= Write amplification =

Write amplification (WA) is an undesirable phenomenon associated with flash memory and solid @-@ state drives (SSDs) where the actual amount of physical information written is a multiple of the logical amount intended to be written .

Because flash memory must be erased before it can be rewritten , with much coarser granularity of the erase operation when compared to the write operation , the process to perform these operations results in moving (or rewriting) user data and metadata more than once . Thus , rewriting some data requires an already used portion of flash to be read , updated and written to a new location , together with initially erasing the new location if it was previously used at some point in time ; due to the way flash works , much larger portions of flash must be erased and rewritten than actually required by the amount of new data . This multiplying effect increases the number of writes required over the life of the SSD which shortens the time it can reliably operate . The increased writes also consume bandwidth to the flash memory which mainly reduces random write performance to the SSD . Many factors will affect the write amplification of an SSD ; some can be controlled by the user and some are a direct result of the data written to and usage of the SSD .

Intel and SiliconSystems (acquired by Western Digital in 2009) used the term write amplification in their papers and publications as early as 2008 . Write amplification is typically measured by the ratio of writes committed to the flash memory to the writes coming from the host system . Without compression , write amplification cannot drop below one . Using compression , SandForce has claimed to achieve a typical write amplification of 0 @.@ 5 , with best @-@ case values as low as 0 @.@ 14 in the SF @-@ 2281 controller .

= = Basic SSD operation = =

Due to the nature of flash memory 's operation , data cannot be directly overwritten as it can in a hard disk drive . When data is first written to an SSD , the cells all start in an erased state so data can be written directly using pages at a time (often 4 ? 8 kilobytes (KB) in size) . The SSD controller on the SSD , which manages the flash memory and interfaces with the host system , uses a logical @-@ to @-@ physical mapping system known as logical block addressing (LBA) and that is part of the flash translation layer (FTL) . When new data comes in replacing older data already written , the SSD controller will write the new data in a new location and update the logical mapping to point to the new physical location . The data in the old location is no longer valid , and will need to be erased before the location can be written again .

Flash memory can only be programmed and erased a limited number of times . This is often referred to as the maximum number of program / erase cycles (P / E cycles) it can sustain over the life of the flash memory . Single @-@ level cell (SLC) flash , designed for higher performance and longer endurance , can typically operate between 50 @,@ 000 and 100 @,@ 000 cycles . As of 2011 , multi @-@ level cell (MLC) flash is designed for lower cost applications and has a greatly reduced cycle count of typically between 3 @,@ 000 and 5 @,@ 000 . Since 2013 , triple @-@ level cell (TLC) flash has been available , with cycle counts dropping to 1 @,@ 000 program @-@ erase (P / E) cycles . A lower write amplification is more desirable , as it corresponds to a reduced number of P / E cycles on the flash memory and thereby to an increased SSD life .

= = Calculating the value = =

Write amplification was always present in SSDs before the term was defined , but it was in 2008 that both Intel and SiliconSystems started using the term in their papers and publications . All SSDs have a write amplification value and it is based on both what is currently being written and what was previously written to the SSD . In order to accurately measure the value for a specific SSD , the selected test should be run for enough time to ensure the drive has reached a steady state condition .

A simple formula to calculate the write amplification of an SSD is :

<formula>

= = Factors affecting the value = =

Many factors affect the write amplification of an SSD . The table below lists the primary factors and how they affect the write amplification . For factors that are variable , the table notes if it has a direct relationship or an inverse relationship . For example , as the amount of over @-@ provisioning increases , the write amplification decreases (inverse relationship) . If the factor is a toggle (enabled or disabled) function then it has either a positive or negative relationship .

= = Garbage collection = =

Data is written to the flash memory in units called pages (made up of multiple cells) . However , the memory can only be erased in larger units called blocks (made up of multiple pages) . If the data in some of the pages of the block are no longer needed (also called stale pages) , only the pages with good data in that block are read and rewritten into another previously erased empty block . Then the free pages left by not moving the stale data are available for new data . This is a process called garbage collection (GC) . All SSDs include some level of garbage collection , but they may differ in when and how fast they perform the process . Garbage collection is a big part of write amplification on the SSD .

Reads do not require an erase of the flash memory , so they are not generally associated with write amplification . In the limited chance of a read disturb error , the data in that block is read and rewritten , but this would not have any material impact on the write amplification of the drive .

= = = Background garbage collection = = =

The process of garbage collection involves reading and rewriting data to the flash memory . This means that a new write from the host will first require a read of the whole block , a write of the parts of the block which still include valid data , and then a write of the new data . This can significantly reduce the performance of the system . Some SSD controllers implement background garbage collection (BGC) , sometimes called idle garbage collection or idle @-@ time garbage collection (ITGC) , where the controller uses idle time to consolidate blocks of flash memory before the host needs to write new data . This enables the performance of the device to remain high .

If the controller were to background garbage collect all of the spare blocks before it was absolutely necessary , new data written from the host could be written without having to move any data in advance , letting the performance operate at its peak speed . The trade @-@ off is that some of those blocks of data are actually not needed by the host and will eventually be deleted , but the OS did not tell the controller this information . The result is that the soon @-@ to @-@ be @-@ deleted data is rewritten to another location in the flash memory , increasing the write amplification . In some of the SSDs from OCZ the background garbage collection only clears up a small number of blocks then stops , thereby limiting the amount of excessive writes . Another solution is to have an efficient garbage collection system which can perform the necessary moves in parallel with the host writes . This solution is more effective in high write environments where the SSD is rarely idle . The SandForce SSD controllers and the systems from Violin Memory have this capability .

= = = Filesystem @-@ aware garbage collection = = =

In 2010 , some manufacturers (notably Samsung) introduced SSD controllers that extended the concept of BGC to analyze the file system used on the SSD , to identify recently deleted files and unpartitioned space . The manufacturer claimed that this would ensure that even systems (operating systems and SATA controller hardware) which do not support TRIM could achieve similar performance . The operation of the Samsung implementation appeared to assume and require an NTFS file system . It is not clear if this feature is still available in currently shipping SSDs from these

manufacturers . Systematic data corruption has been reported on these drives if they are not formatted properly using MBR and NTFS .

= = Over @-@ provisioning = =

Over @-@ provisioning (sometimes spelled as OP , over provisioning , or overprovisioning) is the difference between the physical capacity of the flash memory and the logical capacity presented through the operating system (OS) as available for the user . During the garbage collection , wear @-@ leveling , and bad block mapping operations on the SSD , the additional space from over @-@ provisioning helps lower the write amplification when the controller writes to the flash memory .

The first source of over @-@ provisioning comes from the computation of the capacity and use of gigabyte (GB) as the unit instead of gibibyte (GiB) . Both HDD and SSD vendors use the term GB to represent a decimal GB or 1 @,@ 000 @,@ 000 @,@ 000 (= 109) bytes . Like most other electronic storage , flash memory is assembled in powers of two , so calculating the physical capacity of an SSD would be based on 1 @,@ 073 @,@ 741 @,@ 824 (= 230) per binary GB or GiB . The difference between these two values is 7 @.@ 37 % (= (230 ? 109) / 109 × 100 %) . Therefore , a 128 GB SSD with 0 % additional over @-@ provisioning would provide 128 @,@ 000 @,@ 000 @,@ 000 bytes to the user (out of 137 @,@ 438 @,@ 953 @,@ 472 total) . This initial 7 @.@ 37 % is typically not counted in the total over @-@ provisioning number , and the true amount available is usually less as some storage space is needed for the controller to keep track of non @-@ operating system data such as block status flags . The 7 @.@ 37 % figure may extend to 9 @.@ 95 % in the terabyte range , as manufacturers take advantage of a further grade of binary / decimal unit divergence to offer 1 or 2 TB drives of 1000 and 2000 GB capacity (931 and 1862 GiB) , respectively , instead of 1024 and 2048 GB (as 1 TB = 1 @,@ 000 @,@ 000 @,@ 000 @,@ 000 bytes in decimal terms , but 1 @,@ 099 @,@ 511 @,@ 627 @,@ 776 in binary) .

The second source of over @-@ provisioning comes from the manufacturer , typically at 0 % , 7 % or 28 % , based on the difference between the decimal gigabyte of the physical capacity and the decimal gigabyte of the available space to the user . As an example , a manufacturer might publish a specification for their SSD at 100 , 120 or 128 GB based on 128 GB of possible capacity . This difference is 28 % , 7 % and 0 % respectively and is the basis for the manufacturer claiming they have 28 % of over @-@ provisioning on their drive . This does not count the additional 7 @.@ 37 % of capacity available from the difference between the decimal and binary gigabyte .

The third source of over @-@ provisioning comes from known free space on the drive , gaining endurance and performance at the expense of reporting unused portions , and / or at the expense of current or future capacity . This free space can be identified by the operating system using the TRIM command . Alternately , some SSDs provide a utility that permit the end user to select additional over @-@ provisioning . Furthermore , if any SSD is set up with an overall partitioning layout smaller than 100 % of the available space , that unpartitioned space will be automatically used by the SSD as over @-@ provisioning as well . Yet another source of over @-@ provisioning is operating system minimum free space limits ; some operating systems maintain a certain minimum free space per drive , particularly on the boot or main drive . If this additional space can be identified by the SSD , perhaps through continuous usage of the TRIM command , then this acts as semi @-@ permanent over @-@ provisioning . Over @-@ provisioning often takes away from user capacity , either temporarily or permanently , but it gives back reduced write amplification , increased endurance , and increased performance .

A simple formula to calculate the over @-@ provision of an SSD is :

<formula>

= = TRIM = =

TRIM is a SATA command that enables the operating system to tell an SSD which blocks of

previously saved data are no longer needed as a result of file deletions or volume formatting . 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 or invalid and it will not save those blocks during garbage collection . If the user or operating system erases a file (not just remove parts of it) , the file will typically be marked for deletion , but the actual contents on the disk are never actually erased . Because of this , the SSD does not know that it can erase the LBAs previously occupied by the file , so the SSD will keep including such LBAs in the garbage collection .

The introduction of the TRIM command resolves this problem for operating systems that support it like Windows 7 , Mac OS (latest releases of Snow Leopard , Lion , and Mountain Lion , patched in some cases) , FreeBSD since version 8 @.@ 1 , and Linux since version 2 @.@ 6 @.@ 33 of the Linux kernel mainline . When a file is permanently deleted or the drive is formatted , the OS sends the TRIM command along with the LBAs that no longer contain valid data . This informs the SSD that the LBAs in use can be erased and reused . This reduces the LBAs needing to be moved during garbage collection . The result is the SSD will have more free space enabling lower write amplification and higher performance .

= = = Limitations and dependencies = = =

The TRIM command also needs the support of the SSD . If the firmware in the SSD does not have support for the TRIM command , the LBAs received with the TRIM command will not be marked as invalid and the drive will continue to garbage collect the data assuming it is still valid . Only when the OS saves new data into those LBAs will the SSD know to mark the original LBA as invalid . SSD Manufacturers that did not originally build TRIM support into their drives can either offer a firmware upgrade to the user , or provide a separate utility that extracts the information on the invalid data from the OS and separately TRIMs the SSD . The benefit would only be realized after each run of that utility by the user . The user could set up that utility to run periodically in the background as an automatically scheduled task .

Just because an SSD supports the TRIM command does not necessarily mean it will be able to perform at top speed immediately after a TRIM command . The space which is freed up after the TRIM command may be at random locations spread throughout the SSD . It will take a number of passes of writing data and garbage collecting before those spaces are consolidated to show improved performance .

Even after the OS and SSD are configured to support the TRIM command , other conditions might prevent any benefit from TRIM . As of early 2010 , databases and RAID systems are not yet TRIM @-@ aware and consequently will not know how to pass that information on to the SSD . In those cases the SSD will continue to save and garbage collect those blocks until the OS uses those LBAs for new writes .

The actual benefit of the TRIM command depends upon the free user space on the SSD . If the user capacity on the SSD was 100 GB and the user actually saved 95 GB of data to the drive , any TRIM operation would not add more than 5 GB of free space for garbage collection and wear leveling . In those situations , increasing the amount of over @-@ provisioning by 5 GB would allow the SSD to have more consistent performance because it would always have the additional 5 GB of additional free space without having to wait for the TRIM command to come from the OS .

= = Free user space = =

The SSD controller will use any free blocks on the SSD for garbage collection and wear leveling . The portion of the user capacity which is free from user data (either already TRIMed or never written in the first place) will look the same as over @-@ provisioning space (until the user saves new data to the SSD) . If the user only saves data consuming 1 / 2 of the total user capacity of the drive , the other half of the user capacity will look like additional over @-@ provisioning (as long as the TRIM command is supported in the system) .

= = Secure erase = =

The ATA Secure Erase command is designed to remove all user data from a drive . With an SSD without integrated encryption , this command will put the drive back to its original out @-@ of @-@ box state . This will initially restore its performance to the highest possible level and the best (lowest number) possible write amplification , but as soon as the drive starts garbage collecting again the performance and write amplification will start returning to the former levels . Many tools use the ATA Secure Erase command to reset the drive and provide a user interface as well . One free tool that is commonly referenced in the industry is called HDDErase . Gparted and Ubuntu live CDs provide a bootable Linux system of disk utilities including secure erase .

Drives which encrypt all writes on the fly can implement ATA Secure Erase in another way . They simply zeroize and generate a new random encryption key each time a secure erase is done . In this way the old data cannot be read anymore , as it cannot be decrypted . Some drives with an integrated encryption may require a TRIM command be sent to the drive to put the drive back to its original out @-@ of @-@ box state .

= = Wear leveling = =

If a particular block was programmed and erased repeatedly without writing to any other blocks , that block would wear out before all the other blocks ? thereby prematurely ending the life of the SSD . For this reason , SSD controllers use a technique called wear leveling to distribute writes as evenly as possible across all the flash blocks in the SSD .

In a perfect scenario , this would enable every block to be written to its maximum life so they all fail at the same time . Unfortunately , the process to evenly distribute writes requires data previously written and not changing (cold data) to be moved , so that data which are changing more frequently (hot data) can be written into those blocks . Each time data are relocated without being changed by the host system , this increases the write amplification and thus reduces the life of the flash memory . The key is to find an optimum algorithm which maximizes them both .

= = Separating static and dynamic data = =

The separation of static and dynamic data to reduce write amplification is not a simple process for the SSD controller . The process requires the SSD controller to separate the LBAs with data which is constantly changing and requiring rewriting (dynamic data) from the LBAs with data which rarely changes and does not require any rewrites (static data) . If the data is mixed in the same blocks , as with almost all systems today , any rewrites will require the SSD controller to garbage collect both the dynamic data (which caused the rewrite initially) and static data (which did not require any rewrite) . Any garbage collection of data that would not have otherwise required moving will increase write amplification . Therefore , separating the data will enable static data to stay at rest and if it never gets rewritten it will have the lowest possible write amplification for that data . The drawback to this process is that somehow the SSD controller must still find a way to wear level the static data because those blocks that never change will not get a chance to be written to their maximum P / E cycles .

= = Sequential writes = =

When an SSD is writing large amounts of data sequentially , the write amplification is equal to one meaning there is no write amplification . The reason is as the data is written , the entire block is filled sequentially with data related to the same file . If the OS determines that file is to be replaced or deleted , the entire block can be marked as invalid , and there is no need to read parts of it to garbage collect and rewrite into another block . It will only need to be erased , which is much easier and faster than the read @-@ erase @-@ modify @-@ write process needed for randomly written data going through garbage collection .

= = Random writes = =

The peak random write performance on an SSD is driven by plenty of free blocks after the SSD is completely garbage collected , secure erased , 100 % TRIMed , or newly installed . The maximum speed will depend upon the number of parallel flash channels connected to the SSD controller , the efficiency of the firmware , and the speed of the flash memory in writing to a page . During this phase the write amplification will be the best it can ever be for random writes and will be approaching one . Once the blocks are all written once , garbage collection will begin and the performance will be gated by the speed and efficiency of that process . Write amplification in this phase will increase to the highest levels the drive will experience .

= = Impact on performance = =

The overall performance of an SSD is dependent upon a number of factors , including write amplification . Writing to a flash memory device takes longer than reading from it . An SSD generally uses multiple flash memory components connected in parallel to increase performance . If the SSD has a high write amplification , the controller will be required to write that many more times to the flash memory . This requires even more time to write the data from the host . An SSD with a low write amplification will not need to write as much data and can therefore be finished writing sooner than a drive with a high write amplification .

= = Product statements = =

In September 2008 , Intel announced the X25 @-@ M SATA SSD with a reported WA as low as 1 @.@ 1 . In April 2009 , SandForce announced the SF @-@ 1000 SSD Processor family with a reported WA of 0 @.@ 5 which appears to come from some form of data compression . Before this announcement , a write amplification of 1 @.@ 0 was considered the lowest that could be attained with an SSD . Currently , only SandForce employs compression in its SSD controller .