Using DiskSpd in SQL Server environments

Writer: Robert Beene

Contributors: Jose Barreto, Ramu Konidena

Technical Reviewers: Robert Dorr, Mike Zawadzki, Nitin Verma, Dan Lovinger

Applies To: All Versions of SQL Server except Microsoft Azure SQL Database (MASD)

Topics

Background	Provides general information about how to stress and validate IO subsystems and about the tools that you can use.
General Information	Explains what DiskSpd is, when to use it, and what platforms it supports.
Features	Contains a partial list of the features of DiskSpd.
Open Source	Where to obtain the open source for DiskSpd.
Download	How to obtain DiskSpd.
Contents in Download	Contents of the zip file.
Syntax	Basic syntax for using DiskSpd.
Usage Warnings and Important Notes	Important things to know before using DiskSpd.
Using DiskSpd	Basic explanation of how to use DiskSpd.
Examples	More examples of DiskSpd syntax to use in testing SQL Server environments.
Analyzing the Results	Basic example of interpreting the results of DiskSpd.

BACKGROUND

A number of tools are available to stress and validate the functionality of I/O subsystems. Some of these tools actually simulate SQL Server I/O patterns, and others let you control the I/O patterns submitted to the subsystem. There are tools that confirm the functional validity of a configuration, while others are used only to test performance capacity.

Software that performs a validation test simulates a read and write pattern, but it also verifies that the information was correctly read and written. This type of software is often used to help find root causes of corruption or to help determine whether drivers and hardware are functioning as expected when they’re performing I/O. There is likely to be a speed component to running this test, but this test is more about accuracy than speed.

For the purposes of this document, a performance test tool focuses on simulating the READ and WRITE pattern, but it focuses even more heavily on the throughput and speed at which those operations take place (Load Generator). A performance test determines whether the hardware is capable of meeting the parameters at which it was designed to handle I/O throughput.

The following table compares the most common tools that are currently used to test I/O subsystems in the context of SQL Server.

Table 1 Tools used to test I/O subsystems

Tool	Used to determine	I/O patterns	Provided by
DiskSpd.exe	Performance capacity	User defined— Allows combinations of I/O types	Microsoft
SQLIO.exe[*] *(deprecated – use DiskSpd.exe)*	Performance capacity	User defined— Single I/O type at a time	Microsoft
SQLIOSim	Functional correctness	Simulates SQL Server I/O patterns	Microsoft
SQLIOStress[*] *(deprecated – use SQLIOSim)*	Functional correctness	Simulates SQL Server I/O patterns	Microsoft

SQLIOSim validates the basic functionality of an I/O subsystem under stress. by simulating actual SQL Server I/O patterns and checking results for correctness. SQLIOSim is commonly used by the Microsoft SQL Server support organization to isolate hardware-related corruption problems.

For the purposes of this discussion (performance tuning), DiskSpd is the most appropriate tool.

Top of the Document

INFORMATION

DiskSpd.exe is a versatile storage load generator and performance tool from Microsoft that can be used to run storage performance tests against files, partitions, and physical disks. It can simulate SQL Server I/O activity or more complex, variable access patterns. It can also return results in text or in detailed XML output for use in automated results analysis.

You can use DiskSpd for several configurations from a physical host or virtual machine, using all kinds of storage. These include local disks, LUNs on a SAN, Microsoft Windows Storage Spaces, and SMB file shares.

Top of the Document

[*] DO NOT USE. SQLIO and SQLIOStress are deprecated and will officially be pulled from download site, if they are not already. Any personal copies of the tools should be removed.

FEATURES

Some of the features include:

- Support for 64-bit systems, 32-bit systems, and ARM systems
- Ability to target physical disks in addition to partitions and files
- Variable read/write IO percentage settings
- Custom CPU affinity options
- Consumable XML output option
- Synchronization and tracing functionality
- Results in text (default) or in XML
- Can be executed from command prompt or PowerShell

Top of the Document

OPEN SOURCE

Open Source

DiskSpd is open source (MIT license) and can be found here: https://github.com/microsoft/DiskSpd

Top of the Document

DOWNLOAD

Download details

DISKSPD is provided “as is,” and is no support is offered for any problems encountered when using the tool. Please see the EULA.doc for the DiskSpd license agreement.

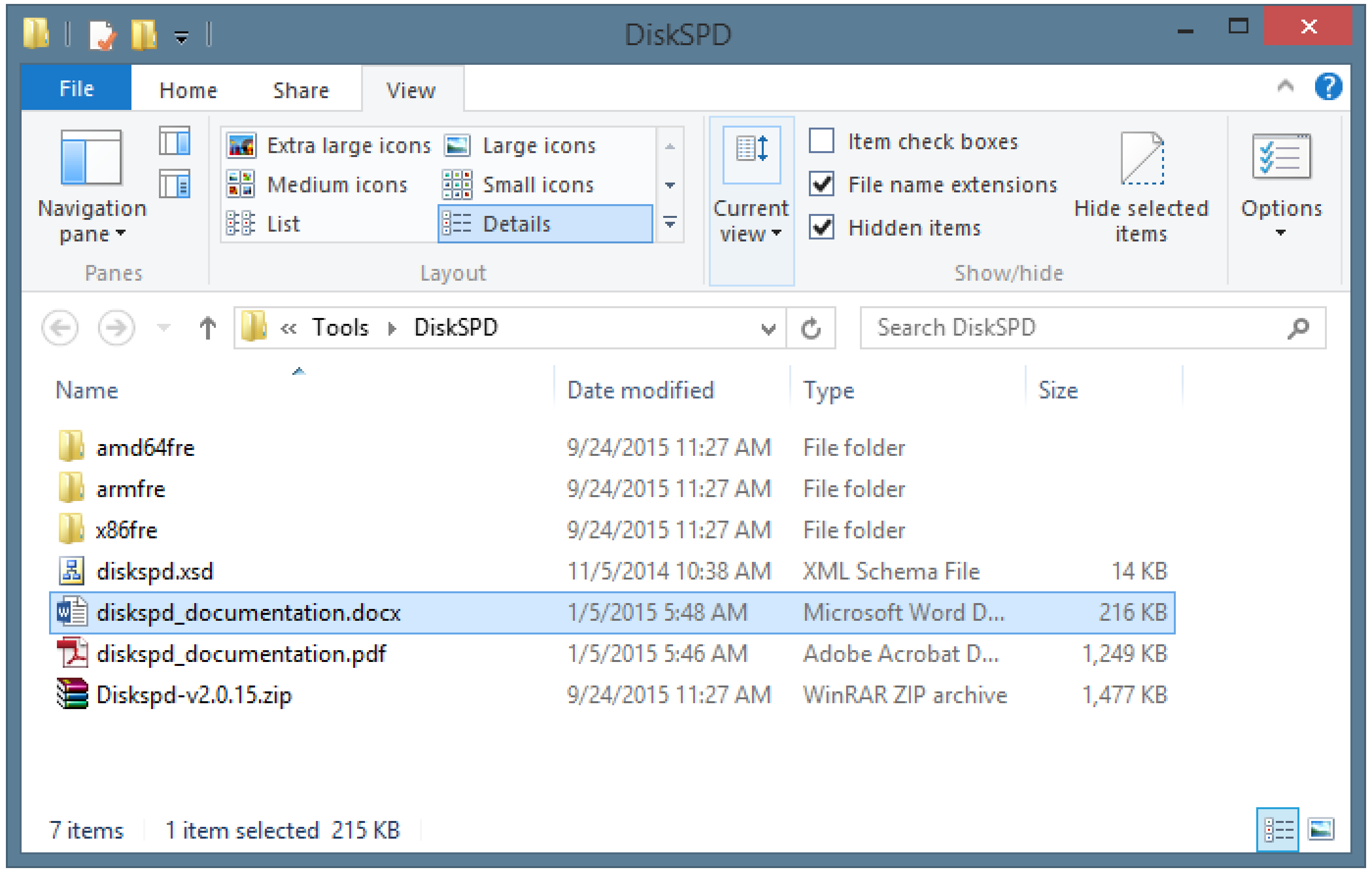
To download DiskSpd, go to DiskSpd, a Robust Storage Testing Tool.

Top of the Document

CONTENTS

The DiskSpd download contains a copy of DiskSpd.exe for each of the three supported architectures, plus documentation as shown in Figure 1.

Figure 1



Name	Date modified	Type	Size
amd64fre	9/24/2015 11:27 AM	File folder	
armfre	9/24/2015 11:27 AM	File folder	
x86fre	9/24/2015 11:27 AM	File folder	
diskspd.xsd	11/5/2014 10:38 AM	XML Schema File	14 KB
diskspd_documentation.docx	1/5/2015 5:48 AM	Microsoft Word D...	216 KB
diskspd_documentation.pdf	1/5/2015 5:46 AM	Adobe Acrobat D...	1,249 KB
Diskspd-v2.0.15.zip	9/24/2015 11:27 AM	WinRAR ZIP archive	1,477 KB

7 items 1 item selected 215 KB

[Top of the Document](#)

SYNTAX

To display a short description of all available options, use the -? parameter.

DiskSpd is a command-line utility that is invoked in the following way:

```
DiskSpd [options] target1 [ target2 [ target3 ...] ]
```

The following example runs a test for 15 seconds using a single thread to drive 100% random 8KiB reads at a depth of 10 overlapped (outstanding) I/Os to a regular file:

```
   DiskSpd -d300 -F1 -w0 -r -b8k -o10 c:\testfile.dat
```

Test targets can be regular files (C:\testfile.dat), partitions (C:), or physical drives (#1). Parameters can be specified as command-line options or in an XML file.

All available options and parameters are enumerated in the table in the “USAGE” section below.

Top of the Document

WARNINGS & PRECAUTIONS

WARNING: When you’re running DiskSpd, be aware that the load put on the environment could affect the performance of other virtual machines on the same physical machine. This could generate lots of load and disturb anyone else using other VMs in the same host, other LUNs on the same SAN, or other traffic on the same network.

WARNING: If you use DiskSpd to write data to a physical disk, you can destroy the data on that disk. DiskSpd does not ask for confirmation. Be careful when you’re using physical disks (as opposed to files) with DiskSpd.

Note: Be aware that uppercase and lowercase parameters have different meanings in DiskSpd.

Note: Make sure there’s nothing else running on the computer. Other running processes can interfere with the results by putting additional load on the CPU, network, or storage.

Note: Run DiskSpd from an elevated command prompt. This will make sure that file creation is fast. Otherwise, DiskSpd will fall back to a slower method of file creation.

Note: You can cancel a DiskSpd run at any time by pressing CTRL+C. DiskSpd exits gracefully and displays all the data it collected before the cancellation, as long as the warm-up period was completed.

Note: The iB notation is an international convention that unambiguously refers to power of 2–based sizing for numbers of bytes, as distinct from powers of 10, which continue to use KB/MB/GB notation.

- 1KiB = 2^{10} = 1,024 bytes
- 1MiB = 1024 KiB = 2^{20} = 1,048576 bytes
- 1GiB = 1024 MiB = 2^{30} = 1,073,741,824 bytes

Top of the Document

USAGE

For complete details about usage and additional examples, see the documentation that's contained in the download. We recommend that you read section 3, "Customizing DiskSpd tests," before using the tool.

In order to know and understand what values to pass for certain parameters for DiskSpd, we must cover more details on SQL Server's READ and WRITE file patterns for the different operations within SQL Server. You will want to make sure that you're familiar with the page and extent sizes in SQL Server. For more information, see Understanding Pages and Extents.

The following table covers the different SQL Server operations.

Note: Make sure that you always use a multiple of 8KiB for data files for those operations that show a range (for example, 8KiB–128KiB).

File type	Operation	READ pattern	WRITE pattern	Threads used	I/O type
Data File	Normal Activity	8KiB up to 128KiB	8KiB up to 128KiB	Based on MaxDOP	Random
	Checkpoint	N/A	64KiB up to 128 KiB	# of Sockets in Computer	Random
	LazyWriter	N/A	64KiB up to 128 KiB	1 per NUMA Node	Random
	Bulk Insert	N/A	8KiB up to 128 KiB	Based on MaxDOP	Sequential
	Backup	1 MB	1 MB	Based on MaxDOP	Sequential
	Restore	64KiB	64KiB	Based on MaxDOP	Sequential
	DBCC Checkdb w/ no repair option	8KiB up to 64KiB	N/A	Based on MaxDOP	Sequential
	Rebuild Index	See Read Ahead	8KiB – 128 KiB	Based on MaxDOP	Sequential
	ReadAhead	Up to 512 KiB	N/A	Based on MaxDOP	Sequential
Log File	Normal Activity	512 bytes - 64KiB	512 bytes - 64KiB	one log writer thread per soft NUMA node with a cap of 4	Sequential

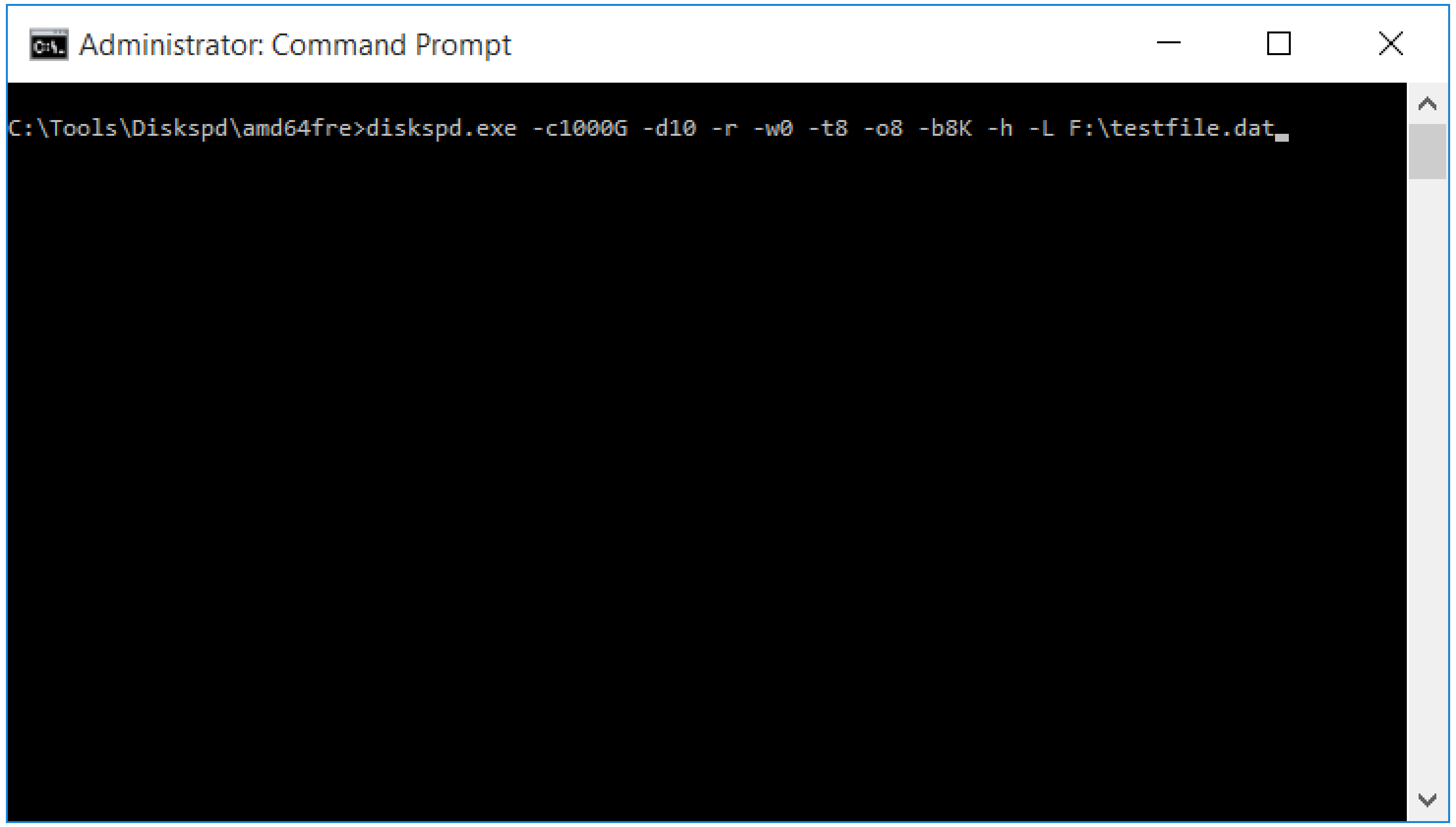
Now that we have a better understanding of the IO patterns for various operations in SQL Server, let's take a look at an sample usage of DiskSpd.

The following example creates:

```
DiskSpd.exe -c1000G –d300 -r -w0 -t8 -o8 -b8K -h -L F:\testfile.dat
```

- -c1000G (A 1000 GiB or 1.073 TB file)
- -d300 (5 Minutes)
- -r (Random IO)
- -w0 (No writes, 100% READ)
- -t8 (8 threads)
- -o8 (8 outstanding IO requests)
- -b8K (Block size is 8KiB)

- -h (Disable both software caching and hardware write caching)
- -L (Measure latency statistics)
- F:\testfile.dat (File path and name to create for test)



```
C:\Tools\Diskspd\amd64fre>diskspd.exe -c1000G -d10 -r -w0 -t8 -o8 -b8K -h -L F:\testfile.dat
```

Note: The test file that's created for the run is not automatically deleted. In this example, a 1000 Gib file is generated. It is important to delete the file afterward to free up the space.

Top of the Document

EXAMPLES

Note: All these examples use the following settings unless where specified:

A single target file sized at 1000 GiB
8 threads for data files
4 threads for Log files (1 per NUMA node up to 4)
300 Second duration
32 outstanding IOs for data files, 8 up to 116 outstanding IOs for log files

Your test may require modifying these settings.

Note: If you want to test LazyWriter Pattern set threads (-t) equal to the number of NUMA nodes on machine. Use 64KiB up to 128KiB WRITES

Patterns	Focus area	Test description	Sample command
Data file patterns	100% 8KiB Random reads	Large area of random concurrent reads of 8-KB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w0 -t8 -o32 -b8K -h -L F:\testfile.dat`
	100% 8KiB Random writes	Large area of random concurrent writes of 8-KB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w100 -t8 -o32 -b8K -h -L F:\testfile.dat`
	60% 8KiB Random READs, 40% 8KiB Random writes	Large area of random concurrent 60% reads and 40% writes of 8-KB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w40 -t8 -o32 -b8K -h -L F:\testfile.dat`

	100% 64KiB Random reads	Large area of random concurrent reads of 64-KB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w0 -t8 -o32 -b64K -h -L F:\testfile.dat`
Also simulates a normal/small checkpoint pattern Recommended threads (-t) equal # of sockets on machine to match checkpoint pattern	100% 64KiB Random writes	Large area of random concurrent writes of 64-KB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w100 -t8 -o32 -b64K -h -L F:\testfile.dat`
	60% 64KiB Random reads, 40% 64KiB Random writes	Large area of random concurrent 60% reads and 40% writes of 64-KB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w40 -t8 -o32 -b64K -h -L F:\testfile.dat`
Simulates a normal/large checkpoint pattern Recommended threads (-t) equal # of sockets on machine to match checkpoint pattern	100% 128KiB Random writes	Large area of random concurrent writes of 128-KB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w100 -t8 -o32 -b128K -h -L F:\testfile.dat`

Simulates a 512KiB Read Ahead Pattern with 5000 outstanding IOs (Enterprise Edition)	100% 512KiB Random reads	Large area of random concurrent reads of 512 KiB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w0 -t8 -o5000 -b512K -h -L F:\testfile.dat`
Simulates a 512KiB Read Ahead Pattern with 128 outstanding IOs (Standard Edition)	100% 512KiB Random reads	Large area of random concurrent reads of 512KiB blocks. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -d300 -r -w0 -t8 -o128 -b512K -h -L F:\testfile.dat`
Log File Patterns Max threads for log files is 1 per NUMA node up to 4. 4 is the max.	100% 64KiB reads	Large area of sequential concurrent reads of 64-KB blocks. 8 outstanding IOs. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -w0 -b64K -F4 -T1b -s8b -o8 -d300 -h F:\testfile.dat`
Log Files can have up to 116 outstanding IO's	100% 64KiB writes	Large area of sequential concurrent writes of 64-KB blocks. 116 Outstanding IOs. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -w100 -b64K -F4 -T1b -s8b -o116 -d300 -h F:\testfile.dat`
	40% 64KiB reads, 60% 64KiB writes	Large area of sequential concurrent 40% Reads, 60% writes of 64-KB blocks. 8 Outstanding IOs. Disable both software caching and hardware write caching.	`DiskSpd.exe -c1000G -w60 -b64K -F4 -T1b -s8b -o8 -d300 -h F:\testfile.dat`

	60% 64KiB reads, 40% 64KiB writes	Run a few separate instances of DiskSpd, but synchronize their start and stop times	`NOTE: You will need a separate Administrator Command prompt for each file plus and additional one to start/stop it.  In this example, you will need 4 administrator command prompts.`
			
			`This simulates random 60% Read, 40% Write for 2 files plus sequential 60% Read, 40% Write to the 3rd file.  All files in this example are 100GiB in size.`
			
			`DiskSpd.exe -c100G -r -w40 -t8 -o32 -b64K -yrMyStartEvent -ypMyStopEvent F:\testfile1.dat`
			
			`DiskSpd.exe -c100G -r -w40 -t8 -o32 -b64K -yrMyStartEvent -ypMyStopEvent F:\testfile2.dat`
			
			`DiskSpd.exe -c100G -w40 -b64K -F4 -T1b -s8b -o8 -yrMyStartEvent -ypMyStopEvent F:\testfile3.dat`
			
			`DiskSpd -yeMyStartEvent`
			
			`rem After a few seconds`
			
			`DiskSpd -yeMyStopEvent`

Top of the Document

ANALYSIS

For complete details on analyzing the results, see the documentation that’s included in the download.

By default, results will be in human-readable text summary, which can also be explicitly specified as -Rtext. An XML summary can be requested by using -Rxml.

Before jumping into the analysis of the results, it is important to know the max IO throughput of the targets in order to determine whether the system is performing at acceptable level. Knowing what to expect from the hardware is key to understanding what values you can expect in the results from this tool. In turn, this lets you know if your results are “good” or “poor.” You will want to not only consider the max throughput of the disk(s) or VHDs themselves, but also of the HBA or I/O card. Also take into consideration any RAID penalties for the different types of RAID used when you determine your max throughput.

For example, assume that you want to a test single SSD drive, and the drive is rated at 500 MB/sec READ or up to 10,000 READ IOPs and 450 MB/sec Write or up to 30,000 WRITE IOPs. Additionally, the I/O card can handle that rate of transfer. In this scenario, we should be able to send that volume of data per second with a reasonable transfer rate. Typical guidelines for determining optimal Average Disk Sec/Transfer for ideal SQL Server performance are shown in Figure 2 below.

Figure 2

Avg Disk Sec\Transfer	SQL performance
<= 10 milliseconds	Ideal.
Between 11 and 20 Milliseconds	Some Performance Impact is Possible
> 20 Milliseconds	SQL Performance is Impacted

Understanding RAID Penalties

It's important, when you're calculating your throughput by using RAID, to consider possible penalties. For example, RAID 5 is a very popular RAID configuration, but it has a write penalty in order to maintain the parity information. When you perform a WRITE under RAID 5, it is a READ, MODIFY, WRITE operation.

For example, suppose you have 1-64KiB WRITE requests, as follows:

READ	the RAID controller must READ the existing 64KiB and read the parity for that stripe.
MODIFY	the RAID controller, will XOR the old data from the parity.
WRITE	the RAID controller, will XOR the new data and write the new parity.

There is a penalty for writes with RAID 5. The penalty is 4 IOs for every IO. The following is the formula for calculating 100% write throughput with RAID 5:

MAX_WRITE * NUMBER_OF_DRIVES / PENALTY

The following table shows a configuration of three SSDs in a RAID 5 configuration where each SSD is rated at 450 MB/sec:

SSD_MAX_WRITE	**450MB**
NUMBER_OF_DRIVES	3
RAID5_PENALTY	4
FORMULA	**MAX_WRITE * NUMBER_OF_DRIVES / PENALTY**
CALCULATION	450MB * 3 / 4
MAX POSSIBLE WRITE THROUGHPUT	**337MB/sec**

Even though each drive can handle up to 450 MB/sec WRITES, the throughput actually drops for WRITES in order to maintain parity. You would want to use the adjusted throughput when evaluating WRITE performance.

COMMON RAID CONFIGURATIONS	
Level	**WRITE penalty**
RAID 0 Striping	1 (or basically no penalty)
RAID 1 Mirroring	2
RAID 5 Striping with Parity	4
RAID 10 Mirroring + Striping	2

Now that we have an understanding of the possible max throughput, we can start examining the results of running DiskSpd to determine whether the throughput is optimal. What follows is the text output that’s generated when you run DiskSpd.

Repeats command line executed

```
Command Line: DiskSpd.exe -c1000G –d300 -r -w0 -t8 -o8 -b8K -h -L F:\testfile.dat
```

Note: For this test, –L for latency tracking was used. You would use –D for IOPs statistics.

Lists the parameters used and any settings

```
Input parameters:

	timespan:   1
	-------------
	duration: 10s
	warm up time: 5s
	cool down time: 0s
	measuring latency
	random seed: 0
	path: 'F:\testfile.dat'
		think time: 0ms
		burst size: 0
		software and hardware write cache disabled
		performing read test
		block size: 8192
		using random I/O (alignment: 8192)
		number of outstanding I/O operations: 8
		thread stride size: 0
		threads per file: 8
		using I/O Completion Ports
		IO priority: normal
```

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

Shows the time of the run, the number of threads, and the number of processors

```
actual test time:       10.00s
thread count:           8
proc count:             12
```

Shows CPU usage total, User mode, Kernel mode, and idle. In this example, CPU usage was very low.

```
CPU |  Usage |  User  |  Kernel |  Idle
-------------------------------------------
   0|   0.94%|   0.16%|    0.78%|  99.06%
   1|   1.09%|   0.16%|    0.94%|  98.91%
   2|   3.44%|   0.31%|    3.12%|  96.56%
   3|   2.97%|   2.34%|    0.62%|  97.03%
   4|   1.41%|   0.78%|    0.62%|  98.59%
   5|   1.87%|   0.62%|    1.25%|  98.12%
   6|   2.34%|   0.78%|    1.56%|  97.66%
   7|   2.03%|   1.09%|    0.94%|  97.97%
   8|   3.28%|   1.72%|    1.56%|  96.72%
   9|   3.59%|   1.09%|    2.50%|  96.41%
  10|   3.28%|   0.31%|    2.97%|  96.72%
  11|   2.97%|   1.87%|    1.09%|  97.03%
-------------------------------------------
avg.|   2.43%|   0.94%|    1.50%|  97.56%
```

Gives IO statistis for the total per thread. In this example, you can see the Average Latency was quite high at over 100 ms.

```
Total IO
thread |       bytes     |     I/Os     |    MB/s   |  I/O per s |  AvgLat  | LatStdDev |  file
-----------------------------------------------------------------------------------------------------
     0 |         5341184 |          652 |      0.51 |      65.20 |  121.331 |   100.379 | F:\testfile.dat (1000GB)
     1 |         5865472 |          716 |      0.56 |      71.60 |  111.852 |    94.201 | F:\testfile.dat (1000GB)
     2 |         5636096 |          688 |      0.54 |      68.80 |  116.697 |   103.136 | F:\testfile.dat (1000GB)
     3 |         5545984 |          677 |      0.53 |      67.70 |  117.712 |   100.470 | F:\testfile.dat (1000GB)
     4 |         5308416 |          648 |      0.51 |      64.80 |  123.625 |   102.807 | F:\testfile.dat (1000GB)
     5 |         5947392 |          726 |      0.57 |      72.60 |  109.959 |    94.134 | F:\testfile.dat (1000GB)
     6 |         5414912 |          661 |      0.52 |      66.10 |  121.703 |   101.714 | F:\testfile.dat (1000GB)
```

```
    7 |       5431296 |          663 |       0.52 |      66.30 |  121.056 |    99.653 | F:\testfile.dat (1000GB)
```

The following total indicates that at 543 IOPs with 4.24 MB/second, there’s an average latency of 117.8 milliseconds (very slow).

```
total:          44490752 |         5431 |       4.24 |     543.10 |  117.808 |    99.620
```

For this run, we know it was a 100% READ test, so READ IO is a duplicate of Total IO.

```
Read IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | LatStdDev |  file
-----------------------------------------------------------------------------------------------------
     0 |       5341184 |          652 |       0.51 |      65.20 |  121.331 |   100.379 | F:\testfile.dat (1000GB)
     1 |       5865472 |          716 |       0.56 |      71.60 |  111.852 |    94.201 | F:\testfile.dat (1000GB)
     2 |       5636096 |          688 |       0.54 |      68.80 |  116.697 |   103.136 | F:\testfile.dat (1000GB)
     3 |       5545984 |          677 |       0.53 |      67.70 |  117.712 |   100.470 | F:\testfile.dat (1000GB)
     4 |       5308416 |          648 |       0.51 |      64.80 |  123.625 |   102.807 | F:\testfile.dat (1000GB)
     5 |       5947392 |          726 |       0.57 |      72.60 |  109.959 |    94.134 | F:\testfile.dat (1000GB)
     6 |       5414912 |          661 |       0.52 |      66.10 |  121.703 |   101.714 | F:\testfile.dat (1000GB)
     7 |       5431296 |          663 |       0.52 |      66.30 |  121.056 |    99.653 | F:\testfile.dat (1000GB)
-----------------------------------------------------------------------------------------------------
total:          44490752 |         5431 |       4.24 |     543.10 |  117.808 |    99.620
```

As expected, WRITE IO was 0.

```
Write IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | LatStdDev |  file
-----------------------------------------------------------------------------------------------------
     0 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
     1 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
     2 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
     3 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
     4 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
     5 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
     6 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
     7 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A | F:\testfile.dat (1000GB)
-----------------------------------------------------------------------------------------------------
total:                 0 |            0 |       0.00 |       0.00 |    0.000 |       N/A


  %-ile |  Read (ms) | Write (ms) | Total (ms)
----------------------------------------------
    min |      1.420 |        N/A |      1.420
   25th |     42.050 |        N/A |     42.050
```

```
     50th |    92.015 |       N/A |    92.015
     75th |   166.870 |       N/A |   166.870
     90th |   253.184 |       N/A |   253.184
     95th |   315.347 |       N/A |   315.347
     99th |   444.655 |       N/A |   444.655
  3-nines |   623.208 |       N/A |   623.208
  4-nines |   688.090 |       N/A |   688.090
  5-nines |   688.090 |       N/A |   688.090
  6-nines |   688.090 |       N/A |   688.090
  7-nines |   688.090 |       N/A |   688.090
  8-nines |   688.090 |       N/A |   688.090
      max |   688.090 |       N/A |   688.090
```

Top of the Document