the Intel and Kingston drives can exhibit significant variations in performance as the number of writes made to the drive increases. Both drives had approximately 150 GB of data written to them in the course of our experiments.

[1] No specs given for random writes.

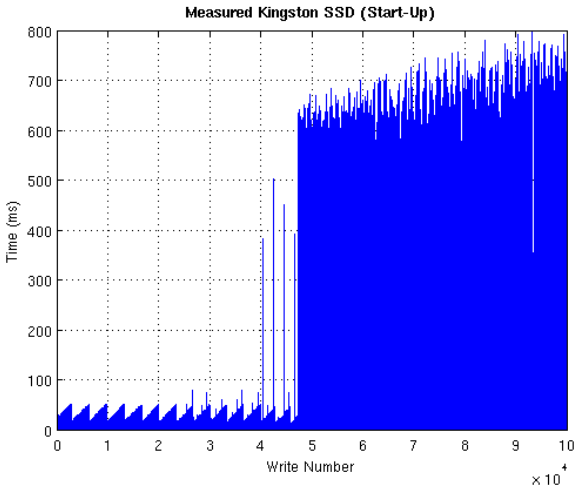

Figure 1: Measured Kingston SSD in start-up. The y axis is the execution time (in ms) of the current write and the x axis is the write number for the current test.

5.1 *Kingston SSD Results*

Figure 1 shows the measured times (in ms) for the first 100000 writes made to the Kingston drive. The "great failure" that occurs at approximately 45000 writes is obvious. After a small initial grace period where latencies are low and all writes are fast, the drive enters a steady-state where latencies are increased and variability is much higher.

Figure 2 shows a close view of 5000 writes taken from the Kingston drive in steady-state, after approximately 140 GB of data have been written to the drive. Writes can be divided into four classes based on their latencies:

1. The vast majority of writes that take fewer than 300 μs.
2. The small peaks, representing latencies of approximately 15 ms.
3. The middle peaks, with latencies around 400 ms.
4. The highest peaks, with latencies mostly around 800 ms, but rarely reaching as high as 1200 ms.

Analysis reveals that 97% of writes take fewer than 300 μs, but the remaining 3% of writes are sufficient to raise the mean write time to 13 ms.

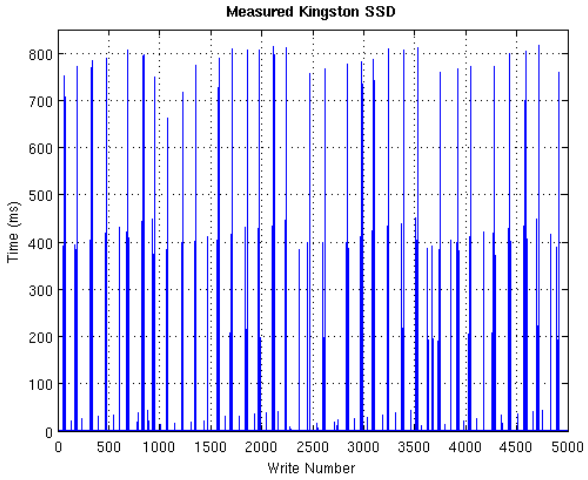

Figure 2: Measured Kingston SSD in steady-state. The y axis is the execution time (in ms) of the current write and the x axis is the write number for the current test.

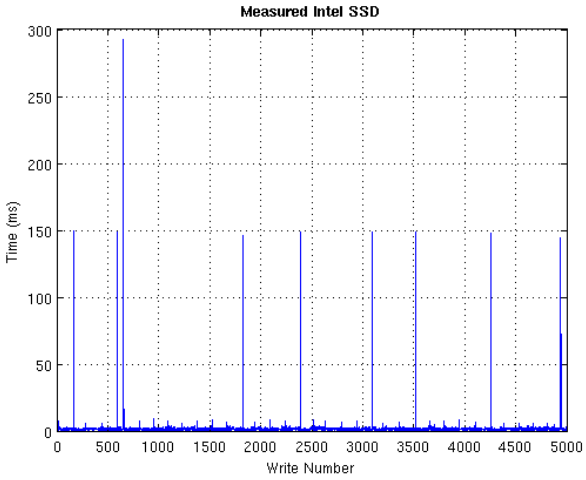

Figure 3: Measured Intel SSD in steady-state. The y axis is the execution time (in ms) of the current write and the x axis is the write number for the current test.

5.2 *Intel SSD Results*

The Intel drive also exhibits a transition between start-up and steady-state, but the change is less dramatic. Figure 3 shows 5000 writes from the drive in steady-state. The steady-state pattern is clearly different from the Kingston drive, indicating that the two drives have very different internal resource management strategies. We did not observe a distinct start-up phase on the Intel drive. This may be due to the internal behavior of the drive, or it may be possible that the drive was slightly used by another experimenter before we received it.

The Intel drive still exhibits high variability between the class of writes taking less than 1.5 ms and the class of writes taking approximately 150 ms, but the most expensive writes are relatively rare – writes with times in excess of 200 ms occur at a rate of only one per 2182 writes. Writes in the middle category, with peaks near 150 ms, occur every 529 accesses, on average. In contrast to the Kingston drive, there is considerable variability among the smallest writes – only 15% of writes take fewer than 300 μs and fully 33% of writes take more than 1.5 ms. In spite of this, the Intel drive still manages to record a mean latency of 1.5 ms, significantly faster than the Kingston. This is likely because the Intel drive is of higher quality than the Kingston drive.

Page size	4 KB
Pages per block	64
Drive size	1 GB
Num. log blocks	128
Page read latency	25.25 μs
Page write latency	101.25 μs
Block erase latency	1500 μs

Table 1: Simulation Parameters

6 Simulating SSD Behavior with DiskSim

Unfortunately, we lack useful documentation for the Kingston and Intel drives, so we are unable to explain the reasons for their observed behaviors. For example, we do not know which FTL algorithms are used by the two drives.

In order to gain additional insight, we used simulation to investigate one particular hybrid FTL algorithm, Fully Associative Sector Translation (FAST) [7]. The simulator [6] is a modification of DiskSim [2] and has the parameters shown in Table 1. Figure 4 shows the behavior of the simulated FAST SSD in steady-state.

6.1 *FAST*

Like most hybrid schemes, FAST uses a mixture of data blocks and log blocks to manage the SSDs internal data.

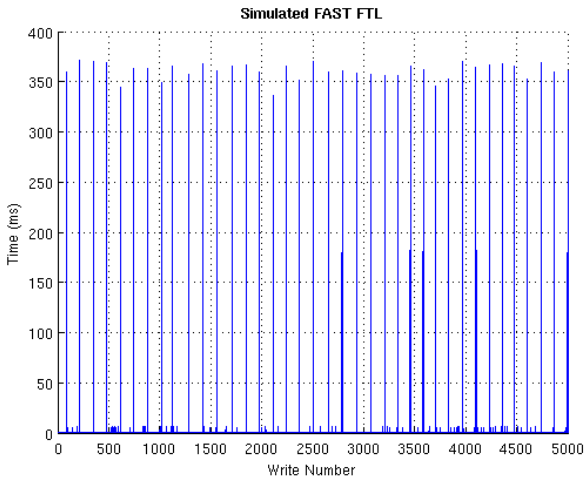

Figure 4: Simulated FAST SSD in steady-state. The y axis is the execution time (in ms) of the current write and the x axis is the number write in the sequence for the current test.

If possible, new data is written directly to a data block. When the desired location in the data block is already occupied, FAST writes to a log block. Under certain circumstances, FAST performs *merge* operations to clear space for new writes.

FAST recognizes two kinds of log blocks: *random* log blocks and one special *sequential* log block. The sequential log is designed to help FAST deal with sequential writes to one block, which may be interspersed with random writes to other blocks. Any write to the first page in a block is directed to the sequential log – we refer to these as *zero-page* writes. If the sequential log already contains data, it is first merged with a data block and cleared to create a clean log for the new write. After the initial zero-page write, any sequential writes to the same active block will be directed to the sequential log. The sequential log is merged and cleared when one of the following occurs:

- The log fills with all of the sequentially written pages from one block.
- A non-sequential write to the active block occurs.
- Another zero-page write to a different block occurs.

Random log blocks are used to log writes that do not go directly to data blocks or to the sequential log block. When all of the random log blocks are filled, FAST performs a *full merge* to garbage collect one data block. Each page written to the log block overwrites one page in an existing data block. In a full merge, all of the pages written to the log block are merged back into their corresponding data blocks. Thus, the cost of the full merge depends primarily on the distinct number of data blocks dirtied by pages in each log block. In the worst case, each page of the log may overwrite a page in a different data block, requiring all of those data blocks to be read and rewritten before completing the merge and freeing the log block.

6.2 *The Effect of Merges*

Knowledge of the underlying FTL algorithm for FAST allows us to explain the behaviors in Figure 4. Most of the writes do not require any merging and complete in the minimum write time of 101 μs (our simulations measure only the physical write time and ignore any overhead for bus contention or data transfer).

There are 64 pages in a block, so on average every 64th request will be a zero-page write. These writes trigger a small merge operation to clear the sequential log block before writing the new zero-page. These small merges cause the small peaks of approximately 5 ms. Each zero-page merge requires the following steps:

- Read any valid data from the old data block.
- Copy the valid data into the current sequential log block.
- Relabel the current sequential log block as the new data block.
- Erase the old data block.

Analytically, we can express the time required for a zero-page merge as (see Table 2 for descriptions of the terms):

$$T_{zero-page} = N_{valid}(T_r + T_w) + T_e. \quad (1)$$

Examining the simulator data shows that $N_{valid} = 32$ in the average case. Using this value, the expected time for a zero-page merge (in microseconds) is

$$T_{zero-page} = 32(25.25 + 101.25) + 1500 = 5548, \quad (2)$$

which agrees with the simulated results. This means that, for our random workload, the sequential log block is practically never used for its intended purpose. Instead, it only adds additional overhead to every 64th write.

Filling all of the random log blocks triggers a full merge. Because writes are randomized over a large address space,

Term	Description
N_{valid}	Number of valid pages in the block
N_{data}	Number of valid data blocks to be merged
T_r	Expected page read time
T_w	Expected page write time
T_e	Expected page erase time

Table 2: Description of Terms used in Modeling Equations

each page in the log block is likely to correspond to a different data block. Thus, each full merge is likely to involve 64 data blocks – the worst possible case.

For each data block involved in the full merge, FAST performs the following steps:

- For each occupied page in the data block, copy the most recent version of that page into a new block.
- Update the mapping table to include the new, updated data block.
- Erase the old data block.

After all data blocks involved in a merge have been processed, FAST finally deletes the log block. Thus, the expected time required for a full merge is given by

$$T_{full} = N_{data}(N_{valid}(T_r + T_w) + T_e) + T_e. \quad (3)$$

For the simulated data, $N_{data} = 64$ and $N_{valid} = 32$. Using these values, we predict an average merge time (in microseconds) of

$$T_{full} = 64(32(25.25+101.25)+1500)+1500 = 356572, \quad (4)$$

which agrees with the simulated results.

Though these analytic models can accurately predict the time required for merges, they are still incomplete. In particular, we have not yet derived an expression for the number of valid pages in a data block when a merge takes place. This is a critical parameter in both equations. However, the models do provide some insight into the FTL algorithm's of the SSD drives in Section 5.

7 Conclusion

We have investigated the performance of two commercial SSDs and one simulated FTL algorithm. Our results reveal several interesting facts about SSD performance on workloads of random writes. First, there is a considerable difference between the very early performance of a drive and its actual performance in steady-state. For this reason, performance measures taken on fresh drives may be far too optimistic. Further, the transition into steady-state occurs early in the life of the drive, after fewer than 50000 writes. This differs somewhat from the "write cliff" phenomenon seen in [4], where performance did not rapidly decrease until after the entire drive had been filled once.

Second, the behavior of the flash translation layer is a critical factor in drive performance, particularly the frequency and severity of merge operations used to free space in the drive. Different FTL algorithms may be better suited to different workloads. Analytic modeling can provide some insight into factors influencing the performance of a particular FTL algorithm, but deriving the exact parameters for each model may be difficult.

Finally, there are several questions we were unable to answer in this project. While the models provide us with some insights into the FTL algorithm's of the Intel and Kingston drives, we do not understand the internal operation of the Intel and Kingston drives, so we cannot provide and explanations for their performance. Our analytic models are incomplete, with some important parameters left to derive. Finally, we do not know how SSD performance will change as cells in the drive begin to wear out and fail.

Acknowledgments

The authors would like to thank Yiying Zhang for her help setting up the SSD drives and machines to run our experiments on. We would also like to thank Aayush Gupta and Nitin Agrawal for providing us with their SSD add-ons to DiskSim, which we used for our simulations.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[2] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, and Contributors. The disksim simulation environment - version 4.0 reference manual. Technical report, Carnegie Mellon, 2008.

[3] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09, pages 181–192, New York, NY, USA, 2009. ACM.

[4] L. G. Harbaugh. Storage smackdown: Hard drives vs ssds. http://www.networkworld.com/reviews/2010/041910-ssd-hard-drives-test.html, April 2010.

[5] D. Jung, J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. Superblock ftl: A superblock-based flash translation layer with a hybrid address translation scheme. *ACM Trans. Embed. Comput. Syst.*, 9:40:1–40:41, April 2010.

[6] Y. Kim, B. Taurus, A. Gupta, and B. Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. In *In Proceedings of the First International Conference on Advances in System Simulation*, Sep 2009.

[7] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Transactions on Embedded Computing Systems*, 6(3), July 2007.

[8] M-Systems. Two technologies compared: Nor vs. nand. In White Paper, 2003.

[9] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, February 1992.