**Analysis of SSD architecture and read/write on Solid State Drive**



MENG XU

DRISHTI GOEL

AESHA ALTHNYAN

CATHALINA FONTENELLE

May 2019

Table of Contents

[**1.Introduction**](#_gjdgxs) **3**

[1.1 Motivation](#_30j0zll) 4

[1.2Scope](#_1fob9te) 4

[1.3Testing Environment & Platforms](#_3znysh7) 5

[**2.BACKGROUND INFORMATION OF SSD**](#_6l7jbhx05zw1) **6**

[2.1 SSD Architecture](#_gmpgvz551rtm) 6

[2.2 Read and Write in SSD](#_itctsseeqjsz) 7

2.3 Wear Leveling 9

[**3.IMPLEMENTATION**](#_p8kme7no03fb) **9**

[3.1 Intel Open Storage Kit](#_tyjcwt) 9

[3.1.1 Implementation Detail](#_3dy6vkm) 10

**4. RESULTS 12**

**5. DISCUSSION AND IMPLICATIONS** 21

6. [**CHALLENGES**](#_2s8eyo1) **22**

**7.0** [**C**](#_17dp8vu)**ONCLUSION 24**

[**References**](#_3rdcrjn) **25**

# 1.Introduction

## 1.1 Motivation

The main motivation for doing this project was to understand the internal architecture and working of a Solid-State Drive. Since we already know that SSD’s are way faster than traditional spinning disks, we wanted to research more into the internal architecture and mechanism of Flash based drives.

One of the key factors which got our attention was the presence of different factors that delimited the SSD performance and we tried to understand and implement these intrinsic factors.

## 1.2 Scope

The project will span across the implementation of different factors that have affect the overall performance of an SSD. These tests are implemented using Intel Open Storage Toolkit. The factors are:

1. Observe individual time performances for sequential and random reads/writes.
2. Effects of increased read randomness on latency.
3. Effects of increased random writes on the performance.
4. Effect of disk cache on the performance.
5. Read and Write Interference with each other.
6. Background operations affecting the performance.
7. Effect of Increased workload randomness (seek range) on performance.
8. Effect of Fragmentation on the performance.

Due to lack of the time, we were able to analyze on the first five factors and the last three factors are kept for the future work.

## 1.3 Testing Environment & Platforms

* Installing Intel Open Storage ToolKit on the system being tested.
* Heracles
  + HGST SN100 1.6TB NVMe 2.5” SSD
  + 120GB Intel DC S3510 2.5” SATA 6Gbps MLC SSD
* Laptop

# 

# 2.BACKGROUND INFORMATION OF SSD

## 2.1 SSD Architecture

A SSD is a small complete, compact system where every component is soldered on a PCB. The components include: flash memories (NAND), microcontroller (SSD Controller), DRAM cache, filtering capacitors to stabilize power supply and temperature sensors. The NANDs are arranged in different channels in order to improve performance. The NAND flash cell is based on the Floating Gate (FG) technology, which contains a MOS transito built with two overlapping gates. One gate is completing surrounded by oxide which allows it to trap electrons, and be able to keep its charge for a long period of time.

Each transistor acts as a memory cell: the Single Level Cell (SLC) stores one bit of information, while the Multi-Level Cell (MLC) stores 2 bits per cell. The main core of the Flash memory is the NAND string, which can be made up of 32, 64 or 128 cells connected in series. Wordlines connect strings of cells together, with all the NAND strings sharing the same wordline (WL), forming a block. Logical pages are made up of cells with the same WL. The capacity of the cell determines the number of pages per WL. The arrangement of cells into strings, pages, blocks and channels are a means of increasing both the storage density and read/write performance.

In NAND, the logical page is the small addressable unit for reading and writing, however, for erasing, the logical block is the smallest unit to be erased.

The NAND interface has become the bottleneck in recent times, with the data access times being much less than the data transfer time. The original interface of SATA provides speeds of approximately 6gbps, but recent developments has produced a PCIe interface with speeds exceeding 24gbps.

The Memory Controller has two main tasks: to provide an interface between the host and flash memories, through the implementation of standard protocols to ensure electrical and logical compatibility between the host and SSD; and to handle data efficiently, maximize transfer speed and help with data integrity and information retention the functions designated to the Flash File System (FFS). The FFS tasks are managed in four (4) main functions, which are implemented in hardware or firmware. These functions are: Wear-leveling Management, Garbage Collection, Error Correction Code and Bad Block Management.

## 2.2 Read and Write in SSD

For the ssd write operation, there are different flows for different situations, but all the write depends on those two characteristics of the NAND flash:NAND Flash must be written in units of page each time, and can only write to idle pages, and cannot overwrite pages that have original content.When erasing data, it can only be erased in blocks due to high voltage.

So, there are two kinds of writing, one is the new write, another one is the update. For the new write, it first need to find a free page and then write data to the idle page and update mapping table after doing the write. For the update, Since the SSD cannot do the overwrite, it need to first find a free page H and read the data in page G to the buffer inside the SSD,and update the updated bytes to the buffer. Then write the data in the buffer to H, after that update G page in mapping table, set to invalid page and last update the H page in the mapping table and add the mapping.

Inside the SSD, over-provisioning and garbage-collection play important role for long time writing. Over-provisioning means that the actual storage space of the SSD is larger than the writable space. For example, an SSD with a usable capacity of 120G may have 128G of actual space. Why do you need over-provisioning? Suppose that there are two blocks in the system, and finally there are two invalid pages left. At this time, to write a new page, according to the NAND principle, two invalid pages must be erased before they can be used. For writing. At this point, you need to use the extra space provided by the SSD to use the garbage-collection method to sort out the available space. So, do the writing in this point, First, find a free block from the over-provisoning space. Copy the all information of Block0 and the first one of Block1 to the free block and then erase Block 0, then Copy Block1's rest to Block0, then Block0 has two free pages and erase Block1, now it’s to do the writing.

# 

**2.3 Wear Leveling**

The wear leveling algorithm keeps track of (1) the ages of all blocks, (2) a timestamp for each block marking the time in which it was last erased, (3) the average length of time it takes a block to be erased, and (4) the current time. Using this information, the WL module can identify particularly young blocks that have not been erased for a very long time, and can target them for static wear leveling.

The type of wear leveling is dynamic wear-leveling strategies. It allow maintaining several free blocks of different ages in each LUN. The overall goal is to associate hot data with young blocks and cold data with old blocks. Temperature detection for pages can be done by :

* 1. assuming the pages migrated in static wear-leveling are cold, and everything else is hot
  2. using information about the temperature of data coming through an open interface from the application.

# 3.IMPLEMENTATION

## 3.1 Intel Open Storage Kit

It contains and iSCSI initiator and target, a SCSI RAM disk, a block-level micro-benchmark, a performance monitor and I/O tracing and replay. It can generate various types of I/O workloads to directly access block devices with different configurations, such as read/writes ratio, random/sequential ratio, request size, time etc. It reports bandwidth, IOPS and latency [1].

## 3.1.1 Implementation Detail

The Intel Storage Toolkit stimulator was installed on a user directory in Heracles and a student laptop running Ubuntu 18.0.4. The commands for changing the configuration and obtaining the results were run on Heracles node 5, on two separate occasions, as well as on the laptop. All four results were analyzed to verify the results and check for any major anomalies. The data obtained from running the commands on Heracles and the laptop is seen in Appendix 1 and 2. Within each ./fitness command, there were 3 iterations for verification.

The ./fitness command is used with various switches to check for the performance test. The below switches have been used for ./fitness for the project implementation [2]:

* ***--wr <int>*** This switch specifies the read to write ratio for the workloads in % “--wr 0” would be all reads and “- -wr 100” would be all writes on the workload.
* ***--wrnd <int>*** This switch specifies the % of random writes among the write operations in the workload. “- -wrnd 0” would be sequential writes in the workload.
* ***-- rrnd<int>*** This switch specifies the % random reads among the write operations in the workload. “- - rrnd 0” would be 100% random reads and “—wrnd 0” would be sequential reads in the workload.
* ***--qdep <int>*** This is the queue depth and specifies in int value as to how many operations can be queued at a time.
* ***-- wrsz <int>*** This switch specifies the write request size in the workload and can be at maximum of 256 Kb for this tool.
* ***--rdsz <int>*** This switch specifies the read request size in the workload and can be at maximum of 256 Kb for this tool.
* ***--wr\_stride<int>*** In Kb is the stride value for the data to be written in the memory during the write operations.
* ***--rd\_stride<int>*** In Kb is the stride value for the data to be read from the memory during the read operations*.*
* ***--Iters <int>*** This switch specifies the number of iterations to be performed to run multiple test for consistency check.
* ***--direct*** This switch if specified, runs the workload directly on the SSD without the influence of the page cache.
* ***--warm <int>*** This switch is used to warm all data blocks to maintain consistency among all bocks rather than hot or cold blocks.
* ***--test <int>*** This is time in seconds that the test should be performed.
* ***-- cap <int>*** In Mb specifies the capacity of the test file to be used for the operations.

The above directions can be utilized in varieties as changes to ./fitness to play out the execution of the task and check for the execution measures with yield in the organization of Throughput in Mb/sec, IOPS and Latency in milliseconds.

The commands and their respective conditions check is as below:

The defaults used for all command to maintain a uniform performance check are:

--qdep 64 \*\*queue depth of 64\*\*

-- wrsz 64 \*\* write request size of 64Kb\*\*

--rdsz 64 \*\* read request size of 64Kb\*\*

--rd\_stride 4 \*\* read stride size of 4Kb\*\*

--wr\_stride 4 \*\* write stride size of 4Kb\*\*

We have played out the accompanying investigations to comprehend the various parts of execution in SSD.

## 4.0 Results and Discussion

**4.1 Random/Sequential Read/Write with caching**

The first test carried out was to determine the random/sequential read/write values, through the command:

*./fitness --file tesr --wr ? --rrnd ? --wrnd ? --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 --rd\_stride 4 --iter 3.*

The values changed in each configuration along with the test results for the parameters of bandwidth, IOPS and latency is seen in Table 1.

***Table 1. Bandwidth, IOPS and Latency results with cache***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | --wr | --rrnd | --wrnd | bandwidth | iops | latency |
| Random read (r-read) | 0 | 100 | -- | 37749.01 | 603979.67 | 0.01 |
| Random write (r-write) | 100 | -- | 100 | 68.13 | 1083.67 | 37.01 |
| Sequential read (s-read) | 0 | 0 | -- | 38473.59 | 615573.33 | 0.02 |
| sequential write(s-write) | 100 | -- | 0 | 87.64 | 1396.00 | 8.91 |

The results which are further illustrated in the Figures 1-3 show that the best performance were obtained with reads, with little difference between random and sequential reads. The read performance was over 500% better than the write performance. Random write performed worse with regards to latency.

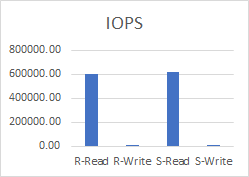
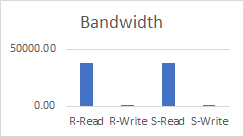


Figure 1. Bandwidth of read/write with cache Figure 2. IOPS of read/write with cache

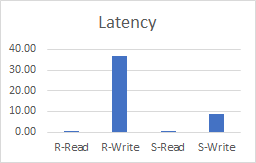


Figure 3. Latency of read/write with cache

**4.2. Random/Sequential Read/Write without caching**

We ran the following base command to determine the bandwidth, IOPS and latency without caching. The --direct parameter was used to ensure the tests were run directly on the SSD.

*./fitness --file test --wr ? --rrnd ? --wrnd ? --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 --rd\_stride 4 --direct --iter 3.*

The trend of the results were very similar to the results with caching as seen in Table 2 and Figures 4-6, except that there were no difference between random and sequential writes. When compared non-caching to caching, the results from caching were by far the best performance.

***Table 2. Bandwidth, IOPS and Latency results without caching***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | --wr | --rrnd | --wrnd | Bandwidth | IOPS | Latency |
| Random read (r-read) | 0 | 100 | -- | 1379.19 | 22060.67 | 2.90 |
| Random write (r-write) | 100 | -- | 100 | 241.22 | 3853.00 | 16.60 |
| Sequential read (s-read) | 0 | 0 | -- | 1367.70 | 21876.67 | 2.92 |
| sequential write(s-write) | 100 | -- | 0 | 243.81 | 3894.67 | 16.43 |

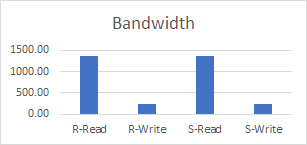


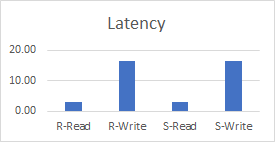
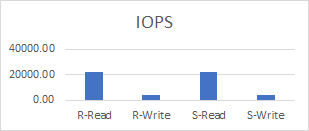
Figure 4. Bandwidth of read/write without cache 

Figure 5. IOPS of read/write without cache Figure 6. Latency of read/write without cache

**4.3. Does Random writes have the worst performance.**

As seen in our first experiment, 100% random writes had the greatest latency by a very wide margin. To test whether all random writes have terrible performance, we tested the performance on a range of random values from 0 to 100 and plotted the values on graphs seen in Figures 7-9. The command used for this test is as follows:

*./fitness --file test --wr 100 --rrnd ? --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 --rd\_stride 4 --iter 3.*

The results shows that random writes values for both bandwidth and IOPS were fairly constant between the range of 10-100%. Latency steadily climbed from 0-90, with a small drop at 100% randomness. The increase in latency while bandwidth and IOPS remain fairly steady would suggest that the lower performance is due to seek time in getting all the pages of the requested information.

***Table 3. Bandwidth, IOPS and Latency results from various levels of random writes***

|  |  |  |  |
| --- | --- | --- | --- |
| Randomness | Bandwidth | IOPS | Latency |
| 0 | 163.69 | 2612.67 | 6.53 |
| 10 | 87.21 | 1388.67 | 9.99 |
| 20 | 77.47 | 1233.33 | 18.42 |
| 30 | 78.94 | 1256.67 | 21.87 |
| 40 | 72.51 | 1153.67 | 29.05 |
| 50 | 76.03 | 1210.00 | 26.52 |
| 60 | 70.40 | 1119.67 | 32.79 |
| 70 | 70.70 | 1124.67 | 31.77 |
| 80 | 72.07 | 1146.67 | 33.52 |
| 90 | 77.93 | 1144.33 | 37.84 |
| 100 | 66.67 | 1060.33 | 30.66 |

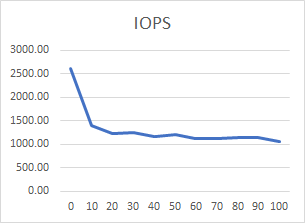
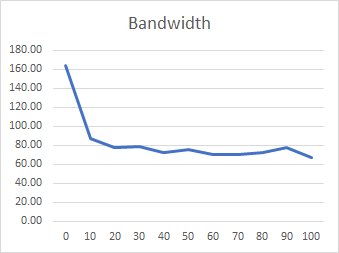


Figure 7. Bandwidth results for random writes from 0-100 Figure 8. IOPS results from random writes 0-100

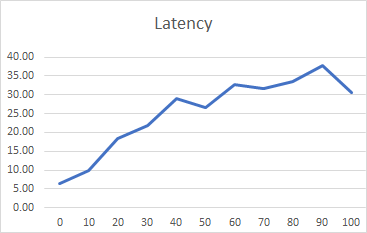


Figure 9. Latency results for random writes range 0-100

**4.4. Are there any interdependencies between read and write**

To test this out, we change the configurations for random/sequential read/write at 25%, 50% or 100%. The results shown in Table 4, and Figures 10-12, that at 50% read and write, there were very little difference between sequential and random read/write. Also, 75% read/25% write gave the best performance. This would suggest that there is an interdependency between read and write, but little difference on random or sequential read/writes.

The command ran with changes in parameters is:

*./fitness --file test --wr ? --rrnd ? --wrnd ? --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 --rd\_stride 4 --iter 3.*

***Table 4. Bandwidth, IOPS and Latency results of read/write interdependence***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| read:write % | --wr | --rrnd | --wrnd | Bandwidth | IOPS | Latency |
| 50:50 | 50 | 0 | 0 | 92.17 | 1468.33 | 43.90 |
| 50:50 | 50 | 0 | 100 | 85.42 | 1360.33 | 46.99 |
| 50:50 | 50 | 100 | 0 | 86.25 | 1373.33 | 46.75 |
| 50:50 | 50 | 100 | 100 | 89.95 | 1433.00 | 44.66 |
| 75:25 | 25 | 0 | 0 | 194.48 | 3105.33 | 20.62 |
| 75:25 | 25 | 100 | 100 | 177.37 | 2831.33 | 22.61 |
| 25:75 | 75 | 0 | 0 | 59.88 | 951.67 | 67.28 |
| 25:75 | 75 | 100 | 100 | 64.34 | 711.33 | 62.04 |

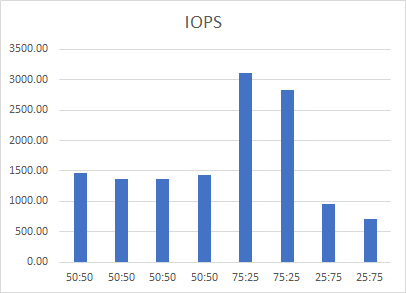
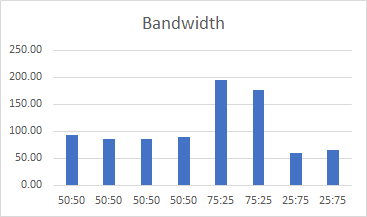


Figure 10. Bandwidth results testing independencies r/w Figure 11. IOPS results testing independencies r/w

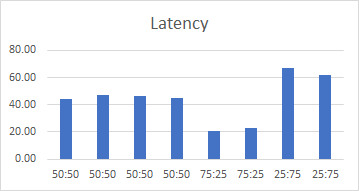


Figure 12. Latency results testing independencies r/w

**4.5. Does random reads have uniform latency**

To determine whether there is uniform latency in random reads, we changed the --rrnd parameter from 10-100 in multiples of 10. --wr was set to zero to ensure all the data was reads and no writes.

*./fitness --file test --wr 0 --rrnd ? --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 --rd\_stride 4 --iter 3.*

The results are shown in Table 5 and Figures 13-15.

***Table 5. Bandwidth, IOPS and Latency results with random reads***

|  |  |  |  |
| --- | --- | --- | --- |
| --rrnd | Bandwidth | IOPS | Latency |
| 10 | 39562.37 | 632993.67 | 0.02 |
| 20 | 38561.03 | 616972.00 | 0.02 |
| 30 | 38823.43 | 621170.67 | 0.02 |
| 40 | 38203.95 | 611258.67 | 0.02 |
| 50 | 37998.82 | 607976.67 | 0.02 |
| 60 | 38283.88 | 612538.00 | 0.02 |
| 70 | 38114.03 | 609820.00 | 0.02 |
| 80 | 38272.93 | 612363.00 | 0.02 |
| 90 | 37837.04 | 605388.33 | 0.02 |
| 100 | 36703.34 | 587249.67 | 0.01 |

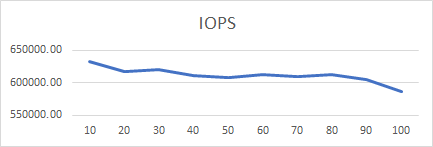
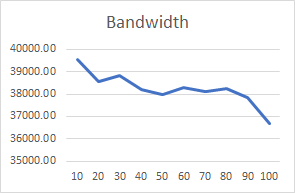


Figure 13. Bandwidth of random reads range 10-100 Figure 14. IOPS of random reads range 10-100

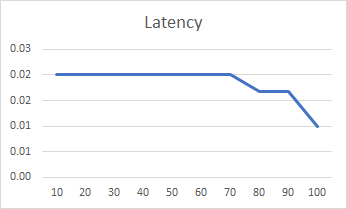


Figure 15. Latency of random reads range 10-100

The data showed that there were slight drops in bandwidth and IOPS as randomness increases. Latency was fairly constant until 80% randomness, then it dropped, which is fairly surprising since both IOPS and bandwidth also dropped. Based on the results, it could be assumed that for the most part, read randomness has no effect on latency.

**4.6. Effect of increasing the workload randomness on SSD**

The determine the effect of workload randomness, we increase the file size generated from the command below changing the --cap parameter. The file size ranges were 512mb, 1gb, 2gb, 4gb, 8gb and 16gb.

*./fitness --file test --wr 100 -wrnd 100 --cap ? --qdep 64 --wrsz 64 --rdsz 64 --wr\_stride 4 --rd\_stride 4 --iter 3.*

The results as seen in Table 6 and Figures 16-18, showed a steady growth in bandwidth and IOPS as the size created increases with a corresponding drop in latency. This would suggest that page size may have an effect on the issue of randomness and latency.

***Table 6. Bandwidth, IOPS and Latency results with different file sizes***

|  |  |  |  |
| --- | --- | --- | --- |
| File size | Bandwidth | IOPS | Latency |
| 512mb | 65.31 | 1038.67 | 38.82 |
| 1gb | 74.26 | 1182.00 | 48.68 |
| 2gb | 92.10 | 1467.00 | 44.00 |
| 4gb | 110.12 | 1755.67 | 32.16 |
| 8gb | 143.07 | 2282.67 | 30.71 |
| 16gb | 208.47 | 3329.00 | 18.57 |

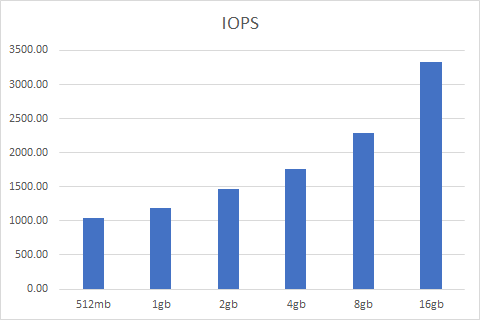
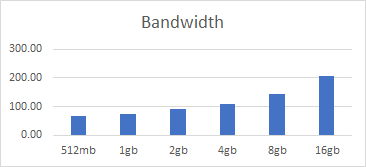


Figure 16. Bandwidth results in increasing workload Figure 17. IOPS results in increasing the workload

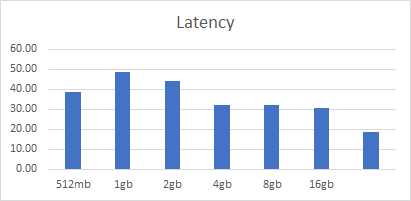


Figure 18. Latency results in increasing the workload

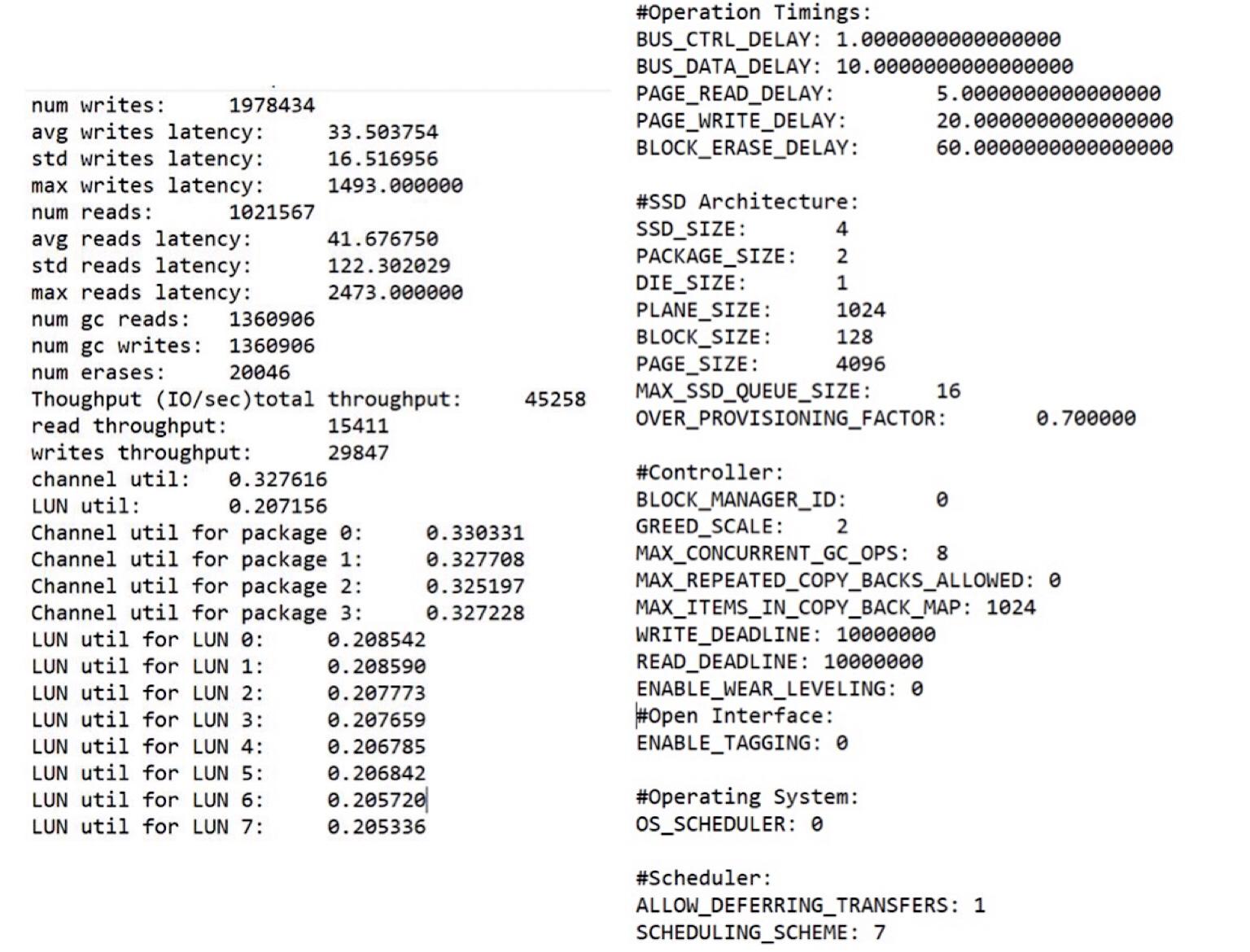
## 5.0 Discussion and implications

In previous hard drives architectures, sequential access has an edge over random access. We found that this is not the case in SSDs, as both sequential and random reads and writes follow the same trend. The performance of reads are by far superior to writes, which may be caused by the overheads associated with writes, writing the content into new pages, marking the old content invalid, erasing the old content, updating the logical table as to the new physical location of the data, to name a few. Although, most of these functions occur in the background they still provide overheads no associated with a read operation. The significantly improved performance with caching indicates that the onboard caching in SSDs has a big impact on performance. The lack of fluctuations in latencies with different ranges of randomness gives the impression that access in SSDs is not a bottleneck and with sufficient bandwidth, pages can be accessed relatively easily using the logical table.

We however, saw a few anomalies in the data which could not be very easily explained. While the architecture of the NAND flash cells with gates and channels is widely known, the working of the microcontroller is still a black box. Most of the functions of the controller is implemented in either firmware or hardware and these algorithms viewed as priopietory by the vendors, with no information on how they work or how they characteristics could be manipulated. Without a full knowledge of the workings of these functions especially the Flash Translation Layer which has such an important role in the workings of the SSD, full grasp of all the inner working will continue to elude us.

## 6.0 Challenges

* At the underlying phase of our execution, we were not ready to run the Intel open storage toolbox because of the absence of underlying support and documentation.
* We attempted to work with various test systems including VSSIM and Eagle tree[3]. Although Eagle tree was easy to install, VSSIM had OS reliance and was just working with Ubuntu 14.04.3. There is just a single activity (the installation process) which is utilized to produce the reads/writes for the disk. As the figure shows, we can only change the configuration of SSD based on the hardware level. Though it contains several algorithms like wear-leveling , garbage collection and so on, it is not able to call in the simulator running. The result which be showed in eagle tree, it’s not clear. First, the result which be showed in the result file doesn’t have any unit which makes really hard for us to do the further analysis when doing the different architecture comparison. Second, besides that, when we change a bigger operation timings in the configuration, the SSD will not do any read/write operation, the same things happened when we change a larger configuration in the SSD hardware architecture. Based on that, we decided not eagle tree for the further analysis. Though the eagle tree itself contains really great algorithms which helps us a lot in how the ssd operations, how the controller do the wear-leveling and scheduler, it’s really hard for us to analysis the performance of SSD by changing the configuration.



## 7.0 Conclusion

We have tried almost all of the components in charge of the execution of a solid-state drive. A portion of these components have a striking impact, others not really.

* Is random write the worst scenario?

As indicated by our tests on Heracles, we can securely say that truly, random write is the worst scenario. We have enough experimentation and chart results to demonstrate our announcement.

* Effects of increased read randomness on latency.

We can say that Read randomness does not influence the latency of an SSD. At the end of the day, the latency stays consistent on random reads.

* Read-write interdependence.

As indicated by our execution, we find that read-write interdependence impacts the execution of the SSD. This can be supported up by the charts and the graphs in that individual part.

## References

|  |  |
| --- | --- |
| [1] | "Intel OST," [Online]. Available: https://sourceforge.net/projects/intel-iscsi/. |
| [2] | OST, "--Help command," [Online]. |
| [3] | E. simulator. [Online]. Available: https://github.com/ClydeProjects/EagleTree. |
| [4] | D. A. K. a. X. Z. F. Chen, " Understanding intrinsic characteristics and system implications of flash memory based solid state drives SIGMETRICS Perform.," vol. 37, pp. 181-192, 2009. |