SSD Write Performance – IOPS Confusion Due to Poor Benchmarking Techniques

Dominique A. Heger
DHTechnologies (DHT)
dheger@dhtusa.com

Introduction

Over the last few years, the proliferation of solid state disk (SSD) based IO solutions triggered an entire benchmarking frenzy, comparing the throughput potential of SSD and hard disk drive (HDD) components, respectively. In most of these performance studies, little is known about the actual HW, OS, and SW setup, benchmark environment, benchmark tool, validation techniques used to asses the benchmark results, or even the actual workload distribution used to conduct the studies. Nevertheless, in most of these studies, compared to HDD drives, rather impressive IOPS SSD values are reported for read and write IO operations, respectively.

The argument made in this article is that most of these studies are highly flawed and skewed. Further, the statement made is that it is pretty useless to compare HDD and SSD solutions in the first place. While comparing different SSD offerings has a lot of merit, discussing the performance delta between an older (HDD) and a rather new technology (SSD) is similar to comparing a horse carriage to a modern car. In a nutshell, the IOPS ceiling of HDD's has not changed significantly since the appearance of the first 15K RPM (EMC) solution in 2002. Further, it is anybodies guess if there ever will be a (production) 20K RPM HDD solution available to the general public. Hence, while the (potential) performance capacity of the fastest SSD's increases almost on a monthly basis, the fastest HDD's remain exactly at the same performance level. Ergo, the HDD to SSD IOPS gap consistently widens over time.

This report addresses the performance behavior of random write IO operations on NAND flash-based SSD drives, elaborating on the importance of comparing SSD IOPS values over a subset, as well as over the entire available Logical Block Address (LBA) space, and discusses the reasons behind the (in this article) reported write IO performance degradation. Next to NAND flash based solutions, the SSD market also provides DRAM based SSD components. DRAM based SSD drives are in general more expensive per GB, but may actually be cheaper per IOPS. As NAND flash based SSD solutions represent the bulk of the installation base, this article only focuses on these SSD devices.

SSD Random IO Write Benchmark – LBA Performance Implications

For this article, a Linux 2.6.38, 4-way (Intel i5 M480 2.76GHz) system with 6GB of RAM and a 160GB Intel X18-M/X25-M SATA (NAND flash) MLC 1.8-Inch SSD drive was being used. The system was configured with the XFS filesystem, the noop Linux IO scheduler, a queue depth of 32, and a Linux (block) IO priority of 4. No additional tuning was performed to the Linux IO OS stack. Other SSD IO studies conducted by DHT disclosed that for most multi-threaded, random and sequential read/write workloads, noop outperformed the other 3 Linux IO schedulers (CFQ, deadline, anticipatory). As a workload generator, FFSB was being used. FFSB was configured with 4 IO threads, each conducting 4KB random IO write operations on files ranging from 64KB up to 128MB. In a first setup, the workload was spread over an 8GB LBA span. Hence, only a small subset of the SSD's LBA space was being used. In a second setup, the same benchmark scenario was executed over the entire LBA space (FFSB was used to age the file system). Each IO setup was benchmarked 10 times. In between the benchmark runs, the system was rebooted. Each benchmark was time-limited at 90 minutes runtime. The coefficient of variation (CV) for both setup runs was less than 4%, hence the

sample data was accepted as being reproducible.

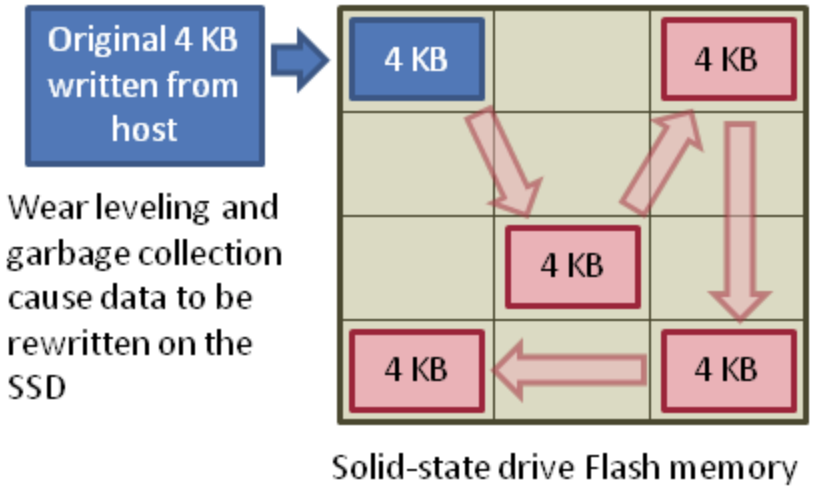
The average IOPS for the random write IO workload for the 8GB and the full LBA span was 3,182 and 342 IOPS, respectively. In other words, utilizing the entire LBA space for the random write IO operations resulted into lowering the IOPS value by approximately 89%! It has to be pointed out that the 342 IOPS (entire LBA space) still reflect a value that outpaces the IOPS potential of a 15K RPM drive. Nevertheless, this simple benchmark study outlines the importance of having a comprehensive benchmark strategy in place while conducting any performance study. Sizing a server system solely based on the IOPS numbers reported for the 8GB span may potentially have catastrophic performance implications while the IO subsystem ages. If an SLA calls for less than the 342 random write IOPS reported in this study, there would obviously be no actual concern. In other words, the 342 random IO write operations per second depict the lower bound on the capacity (throughput) behavior for this particular SSD (for the benchmarked 4KB random write workload scenarios).

SSD Technology

The next few paragraphs elaborate on how write operations are affected by today's state of the SSD NAND flash technology. In other words, the 2nd part of this article explains the rather vast IOPS delta observed via the 2 benchmark scenarios. To reiterate, while the actual workload distribution between the 2 benchmark setups remained the same, the spread or span on the SSD (either utilizing a small subset or the entire LBA) drove the IO performance discrepancy.

One of the 1st performance issues to discuss is labeled as SSD write amplification (WA). Write amplification is an undesirable phenomenon associated with SSD's and flash memory. Flash memory has to be erased prior to being re-written. The process to perform these operations results in moving (or re-writing) user data, as well as metadata more than once. This multiplying effect increases the number of write operations required over the life-span of an SSD, which in effect shortens the time an SSD can reliably operate. The increased number of write operations also consumes bandwidth to the flash memory, which has a detrimental impact on random write performance to the SSD. To reiterate, an SSD experiences write amplification as a result of garbage collection (GC) and wear leveling (see paragraph below), thereby increasing the number of write operations to the drive (see Figure 1).

Figure 1: SSD Write Amplification

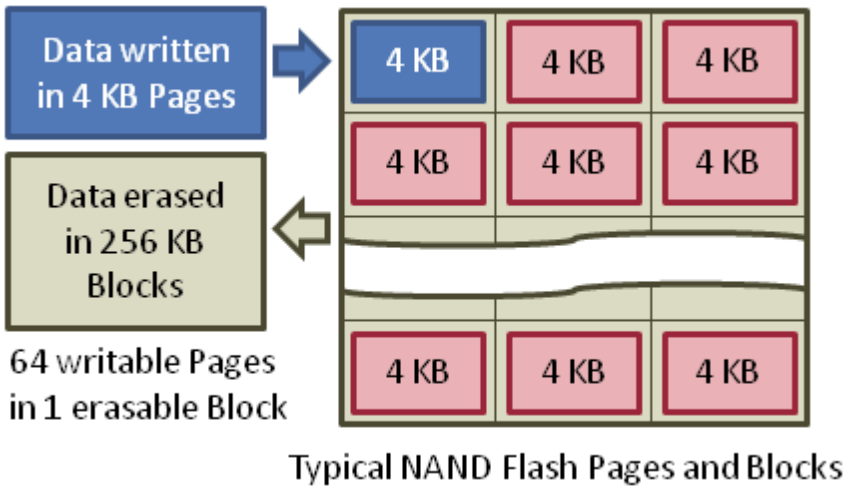


Note: Graph 1 courtesy of SSD Wikipedia

It has to be pointed out that every SSD includes a controller that incorporates the electronics that bridge the NAND memory components to the host computer. The controller is an embedded processor that executes firmware-level code. The controller is one of the most important SSD performance features. As soon as every block of an SSD has been written to, the SSD controller will have to return some of the blocks that no longer hold any current data (stale blocks). These blocks have to be erased so that new data can be written into them, a process also known as garbage collection (GC). All SSD's include some level of garbage collection. The big question though is when and how efficient the GC feature operates. With SSD's, data is written to the flash memory in units called pages (pages are made up of multiple cells). However, the memory can only be erased in larger units called blocks (blocks are made up of multiple pages – see Figure 2). If the data in some of the pages of the block are no longer needed, all of the other pages with good data in that block must be read and re-written into newly erased blocks. At that point, the extra pages not re-written are available for new data to be written into. This process of having to re-write data (that has already been sent from the host system sometime in the past) is a major part of the write amplification issue already discussed in this report.

Read requests do not require an erase of the flash memory and hence, are generally not associated with write amplification. The only exception here would be a *read disturb* operation. Each flash cell holds a certain voltage. When the controller requests the voltage from a cell, the act of making the request introduces the potential for altering the stored voltage. If a cell is read too many times, its stored value may change. Read disturb is the process under which stored data is slightly altered each time it is read, and read disturb management is a technique for monitoring and managing the number of times a cell has been read. Once a particular cell approaches the point where it has been read too many times, the controller moves the data in that cell to another location, resetting the cycle to start the count all over again.

Figure 2: SSD Write and Erase Cycle



Note: Graph 2 courtesy of SSD Wikipedia

Due to the nature of the flash memory's operation cycle, data can not be directly over-written (as it can be done with HDD's). As data is first written to a new SSD, all the cells start off in an erased state so that data can immediately be written (using one page at a time, most pages are either 4KB or 8KB). The SSD controller, which manages the flash memory and the interfaces to the host system,

uses a logical to physical mapping process that is known as logical block addressing (LBA). From a technical perspective, the LBA abstraction layer is part of the flash translation layer (FTL). In a scenario where new data has to replace older data, the SSD controller will write the new data in a new location and update the logical mapping to reference the new (physical) location. At this point, the old location no longer holds any valid data, but eventually the location has to be erased prior to being written to again. Further, flash memory can only be programmed and erased a limited number of times. Such a scenario is referred to as the maximum number of program/erase cycles (the P/E cycles) the device can sustain over the life of the flash memory. In a nutshell, a lower WA is more desirable to reduce the number of P/E cycles on the flash memory, and thereby increases the life expectation of the SSD.

From a capacity planning perspective, it is paramount to understand how SSD *over-provisioning* is implemented. Over-provisioning also has performance implications, albeit indirectly. Over-provisioning (OP) refers to the delta between (1) the physical capacity of the flash memory and (2) the logical SSD capacity as presented via the host operating system. This additional space (the OP part) is used by the SSD's for garbage collection, wear-leveling, or bad block mapping scenarios. The 1st level of OP is the result of the computation of the actual capacity, and the usage of GB units as the binary prefix. Both HDD and SDD vendors use the term GB to represent a *decimal* GB or 1,000,000,000 (10^9) bytes. Flash memory (as most other electronic storage) is assembled in multiples of 2, hence calculating the physical capacity of an SSD would be based on a value of 1,073,741,824 (2^30) per *binary* GB. The delta between these 2 values equals to 7.37%. Hence, a 128GB SSD with 0% OP would provide 128,000,000,000 bytes to the user community.

The 2nd level of OP actually comes from the manufacturers. This level of OP is typically 0%, 7%, or 28% based on the delta between the decimal GB of the physical capacity and the decimal GB of the available space to the user community. To illustrate, a manufacturer might publish a user-addressable capacity specification for an SSD as being 100GB or 120GB based on an actual SSD capacity of 128GB. The delta (in this example 28% or 7%, respectively) serves as the baseline for a manufacturer to state that a particular SSD has either 28% or 7% OP available. The 2nd level of OP does not count for the stipulated 7.37% of available capacity discussed in level 1. The 3d level of OP is basically attributed to the user community. Some SSD solutions allow the user to select additional OP space to improve the endurance and performance behavior of the drive at the expense of the total capacity potential. Some vendors incorporate that feature directly while others utilize OS partitions that are created smaller than the actual full user capacity of the SSD. To reiterate, while OP is not considered a direct performance factor, OP does reduce the actual usable user capacity. Therefore, OP allows for better managing write amplification, as well as increasing the endurance behavior of the SSD and hence, indirectly impacts the SSD's performance potential.

Synopsis of the SSD Write Performance Implications

To summarize, the discussed performance results for the benchmarked Intel SSD can be explained by how data is read/written to/from an SSD NAND flash drive. As discussed, in SSD terms, NAND flash memory cells are grouped on pages, and these pages are grouped into blocks. Generally speaking, 1 page is normally 4KB in size, whereas a block is 512KB in size. While it is possible to read or write flash NAND on a per page basis, only one block at a time can be deleted. Further, when a file is deleted (at the host level), the SSD is not actually aware of that operation, as the remove operation is typically only flagged at the systems (file) level. Therefore, the SSD is only able to compute the invalidity of a piece of data on a page when the page (actually the block) is over-written. The resulting phenomenon is pretty straight forward. A new 4KB file write operation onto a page that has already been used, results into the SSD to read the entire block, delete it, and then re-write all the necessary pages in that block. Ergo, to write 4KB may result into reading as much as 512KB, delete 512KB, and finally write 512KB. Such an IO scenario obviously has to have a detrimental impact on (especially small, random) IO performance.

Another SSD performance challenge is page addressing on flash memory, as the controller has to translate between the logical addresses utilized by the file system, and the physical addresses of the corresponding flash pages. With SSD's, there is no fixed mapping (as with HDD's), as the flash controller allocates pages in a way to balance performance (utilizing techniques such as write combining) and optimizing the lifespan of the cells (wear leveling). Write combining is based on a rather simple concept, but it adds overhead to the SSD controller. In write combining, multiple write requests are collected by the controller prior to being written to the drive. The goal is to bundle several small write requests into a single, larger write operation while hoping that the neighboring pages are likely be changed at the same time, and that these pages actually belong to the same file. This technique may significantly reduce the write amplification factor, which should improves write performance. Success (aka improved performance) though depends on how the data is actually sent to the drive, whether the data chunks are part of the same file, and the likelihood of the data being changed/erased at the same time. The SSD controller attempts to reorganize the allocation table as efficiently as possible to support write combining. But, as soon as all the pages have been used once (no free pages left), no actual performance gain is feasible anymore. That is the point where the SSD TRIM function (potentially) comes into play.

TRIM is a SATA command that enables the OS to notify an SSD which blocks of previously saved data are no longer needed (as a result of delete/remove or format requests). Hence, when an LBA is replaced by the OS, as with an overwrite of a file, the SSD knows that the original LBA can be marked as stale, and will not save these blocks during the garbage collection cycle. If a user deletes a file (not just removes part of a file), the file will typically be marked for deletion (the file descriptor is changed), but the actual contents on disk is not erased. Because of that, the SSD does not know that the LBA's that the file previously occupied can be erased, so the SSD will continue garbage collecting them. It has to be pointed out though that just because an SSD supports the TRIM command, that does not necessarily imply that the drive will perform at top-speed immediately after the TRIM operation. The space which is freed-up after the TRIM command was executed, may reflect random locations that are spread throughout the SSD. In reality, it normally takes a number of passes of writing data and garbage collecting before those spaces are consolidated, and the SSD drive shows improved performance. Further, not all systems support TRIM. And even after the OS and the SSD components are configured to support the TRIM command, other conditions may prevent TRIM to have an impact. To illustrate, some database systems and RAID solutions are not yet TRIM-aware, and subsequently do not know how to pass that information onto the SSD. In environments like that, the SSD will continue saving and garbage collecting those blocks until the OS utilizes the LBA's for new write requests.

Summary

Comparing SSD and HDD performance is basically useless. Traditional HDD benchmarks are focused on quantifying IO performance revolving around internal transfer speed, rotational latency, or disk seek time behavior. As SSD solutions do not spin or seek, the performance comparison becomes a moot point. However, SSD solutions have their own performance challenges, and comparing the performance behavior of competing SSD drives has a lot of merit (the same holds true for HDD's). The argument made is that benchmarking SSD solutions has to be conducted in several stages, by basically aging the IO environment. In other words, the same workload behavior has to be executes across different LBA sizes. It is suggested that the last set of benchmarks to execute has to be the one where the entire SSD LBA size is utilized.

While beyond the scope of this article, it has to be pointed out that other IO benchmarks conducted by DHT utilizing a variety of SSD solutions showed a (measurable) performance degradation for random write requests when larger LBA chunks were used. It also has to be pointed out that IO performance on an empty (newly created) HDD file system is normally superior to the

performance behavior of an aged HDD based file system (workload dependent of course). To benchmark any systems component is a daunting task. A sound benchmark process has to be in place, discussing all the goals and objectives. Unfortunately, most benchmarks are executed in an environment lacking documentation, proper OS and HW setup and tuning, sound configuration of the workload generators, as well as reporting on the statistical analysis of the benchmark runs. This holds true even for some of the major SSD manufacturers, who do not disclose SSD IO performance numbers over varying LBA spreads. In most cases, only one set of performance numbers per SSD is published.

References

- Hu, (2009). "Write Amplification Analysis in Flash-Based Solid State Drives", IBM
- Smith, (2009). "SSDs: The Devil is in the Preconditioning Details", Flash Memory
- Lucchesi (2009). "SSD Flash drives enter the enterprise", Silverton Consulting
- Layton, (2009). "Anatomy of SSDs", Linux Magazine
- Bagley, (2009). "Over-provisioning: a winning strategy or a retreat?", StorageStrategies
- Heger, Rao, “Flexible File System Benchmark”, Sourceforge
- Johnson, (2005) “Performance Tuning for Linux Servers”, IBM Press
- Jeremic, (2009), “The Pitfalls of Deploying SSD RAIDs”, University of Rostock
- SSD & HDD Wikipedia