Violin MEMORY

Best Practices

Best Practices Guide for Using the DiskSpd Performance Test Tool

Document Number: 535-0264-00 Rev 01
May 2016

LEGAL NOTICE

Violin Memory, Inc.
4555 Great America Parkway
Santa Clara, CA 95054
USA

Table of Contents

1. Challenges Using SQLIO with Modern SANs

SQLIO has long been the standard SAN benchmarking tool in the Microsoft world. But, as times have changed, so do the technologies we use. In the case of benchmarking tools, Microsoft decided to release a whole new tool, DiskSpd, over upgrading their existing one, SQLIO. Given the wide array of challenges it makes sense that they decided to start from scratch. The following is a list of the more prominent issues with using SQLIO as a SAN benchmarking tool on modern SANs.

1.1. Workload Mixes

SQLIO is limited to running only 100% read or 100% write workloads. Almost no real-world workload runs with these ratios so the results are mostly inapplicable. DiskSpd allows for an integer based percentage to be specified for the write ratio. So, a –w30 would result in a 70% read / 30% write workload. This is one of the more common workloads tested for anything from applications to databases.

1.2. Affinity / NUMA

SQLIO was not programmed to be NUMA aware and thus can run into inflated IO latencies due to improper thread allocation and inefficient memory access. Modern programs like SQL Server are NUMA aware, follow different memory access patterns and thus a SQLIO test would not adequately reflect real world application performance. Modern SANs and applications can deliver much higher performance factors with architectural awareness that can be obscured by older benchmarking tools like SQLIO.

1.3. Detailed Reporting

SQLIO has a basic set of reporting output that tracks only the overall throughput, IOs per second and the latency of the workload. As the workloads are restricted to all-reads or all-writes it is not possible to see the SAN's performance during real world workloads. DiskSpd reports separately on the read and write statistics as well as includes the CPU utilization and statistics per thread and per file. DiskSpd also improves reporting on the IO latency by adding a standard deviation to help administrators better understand the distribution of latencies. Not all IOs return at the speed of the average or the maximum so it is very helpful to know how close to the average that most IOs return. The tighter the standard deviation the more consistent the SAN and the more consistent the workload's performance will be in production.

1.4. Unique Data / Deduplication Testing

Flash storage testing commonly includes testing with data deduplication as well. Deduplication technology has been around a while but it randomizes the data physically stored. Random IO can significantly degrade hard disk drive performance but does not affect flash storage technology. This allows for significant cost reductions while not enduring the significant performance degradation that it can cause hard disk drives.

To test data deduplication requires a benchmarking tool that can generate files with a unique data pattern. SQLIO produces test files of all-spaces. These files tend to get deduplicated down to a single storage block and this can have varying effects on the SANs being tested. Some can report artificially higher performance as they are never actually doing the IO or can report artificially lower performance as they are bottlenecked around the reading and writing of one block and not accessing the rest of the SAN.

DiskSpd was developed to allow for files to be created with an all-unique file pattern. This creates a real-world layout on storage and delivers realistic benchmarking results.

2. Using DiskSpd

While the tool is new, it is not that unlike SQLIO. They are both command line tools that have a series of input parameters. They both require source files to be generated before any benchmarking runs and can be scripted to output their results to files.

2.1. Downloading the Tool

The following link is to the download page hosted by Microsoft. There is a download button for the zip file which shows the current version. As of the writing of this paper the version is 2.0.15 meaning the download file is named "Diskspd-v2.0.15.zip".

https://gallery.technet.microsoft.com/DiskSpd-a-robust-storage-6cd2f223

2.2. Unpacking

Once the zip file has been downloaded and moved to the benchmarking test host, unzip the file into a directory that is easy to get back to. Inside the newly unpacked structure are three directories corresponding to different processor architectures. Most Windows hosts are 64 bit these days so copy the diskspd.exe file from the "amd64fre" directory and place it into the root folder or whatever folder is planned to be used for storing the testing tools and results.

2.3. Create Test Files

With SQLIO, it was required to use the param.txt file in order to create the source data files. This is no longer the case. With DiskSpd, the source files will be created prior to executing the workload and multiple files can be referenced from the command line.

In order to create a source file, include the –c input parameter along with a size. It is recommended to use multiple files (at least 2) over multiple LUNs as this properly simulates production workloads. If a file has already been created then the –c parameter is not required. It is recommended to keep the –c parameter in the command line all throughout testing so that later review of the testing can be better understood.

```
C:\DiskSpd\diskspd.exe -c1024G -h -L –d5 -r -w0 –t16 –o16 –b4K J:\testfile.dat K:\testfile.dat L:\testfile.dat
M:\testfile.dat
```

The above command will create a total of 4 files on the J, K, L and M volumes of 1024GB each. Most of the input parameters are not needed for the file building step as they have defaults and the ending run will not have its data evaluated. But, the above command will get you used to the normal set of parameters to use. The number of threads, batch size and number of outstanding IOs are not relevant as this is just the file creation step.

When done with the file creation it will execute a test for only 5 seconds before finishing (-d5). When the short test is done then the real testing can begin.

2.4. Common Parameters

While there are many advanced features and settings that can be used for niche testing, there are a standard set of parameters that encompass the vast majority of SAN benchmark testing.

2.4.1. Block Size (-b)

This is the IO block size for reads and writes. With DiskSpd the input format now includes a magnitude qualifier. Batches can be set in kilobytes (-b4k, -b8k, etc.), megabytes (-b1M, etc.) or Gigabytes (-2G). Please refer to the Best Practices section to learn more about what to choose for this parameter.

2.4.2. Read / Write Ratio (-w)

The –w parameter allows for the specification of the write percentage. So, -w0 would be zero writes so therefore 100% reads. A –w30 would be 30% writes and 70% reads.

2.4.3. Threads (-t)

The number of threads to spawn for IO processing (-t16). Each file will get this number of threads so plan accordingly. See the Best Practices section for more learning as to how to use this parameter effectively.

2.4.4. IOs Outstanding (-o)

Each thread launched will execute this number of IO requests (-o16). It will maintain this number of in-flight IOs. See the Best Practices section for better understanding on how to use parameter along with the threads and batch size parameters to drive complete workloads.

2.4.5. Random or Sequential (-r for random, empty for sequential)

This is a switch parameter that creates random file access when the (-r) parameter is included and sequential file access when it is not included.

2.4.6. Duration (-d)

This is the number of seconds to execute the test (-d60). This is similar to SQLIO's –s parameter.

2.4.7. Latency (-L)

This switch parameter allows for advanced latency reporting including per thread, per file and standard deviation based details.

2.4.8. Caching (-h)

This parameter turns off caching that may interfere with the SANs benchmarking performance.

2.4.9. Source File Size (-c)

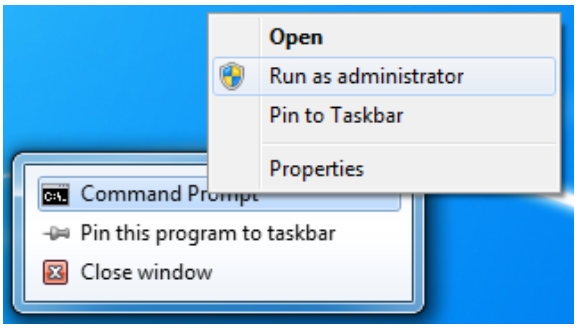
Use the –c parameter to specify the source file size. –c100G would create a 100 gigabyte source file. See the Best Practices to better understand the number of files and file sizes to use.

3. Best Practices

Performance benchmarking is a journey. Each system will have its own optimal parameters and niche performance outcomes. It is optimal to not use a basic grid script combining many different combinations of threads, outstanding IO, batch sizes and read/write ratios as this will take a considerable time to complete all the different combinations and not allow for the following of where the data takes you. Also, many parameters should be static based upon workload profiles or system architecture so running numbers of threads, for example, that do not correspond to the number of cores on a single CPU or in the full system will not provide real world results. Thus, it is best to start simple and try different combinations to see where the data takes you. Below are some general and DiskSpd specific benchmarking best practices.

3.1. Run From an Elevated Command Prompt

Especially for the file creation step, this can lead to increased performance. To elevate, launch the PowerShell or command windows with the "Run as Administrator" option.



3.2. Configure Enough CPU Resources

With high end performance testing it can take significant CPU resources in order to process hundreds of thousands of IO packets per second. This may require more resources than you are used to. Be sure to monitor the overall CPU consumption, the consumption per core and the consumption per thread in the DiskSpd output.

For virtual machine based testing this can be a more significant concern as it is likely that the virtual machine used for testing is not configured to utilize the entire CPU resources of the host. In this case, it is recommended to allocate at least one full CPU. If less resources are going to be allocated to the VM during production then a lower benchmarking peak output should be understood.

One benchmarking theory is to peak out the host resources. This is normally done to test the SAN's peak performance instead of a testing a specific workload. It is common for flash arrays to have a higher performance than one host can drive so if full-SAN testing is the point of the test, allocate several hosts to make sure the bottleneck is the SAN and not the host. Host servers also have many bottlenecking points (CPU, IOH, HBA ports, PCIe bus, QPI links, etc.) that are likely to be well below what a flash SAN handle so be sure to monitor all aspects of the test and make sure the appropriate bottleneck is found for each test.

Another benchmarking theory is to test a specific workload. In this process the workload is profiled and duplicated. For this type of testing it is recommended to not drive the benchmarking tool up to the maximum of the CPU as this would not allow for any remaining CPU resources for the application to do work with the data. Each application is different but driving the total CPU consumption, for just IO, past 25% of the total CPU resources is not leaving a realistic amount of resources left for application

processing.

3.3. Align Threads to CPU Cores

At the lower latencies that are achievable by flash storage, tuning the number of threads to align with the number of CPU cores in use can lead to significant gains. For example, if there are 4 CPU cores and 5 threads configured in a test then as the operating system seeks to even out the per-core usage it will move the threads around between the cores and force the threads to be hoping through L3 cache. This time delay can be in the milliseconds, well above the microsecond response times by modern flash SANs. So, it is recommended to aim for "sticky threads" in which each thread lives on a core and does not have to hop between them. This can be done by allocating one thread per CPU core in use. In general it is recommended to use 1 CPU for testing and use the same number of threads as there are logical cores in that 1 CPU. It is also recommended to use the CPU that is closest to the IOH (IO Hub, or PCIe controller). Check with your host documentation to see which CPU that is. Commonly it is the 1st CPU or CPU 0.

3.4. Use Multiple LUNs by Using Multiple files

As SANs get faster it is common to start running into bottlenecks that were commonly not seen before. It is also a best practice to aim for parallelization through all layers. As such, it is a best practice to utilize multiple LUNs when doing high performance flash storage benchmarking. As a rule of thumb, 4 to 8 LUNs should make sure that anything single threaded per LUN or file will be worked around via the use of multiple LUNs and files (one file per LUN should be sufficient).

For each file used in a test DiskSpd will launch the full number of threads requested as discussed later in the Threads and Outstanding I/Os section. The number of files should be coordinated with the number of threads and outstanding I/Os for optimal results.

It is generally not required to have more than one file per LUN. But, it is highly suggested that the total file space used across all of the testing files is many hundreds of gigabytes to make sure that all I/Os are actually processed down to the storage media and that any deduplication or caching mechanisms do not produce misleading statistics. Also, if a file is too small then the random block ID generator could run into conflicts on writes and thus stunt a performance test. 40GBs should be the smallest test footprint with 100GB being the recommended minimum. Unless otherwise obstructed, aim for a footprint in the multiple hundred GB range for peak performance benchmarks and multiple TB range for sustained benchmark testing.

3.5. Deduplication Testing

Data reduction has been a common technology to test as flash storage has come into the main stream storage market. Deduplication is the most common, and most impactful, of the data reduction technologies (compression being the other). When testing data deduplication it is required that the source file be at least partially unique data. Older tools, like SQLIO, only create files of repeating values that deduplicate down to just one block. This can dramatically alter the perceived performance of SANs as they either do not do the actual read or write operation or they operate on only just one block over and over. It is best practice to both create files with unique data and use a large file space (few large files or many medium files). Four files of 500GBs each should be a sufficient target.

3.6. Run Durations

For peak performance benchmark tests, the test run's resulting average and standard deviation results need time to aggregate over many samples and reduce the impacts of the startup and cool down periods. It is recommended that peak performance tests run at least 5 minutes to make sure that sufficient samples are gathered. For sustained testing, see the next section.

3.7. Sustained Versus Peak

Every flash storage product has garbage collection, wear leveling and error correction processes that can alter performance and will most likely perform differently in the beginning moments of a test versus later. Make sure to run long-duration tests to make sure that true production performance has been determined.

For sustaining performance benchmark tests, the tests will need to be run until the array settles into its sustained performance mark. This can vary depending on the workload and the array configuration. It is recommended to run mixed workloads and heavy-write workloads for many hours, even days, to find the true sustaining performance mark.

Some vendors have the ability to alter the configurations of the arrays to better suit certain workloads, such as heavy bursts of writes. Make sure that the vendor has been properly informed of the production workloads to ensure that the appropriate array configurations have been deployed.

3.8. Threads and Outstanding IOs

This section deals with how to tune the benchmark configurations to determine the performance of the storage tier. There are two throttles that can be used to increase or decrease the benchmark performance: Threads and Outstanding I/Os. Below are descriptions on how to best utilize these two settings to align the DiskSpd tool to how Windows and servers operate in order to achieve optimum results.

3.8.1. Threads

The –t parameter sets the number of parallel threads that will process the I/O requests and cumulate the testing results. This number will be launched per file so if there are four files and eight threads then 32 actual processing threads will get launched.

Windows attempts to even out hardware resources for both wear & tear, thermal regulation and optimum performance. As a part of this, Windows will attempt to even processing over the cores in each CPU while attempting to retain appropriate NUMA affinity. This translates into a core-hopping scenario, in which threads release an I/O request on one core while receiving the returned I/O on a different core, when the number of worker threads does not align with the number of cores per CPU. So, if the host being tested on has two CPUs, each with 8 cores, then setting the number of threads to something like 12 would mean that each thread would be hoping around between cores, having to pass the working data between L3 cache and slowing down the perceived latency per I/O and reducing the overall number of IOPs in the test. It is thus recommended to always set the number of threads equal to the number of physical cores in the first CPU and only change the I/Os outstanding as the throttle. See the next section for further explanation.

3.8.2. Outstanding I/Os

The –o parameter sets the number of I/Os that each thread can have outstanding. This is sometimes referred to as a queue depth for the processing threads. This is the maximum number of in-flight I/Os that each thread can have.

Outstanding I/Os should be the primary throttle configuration as the number of files, LUNs and threads should be determined

based upon the hardware and workload being tested. This leaves I/Os outstanding as the remaining dial with which to adjust performance.

There are two types of testing processes that can be followed: grid testing and manual testing. With grid testing, a script is written that tries every combination of each configuration and the overall results are reviewed later to determine which settings yielded the best results. This takes a long time to run and can miss the optimal settings. It is thus recommended to use the manual approached when available.

The manual testing process is the recommended approach. With this method the tester will start off with the appropriate number of LUNs, files, batch size and threads as discussed in other parts of this paper and then the I/Os outstanding will be set to 1 and incremented upwards, in small units, until peak IOPs or throughput has been achieved and the latency noted. This allows for a faster approach as the too high configurations are skipped and the lower settings can be cancelled after a quick check of the performance. So, a 5 minute run can be cut off after 15-30 seconds if the results are sub-optimal. Once the optimal numbers are found then a 5 minute run can be executed at that setting for official results. This process also helps identify the exact peak settings and exact latency at peak versus the inflated latencies with over saturated configurations.

3.9. Block Sizes

The block size for each test should be pre-determined based on the profile of the production workload it represents. While tests like 4k 100% read IOPs are commonly cited in marketing, there are almost no real world application use case that performs like this. So, it is recommended to profile real production workloads and emphasize the results from these types of tests over "hero" or non-production workload tests.

Common real-world workloads would be:

- SQL Database:
  - Data access:
    - OLTP: 70/30 ratio 8k batches
    - DW/BI: 70/30 ratio 64k/128k batches
  - Tempdb: 50/50 ratio 64k batches
  - Log: 100 writes 4k batches

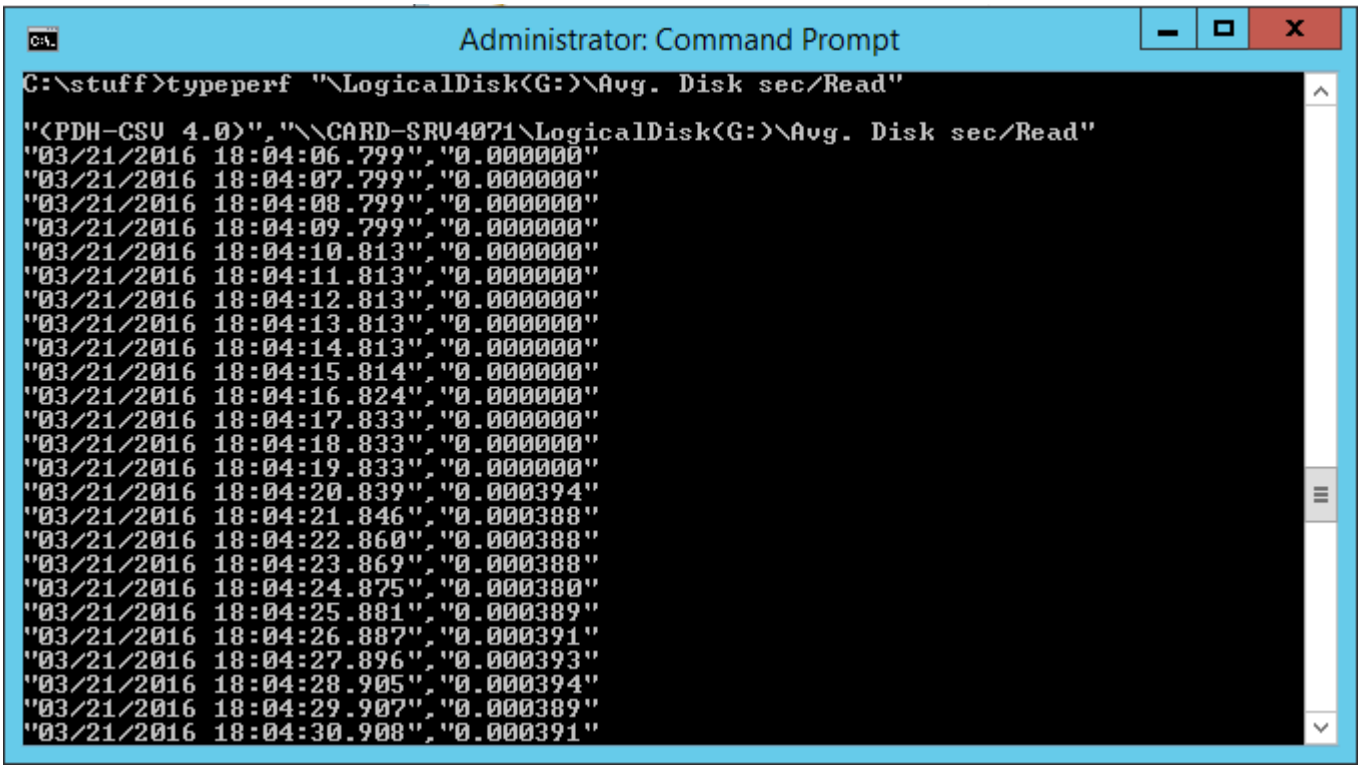
For IOPs testing it is common to use 4k or 8k batches. This configuration strains the storage and transport tiers ability to handle large numbers of requests. This helps represent use cases like OLTP databases or small application work.

4. Typeperf: Accurate Latency Measurement

4.1. Purpose of typeperf

Flash storage has brought performance to a whole new level in the data center. As such, some of the older tools have had to be upgraded or replaced. In the case of Performance Monitor (perfmon) the tool allows for adequate measurement of ultra-low latency storage, like flash devices, when collecting the data through a Data Collector. But, in the real-time GUI, it rounds to the nearest millisecond. For storage devices capable of performing below 1ms it does not due justice. In fact, it can be misleading. If an ultra-fast storage device was performing at 500 (.5ms) microseconds then it would round up in the GUI to 1 millisecond which could make it look to perform similarly to another device running at 1400 microseconds (1.4ms).

For accurate measurement, it is recommended that a tool other than the perfmon's GUI be used to review real-time latency measurement when doing storage benchmarking. A Data Collector can store all of the data for later use, but if real-time monitoring is desired then it is recommended to use typeperf. This is another Microsoft built tool shipped with all Windows versions that allows for the output of the monitoring objects directly to the command window. Typeperf will report latencies down to the microsecond, seconds to 6 digits. This allows for proper monitoring of ultra-fast storage devices.



4.2. Executing typeperf

The key statistic that needs real-time reporting with the proper significant digits is latency. To get this running, open a command window. Typeperf is in the path so it can be ran from any location. It can only report on reads or writes at a time so if needed, open two command windows. One for reads and one for writes. For the read command window, execute the following:

C:\ >typeperf "\LogicalDisk(*)\Avg. Disk sec/Read"

This will then start the output at about 1 per second. The time stamp is outputted along with the object data (latency). The tool can be stopped with a Ctrl+C. For writes, use the counter for write latency:

C:\ >typeperf "\LogicalDisk(*)\Avg. Disk sec/Write"

Note that the above examples pull all volumes. If it is desired to just monitor one volume, change the LogicalDisk object to contain the drive letter like this example filtering for the G drive:

C:\ >typeperf "\LogicalDisk(G:)\Avg. Disk sec/Write"

5. Examples

5.1. Testing Input

The following are some examples:

- 8k blocks, random, 6 threads, 16 outstanding I/Os at 70/30 r/w ratio over two 100GB sized files, each on a separate LUN.
  - C:\DiskSpd>Diskspd.exe -b8K -d300 -h -L -o16 –t6 -r -w30 -c100G G:\testfile.dat H:\testfile.dat
- 64k blocks, sequential (removing –r), 6 threads, 12 outstanding I/Os at 50/50 r/w ratio over two 200GB sized files, each on a separate LUN.
  - C:\DiskSpd>Diskspd.exe –b64K -d300 -h -L -o12 –t6 –w50 –c200G G:\testfile.dat H:\testfile.dat
- 4k blocks, random, 8 threads, 24 outstanding I/Os at 100% reads over one 100GB sized files, each on a separate LUN.
  - C:\DiskSpd>Diskspd.exe –b4K -d300 -h -L –o24 -t8 -r -w0 -c100G G:\testfile.dat

5.2. Testing Output

The following is output from a run on a 56 core based host. Note that DiskSpd only tracks the first 32 cores.

- Latency is a histogram near the bottom in RED. The minimum latency (min) is the most commonly used latency stat. Note the histogram's distribution of latency. Larger maximum and wider distributions are commonly considered less desirable than storage with a lower peak latency and tighter distributions.
- IOPs and throughput are in BLUE. Note that statistics are tracked per file and per thread, per file and per workload (reads vs writes). For mixed workloads make sure to review the read versus write statistics as well as the total statistics.

```
C:\stuff>Diskspd.exe -b4K -d30 -h -L -o32 -t2 -r -w30 -c100G G:\testfile.dat
H:\testfile.dat

Command Line: Diskspd.exe -b4K -d30 -h -L -o32 -t2 -r -w30 -c100G G:\testfile.dat
```

```
H:\testfile.dat

Input parameters:

        timespan:   1
        -------------
        duration: 30s
        warm up time: 5s
        cool down time: 0s
        measuring latency
        random seed: 0
        path: 'G:\testfile.dat'
                think time: 0ms
                burst size: 0
                software and hardware write cache disabled
                performing mix test (write/read ratio: 30/100)
                block size: 4096
                using random I/O (alignment: 4096)
                number of outstanding I/O operations: 32
                thread stride size: 0
                threads per file: 2
                using I/O Completion Ports
                IO priority: normal
        path: 'H:\testfile.dat'
                think time: 0ms
                burst size: 0
                software and hardware write cache disabled
                performing mix test (write/read ratio: 30/100)
                block size: 4096
                using random I/O (alignment: 4096)
                number of outstanding I/O operations: 32
                thread stride size: 0
                threads per file: 2
                using I/O Completion Ports
                IO priority: normal


Results for timespan 1:
*******************************************************************************

actual test time:       30.01s
thread count:           4
proc count:             32

CPU |  Usage |  User  |  Kernel |  Idle
-------------------------------------------
   0|  80.87%|   7.92%|   72.95%|  19.11%
   1|  81.18%|   8.12%|   73.06%|  18.80%
   2|  84.46%|   9.22%|   75.24%|  15.52%
   3|  84.25%|   8.85%|   75.40%|  15.73%
   4|   1.25%|   0.52%|    0.73%|  98.68%
   5|   0.00%|   0.00%|    0.00%|  99.93%
   6|   0.10%|   0.10%|    0.00%|  99.82%
   7|   0.00%|   0.00%|    0.00%|  99.93%
```

```
   8|   0.05%|   0.05%|    0.00%|  99.93%
   9|   0.21%|   0.00%|    0.21%|  99.77%
  10|   0.00%|   0.00%|    0.00%|  99.93%
  11|   0.00%|   0.00%|    0.00%|  99.93%
  12|   0.00%|   0.00%|    0.00%|  99.93%
  13|   0.10%|   0.10%|    0.00%|  99.82%
  14|   0.00%|   0.00%|    0.00%|  99.93%
  15|   0.00%|   0.00%|    0.00%|  99.93%
  16|   0.52%|   0.00%|    0.52%|  99.41%
  17|   0.00%|   0.00%|    0.00%|  99.93%
  18|   0.00%|   0.00%|    0.00%|  99.93%
  19|   0.00%|   0.00%|    0.00%|  99.93%
  20|   0.21%|   0.05%|    0.16%|  99.72%
  21|   0.00%|   0.00%|    0.00%|  99.93%
  22|   0.00%|   0.00%|    0.00%|  99.93%
  23|   0.00%|   0.00%|    0.00%|  99.93%
  24|   0.05%|   0.00%|    0.05%|  99.88%
  25|   0.00%|   0.00%|    0.00%|  99.93%
  26|   0.00%|   0.00%|    0.00%|  99.93%
  27|   0.00%|   0.00%|    0.00%|  99.93%
  28|   0.00%|   0.00%|    0.00%|  99.93%
  29|   0.00%|   0.00%|    0.00%|  99.93%
  30|   0.00%|   0.00%|    0.00%|  99.93%
  31|   0.00%|   0.00%|    0.00%|  99.93%
----------------------------------------------------------------------------
avg.|  10.41%|   1.09%|    9.32%|  89.52%

Total IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | L
atStdDev |  file
------------------------------------------------------------------------------
     0 |      7249870848 |      1769988 |     230.42 |   58987.33 |    0.542 |
   0.322 | G:\testfile.dat (100GB)
     1 |      7261163520 |      1772745 |     230.78 |   59079.21 |    0.541 |
   0.324 | G:\testfile.dat (100GB)
     2 |      7605411840 |      1856790 |     241.72 |   61880.13 |    0.516 |
   0.252 | H:\testfile.dat (100GB)
     3 |      7609827328 |      1857868 |     241.86 |   61916.05 |    0.516 |
   0.251 | H:\testfile.dat (100GB)
------------------------------------------------------------------------------
total:       29726273536 |      7257391 |     944.78 |  241862.71 |    0.528 |
   0.289

Read IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | L
atStdDev |  file
------------------------------------------------------------------------------
     0 |      5072539648 |      1238413 |     161.22 |   41271.85 |    0.531 |
   0.319 | G:\testfile.dat (100GB)
     1 |      5081006080 |      1240480 |     161.49 |   41340.73 |    0.531 |
   0.329 | G:\testfile.dat (100GB)
     2 |      5323177984 |      1299604 |     169.18 |   43311.12 |    0.507 |
   0.252 | H:\testfile.dat (100GB)
     3 |      5324611584 |      1299954 |     169.23 |   43322.79 |    0.506 |
   0.248 | H:\testfile.dat (100GB)
```

```
------------------------------------------------------------------------------
total:       20801335296 |      5078451 |     661.12 |  169246.49 |    0.518 |
   0.289

Write IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | L
atStdDev |  file
------------------------------------------------------------------------------
     0 |      2177331200 |       531575 |      69.20 |   17715.48 |    0.566 |
   0.328 | G:\testfile.dat (100GB)
     1 |      2180157440 |       532265 |      69.29 |   17738.48 |    0.565 |
   0.312 | G:\testfile.dat (100GB)
     2 |      2282233856 |       557186 |      72.54 |   18569.00 |    0.539 |
   0.250 | H:\testfile.dat (100GB)
     3 |      2285215744 |       557914 |      72.63 |   18593.26 |    0.540 |
   0.256 | H:\testfile.dat (100GB)
------------------------------------------------------------------------------
total:        8924938240 |      2178940 |     283.66 |   72616.22 |    0.552 |
   0.288


  %-ile |  Read (ms) | Write (ms) | Total (ms)
----------------------------------------------
    min |      0.202 |      0.185 |      0.185
   25th |      0.391 |      0.420 |      0.399
   50th |      0.450 |      0.483 |      0.460
   75th |      0.553 |      0.596 |      0.567
   90th |      0.766 |      0.812 |      0.781
   95th |      0.936 |      0.974 |      0.949
   99th |      1.486 |      1.521 |      1.497
3-nines |      2.454 |      2.485 |      2.464
4-nines |      8.010 |      7.650 |      7.870
5-nines |     24.608 |     24.539 |     24.551
6-nines |     42.137 |     35.227 |     42.052
7-nines |     42.775 |     42.122 |     42.775
8-nines |     42.775 |     42.122 |     42.775
    max |     42.775 |     42.122 |     42.775
```