



Vdbench
Users Guide

Version: 5.02
February 2010

Author: Henk Vandenberg
(<http://blogs.sun.com/henk/>)

Copyright © 2010 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved. U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms. Sun, Sun Microsystems, the Sun logo, Solaris and Vdbench are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. This product is covered and controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

Copyright © 2010 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés. L'utilisation est soumise aux termes de la Licence. Sun, Sun Microsystems, le logo Sun, Solaris et Vdbench sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd. Ce produit est soumis à la législation américaine en matière de contrôle des exportations et peut être soumis à la réglementation en vigueur dans d'autres pays dans le domaine des exportations et importations. Les utilisations, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers les pays sous embargo américain, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exhaustive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

TABLE OF CONTENTS :

1 VDBENCH: DISK I/O WORKLOAD GENERATOR.....	9
1.1 Introduction.....	9
1.2 Objective.....	9
1.3 Terminology.....	10
1.4 Summary of changes since release 5.01.....	10
1.4.1 Changes to 501 to File system testing (FSD/FWD):.....	10
1.4.2 Changes since 501 to Raw I/O testing (SD/WD).....	11
1.4.3 General changes since 501:.....	12
1.5 Summary of changes since release 5.00.....	14
1.6 Summary of changes since release 4.07.....	14
1.7 Installing Vdbench.....	15
1.8 How to start Vdbench:.....	15
1.9 Execution Parameter Overview.....	16
1.9.1 Execution Parameters.....	16
1.9.2 Parameter File(s).....	18
1.9.2.1 General Parameters: Overview.....	18
1.9.2.2 Host Definition (HD) Parameter overview.....	20
1.9.2.3 Replay Group (RG) Parameter Overview.....	20
1.9.2.4 Storage Definition (SD) Parameter Overview.....	21
1.9.2.5 File system Definition (FSD) Parameter Overview.....	22
1.9.2.6 Workload Definition (WD) Parameter Overview.....	22
1.9.2.7 File system Workload Definition (FWD) Parameter Overview.....	22
1.9.2.8 Run Definition (RD) Parameter Overview (For raw i/o testing).....	23
1.10 Report files.....	23
1.11 Execution Parameter Detail.....	26
1.11.1 '-f xxx ': Workload Parameter File(s)	26
1.11.2 '-oxxx': Output Directory	27
1.11.3 '-v': Activate Data Validation.....	28
1.11.4 '-j': Activate Data Validation and Journaling.....	28
1.11.5 '-s': Simulate Execution.....	28
1.11.6 '-k': Kstat Statistics on Console.....	28
1.11.7 '-m nn': Multi JVM Execution.....	29
1.11.8 '-t': Sample Vdbench execution.....	29
1.11.9 '-e nn' Override elapsed time.....	29
1.11.10 '-I nn' Override report interval time.....	30

1.11.11 '-w nn' Override warmup time.....	30
1.12 Vdbench utility functions.....	31
1.12.1.1 sds: Generate Vdbench SD parameters.....	31
1.12.1.2 dvpost: Data Validation post processing.....	31
1.12.1.3 jstack: Display java execution stacks of active Vdbench runs.....	31
1.12.1.4 rsh: Vdbench RSH daemon.....	31
1.12.1.5 print: Print any block on any lun or file.....	31
1.12.1.6 edit: Simple full screen editor, or 'back to the future'.....	32
1.12.1.7 compare: Compare Vdbench test results.....	32
1.12.1.8 parse: Parse Vdbench flatfile.....	32
1.12.1.9 gui: Vdbench demonstration GUI.....	32
1.13 General Parameter Detail.....	33
1.13.1 'include=parmfile'	33
1.13.2 'data_errors=xxx': Terminate After Data Validation or I/O Errors.....	33
1.13.3 'force_error_after': Simulate Data Validation Error.....	34
1.13.4 'swat=xxx': Create performance charts using Swat.....	34
1.13.5 'start_cmd=' and 'end_cmd=' command or script at the beginning/end of Vdbench or beginning/end of each run.....	34
1.13.6 'patterns=dirname': Data Patterns to be Used.....	35
1.13.7 'compression=nn': Set compression for data patterns.....	35
1.13.8 'port=nnnn': Specify port number for Java sockets.....	36
1.13.9 'create_anchors=yes': Create anchor parent directory.....	36
1.13.10 'report=': Generate extra SD reports.....	36
1.13.11 'histogram=': set bucket count and bucket size for response time histograms.....	37
1.13.12 'formatxfersize=nnnn'.....	37
1.14 Replay Group (RG) Parameter Overview.....	37
1.15 Host Definition parameter detail.....	38
1.15.1 'hd=host_label'	38
1.15.2 'system=system_name'	38
1.15.3 'jvms=nnn'	38
1.15.4 'vdbench=vdbench_dir_name'	38
1.15.5 'shell=rsh ssh vdbench'	38
1.15.6 'user=xxxx'	39
1.15.7 'mount=xxx'	39
1.16 Storage Definition Parameter Detail.....	40
1.16.1 'sd=name': Storage Definition Name.....	40
1.16.2 'lun=lun_name': LUN or File Name.....	40
1.16.3 'host=name'	41
1.16.4 'count=(nn,mm)'	41
1.16.5 'size=nn: Size of LUN or File.....	41
1.16.6 'range=nn': Limit Seek Range.....	42
1.16.7 'threads=nn': Maximum Number of Concurrent outstanding I/Os.....	42
1.16.8 'hitarea=nn': Storage Size for Cache Hits.....	42

1.16.9 'journal=name': Directory Name for Journal File.....	42
1.16.10 'offset=': Don't start at byte zero of a LUN.....	43
1.16.11 'align=': Determine lba boundary for random seeks.....	43
1.16.12 'openflags=': control over open and close of luns or files.....	44
1.17 Workload Definition Parameter Detail.....	45
1.17.1 'wd=name': Workload Definition Name.....	45
1.17.2 'host=host_label'.....	45
1.17.3 'sd=name': SD names used in Workload.....	46
1.17.4 'rdpct=nn': Read Percentage.....	46
1.17.5 'rhpct=nn' and 'whpct=nn': Read and Write Hit Percentage.....	46
1.17.6 'xfersize=nn': Data Transfer Size.....	46
1.17.7 'skew=nn': Percentage skew.....	47
1.17.8 'seekpct=nn': Percentage of Random Seeks.....	47
1.17.9 'range=nn': Limit Seek Range.....	48
1.17.10 'iorate=' Workload specific I/O rate.....	48
1.17.11 'priority=' Workload specific I/O priority.....	48
1.18 Run Definition for raw I/O parameter detail.....	50
1.18.1 'rd=name': Run Name.....	50
1.18.2 'wd=': Names of Workloads to Run.....	51
1.18.3 'sd=xxx'.....	51
1.18.4 'iorate=nn': One or More I/O Rates.....	51
1.18.5 'curve=nn': Define Data points for Curve.....	52
1.18.6 'elapsed=nn': Elapsed Time.....	52
1.18.7 'interval=nn': Reporting Interval.....	52
1.18.8 'warmup=nn': Warmup period.....	52
1.18.9 'distribution=xxx': Arrival Time Distribution.....	53
1.18.10 'pause=nn': Sleep 'nn' Seconds.....	53
1.18.11 Workload parameter specification in a Run Definition.....	54
1.18.11.1 'sd=xxx' Specify SDs to use.....	54
1.18.11.2 '(for)xfersize=nn': Create 'for' Loop Using Different Transfer Sizes.....	54
1.18.11.3 '(for)threads=nn': Create 'for' Loop Using Different Thread Counts.....	55
1.18.11.4 '(for)rdpct=nn': Create 'for' Loop Using Different Read Percentages.....	55
1.18.11.5 '(for)rhpct=nn': Create 'for' Loop Using Different Read Hit Percentages.....	55
1.18.11.6 '(for)whpct=nn': Create 'for' Loop Using Different Write Hit Percentages.....	56
1.18.11.7 '(for)seekpct=nn': Create 'for' Loop Using Different Seek Percentages.....	56
1.18.11.8 '(for)hitarea=nn': Create 'for' Loop Using Different Hit Area Sizes.....	56
1.18.11.9 '(for)compression=nn': Create 'for' Loop Using Different compression rates.....	57
1.18.11.10 Order of Execution Using 'forxxx' Parameters.....	57
1.19 Data Validation and Journaling.....	58
1.20 Swat I/O Trace Replay.....	61
1.21 Complete Swat I/O Replay Example.....	62
1.22 File system testing.....	63

1.22.1 Directory and file names.....	63
1.22.2 File system sample parameter file.....	64
1.23 File System Definition (FSD) parameter overview:.....	65
1.24 Filesystem Workload Definition (FWD) parameter overview:.....	65
1.24.1 Multi-host parameter replication.....	66
1.25 Run Definition (RD) parameters for file systems, overview.....	66
1.26 File System Definition (FSD) parameter detail:.....	67
1.26.1 'fsd=name': Filesystem Storage Definition name.....	67
1.26.2 'anchor=': Directory anchor.....	67
1.26.3 'count=(nn,mm)': Replicate parameters.....	67
1.26.4 'shared=': FSD sharing.....	67
1.26.5 'width=': Horizontal directory count.....	68
1.26.6 'depth=': Vertical directory count.....	68
1.26.7 'files=': File count.....	68
1.26.8 'sizes=': File sizes.....	68
1.26.9 'openflags=': Optional file system 'open' parameters.....	69
1.26.10 'total_size=': Create files up to a specific total file size.....	69
1.26.11 'workingsetsize=nn' or 'wss=nn'.....	69
1.27 File system Workload Definition (FWD) detail.....	70
1.27.1 'fwd=name': File system Workload Definition name.....	70
1.27.2 'fsd=': which File System Definitions to use.....	70
1.27.3 'fileio=': random or sequential I/O.....	70
1.27.4 'rdpct=nn': specify read percentage.....	70
1.27.5 'stopafter=': how much I/O?.....	70
1.27.6 'fileselect=': which files to select?.....	71
1.27.7 'xfersizes=': data transfer sizes for read and writes.....	71
1.27.8 'operation=': which file system operation to execute.....	71
1.27.9 'skew=': which percentage of the total workload.....	71
1.27.10 'threads=': how many concurrent operations for this workload.....	71
1.28 Run Definition (RD) parameters for file system testing, detail.....	73
1.28.1 'fwd=': which File system Workload Definitions to use.....	73
1.28.2 'fwdrate=': how many operations per second.....	73
1.28.3 'format=': pre-format the requested directory and file structure	74
1.28.4 'operations=': which file system operations to run.....	75
1.28.5 'foroperations=': create 'for' loop using different operations.....	75
1.28.6 'fordepth=': create 'for' loop using different directory depths.....	75
1.28.7 'forwidth=': create 'for' loop using different directory widths.....	76
1.28.8 'forfiles=': create 'for' loop using different amount of files.....	76
1.28.9 'forsizes=': create 'for' loop using different file of sizes.....	76
1.28.10 'fortotal=': create 'for' loop using different total file sizes.....	77
1.28.11 'forwss=': 'for' loop using working set sizes.....	77

1.29 Multi Threading and file system testing.....	78
1.30 Operations counts vs. nfsstat counts:.....	79
1.31 Report file examples.....	80
1.31.1 summary.html.....	80
1.31.2 summary.html for file system testing.....	81
1.31.3 SD_name.html.....	81
1.31.4 logfile.html.....	82
1.31.5 kstat.html.....	82
1.31.6 histogram.html.....	83
1.31.7 nfs3/4.html.....	84
1.31.8 flatfile.html.....	84
1.32 Sample parameter files.....	85
1.32.1 Example 1: Single run, one raw disk.....	85
1.32.2 Example 2: Single run, two raw disk, two workloads.....	85
1.32.3 Example 3: Two runs, two concatenated raw disks, two workloads.....	85
1.32.4 Example 4: Complex run, curves with different transfer sizes.....	86
1.32.5 Example 5: Multi-host.....	86
1.32.6 Example 6: Swat I/O trace replay.....	87
1.32.7 Example 7: File system test.....	87
1.33 Permanently override Java socket port numbers.....	88
1.34 Java Runtime Environment.....	89
1.35 Solaris.....	89
1.36 HP/UX.....	89
2 VDBENCH WORKLOAD COMPARE.....	90
3 VDBENCH FLATFILE SELECTIVE PARSING.....	91
4 VDBENCH SD PARAMETER GENERATION TOOL.....	92
5 VDBENCH DATA VALIDATION POST-PROCESSING TOOL.....	94
6 VDBENCH GUI DOCUMENTATION.....	97
6.1 Overview.....	97
6.2 Entering Operating Parameters.....	97
6.3 Storage Device Selection.....	98

<u>6.4 File Selection.....</u>	<u>99</u>
<u>6.5 Storage Device and File Definition Deletion.....</u>	<u>100</u>
<u>6.6 Workload Definition Specification.....</u>	<u>100</u>
<u>6.7 Workload Definition Modification.....</u>	<u>101</u>
<u>6.8 Workload Definition Deletion.....</u>	<u>101</u>
<u>6.9 Run Definition Specification.....</u>	<u>102</u>
<u>6.10 Running Vdbench.....</u>	<u>102</u>
<u>6.11 Starting a Run.....</u>	<u>103</u>
<u>6.12 Stopping a Run.....</u>	<u>103</u>
<u>6.13 Application Menus.....</u>	<u>103</u>
6.13.1 File Menu.....	103
6.13.2 Help Menu.....	104

1 Vdbench: Disk I/O Workload Generator

Getting started with Vdbench:

- [Installation Instructions](#)
- [Vdbench GUI Documentation](#)
- [Execution](#)
- [Example parameter files](#)

1.1 Introduction

Vdbench is a disk and tape I/O workload generator to be used for testing and benchmarking of existing and future storage products.

Vdbench is written in Java with the objective of supporting Sun heterogeneous attachment. At this time Vdbench has been tested on Solaris Sparc and x86, Windows NT, 2000, 2003, 2008 and XP, HP/UX, AIX, Linux, Mac OS X, zLinux, and native VmWare

Note: one or more of these platforms may not be available for this latest release, this due to the fact that a proper system for a Java JNI C compile may not have been available at the time of distribution. In this case there will be a 'readme.txt' file in the OS specific subdirectory, asking for a volunteer to do a small Java JNI C compile.

1.2 Objective

The objective of Vdbench is to generate a wide variety of controlled storage I/O workloads, allowing control over workload parameters such as I/O rate, LUN or file sizes, transfer sizes, thread count, volume count, volume skew, read/write ratios, read and write cache hit percentages, and random or sequential workloads. This applies to both raw disks and file system files and is integrated with a detailed performance reporting mechanism eliminating the need for the Solaris command *iostat* or equivalent performance reporting tools. Vdbench performance reports are web accessible and are linked using HTML. Just point your browser to the *summary.html* file in the Vdbench output directory.

There is no requirement for Vdbench to run as root as long as the user has read/write access for the target disk(s) or file system(s) and for the output-reporting directory.

Non-performance related functionality includes data validation with Vdbench keeping track of what data is written where, allowing validation after either a controlled or uncontrolled shutdown.

1.3 Terminology

- **Execution** parameters control the overall execution of Vdbench and control things like parameter file name and target output directory name.
- **Raw I/O workload** parameters describe the storage configuration to be used and the workload to be generated on it. The Workload parameters include **General**, **Host Definition (HD)**, **Replay Group (RG)**, **Storage Definition (SD)** and **Run Definition (RD)**. A **Run** is the execution of one workload requested by a Run Definition. Multiple **Runs** can be requested within one Run Definition.
- **File system Workload** parameters describe the file system configuration to be used and the workload to be generated on it. The Workload parameters include **General**, **Host Definition (HD)**, **File System definition (FSD)**, **File system Workload Definition (FWD)** and **Run Definition (RD)** parameters and must always be entered in the order in which they are listed here. A **Run** is the execution of one workload requested by a Run Definition. Multiple **Runs** can be requested within one Run Definition.
- **Replay**: This Vdbench function will replay the I/O workload traced with and processed by the Sun StorageTek™ Workload Analyzer Tool (Swat).
- **Master and Slave**: Vdbench runs as two or more Java Virtual Machines (JVMs). The JVM that you start is the master. The master takes care of the parsing of all the parameters, it determines which workloads should run, and then will also do all the reporting. The actual workload is executed by one or more Slaves. A Slave can run on the host where the Master was started, or it can run on any remote host as defined in the parameter file. See also '[-m nn](#)': [Multi JVM Execution](#)
- **Data Validation**: Though the main objective of Vdbench has always been to execute storage I/O workloads, Vdbench also will help you identify data corruptions on your storage.
- **Journaling**: A combination of Data Validation and Journaling allows you to identify data corruption issues across executions of Vdbench. See Data Validation and Journaling.

1.4 Summary of changes since release 5.01

1.4.1 Changes to 501 to File system testing (FSD/FWD):

1. Complete Data Validation and Journaling for File system testing.
2. format= parameter enhancements, including 'format=restart'.
3. new openflags= options, including fsync and directio.
4. openflags= now available on FSD, FWD, and RD.
5. fwd=format special FWD to set xfersize=, threads=, and openflags= for format runs.
6. The format has changed from first creating all files as empty and then filling them with the requested amount of file size, to now create and then immediately fill.
7. 'flatfile.html' now also created for file system tests.
8. './vdbench compare' can now also be used for output of file system tests.
9. Vdbench now has skew control just like it has always had for raw workloads: if you want 50/50 read/write, Vdbench will make sure that those objectives are met and no longer will let faster workloads dominate the slower ones.

10. If requested, Vdbench now will create mountpoint directories and mount file systems, and will even change mount options in between runs.
11. File system workloads now report a response time histogram. These histograms are reported for each FSD, each FWD, and each RD.
12. There are now detail reports for each FSD and FWD instead of only the totals.
13. Just like with 'sd=', 'fsd=' now also supports the 'count=(start,###)' parameter to quickly define multiple FSDs.
14. 'fsd=xxx,shared=yes' parameter to allow multiple clients and/or JVMs to use the same FSD instead of requiring a unique FSD for each host or JVM.
15. 'fsd=xxx,workingsetsize=' (or wss=) parameter to force Vdbench to use only a specific subset of the total amount of file space that has been created.
16. 'fsd=xxx,distribution=all' parameter forcing Vdbench to create files under each directory instead of only in the lowest level directory.
17. 'fwd=xxx,stopafter=' parameter now allows you to specify a percentage, e.g. stopafter=20%, where, during a random I/O workload, Vdbench will switch to a new file after processing 20% of the file's data.
18. 'fwd=xxx,stopafter=' can be used to simulate an 'append' to disk files.
19. 'fsd=xxx,size=(100m,0)' A new option that instead of creating fixed file sizes it will create random file sizes, with an average of the requested file size.
20. Vdbench now keeps track of the file structure that it has created or is creating. If for instance you created 10 million files in a previous execution it would waste a lot of time in the next execution verifying 'which file exists already, how long is each file? Etc'. Now Vdbench just keeps the info available in file 'vdb_control.file' placed directly in the FSD anchor.
21. Changes to file system reporting: the 'ops' column has been renamed to the 'ReqstdOps' column. I also added a 'total mb/sec' column. The 'ReqstdOps' column reports only the – requested—operations. For instance if you ask for reads to be done this column contains only the read statistics though the 'open' and 'close' statistics are still reported separately in their own columns.
22. The anchor directory will now always be created if it is not there yet. The parent directories only will be created however when you add the 'create_anchors=yes' parameter.
23. New parameter to allow a sequential file to be deleted before it is rewritten. 'fwd=xxx,...,fileio=(sequential,delete)'.
24. New 'fwd=xxx,...,rdpct=xx' parameter to allow reads and writes to a dataset to be mixed. This does not make much sense for sequential I/O (read block1, write block2, read block3, write block4, etc) but has use for random I/O.

1.4.2 Changes since 501 to Raw I/O testing (SD/WD)

1. When Vdbench finds disk files that do not exist, all files will be created during one single format run instead of one format run per file, improving throughput.
2. If Vdbench finds a disk file that is too small for the size= values used it will expand that file, also just in one single format run. This equates almost to the new 'format=restart' option of file system testing.
3. A new formatxfersize= parameter to control the data transfer size used during above format runs.

4. openflags= now available on SD, WD and RD.
5. On Windows any error code returned from the GetLastError() function will now be properly translated into text using the getWindowsErrorText() API call.
6. new openflags= options, including fsync() and directio()
7. Garbage in the /dev/rdisk/ directory could cause Vdbench to abort because of a non-zero return code from the 'ls' command. Fixed.
8. The 'L' from the 'ls -lL /dev/rdisk' command has been removed. This could cause temporary hangs if there was a device that was in an error state. This hang would occur at the start of a run where Vdbench (on Solaris) was interrogating the storage configuration.
9. Vdbench now no longer creates a report for each separate SD, once for each host and once for each slave. Running 200 SDs against 8 slaves (on one host) now has 1800 less report files. Override this using the 'report=slave' or 'report=host' parameter.
10. I finally bit the bullet. On Linux when opening devices starting with '/dev/', you are now required to specify 'openflags=o_direct' to force raw I/O. If you really need to run against a /dev/ device while not doing raw I/O, I coded in a trick, let me know if you need it.
11. When more than one Workload Definition (WD) is specified, Vdbench will now generate a new Workload Report.
12. You now can specify specific target I/O rates for each Workload Definition.
13. You now can specify an I/O priority for each Workload Definition.
14. Vdbench will no longer create a new file that starts with /dev/.
15. You can now specify a complete workload without ever specifying a Workload Definition by entering (most of) the parameters under the Run Definition.
16. ./vdbench sds now also works for Linux and Windows

1.4.3 General changes since 501:

1. Journal recovery now is aware of those write operations that were pending at the time of the Vdbench or system shutdown and will analyze the status of those outstanding I/Os to see whether those blocks contain the old contents (from before the write) or the new contents, though at this time Vdbench is not able to completely verify the contents if half of the block contains 'new' data, with the other half containing 'old' data. This scenario will happen when a write is interrupted, for instance due to a power outage.
2. All platforms: a check has been added in the JNI read and write functions to make sure that IF an I/O error is returned the 'errno' variable contains a non-zero value.
3. Java 1.5 is now required. It already was required, but the check was not done.
4. Added a [new 'print'](#) function to be used for collecting diagnostics data after a Data Validation error.
5. After a Data Validation error discovered during the pre-read of a write, the block will not be overwritten (this was a bug).
6. 'host=(host_name,host_label' has been replaced by 'hd=host_label,system=host_name' (HD=Host Definition). This was done to keep syntax in line with all other SD/WD/FSD/FWD/RD parameters. However, if you use the older format it will still be accepted for a while.

7. A new 'include=filename' parameter, allowing Vdbench to dynamically insert multiple parameter files. This in addition to the already existing './vdbench -f file1 file2 file*n*' execution parameters
8. To assist with the creation of complex parameter files, any time that Vdbench finds '\$host' in a line of parameters, that whole line is repeated once for each defined Host Definition, repeating '\$host' each time with the host label defined in the HD.
9. New 'vdbench jstack' function. When running from a java JDK, Vdbench will print out the current Java execution stacks for all currently running JVMs. Great for debugging Vdbench hangs when you see any.
10. Vdbench compare now allows you to compare output directories where the amount of runs completed does not match.
11. Small enhancement to using the '-s' (simulate) execution parameter. This now displays the run names and 'forxxx' parameters for each run that you have requested.
12. All Data Validation and Journaling options now can be specified inside of the parameter file instead of only on the command line.
13. A new 'warmup=' parameter giving you control over the amount of reporting intervals that will be excluded from run totals.
14. Introduced 'forxfersize=(512k-1k,d)', the reverse of 'double the previous value'. Can also be used for other forxx parameters.
15. The data pattern generation for Data Validation has improved. Instead of using only the lba and the DV key as a seed value I now also add the SD or FD name to the seed. This allows a greater data pattern diversity when using multiple SDs.
16. Journals can now be done on raw devices to make sure that possible file system issues cannot compromise the integrity of the journal file. Add 'journal=/dev/xxxx' to the SD or FSD. The 'map' file (backup copy of the raw DV map) will in this case be written to the current directory.
17. The journal file format has been rewritten to allow the integrity of the journal file to be checked.
18. Resolved a loop when Journal recovery was done on a system with a single cpu. They still exist ☺
19. 'Journal recovery only'. Allows you to recover journals and validate all data without executing a workload.
20. The 'patterns=' option has been changed to only support a 'default' pattern.
21. To accommodate synchronous journal writes, the journal file is now opened using the 'O_SYNC' parameter to make sure that a file's metadata also is updated.
22. A Data Validation Post-processing GUI. The amount of data reported in the errorlog.html file after a DV error can be so overwhelming that it becomes difficult to filter out useful information.

1.5 Summary of changes since release 5.00

1. Support for zLinux
2. 64bit support for Solaris, Linux and zLinux.
3. Over-use of threads in multi JVM runs has been resolved
4. <http://blogs.sun.com/henk/> will be used for needed general communications.
5. Solved memory over-allocation caused by unneeded retention of nfsstat statistics.
6. Data Validation now also available for tape.
7. Including now a set of sample parameter files in the /examples/ directory.
8. Replaced file example5 with a proper multi-host example.
9. Data patterns for Data Validation now created using a Linear Feedback Shift Register Sequence to force more data uniqueness.
10. The date field in the column header now is refreshed at midnight.
11. Added a new /examples/ directory with about thirty sample parameter files.
12. Vdbench also runs on native VmWare. See script 'Vdbench.bash' for instructions.
13. './vdbench parseflat': allows selective parsing of flatfile.html (raw functionality only).

1.6 Summary of changes since release 4.07

1. Streamlined multi host processing, allowing central control and reporting over concurrent workloads executed on multiple hosts.
2. The amount of reporting has tripled. There will be one copy of each report for each slave, each host, and then a total for the complete run.
3. '[count=](#) SD parameter' to allow quick definition of ranges of SDs.
4. '[./vdbench sds](#)' a Solaris GUI to assist with the creation of SD parameters and the creation of other commands or lists that require frequent manipulation of different lun names.
5. 'range=' parameter allowing you to run workloads on subsets of an SD.
6. '-l' execution parameter to 'loop' through all runs over and over again.
7. Improvements around tape handling, properly recognizing EOT.
8. Vdbench does tape reads only if the same tape (real or virtual) has been written in the same Vdbench execution. This is done to allow proper recognition of EOT.
9. New sd= parameter in a Run Definition (RD).
10. New '[swat=](#) parameter' allowing the automatic creation of performance charts using Sun StorageTek™ Workload Analysis Tool.
11. Something that already existed but was not documented: a file, *swat_mon.txt* that can be imported into Swat Performance Monitor (SPM) for the creation of performance charts.
12. Vdblite has been removed. Processors have become fast enough to eliminate the need to save a few cycles.
13. The '-vq' parameter has been removed. This was more of an 'oops', but considering the fact that processors have become much faster this probably won't be missed.
14. The 'ciorate' parameter has been removed. No feedback was ever provided about this option so it is no longer supported.
15. The 'steady_rate' parameter has been removed for the same reason.
16. The 'rdpct=copy' option has been removed.
17. The 'fsd=xxx,cleanup=' parameter has been removed.

1.7 Installing Vdbench

Vdbench is packaged as a *tar* file or as a *zip* file. Untar or unzip the file and you're ready to go. Both tar and zip files contain everything you need for both Windows and Unix systems.

1.8 How to start Vdbench:

You can do a very quick simple test without even having to create a parameter file:

Unix: /home/vdbench/vdbench -t

Windows: c:\vdbench\vdbench -t

After this, use your favorite web browser to look at /home/vdbench/output/summary.html and you'll see the reports that Vdbench creates.

To start Vdbench:

- Unix: /home/vdbench/vdbench -f parmfile
- Windows: c:\vdbench\Vdbench.bat -f parmfile

You can find some simple example parameter files here: [link to some sample parameter files:](#)

1.9 Execution Parameter Overview

vdbench [- fxxx yyy zzz] [-o xxx] [-c] [-l] [-s] [-k] [-e nn] [-i nn] [-w nn] [-m nn] [-v] [-vr] [-vw] [-vt] [-j] [-jr] [-jm] [-jn] [-jro] [-p nnnn] [-t] [-l] [-]

Execution parameters must be specified individually: Enter ‘-v -fparmfile’, and not ‘-vfparmfile’.

or,

./vdbench [compare] [gui] [dvpost] [edit] [jstack] [parse] [print] [sds] [rsh] for some Vdbench utility functions.

Here is a brief description of each parameter, with a link to a more detailed description:

1.9.1 Execution Parameters

See also [Execution Parameter Detail](#).

compare	Start Vdbench workload compare
dvpost	Post-processing of output generated by Data Validation
edit	Primitive full screen editor, syntax ‘./vdbench edit file.name’.
gui	Start the Vdbench GUI
jstack	Create stack trace. Requires a JDK.
parse(flat)	Selective parsing of flatfile.html
print	Print any block on any disk or disk file
rsh	Start Vdbench RSH daemon (For multi-host testing)
sds	Start Vdbench SD parameter generation tool (Solaris)
-f xxx yyy zzz	Workload parameter file name(s) . One parameter file is required.
-o xxx	Output directory for reporting. Default is ‘output’ in current directory.
-t	Create a 5 second sample workload on a small disk file (for demo).
-e nn	Override ‘ elapsed ’ parameters in Run Definitions (RD)
-i nn	Override ‘ interval ’ parameters in Run Definitions (RD)
-w nn	Override ‘warmup’ parameters in Run Definitions (RD).
-m nn	Override the amount of concurrent JVMs to run workload
-v	Activate data validation .
-vr	Activate data validation , immediately re-read after each write.
-vw	Activate data validation , but don’t read before write.
-vt	Activate data validation , keep track of each write timestamp (memory intensive)
-j	Activate data validation with journaling .
-jr	Recover existing journal , validate data and run workload

-jro	Recover existing journal, validate data but do not run requested workload.
-jm	Activate journaling, but only write the journal maps.
-jn	Activate journaling, but use asynchronous writes to journal.
-s	Simulate execution. Scans parameter files and displays run names.
-k	Solaris only: Report kstat statistics on console .
-c	Clean (delete) existing FSD file system structure at start of run.
-p nnnn	Override Java socket port number (default 5570).
-l	After the last run, start over with the first run. This is an endless loop.
-	Command line input for Vdbench. '-f' files will be ignored.

1.9.2 Parameter File(s)

The parameter files entered will be read in the order specified. All parameters have a required order as defined here: **General**, **HD**, **RG**, **SD**, **WD** and **RD**, or for file system testing: **General**, **HD**, **FSD**, **FWD** and **RD**.

Note that not all types of parameters are always needed.

There is however one parameter that can be anywhere: include=/parm/file/name

When this parameter is found, the contents of the file name specified will be copied in place.

Example: include=/complicated/workload/definitions.txt

You can use as many includes as needed, though overuse of this parameter will make it very difficult to take a quick look at a parameter file to see what's being requested. File 'parmfile.html' in the output directory will show you the final results of everything that has been included.

1.9.2.1 General Parameters: Overview

These parameters must be the *first* parameters in the parameter file.

See also [General Parameter Detail](#).

data_errors=nn	Terminate after 'nn' read/write/data validation errors (default 50)
data_errors=cmd	Run command or script 'cmd' after first read/write/data validation error, then terminate .
force_error_after=(nn,mm)	Simulate data validation error after nn reads. Simulate 'mm' errors (default 1)
start_cmd=cmd	Execute command or script at the beginning of the first run
end_cmd=cmd	Execute command or script at the end of the last run
compression=nn	Specify the compression percentage of the data pattern used for writes to each SD. Not used for file system workloads.
port=nn	Override the Java socket port number .
swat=(xx,yy)	Call Swat to create performance charts .
formatxfersize=	Specify xfersize used when creating or expanding an SD file.
create_anchors=yes	Create parent directories for FSD anchor.
report=host_detail report=slave_detail	Specifies which SD detail reports to generate. Default is SD total only.
histogram=(default,"x,y")	For file system testing. Experimental. Needs feedback. Default: histogram=(default,"1-2147483647,d")
validate=yes	(-vt) Activate Data Validation . Options can be combined: validate=(x,y,z)
validate=read_after_write	(-vr) Re-reads a data block immediately after it was written.

validate=no preread	(-vw) Do not read before rewrite.
validate=time	(-vt) keep track of each write timestamp (memory intensive)
journal=yes	Activate Data Validation and Journaling :
journal=recover	Recover existing journal , validate data and run workload
journal=only	Recover existing journal, validate data but do not run requested workload.
journal=noflush	Use asynchronous I/O on journal files

1.9.2.2 Host Definition (HD) Parameter overview

These parameters are ONLY needed when running Vdbench in a multi-host environment or if you want to override the number of JVMs used in a single-host environment.

See also [Host Definition parameter detail](#).

hd=default	Sets defaults for all HDs that are entered later
hd=localhost	Sets values for the current host
hd=host_label	Specify a host label.
system=host_name	Host IP address or network name , e.g. xyz.customer.com
vdbench=vdbench_dir_name	Where to find Vdbench on a remote host if different from current.
jvms=nnn	How many slaves to use. See Multi JVM execution .
shell=rsh ssh vdbench	How to start Vdbench on a remote system.
user=xxxx	Userid on remote system Required.
clients=nn	This host will simulate the running of multiple 'clients'. Very useful if you want to simulate numerous clients for file servers without having all the hardware.
mount="mount xxx ..."	This mount command is issued on the target host after the possibly needed mount directories have been created.

1.9.2.3 Replay Group (RG) Parameter Overview

See also [Swat I/O Trace Replay](#).

rg=name	Unique name for this Replay Group (RG).
devices=(xxx,yyy,...)	The device numbers from Swat's flatfile.bin.gz to be replayed.

Example: rg=group1,devices=(89465200,6568108,110)

Note: Swat Trace Facility (STF) will create Replay parameters for you. Select the 'File' 'Create Replay parameter file' menu option.

1.9.2.4 Storage Definition (SD) Parameter Overview

See also [Storage Definition Parameter Detail](#).

This set of parameters identifies each physical or logical volume manager volume or file system file used in the requested workload. Of course, with a file system file, the file system takes the responsibility of all I/O: reads and writes can and will be cached (see also `openflags=`) and Vdbench will not have control over *physical* I/O. However, Vdbench can be used to test file system file performance (See also [File system testing](#)).

Example: `sd=sd1,lun=/dev/rdisk/c0t0d0s0,threads=8`

<code>sd=default</code>	Sets defaults for all SDs that are entered later.
<code>sd=name</code>	Unique name for this Storage Definition (SD).
<code>host=name</code>	Name of host where this SD can be found. Default 'localhost'
<code>lun=lun_name</code>	Name of raw disk or tape or file system file. Optional unless you want Vdbench to create a disk file for you.
<code>count=(nn,mm)</code>	Creates a sequence of SD parameters.
<code>size=nn</code>	Size of the raw disk or file to use for workload.
<code>range=(nn,mm)</code>	Use only a subset ' range=nn ': Limit Seek Range of this SD.
<code>threads=nn</code>	Maximum number of concurrent outstanding I/O's for this SD. Default 8
<code>hitarea=nn</code>	See read hit percentage for an explanation. Default 1m.
<code>journal=xxx</code>	Directory name for journal file for data validation
<code>replay=nnn</code>	Device number(s) to select for Swat I/O replay
<code>replay=(group,...)</code>	Replay Group(s) using this SD.
<code>resetlun=nnn</code>	Issue <code>ioctl</code> (USCSI RESET) every nnn seconds. Solaris only
<code>resetbus=nnn</code>	Issue <code>ioctl</code> (USCSI RESET_ALL) every nnn seconds. Solaris only
<code>offset=nnn</code>	At which offset in a lun to start i/o.
<code>align=nnn</code>	Generate logical byte address in ' nnn ' byte boundaries , not using default 'xfersize' boundaries.
<code>openflags=(flag,...)</code>	Pass specific flags when opening a lun or file

1.9.2.5 File system Definition (FSD) Parameter Overview

See [Filesystem Definition \(FSD\) parameter overview](#)

1.9.2.6 Workload Definition (WD) Parameter Overview

See also [Workload Definition Parameter Detail](#).

The Workload Definition parameters describe what kind of workload must be executed using the storage definitions entered.

Example: wd=wd1,sd=(sd1,sd2),rdpct=100,xfersize=4k

wd=default	Sets defaults for all WDs that are entered later.
wd=name	Unique name for this Workload Definition (WD)
sd=xx	Name(s) of Storage Definition(s) to use
host=host label	Which host to run this workload on. Default localhost.
rdpct=nn	Read percentage . Default 100.
rhpc=nn	Read hit percentage . Default 0.
whpc=nn	Write hit percentage . Default 0.
xfersize=nn	Data transfer size . Default 4k.
skew=nn	Percentage of skew that this workload receives from the total I/O rate.
seekpct=nn	Percentage of random seeks . Default seekpct=100 or seekpct=random.
range=(nn,nn)	Limit seek range to a defined range within an SD.
openflags=(flag,...)	Pass specific flags when opening a lun or file.
priority=nn	I/O priority to be used for this workload.
iorate=nn	Dedicated fixed I/O rate for this workload.

1.9.2.7 File system Workload Definition (FWD) Parameter Overview

See [Filesystem Workload Definition \(FWD\) parameter overview](#)

1.9.2.8 Run Definition (RD) Parameter Overview (For raw i/o testing)

See also [Run Definition Parameter Detail](#).

The Run Definition parameters define which of the earlier defined workloads need to be executed, what I/O rates need to be generated, and how long the workload will run. One Run Definition can result in multiple actual workloads, depending on the parameters used.

Example: rd=run1,wd=(wd1,wd2),iorate=1000,elapsed=60,interval=5

rd=default	Sets defaults for all RDs that are entered later.
rd=name	Unique name for this Run Definition (RD).
wd=xx	Workload Definitions to use for this run.
sd=xxx	Which SDs to use for this run (Optional).
iorate=(nn,nn,nn,...)	One or more I/O rates .
iorate=curve	Create a performance curve .
iorate=max	Run an uncontrolled workload.
curve=(nn,nn,...)	Data points to generate when creating a performance curve.
elapsed=nn	Elapsed time for this run in seconds. Default 30 seconds.
interval=nn	Reporting interval in seconds. Default 'min(elapsed/2,60)'
warmup=nn	Override warmup period .
distribution=x	I/O arrival calculations: exponential, uniform, or deterministic.
pause=nn	Sleep 'nn' seconds before starting next run.
forxfersize=nn	Multiple runs for each data transfer size .
forthreads=nn	Multiple runs for each read thread count .
forrdpct=nn	Multiple runs for each read percentage .
forhpct=nn	Multiple runs for each read hit percentage .
forwhpct=nn	Multiple runs for each write hit percentage .
forseekpct=nn	Multiple runs for each seek percentage .
forhitarea=nn	Multiple runs for each hit area size .
forcompression=nn	Multiple runs for each compression percentage.
replay=filename	File name used for Swat I/O replay (file 'flatfile.bin' from Swat)
start cmd=cmd	Execute command or script at the beginning of the first run
end cmd=cmd	Execute command or script at the end of the last run
openflags=xxxx	Pass specific flags when opening a lun or file

1.10 Report files

HTML files are written to the directory specified using the '-o' execution parameter.

These reports are all linked together from one starting point. Use your favorite browser and point at 'summary.html'.

The following ([see the report file examples](#)) are created.

summary.html	Contains workload results for each run and interval. Summary.html also
------------------------------	--

	contains a link to all other html files, and should be used as a starting point when using your browser for viewing. For file system testing see summary.html for file system testing From a command prompt in windows just enter 'start summary.html'; on a unix system, just enter 'firefox summary.html &'.
<i>hostx.summary.html</i>	Identical to summary.html, but containing results for only one specific host. This report will be identical to summary.html when not used in a multi-host environment.
<i>hostx-n.summary.html</i>	Summary for one specific slave.
logfile.html	Contains a copy of most messages displayed on the console window, including several messages needed for debugging.
<i>hostx_n.stdout.html</i>	Contains logfile-type information for one specific slave.
<i>parmfile.html</i>	Contains a copy of the parameter file(s) from the '-f parmfile' execution parameter.
<i>parmscan.html</i>	Contains a running trail of what parameter data is currently being parsed. If a parsing or parameter error is given this file will show you the latest parameter that was being parsed.
sdbname.html	Contains performance data for each defined Storage Definition.
<i>hostx.sdbname.html</i>	Identical to sdbname.html, but containing results for only one specific host. This report will be identical to sdbname.html when not used in a multi-host environment. This report is only created when the 'report=host detail' parameter is used.
<i>hostx_n.sdbname.html</i>	SD report for one specific slave. . This report is only created when the 'report=slave detail' parameter is used.
kstat.html	Contains Kstat summary performance data for Solaris
<i>hostx.kstat.html</i>	Kstat summary report for one specific host. This report will be identical to kstat.html when not used in a multi-host environment.
<i>host_x.instance.html</i>	Contains Kstat device detailed performance data for each Kstat 'instance'.
nfs3/4.html	Solaris only: Detailed NFS statistics per interval similar to the nfsstat command output.
flatfile.html	A file containing detail statistics to be used for extraction and input for other reporting tools. See also Parse Vdbench flatfile
<i>errorlog.html</i>	Any i/o errors or Data Validation errors will be written here. This file serves as input to the './Vdbench dvpost' post-processing utility .
<i>swat_mon.txt</i>	This file can be imported into the Swat Performance Monitor allowing you to display performance charts of a Vdbench run.
<i>messages.html</i>	For Solaris only. At the end of a run the last 500 lines from /var/adm/messages are copied here. These messages can be useful when certain i/o errors or timeout messages have been displayed.
<i>fwdx.html</i>	A detailed report for each File system Workload Definition (FWD).
<i>wdx.html</i>	A separate workload report is generated for each Workload Definition (WD) when more than one workload has been specified.
histogram.html	For file system workloads only. A response time histogram reporting

	response time details of all requested FWD operations.
<i>fsdx.histogram.html</i>	A response time histogram for each FSD.
<i>fwdx.histogram</i>	A response time histogram for each FWD.

1.11 Execution Parameter Detail

1.11.1'-f xxx ': Workload Parameter File(s)

The workload parameter file(s) contains all the workload parameters.

Vdbench workload parameters can also be passed as command line parameters. All command line input entered after the first blank separated 'hyphen' (" - ") will be used as a replacement for the workload parameter file. Since parentheses are special characters in Unix shells, (signifying a sub shell) you must either escape parentheses or enclose parameters in either single or double quotes; e.g., vdbench -o output - "sd=... wd=... rd=.."

Command line parameters are used as a *replacement* for the '-f parmfile' option, not an addition.

There are five groups of parameters in the file: General (optional), Host Definition (**HD**) (optional), Storage Definition (**SD**), Workload Definition (**WD**), and Run Definition (**RD**). For File system testing this will be General (optional), Host Definition (**HD**) (optional), File System Definition (**FSD**), File system Workload Definition (**FWD**), and Run Definition (**RD**). These groups must be entered in the order defined here.

Each parameter has a keyword followed by one or more sub parameters. Most keywords (and alphanumeric sub parameter) can be abbreviated to its shortest unique value with a minimum of two characters. For example xfersize=512 can be abbreviated to xf=512. Sub parameters can be coded with a single value 'iorate=1000', or with multiple values 'iorate=(100,200,300)'. Multiple values must always be enclosed within parentheses. A set of sub parameters must be either numeric, or alphanumeric, not a mix. Not all keywords accept multiple sub parameters, but the documentation will make clear which parameters will accept them. Keywords may be entered in mixed case: e.g. 'Xfersize=4k'. When using embedded blanks or other special characters ('-' or '=') you must encapsulate the parameters in double quotes.

Numeric parameters allow definition in (k)ilobytes, (m)egabytes, (g)igabytes and (t)erabytes. k/m/g/t may be specified in lower or upper case. One kilobyte equals 1024 bytes. Time values may also be entered as minutes or hours; e.g., 'elapsed=7200 is equivalent to 'elapsed=120m' or 'elapsed=2h'.

Multiple numeric values can be entered as follows:

- keyword=(1,2,3,4,5,6,7,8,9,10,..)	Individual values
- keyword=(1-10,1)	Range, from 1 to 10, incremented by 1 (1,2,3,4,5,6,7,8,9,10)
- keyword=(1-64,d)	Doubles: from 1 to 64, each successive value doubled (1,2,4,8,16,32,64)
- keyword=(64,1,d)	Reverse double (divide by two)

A detailed parameter scan report is written to *output/parmscan.html*. When there are problems with the interpretation of the keywords and sub parameters, looking at this file can be very helpful because it shows the last parameter that was read and interpreted. A complete copy of the input parameters is written to *output/parmfile.html*. This is done so that when you look at a Vdbench output directory you will see exactly what workload was executed -- no more guessing trying to remember 'what did I run 6 months ago'.

Comments: a line starting with '/', '#' or '*' and anything following the first blank on a line is considered a comment. Also, a line beginning with 'eof' is treated as end of file, so whatever is beyond that will be ignored.

Continuation: If a line gets too large you may continue the parameters on the next line by ending the line with a comma and a blank and then start the next line with a new keyword.

Vdbench allows for the specification of multiple parameter files. This allows for instance the separation of SD parameters from WD and RD parameters. Imagine running the same workload on different storage configurations. You can then create one WD and RD parameter file, and multiple SD parameter files, running it as follows: "vdbench -f sd_parmfile wd_rd_parmfile".

include=/parm/file/name

When this parameter is found, the contents of the file name specified will be copied in place.

Example: include=/complicated/workload/definitions.parmfile

1.11.2'-xxxx': Output Directory

Vdbench writes all its HTML files to this output directory. The directory will be created if it does not exist. An already existing directory will be reused after first deleting all existing HTML files. Reused directories can contain HTML files that were placed there by an earlier execution of Vdbench, and may not be related in any way to the new execution. Leaving these files around could cause confusion, so the old HTML files are deleted.

If you do not want to reuse an output directory, you may add a '+' after the directory name: e.g. '-o dirname+'. If 'dirname' does not exist it will be created. If it *does* exist, the directory name is incremented by one to 'dirname001', and if that directory name is available it is created. And so on until 'dirname999', after which Vdbench will stop.

You may also request that a timestamp be added to the output directory name: '-o output.tod' will result in a directory named 'output.yymmdd.hhmmss'.

This dynamic creation of new output directory names is very useful if you don't want to accidentally overwrite this very important test that just took you 24 hours. ☺

Default: '-f output'.

1.11.3'-v': Activate Data Validation

This execution parameter activates Data Validation. Each write of a block will be recorded, and after the next read to the same block, the block's old contents will be validated. The next write to the block will cause the block to be read first and then validated.

Option '-vr' can be used to do a read and validate immediately after each write, versus normally only validating when a block of data is scheduled for the next read or the next write. When doing I/O against a large LUN it can normally take quite a while before a block is referenced again.

So, at times this may be useful to get a quick confirmation that the data is correct.

Be aware, however, that reading a block immediately after a write likely will only show that the data reached the controller cache and there is no proof that the data ever reached the physical disk drives.

Option '-vt' will save the timestamp of the last successful read or write in memory (no journaling available). When that data block fails this timestamp will be reported. Knowing the time of day that the block was good can help you identify which error injection might have caused the problem. Beware: this requires 8 more bytes of memory per data block (memory needs for 512 byte blocks could therefore be prohibitive).

See [Data Validation and Journaling](#) for a more detailed description of data validation.

Data Validation can also be activated using the 'validate=yes' parameter in the parameter file.

1.11.4'-j': Activate Data Validation and Journaling

Journaling allows data validation to continue after Vdbench or the operating system terminates.

'-j' creates a new journal file or overwrites an existing one. Specify '-jr' to recover an existing journal. '-jn' prevents a flush to disk on journal writes (by default, each journal write is flushed directly to disk(synchronous i/o)). However, be aware that if the operating system terminates without a proper shutdown, the unflushed journal file may be incomplete.

See [Data Validation and Journaling](#) for a more detailed description of data validation.

Journaling can also be activated using the 'journal=yes' parameter in the parameter file.

1.11.5'-s': Simulate Execution

Vdbench can create large and complex workloads. A simulation run scans and interprets the parameter file(s), but does not execute the workloads.

The output directory name specified with the '-o' parameter will have '.simulate' suffixed to it.

1.11.6'-k': Kstat Statistics on Console

On Solaris systems, kstat performance statistics are reported to *kstat.html*. To allow these statistics also to be written to the active console window, specify '-k'.

Vdbench does its utmost to match the requested LUN and/or file names with correct Kstat information. Veritas VxVm, QFS, SVM, and ZFS are supported, but there are situations where

Vdbench has some problems. When Vdbench fails to find the correct Kstat information, execution continues, but without using Kstat.

Solaris device names (/dev/rdisk/cxtxdx) are translated to Kstat instance names using the output of iostat. Output of 'iostat-xd' is matched with output of 'iostat -xdn', and the device and instance names are taken from there. If a device name cannot be translated to the proper Kstat instance name this way there is possibly a bug in Solaris that needs to be resolved.

1.11.7'-m nn': Multi JVM Execution

Depending on the processor speed, there is a maximum number of IOPS or maximum thread count that a single Java Virtual Machine (JVM) can handle. Since Java runs as a single process, it is bound by what a single process can do. To alleviate this problem, Vdbench starts an extra copy of itself (called a slave) for each requested 5000. The default 5000 was set several years ago when testing with 300 MHz or slower systems. The newer faster systems can handle many more iops per JVM.

The maximum number of JVMs started this way is limited by the number of processors (Solaris only) and by the number specified with the '-m nn' parameter or 'hd=hostname,jvms=nn', of which the default is 8.

Each workload is executed on each JVM or slave, except for sequential workloads. Running sequential workloads on each slave would result in the same sequential blocks being read by each slave, something that makes for nice performance numbers, but that does not really represent an accurate sequential workload. Sequential workloads therefore are spread round robin over each available JVM/slave.

The JVM count can also be set (and that is the preferred method) using the hd=hostname,jvms=nn parameter.

1.11.8'-t': Sample Vdbench execution.

When running './vdbench -t' Vdbench will run a hard-coded sample run. A small temporary file is created and a 50/50 read/write test is executed for just five seconds.

This is a great way to test that Vdbench has been correctly installed and works for the current OS platform without the need to manually create a parameter file.

1.11.9'-e nn' Override elapsed time

This parameter can be used to temporarily override the value of any elapsed= parameters specified in the parameter file. This can be very useful to quickly discover problems. For instance, you just created a 24hour test run, including multiple Run Definitions (RDs). If there is a problem you don't want to find that out after hours and hours of running. Just run the test with only a few seconds or minutes of elapsed time to see how things work for you.

1.11.10 '-l nn' Override report interval time.

This overrides all interval= values specified in the parameter file. See also '-e nn' above.

1.11.11 '-w nn' Override warmup time.

This overrides all warmup= values specified in the parameter file. See also '-e nn' above.

1.12 Vdbench utility functions.

Vdbench has several small utility functions to help you with your day-to-day Vdbench testing. Just enter ‘./vdbench xxx’, for instance ./vdbench sds’.

1.12.1.1sds: Generate Vdbench SD parameters.

Sick and tired of entering 200 50+ hexadecimal Solaris device names without any typos? Run ./vdbench sds and Vdbench will do it for you.

See [Vdbench SD parameter generation tool](#). This works for Solaris, Linux and Windows,

1.12.1.2dvpost: Data Validation post processing

Running ‘./vdbench dvpost’ brings up a GUI that allows you to zoom in on the Data Validation errorlog.html file. This allows you to ‘quickly’ skim through possibly thousands and thousands of lines of data generated when a data corruption is recognized by Vdbench.

1.12.1.3jstack: Display java execution stacks of active Vdbench runs.

Like all software, there always is a chance that there is a problem or bug somewhere that has not been anticipated. Code hangs are very difficult to fix if you don’t know where the problem is. ‘./vdbench jstack’ will print out the active Java execution stack of all currently running Java programs. Run this before killing Vdbench.

Must be run using the same user ID used for Vdbench.

1.12.1.4rsh: Vdbench RSH daemon.

Not every OS has an RSH daemon available (windows for instance) for testing, and some times getting the available RSH or SSH to do what you want just does not work.

For those situations Vdbench has his own (primitive) RSH daemon. It only works for Vdbench. Just run ./vdbench rsh’ once on the target system, and Vdbench will open a socket that will be used to start Vdbench slaves on that host and return its stdout and stderr output.

1.12.1.5print: Print any block on any lun or file.

Especially when running into data corruption issues identified by Data Validation being able to see what the current contents is of a data block can be very useful.

Syntax: `./vdbench print device lba xfersize [-q]`

device: any device or file name.
 lba: logical byte address. May be prefixed with 0x if this is hexadecimal. You may also specify k or m for kilobytes or megabytes.
 xfersize: length of block to print.

While printing Vdbench also does a quick Data Validation on the block being printed and reports differences found. To suppress this, add '-q' (quiet) as an extra execution parameter.

1.12.1.6edit: Simple full screen editor, or 'back to the future'.

It was around 1978 that I went away from line editors, never expecting to have to go back there. When starting to work with Unix systems back in 2000 all I found was vi. By that time I had lost all my Neanderthal habits, so vi just wasn't the way to go for me. It took me less than half an hour to write a full screen editor using Java, and here it is: Just run `./vdbench edit /file/name` and the convenience of the 21st century will be with you.☺

1.12.1.7compare: Compare Vdbench test results.

This function compares two sets of Vdbench output directories and shows the delta iops and response time and optionally the data rate in 9 different colors: light green is good, dark green is better, red is bad, etc. See also: [Vdbench Workload Compare](#).

1.12.1.8parse: Parse Vdbench flatfile.

The Vdbench flatfile parser is a simple program that takes the flatfile, picks out the columns and rows that the user wants, and then writes it to a tab delimited file. See [Vdbench flatfile selective parsing](#).

1.12.1.9gui: Vdbench demonstration GUI.

A simple GUI is available to demonstrate the use of Vdbench raw I/O workloads using SD, WD and RD parameters. This GUI supports only a very small subset of what Vdbench can do. See also [Vdbench GUI documentation](#).
 The fastest way to get some useful parameter files is by looking inside of the `/examples/` directory.

1.13 General Parameter Detail

1.13.1 'include=parmfile'

As specified earlier, there is a specific order in which these parameters must be specified, with the 'General Parameters' at the beginning of the parameter file.

There is however ONE exception: the 'include=*parmfile*' parameter. This can reside anywhere. It behaves like the good old fashioned #include statement for C code.

The file name specified will be inserted in-line into what is currently being read. You can use as many include= parameters needed.

One use of this parameter could be if you want to keep the SD or FSD parameters separated from the rest of the parameters file. SDs and FSDs typically change frequently, while the rest of the parameters stay unchanged.

A different use could be for instance if you have created a fixed, complex 'Application X' set of workloads and you just want to include this existing workload in a new test run.

For instance include=payroll or include=email.

If you code only a file name and not a directory name, Vdbench will look for the file name in the directory of the file containing the current 'include=' statement.

1.13.2 'data_errors=xxx': Terminate After Data Validation or I/O Errors

Vdbench by default will abort after 50 data validation or read/write errors. If one or more but less than the specified amount of errors occur, Vdbench at the end of the run (elapsed=) will abort.

There are three ways to define the error count:

data_errors=nn	Causes an abort after nn validation errors. Default 50
data_errors="script_name"	Causes this command/script to be executed after the first error, followed by an abort. This command can be used for diagnostic data collection and/or system dumps.
data_errors=(nn,mm)	Terminates the Vdbench run after 'nn' errors, or 'mm' seconds after the last error. This gives Vdbench the chance to report more errors, but eliminates the possibility that Vdbench keeps running unnoticed for a long periods of time after an error.

The command syntax under 'data_errors=script_name' can be as follows:
 data_errors= "script_name \$output \$lun \$lba \$size". After the error, the script is called with substituted values for 'output directory name', 'LUN name', 'lba', and data 'transfer size'. This allows for quicker collection of diagnostic data.

1.13.3"force_error_after': Simulate Data Validation Error

The 'force_error_after=(nn,mm)' option simulates a data validation error after 'nn' successful read operations. This option allows you for instance to test the 'data_errors=script_name' option. This option can be also very useful to see what happens when a real Data Validation error occurs.

Vdbench will by default simulate only one error, but you may override this by using a second 'mm' parameter: force_error_after=(500,10).

1.13.4'swat=xxx': Create performance charts using Swat.

This is only available for SD/WD raw I/O workloads.

Swat (Sun StorageTek™ Workload Analysis Tool) has a batch utility that allows you to automatically create a set of JPG files of performance charts.

Vdbench will automatically call that batch utility when you add the following parameters in the parameter file:

swat=(*where,which*)

where: the directory where Swat is installed

which: a file containing the proper parameters for the Swat batch utility, with a default of 'swatcharts.txt' which contains the parameters needed to create a Basic Performance chart, Response Time chart, and an IOPS chart.

These JPG files will be placed in subdirectory 'charts' in the Vdbench output directory. There will also be a link in the summary.html file to all these charts.

1.13.5'start_cmd=' and 'end_cmd=' command or script at the beginning/end of Vdbench or beginning/end of each run.

There are two places where you may use this parameter: Either as a general parameter specified at the beginning of a parameter file, or as part of a Run Definition (RD).

When used as a general parameter, 'start_cmd=cmd': 'cmd' will be executed right before the first run; 'end_cmd=cmd': 'cmd' will be executed immediately after the last run is finished.

When used as a Run Definition parameter, , 'start_cmd=*cmd*': '*cmd*' will be executed right before each run; 'end_cmd=*cmd*': '*cmd*' will be executed immediately after each run is finished.

These commands or scripts can be used for anything you like. For example, 'uname -a' and 'psrinfo' come to mind.

A string containing '\$output' in the requested command will be replaced by the output directory name requested using the '-o' execution parameter, allowing the command/script access to this directory name.

When starting Vdbench, the command 'config.sh *output.directory*' will always be executed. This allows any kind of preprocessing to be done, or in the default case, the gathering of configuration specific data. After this, 'my_config.sh *output.directory*' will be executed. 'my_config.sh' is there for you to use, and will not be replaced by a (re)installation of Vdbench.

If you never want either of these scripts to be executed, create file 'noconfig' in the Vdbench directory.

Note: Sometimes, depending on which system you are running on, config.sh can take a few seconds or some times minutes. If you don't need this to run, simply create file 'noconfig' in the Vdbench directory.

Note: These commands will only be executed on the localhost; the host where the Master is running. At this time there are no provisions to run this on a remote host.

1.13.6'patterns=dirname': Data Patterns to be Used

By default, each block written contains within each 32-bit word the value of its relative word offset (0,1,2,3,4,5,...,nnnn).

To request a different data pattern, specify 'patterns=*dirname*', where *dirname* is a directory that contains file 'default'. The content of this 'default' file is placed in the write buffer. If the pattern file does not contain enough data to fill the write buffer, the contents of the pattern file will be repeated until the buffer is full.

Data Validation and compression generate their own data pattern.

This parameter is only used for SD/WD workloads.

In earlier versions of Vdbench the patterns directory could contain data patterns for up to 126 Data Validation keys. That is no longer supported. Only the 'default' data pattern is used.

1.13.7'compression=nn': Set compression for data patterns

Normally for disk operations it does not matter what is being read or written. For tape it can have a great impact on the performance. The default data pattern written by Vdbench compresses very well, which will impact the usefulness of the performance test. 'compression=nn' sets the percentage of data that is left after compression, for instance 'compression=20' shows that on average 100 bytes are compressed down to 20.

Understanding that there are different compression algorithms this parameter cannot guarantee the ultimate results. The data patterns implemented are based on the use of the 'gzip' utility, and with that in mind the accuracy for this implementation is plus or minus 5%.

You may experiment with this, writing to some disk files and then compressing them with gzip.

If you want a higher accuracy for the compression, use the 'patterns=' parameter above. Just give Vdbench byte-for-byte and Vdbench will write it to your disk.

This parameter may be overridden using the 'forcompression=' parameter.

This parameter is only used for SD/WD workloads.

1.13.8 'port=nnnn': Specify port number for Java sockets.

Vdbench communicates between the master and slave JVMs using Java sockets, using port 5570. If on your system this port is already used by something else you may override this setting using the 'port=nnn' parameter.

Since with Vdbench you can create and execute as many different workloads as you can think of, running more than one Vdbench run at the same time should not be necessary. However, if you choose to do so your Vdbench could run into connection problems if it tries to connect to the same port that another Vdbench execution already is using. To eliminate this risk, Vdbench will increment the port number by one when it gets a connection failure to see if that connection will be successful. Vdbench will do this a maximum of ten times before it gives up. The end result is that it will try to use ports 5570 through 5579 (default).

You can also use the '-p nnnn' execution parameter to override the port number, or override the used port numbers permanently using the [portnumbers.txt](#) file.

1.13.9 'create_anchors=yes': Create anchor parent directory

The anchor directory for a File System Definition (FSD) is automatically created if it does not exist. However, its parent directories will NOT be created. Use 'create_anchors=yes' to also include the creation of the parent directories.

1.13.10 'report=': Generate extra SD reports.

By default Vdbench will create a separate report with detailed statistics for each SD. Starting Vdbench 5.02 Vdbench will no longer create SD reports for each slave or host (200 SDs over 8 slaves plus one host creates 1800 reports!). To still create these detail reports specify report=host_detail or report=slave_detail or abbreviated report=(host,slave) depending on your needs.

1.13.11 'histogram=': set bucket count and bucket size for response time histograms.

This option was created for file system testing to help figure out what portion of the generated I/O came from cache or from disk. The code is all there and works fine, but I need to see some real test results before I consider this 'done'.

Though I am considering allowing multiple different histograms to be specified, for instance one for FSDs, one for SDs, etc, for now I only have one coded. The proper feedback will help me figure out what the next step should be.

Default: histogram=(default,"1-2147483647,d") *(Make sure you use the double quotes here)*

The syntax for the bucket count and ranges is close to the values that you can specify for most of the *forxxx* parameters: Specific numeric, "10,20,30,40", incrementing ranges "10-100,10", or doubling ranges "10-1000,d", or a combination of all of these for instance:

histogram=(default,"1-10,1,11-100,10,101-1000,d")

1.13.12 'formatxfersize=nnnn'

When Vdbench creates new or expands existing disk files specified using SD parameters the default is xfersize=64k. To override this default, specify formatxfersize=nnn.

For file names specified using FSD parameters, use the ['fwd=format' parameter](#).

1.14 Replay Group (RG) Parameter Overview

rg=name	Unique name for this Replay Group (RG).
devices=(xxx,yyy,...)	The device numbers from Swat's flatfile.bin.gz to be replayed.

A Replay Group is a group of devices obtained from Swat whose I/O workload must be replayed on one of more SDs. Vdbench will obtain the maximum lba used for each device from flatfile.bin.gz, and will place them on the target luns, straddling luns if needed.

Example:

```
rg=group1,devices=(89465200,6568108,110)
```

```
rg=group2,devices=(200,300)
```

```
sd=sd1,lun=/dev/rdisk/cctxdxsx,replay=group1
```

```
sd=sd2,lun=/dev/rdisk/cytydysy,replay=group1
```

```
sd=sd3,lun=/dev/rdisk/cztzdysz,replay=group2
```

These Replay Groups make Vdbench work like a primitive volume manager.

A Vdbench replay can also be created by Swat Trace Facility (STF using the 'File' 'Create replay parameter file' menu option.

1.15 Host Definition parameter detail

1.15.1 'hd=host_label'

The host label is used for cross-referencing and reporting.

Example: `hd=localhost` or `hd=systemA`

Use 'localhost' when you want to set values for the *current* host.

`hd=default` specifies the default settings to be used for all later HD parameters.

Note that the contents of this parameter have changed since Vdbench 5.00 and 5.01. The old contents however will be accepted for a while to prevent version compatibility issues.

For a complete example, [see example 5](#) below.

1.15.2 'system=system_name'

When running Vdbench in multi-host mode you specify here the system name of the system's IP address, e.g. `system=x.y.z.com` or `system=12.34.56.78`

1.15.3 'jvms=nnn'

This parameter tells Vdbench how many JVMs to use on this host. See '[Multi JVM Execution](#)'.

1.15.4 'vdbench=vdbench_dir_name'

This tells Vdbench where it can find its installation directory on a remote host. Default: the same directory as currently used. Use double quotes (") when a directory name has embedded blanks, for instance on windows systems).

1.15.5 'shell=rsh | ssh | vdbench'

For multi-host execution Vdbench by default uses RSH. You can optionally use SSH. If your target system does not have RSH or if you can't get the proper settings on the local or remote systems to get RSH or SSH to work you can use Vdbench's own RSH daemon by specifying `shell=vdbench`. On the target remote system you must do a one time start of '`./vdbench rsh`' to start the [Vdbench RSH daemon](#). This daemon of course will only work with Vdbench, and it is simply a small program that uses Java sockets to start a command and receive stdout/stderr output back. See also '[portnumbers.txt](#)'.

Note: If you have the MKS toolkit installed on your Windows system you may want to remove the MKS version of RSH. Experience has shown that the stdout and stderr output streams created

by this version of RSH do not close properly therefore preventing Vdbench from recognizing the completion of a remote copy of Vdbench.

1.15.6 'user=xxxx'

Used for RSH and SSH. Note: it is the user's responsibility to properly define the RSH or SSH security settings on the local and on the remote hosts.

1.15.7 'mount=xxx'

This parameter is mainly useful when doing some serious multi-host testing. If you for instance have 20 target clients that you have connected all to the same file system it is nice not to have to manually create all these mount points and issue the mount commands.

The 'mount=' parameter can be used in two places:

- As part of a Host Definition (HD).
- As part of a Run Definition (RD).

When used as a Host Definition parameter you specify the complete mount command that you want issued on the remote system, e.g.:

```
mount="mount -o forcedirectio /dev/dsk/c2t6d0s0 /export/h01"
```

Vdbench will create the mount point directory if needed, in this example /export/h01 and then will issue the mount command.

When used as a Run Definition parameter (RD), you only specify the mount options, e.g.

```
mount="-o noforcedirectio".
```

Vdbench will replace the (possible) mount options as specified as part of the Host Definition with the newly specified mount options.

When you code 'mount=reset', the original mount command as specified will be executed.

Note, that for normal file system testing operations, each host will need his own FSD parameter, unless the 'shared=yes' FSD parameter is used in which case all hosts can use the same.

1.16 Storage Definition Parameter Detail

1.16.1'sd=name': Storage Definition Name

'sd=' uniquely identifies each Storage Definition. The SD name is used by the Workload Definition (WD) and Run Definition (RD) parameters to identify which SDs to use for its workload.

When you specify 'default' as the SD name, the values entered will be used as default for all SD parameters that follow.

The 'name' can contain any free-format name, special characters not allowed.

1.16.2'lun=lun_name': LUN or File Name

'lun=' describes the name of the raw disk or the file name of the file system file to use. Be careful that you do not specify any disk that contains data that you do not want to lose. Been there, done that ☺. This is the main reason why Vdbench does not require root access. You don't want to lose your root disk.

Please don't think, "I'll only be reading the customer's currently active production disks". One accidental rdpct= with a value different than 100 and all data will be gone. That has happened at least twice!

lun=/dev/rdisk/c0t0d0s0	Name of raw Solaris disk to be used for this SD.
lun=/dev/vx/rdisk/c0t0d0s0	Name of a raw VXVM volume
lun=/dev/rmt/x	Name of a Solaris tape drive
lun=/dev/rmt/,count=(0,5)	Name of Solaris tape drives 0-4.
lun=/home/dir/filename	Solaris file system filename. No control over physical I/O guaranteed since the file system may use system cache.
lun=\\.\d:	Name of raw mounted Windows disk to be used for this SD.
lun=\\.\PhysicalDrive1	Physical number of raw Windows disk to be used for this SD.
lun=c:\temp\filename	Windows: file system file name.
lun=\\.\c:\temp\filename	Windows: raw access to file.
lun=\\.\Tape0	Windows: tape drive 0.

The raw disk or the file system file will be opened for input only unless 'rdpct=' is specified with any value other than 100 (default). This allows Vdbench to execute with read-only access to disks or files.

Block zero will never be accessed to prevent the volume label from being overwritten. This is accomplished by never allowing Vdbench to generate a seek address of zero, no matter how large the data transfer size is. This also implies that block zero will never be read.

Note: a request to read a tape, both real and virtual, will only be accepted if the tape is written earlier in the same Vdbench execution. This is necessary so that Vdbench knows how many blocks have been written and with that information then can properly identify EOT.

Note: some volume managers will return a NULL block each time that a block is read that has never been written. Though of course this gives wonderful performance numbers the disk is never read and this therefore is NOT a valid workload. When you have a volume manager like this, make sure that you first pre-format the whole volume using:

```
sd=sd1,lun=xx
```

```
wd=wd1,sd=*,xf=1m,seekpct=eof,rdpct=0
```

```
rd=rd1,wd=*,iorate=max,elapsed=100h,interval=60
```

(The elapsed time must be large enough to completely format the volume. Vdbench will stop when done, but if the elapsed time is too short Vdbench may terminate too early before the volume has been completely formatted).

1.16.3 'host=name'

This parameter is only needed when you do a multi-host run where the lun names on each host are different. For instance if a lun is /dev/rdisk/a on hosta but it is named /dev/rdisk/b on hostb then you'll have to tell Vdbench about it.

The 'lun' and 'host' parameters in this case have to be entered in pairs, connecting a lun name to a host name, e.g.:

```
sd=sd1,lun=/dev/rdisk/a,host=hosta,lun=/dev/rdisk/b,host=hostb
```

By default Vdbench assumes that the lun names on each host are identical.

1.16.4 'count=(nn,mm)'

This parameter allows you to quickly create a sequence of SDs, e.g. sd=sd,lun=/dev/rmt/,count=(0,8) results in sd0-sd7 for /dev/rmt/0-7.

You may also specify an asterix:

```
sd=sd,lun=/dev/rmt/*cbn,count=(0,2)
```

This will result in devices 0cbn and 1cbn being selected.

1.16.5 'size=nn: Size of LUN or File'

'size=' describes the size of the raw disk or file. You can enter this in bytes, kilobytes, megabytes, gigabytes or terabytes (k/m/g/t). If not specified, the size will be taken from the raw disk or from the file. Vdbench supports addresses larger than 2GB.

If this is a non-existing file, or an existing file that is not large enough, a separate Vdbench run named 'File_format_or_append_for_sd=' is automatically executed that will do a sequential write or append to the file until the file is full. This replaces the need to create a new file using *mkfile* or other utility.

Note: to prevent accidentally creating a huge file in the /dev/rdisk/ directory because an incorrectly entered 50+ digit lun name, Vdbench will refuse to create new file names that start with /dev/. Bad things happen when your root directory fills up ☺

1.16.6'range=nn': Limit Seek Range

By default, the whole SD will be used. To limit the seek range for a workload, specify the starting and ending range of the SD: 'range=(40,60)' will limit I/O activity starting at 40% into the SD and ending at 60% into the SD. If the parameter value is larger than 100, values will be considered given in *bytes* instead of in *percentages*; e.g., 'range=(1g,2g)'.

1.16.7'threads=nn': Maximum Number of Concurrent outstanding I/Os

'threads=nn' specifies the maximum number of concurrent I/O's that can be outstanding for this SD. Be aware that depending on the storage subsystem, some of these I/Os may be queued by the operating system.

Warning: If you specify a LUN that physically consists of multiple disk drives, the thread count determines the maximum concurrency for the LUN, not for the disks. A total of 8 concurrent threads for a total of 16 physical disks will not allow for much concurrency. Also, for Solaris, make sure that your *sd_max_throttle* parameter in */etc/system* allows the requested amount of concurrency.

Note: be aware that the maximum concurrency will only occur when there is enough demand. Requesting 10 iops against a device that can handle 1000 iops will not give you the concurrency that you request with the thread= parameter.

This parameter may be overridden using the ['forthreads='](#) parameter.

1.16.8'hitarea=nn': Storage Size for Cache Hits

See [read hit percentage](#) for an explanation. Default value is 1MB.

1.16.9'journal=name': Directory Name for Journal File

Used with journaling only. Journal files are needed for each Storage Definition and are created by default in the current directory. The file names are 'sdname.jnl' and 'sdname.map', where 'sdname' is the name of the SD.

See [Data Validation and Journaling](#) for a more detailed description of data validation and journaling.

When as part of the Data Validation and journaling testing you bring down your OS it is imperative that all writes to the journal file are synchronous. If your OS or file system does not handle this properly you could end up with a corrupted journal file. A corrupted journal file means that the results will be unpredictable during journal recovery.

Journaling therefore now allows you to specify a RAW device, e.g. `journal=/dev/xxxx`, bypassing the possibly faulty file system code.

1.16.10 'offset=': Don't start at byte zero of a LUN

Vdbench always starts at the beginning of a LUN, but some times it is needed to modify that. Some times a LUN does not start at an exact physical stripe boundary and this parameter allows you do adjust for that. The offset is in bytes and must be a multiple of 512.

Note: Vdbench never accesses block zero on any SD. This has been done to make sure that it never overwrites a volume label and/or vtoc.

1.16.11 'align=': Determine lba boundary for random seeks.

Whenever Vdbench generates an LBA it by default is always on a block boundary (`xfersize=`). Use the 'align=' parameter to change that to always generate an LBA on a different alignment. The align= value is in bytes and must be a multiple of 512.

1.16.12 'openflags=': control over open and close of luns or files

(Solaris and Linux only) This parameter allows you to control what parameters are passed to the system's open and close functions. By default write operations are handled according to how the file system is mounted or for raw devices, how the device normally operates. This can mean that a write operation completes as soon as the data is stored in system cache. This makes for very good performance, but does not really exercise the storage.

Openflags can be specified for SD, WD, FSD, FWD, and RD parameters.

Options (you can create any combination of these)

Solaris:	(xx_SYNC descriptions found in man open)
o_dsync	Write I/O operations on the file descriptor complete as defined by synchronized I/O data integrity completion
o_rsync	Read I/O operations on the file descriptor complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in oflag, all I/O operations on the file descriptor complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in oflag, all I/O operations on the file descriptor complete as defined by synchronized I/O file integrity completion.
o_sync	Write I/O operations on the file descriptor complete as defined by synchronized I/O file integrity completion.
0x.....	Any hex value, to be passed to the open() function.
fsync	Call fsync() before the file is closed.
directio	Calls the directio() function after the file is opened, using 'DIRECTIO_ON'
directio_off	Calls the directio() function after the file is opened, using 'DIRECTIO_OFF'. This one is meant to be used if a previous failed I/O run left the target file name with directio active and you want to forcibly remove that status.
clear_cache	When using directio() for an NFS mounted file, any data still residing in file system cache will continue to be used, circumventing the directio() request. This option will forcibly clear any existing data from cache using mmap() functions.

Linux:	
o_direct	Gives you raw access to a full volume (no partition). This parameter is <i>required</i> when using /dev/xxx volumes
o_dsync o_rsync o_sync	These three all pass '0x010000' to the Linux open() function.
fsync	Call fsync() before the file is closed.

1.17 Workload Definition Parameter Detail

The Workload Definition parameters describe what kind of workload must be executed using the storage definitions entered. Note that a lot of these parameters can be overridden within a Run definition (RD) using ‘forxxx=’ parameters.

Example: wd=wd1,sd=(sd1,sd2),rdpct=100,xfersize=4k

wd=default	Sets defaults for all WDs that are entered later.
wd=name	Unique name for this Workload Definition (WD)
host=host_label	Specify here which host you want this to run on.
sd=xx	Name(s) of Storage Definition(s) to use
rdpct=nn	Read percentage . Default 100.
rhpc=nn	Read hit percentage . Default 0.
whpct=nn	Write hit percentage . Default 0.
xfersize=nn	Data transfer size . Default 4k.
xfersize=(nn,%%,...	Data transfer size distribution.
skew=nn	Percentage of skew that this workload receives from the total I/O rate.
seekpct=nn	Percentage of random seeks . Default seekpct=100 or seekpct=random.
range=(nn,nn)	Limit seek range to a defined range within an SD.
openflags=	See openflags=
iorate=nn	Workload specific I/O rate
priority=	Workload specific I/O priority .

1.17.1 'wd=name': Workload Definition Name

'wd=name' uniquely identifies each Workload Definition. The WD name is used by the Run Definition parameters to identify which workloads to execute. When you specify ‘default’ as the WD name, the values entered will be used as default for all WD parameters that follow.

1.17.2 'host=host_label'

This parameter is only needed for multi-host runs where you do not want each workload to run on each host. For example:

wd=wd1,host=hosta,....

wd=wd2,host=hostb,...

1.17.3'sd=name': SD names used in Workload

'sd=' selects a specific SD for this workload. A single SD name can be specified as 'sd=sd1', multiple SDs can be specified as either 'sd=(sd1,sd2,sd3,...)', a range as in 'sd=(sd1-sd99)', using a wildcard character 'sd=sd*', or a combination of these.

1.17.4'rdpct=nn': Read Percentage

'rdpct=' specifies the read percentage of the workload. rdpct=100 means 100% read; rdpct=0 means 100% write. rdpct=80 means a read/write ratio of 4:1, etc.

The default is 100% read. If there are no workloads for a specific SD with a read percentage other than 100 this SD is opened for input only.

This parameter may be overridden using the ['forrdpct='](#) parameter.

1.17.5'rhpc=nn' and 'whpc=nn': Read and Write Hit Percentage

'rhpc=' and 'whpc=' specify the cache hit percentage that Vdbench will attempt to generate. This parameter is only useful when accessing raw devices or file systems mounted with 'direct I/O' (or using 'openflags'). For this to work, each volume is divided into two parts: the first one-megabyte of storage will be accessed for cache hits and is called the **hit area**. The remaining space on the SD will be accessed for cache misses and is called the **miss area**.

When Vdbench needs to generate a cache hit, it generates an I/O to the hit area, assuming that the data accessed is, or soon will be, residing in cache. Cache misses will be targeted toward the miss area, assuming that the miss area is large enough to ensure that most random accesses are cache misses. As you can see, this will only be useful when Vdbench has control over which physical volume and physical blocks will be ultimately read or written (so no file system cache).

The ['hitarea=nn'](#) and ['forhitarea=nn'](#) parameters have been created to allow control over a volume's cache working set size. Some cached storage subsystems have different performance characteristics if too small a subset of the available cache is used. The total size of the LUN/SD must be at least 4 times the hitarea size.

These parameters may be overridden using the ['forrhpc='](#) or ['forwhpc='](#) parameters.

1.17.6'xfersize=nn': Data Transfer Size

'xfersize=' specifies how much data is transferred for each I/O operation; allows (k)ilo and (m)ega bytes.

Example: xfersize=4k (default)

You may also specify a distribution of data transfer sizes. Specify pairs of transfer size and percentages; the total of the percentages must add up to 100.

Example: xfersize=(4k,10,8k,10,16k,80)

This parameter may be overridden using the ['forxfersize'](#) parameter.

1.17.7'skew=nn': Percentage skew

'skew=' specifies the percentage of the run's total I/O rate that will be generated for this workload. By default the total I/O rate will be evenly divided among all workloads. However, if the skew value is nonzero, a percentage of the requested I/O rate equal to the percentage skew value will be apportioned to one workload, with the remaining skew evenly divided among the workloads that have no skew percentage specified. The total skew for all workloads used in a Run Definition must equal 100%.

Example: 5 workload definitions, one workload specifies 'skew=60'. This workload receives 60 percent of the requested I/O activity while the other 4 workloads each receive 10% of the requested I/O rate, with a total of 100%.

Vdbench generates I/O workloads by sending new I/O requests to a volume's (SD) internal work queue. This work queue has a maximum queue depth of 2000 per SD. If an SD cannot keep up with it's requested workload and the queue fills up, Vdbench will not be able to generate new I/O requests for this and all other SDs until space in the queue becomes available again. This means that if you send 1000 IOPS to an SD that can handle only 100 IOPS, and 50 IOPS to a similar device, the queue for the first device will fill up, and I/O request generation for the second device will be held up. This has been done to enable Vdbench to preserve the requested workload skew while still allowing for a temporary 'backlog' of requested I/Os.

To accommodate users that still would like to run an *uncontrolled* workload see ['iorate=max'](#).

1.17.8'sseekpct=nn': Percentage of Random Seeks

'seekpct=' specifies how often a seek to a random lba will be generated:

seekpct=100 or seekpct=random	Every I/O will go to a different random seek address.
seekpct=0 or seekpct=sequential	There will be NO random seeks, and the run will therefore be purely sequential. When the end of the volume or file is reached, Vdbench will continue at the beginning, unless 'seekpct=eof' (see below) is specified. Be aware that if the volume size is smaller than the cache size, continued processing will be all cache hits.
seekpct=20	On average, 20% of the I/O operations will start at a new random seek address. This means that on average there will be one random

	seek, with 5 consecutive blocks of data transferred.
seekpct=-1 or seekpct=eof	A negative <i>one</i> value causes a sequential workload to terminate as soon as end-of-file is reached. Vdbench will continue until all workload using seekpct=eof have reached EOF.

The randomizer used to generate a seek address is seeded using the host's time of day in micro seconds multiplied by the relative position of the Storage Definition (SD) defined for the workload (WG). For sequential processing, Vdbench always starts at the *second* block as defined by the data transfer size (block zero is never used).

This parameter may be overridden using the [‘forseekpct=’](#) parameter.

1.17.9‘range=nn’: Limit Seek Range

By default, the whole SD will be used. To limit the seek range for a workload, specify the starting and ending percentage of the SD: ‘range=(40,60)’ will limit I/O activity starting at 40% into the SD and ending at 60% into the SD. If the parameter value is larger than 100, values will be considered given in bytes instead of in percentages; e.g., ‘range=(1g,2g)’.

See also sd=xxx,range=

1.17.10‘iorate=’ Workload specific I/O rate.

Normally the I/O rate for a workload is controlled by the rd=xxx,iorate= parameter, together with the workload skew= parameter. With the workload specific iorate= parameter you can now give a FIXED I/O rate to a workload, while the other workloads continue to be controlled by the rd=xxx,iorate= and the workload skew parameters.

When used, ‘priority=’ must also be specified..

This option was initially created to test a ‘what if’: “If I run a Video On Demand (VOD) workload , what will the impact on performance be if I add some maintenance or video editing workload?”.

1.17.11‘priority=’ Workload specific I/O priority.

All I/O in Vdbench is scheduled using the expected I/O arrival times, which is obtained from the requested I/O rate. For instance, iorate=100 starts a new I/O on average every $1000/100 = 10$ milliseconds. There are no I/O priorities within devices or workloads.

The new ‘priority=’ parameter attempts to change this.

Normally I/O in Vdbench is handled using internal 'work' fifo queues. When the priority= parameter is used, any work in a higher priority fifo queue is processed before any lower priority fifo is checked causing I/O for that higher priority workload to be started first.

When any priority is specified, ALL workloads will have to have a priority specified. Priorities go from 1 (highest) to 'n' (lowest), and must be in sequence (priorities must be for instance 1 and 2, not 1 and 3).

Warning though: allowing for a specific workload iorate and priority does not guarantee that if you ask for 100 IOPS and your system can do only 50 that Vdbench magically gets the workload done ☺.

The iorate and priority parameters are brand-new. I'd love to get some feedback if you use these.

1.18 Run Definition for raw I/O parameter detail

For parameters specific to file system testing, see [RD parameters for file system testing, detail](#). The Run Definition parameters specify which of the earlier defined workloads need to be executed, which I/O rates need to be generated, and how long to run the workloads. One Run Definition can result in multiple actual workloads, depending on the parameters used.

Example: rd=run1,wd=(wd1,wd2),iorate=1000,elapsed=60,interval=5

rd=default	Sets defaults for all RDs that are entered later.
rd=name	Unique name for this Run Definition (RD).
wd=xx	Workload Definitions to use for this run.
sd=xxx	New with Vdbench502: specify which SDs to use .
iorate=(nn,nn,nn,...)	One or more I/O rates .
iorate=curve	Create a performance curve .
iorate=max	Run an uncontrolled workload.
curve=(nn,nn,...)	Data points to generate when creating a performance curve.
elapsed=nn	Elapsed time for this run in seconds. Default 30 seconds.
interval=nn	Reporting interval in seconds. Default 'min(elapsed/2,60)'
warmup=nn	Warmup time , excluding the first n intervals from run total.
distribution=x	I/O arrival calculations: Default exponential.
pause=nn	Sleep 'nn' seconds before starting next run.
(for)xfersize=nn	Multiple runs for each data transfer size .
(for)threads=nn	Multiple runs for each read thread count .
(for)rdpct=nn	Multiple runs for each read percentage .
(for)hpct=nn	Multiple runs for each read hit percentage .
(for)whpct=nn	Multiple runs for each write hit percentage .
(for)seekpct=nn	Multiple runs for each seek percentage .
(for)hitarea=nn	Multiple runs for each hit area size .
(for)compression=nn	Multiple runs for each compression percentage.
replay=filename	File name used for Swat I/O replay (file 'flatfile.bin' from Swat)

1.18.1 'rd=name': Run Name

'rd=name' defines a unique name for this run. Run names are used in output reports to identify which run is reported.

When you specify 'default' as the RD name, the values entered will be used as default for all SD parameters that follow.

1.18.2 'wd=': Names of Workloads to Run

'wd=' identifies workloads to run. Specify a single workload as 'wd=wd1' or multiple workloads either by entering them individually 'wd=(wd1,wd2,wd3)', a range 'wd=(wd1-wd3)' or by using a wildcard character: 'wd=wd*'.

The total [skew percentage](#) specified for all requested workloads must equal 100.

1.18.3 'sd=xxx'

Normally you specify the SDs to be used as part of a Workload Definition (WD) parameter. However, when you specify all workload parameters using the available 'forxx' RD options, the WD parameter really is not necessary at all, so you can now specify the SD parameters during the Run Definition.

You can specify both the WD and the SD parameters though. In that case the SD parameters specified here will override the SD parameters used when you defined the WD= workload. If you do not specify the WD parameter, all defaults as currently set for the Workload definition are used.

1.18.4 'iorate=nn': One or More I/O Rates

- iorate=100	Run a workload of 100 I/Os per second
- iorate=(100,200,...)	Run a workload of 100 I/Os per second, then 200, etc.
- iorate=(100-1000,100)	Run workloads with I/O rates from 100 to 1000, incremented by 100.
- iorate=curve	Run a performance curve. See below.
- iorate=max	Run the maximum uncontrolled I/O rate possible. See below.

'iorate=curve': Run a performance curve.

- When no skew is requested for any workload involved, determine the maximum possible I/O rate by running 'iorate=max'. After this 'max' run, the target skew for the requested workloads will be set to the skew that was observed. Workloads of 10%, 50%, 70%, 80%, 90% and 100% of the observed maximum I/O rate and skew will be run. Target I/O rates above 100 will be rounded up to 100; I/O rates below 100 will be rounded up to 10. Curve percentages can be overridden using the '[curve=](#)' parameter.
- When any skew is requested, do the same, but *within* the requested skew.

'iorate=max': Run the maximum I/O rate possible.

- When **no** skew is requested, allow each workload to run as fast as possible without controlling skew.

- When **any** skew is requested, allow all workloads to run as fast as possible *within* the requested skew.

During a Vdbench replay run, the I/O rate by default is set to the I/O rate as it is observed in the trace input. This I/O rate is reported on the console and in the file *logfile.html*.

If a different I/O rate is requested, the inter-arrival times of all the I/Os found in the trace will be adjusted to allow for the requested change.

1.18.5'curve=nn': Define Data points for Curve

The default data points for a performance curve are 10, 50, 70, 80, 90, and 100%. To change this, use the `curve=` parameter. Example: `curve=(10-100,10)` creates one data point for each 10% of the maximum I/O rate. Target I/O rates above 100 will be rounded up to 100; I/O rates below 100 will be rounded up to 10.

Remember: each data point takes 'elapsed=nn' seconds!

1.18.6'elapsed=nn': Elapsed Time

This parameter specifies the elapsed time in seconds for each run. This value needs to be at least twice the value of the reporting interval below. Each requested workload runs for 'elapsed=' seconds while detailed performance interval statistics are reported every 'interval=' seconds. At the end of a run, a total is reported for all intervals *except* for the first interval. The requirement to have at least 2 intervals allows us to have at least 1 reporting interval included in the workload total.

1.18.7'interval=nn': Reporting Interval

This parameter specifies the duration in seconds for each reporting interval. At the end of each reporting interval, all statistics that have been collected are reported.

Note: when doing very long runs for instance over the weekend, the amount of detail data reported can become overwhelming. A more reasonable interval duration may be appropriate, for instance 60 seconds. However, if you have a poorly performing storage device that never seems to be able to get to a stable workload I/O rate you may want to choose the lowest reporting interval possible. The longer your reporting interval, the more you will hide the performance problems.

1.18.8'warmup=nn': Warmup period

By default Vdbench will exclude the first interval from its run totals. The warmup value will cause the first 'warmup/interval' intervals to be excluded. Using this parameter also changes the

meaning of the 'elapsed=' parameter to mean 'run elapsed= seconds *after* the warmup period completes'.

1.18.9'distribution=xxx': Arrival Time Distribution

Specify 'distribution=exponential' for an exponential distribution, and 'distribution=uniform' for a uniform distribution. The default distribution is exponential. Remember, you may abbreviate to 'di=un'.

In simple terms, an exponential arrival rate is a good distribution to achieve lots of I/O bursts. It is a good method to approximate a large application with many users.

An exponential arrival rate is a classic modeling approach to describe a general arrival distribution of independent events. It causes significant queuing to occur even when the device utilization is only 50% busy.

1.18.10'pause=nn': Sleep 'nn' Seconds

When doing multiple runs, the 'pause=nn' parameter causes Vdbench to go to sleep for 'nn' seconds before it starts the next run. This parameter is ignored for the first run. 'pause=nn' can be used to allow a storage controller some time to catch its breath and complete things like emptying cache.

1.18.11 Workload parameter specification in a Run Definition.

The original objective of all the forxxx parameters was to allow a user to override most of all Workload Definition (WD) parameters specified earlier to create complex, varying workloads, like forxfersize=(1k-1m,d).

Once I observed that some users were specifying here frequently just single parameters like forx=16k I realized that this started making the Workload Definition obsolete. Yes, you'll always need multiple WDs to allow the running of *different concurrent* workloads, but for a workload without any forxxx variations WDs were not really needed anymore.

To make life easier for my users I then added the sd= parameter to a Run Definition, and from that point on indeed the WD became obsolete for this type of non-varying run.

The next step then was of course to not even ask you to specify forxfersize=, but just simply xfersize=. Internally of course Vdbench still treats it as a forxxx parameter, but as far as the parameter definition, you can now just specify forxfersize=4k.

Previous:

```
sd=sd1,lun=/dev/xxx
wd=wd1,sd=*,rdpct=100,xfersize=4k
rd=rd1,wd=wd1,iorate=max,elapsed=60,interval=1
```

New (And of course you can still code xfersize=(4k,8k):

```
sd=sd1,lun=/dev/xxx
rd=rd1,sd=*,iorate=max,elapsed=60,interval=1,rdpct=100,xfersize=4k
```

1.18.11.1 'sd=xxx' Specify SDs to use

If you did not specify a Workload Definition you may specify the SDs to be used here.

If you use both the sd= and wd= parameters, this will override the SDs specified in the Workload Definition.

1.18.11.2 '(for)xfersize=nn': Create 'for' Loop Using Different Transfer Sizes

The 'forxfersize=' parameter is an override for all workload specific xfersize parameters and allows multiple automatic executions of a workload with different data transfer sizes.

- forxfersize=4k	One run with 4k transfer size.
- forxfersize=(4k,8k,12k,16k)	One run each for 4k, 8k, etc.
- forxfersize=(4k-32k,4k)	One run each from 4k to 32k in increments of 4k.
- forxfersize=(1k-128k,d)	One run each from 1k to 128k, each time doubling the

transfer size. (1k,2k,4k,8k,etc).

See [Order of Execution](#) for information on the execution order of this parameter.

1.18.11.3'(for)threads=nn': Create '*for*' Loop Using Different Thread Counts

The 'forthreads=' parameter is an override for all storage definition specific thread parameters and allows multiple automatic executions of a workload with different numbers of threads.

- forthreads=32	Do one run with 32 threads for each SD.
- forthreads=(1,2,3,4)	Do one run each with 1, 2, 3 and 4 threads.
- forthreads=(1-5,1)	Do one run each from 1 to 5 threads in increments of one.
- forthreads=(1-64,d)	Do one run each from 1 to 64 threads, each time doubling the thread count.

See [Order of Execution](#) for information on the execution order of this parameter.

1.18.11.4'(for)rdpct=nn': Create '*for*' Loop Using Different Read Percentages

The 'forrdpct=' parameter is an override for all workload specific *rdpct* parameters, and allows multiple automatic executions of a workload with different read percentages.

- forrdpct=50	Do one run with 50% reads
- forrdpct=(10-100,10)	Do one run each with read percentage values ranging from 10 to 100 percent, incrementing the read percentage by 10 percent each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.18.11.5'(for)rhpc=nn': Create '*for*' Loop Using Different Read Hit Percentages

The 'forrhpc=' parameter is an override for all workload specific *rhpc* parameters, and allows multiple automatic executions of a workload with different read hit percentages.

- forrhpc=50	Do one run with 50% read hits
- forrhpc=(10-100,10)	Do one run each with read hit percentage values ranging from 10 to 100 percent, incrementing the read hit percentage by 10 percent each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.18.11.6'(for)whpct=nn': Create '*for*' Loop Using Different Write Hit Percentages

The 'forwhpct=' parameter is an override for all workload specific *whpct* parameters, and allows multiple automatic executions of a workload with different write hit percentages.

- forwhpct=50	Do one run with 50% write hits
- forwhpct=(10-100,10)	Do one run each with write hit percentage values ranging from 10 to 100 percent, incrementing the write hit percentage by 10 percent each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.18.11.7'(for)seekpct=nn': Create '*for*' Loop Using Different Seek Percentages

The 'forseekpct=' parameter is an override for all workload specific *seekpct* parameters, and allows multiple automatic executions of a workload with different seek percentages.

- forseekpct=50	Do one run with 50% seek
- forseekpct=(10-100,10)	Do one run each with seek percentage values ranging from 10 to 100 percent, incrementing the seek percentage by 10 percent each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.18.11.8'(for)hitarea=nn': Create '*for*' Loop Using Different Hit Area Sizes

The 'forhitarea=' parameter is an override for all Storage Definition hit area parameters, and allows multiple automatic executions of a workload with different hit area sizes:

- forhitarea=4m	Do one run with 4MB hit area
- forhitarea=(10m-100m,10m)	Do one run each with hit area values ranging from 10 MB to 100 MB, incrementing the hit area by 10 MB each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.18.11.9'(for)compression=nn': Create '*for*' Loop Using Different compression rates.

The 'forcompression=' parameter is an override for the compression= parameter, and allows multiple automatic executions of a workload with different compression percentages.

1.18.11.10Order of Execution Using 'forxxx' Parameters

The order in which the 'forthreads=', 'forxfersize=', 'forrdpct=', 'forrhpct=', 'forwhpct=', 'forseekpct=' and 'forhitarea=' parameters are found in the input will determine the order in which the requested workloads will be executed.

Treat this as a sequence of embedded 'for' loops. Using 'forthreads=' and 'forxfersize' as an example:

- for (all threads) { for (all xfersizes) { for (all I/O rates) } } versus
- for (all xfersizes) { for (all threads) { for (all I/O rates) } } depending on what parameter came first.

1.19 Data Validation and Journaling

Data validation should not to be used during a performance run. The processor overhead can impact performance results.

Before I start I want to answer a question that has come up a few times: “why use Vdbench to check for data corruptions? I can just write large files, calculate a checksum and then re-read and compare the checksums. “

Yes, of course you can do that, but is that really good enough? All you’re doing here is check for data corruptions during sequential data transfers. What about random I/O? Isn’t that important enough to check? If you write the same block X times and the contents you then find are correct, doesn’t it mean that you could have lost X-1 consecutive writes without ever noticing it?

You spent 24 hours writing and re-reading large sequential files, which block is the one that’s bad? When was that block written and when was that block read again? Yes, it is nice to say: I have a bad checksum over the weekend. It is much more useful to say “I have a specific error in a specific block, and yes, I know when it was written and when it was found to be in error”, and by the way, this bad block actually came from the wrong disk.”

See [data_errors=](#) for information about terminating after a data validation error.

Data validation works as follows: Every write operation against an SD or FSD will be recorded in an in-memory table. Each 512-byte sector in the block that is written contains an 8-byte logical byte address (LBA), and a one-byte data validation key. The data validation key is incremented from 1 to 126 for each write to the same block. Once it reaches the value **126**, it will roll over to **one**. **Zero** is an internal value indicating that the block has never been written. This *key* methodology is developed to identify lost writes. If the same block is written several times without changing the contents of the block it is impossible to recognize if one or more of the writes have been lost. Using this key methodology we will have to lose exactly 126 consecutive writes to a block without being able to identify that writes were lost.

After a block has been written once, the data in the block will be validated after each read operation. A write will always be prefixed by a read so that the original content can be validated. Use of the '-vr' execution parameter (or validate=read parameter file option) forces each block to be read immediately after it has been written. However, remember that there is no guarantee that the data has correctly reached the physical disk drive; the data could have been simply read from cache.

Since data validation tables are maintained in memory, data validation will normally not be possible after Vdbench terminates, or after a system crash/reboot. To allow continuous data validation, use journaling.

Journaling: to allow data validation after a Vdbench or system outage, each write is recorded in a journal file. This journal file is flushed to disk using synchronous writes after each update (or we would lose updates after a system outage). Each journal update writes 512 bytes to its disk. Each journal entry is 8 bytes long, thereby allowing 63 entries plus an 8-byte header to be recorded in one journal record. When the last journal entry in a journal record is written, an additional 512

bytes of zeros is appended, allowing Vdbench to keep track of end-of-file in the journal. A journal entry is written *before* and *after* each Vdbench write.

Since each Vdbench workload write will result in two *synchronous* journal writes, journaling will have an impact on throughput/performance for the Vdbench workload. It is highly recommended that you use a disk storage unit that has write-behind cache activated. This will minimize the performance impact on the Vdbench workload. To allow file system buffering on journal writes, specify '-jn' or '-jrn' (or journal=noflush in your parameter file) to prevent forced flushing. This will speed up journal writes, but they may be lost when the system does not shut down cleanly.

It is further recommended that the journals be written to what may be called a 'safe' disk. Do not write the journals to the same disk that you are doing error injection or other scary things on! With an unreliable journal, data validation may not work.

At the start of a run that requests journaling, two files are created: a map backup file, and a journal file. The contents of the in-memory data validation table (map) are written to both the backup and the journal file (all key entries being zero). Journal updates are continually written at the end of the journal file. When Vdbench restarts after a system failure and journal recovery is requested, the original map is read from the beginning of the journal file and all the updates in the journal are applied to the map. Once the journal file reaches end of file, all blocks that are marked 'modified' will be read and the contents validated.

Next, the in-memory map is written back to the beginning of the journal file, and then to the backup file. Journal records will then be written immediately behind the map on the journal file. If writing of the map to the journal file fails because of a system outage, the backup file still contains the original map from the start of the previous run. If during the next journal recovery it is determined that not all the writes to the map in the journal file completed, the map will be restored from the backup file and the journal updates again are applied from the journal entries that still reside in the journal file *after* the incomplete map.

After a journal recovery, there is one specific situation that needs extra effort. Since each write operation has a *before* and *after* journal entry, it can happen that an *after* entry has never been written because of a system outage. In that case, it is not clear whether the block in question contains *before* or *after* data. In that case, the block will be read and the data that is compared may consist of either of the two values, either the *new* data or *old* data.

When a block is larger than one sector (512 bytes) it can happen that the beginning of that block contains *new* data and the remainder of the block contains *old* data. Data Validation reports this situation, but at this time is not able to compare the *old* data. It could be that there is a corruption in the *old* data, and that at this time will not be recognized. This functionality hopefully will be added in 5.03.

Note: I understand that any storage device that is interrupted in the middle of a write operation must have enough residual power available to complete the 512-byte sector that is currently being written, or may be ignored. That means that if one single sector contains both old and new data that there has been a data corruption.

Once the journal recovery is complete, all blocks that are identified in the map as being written to at least once are read sequentially and their contents validated.

During normal termination of a run, the data validation map is written to the journal. This serves two purposes: end of file in the journal file will be reset to just after the map, thus preserving disk space, (at this time unused space is not freed, however) and it avoids the need to re-read the whole journal and apply it to the starting map in case you need to do another journal recovery.

Note: since the history of all data that is being written is maintained on a block by block level using different data transfer sizes within a Vdbench execution has the following restrictions:

- Different data transfer sizes are not allowed when using journaling
- Without journaling, different data transfer sizes are allowed, but switching to a different transfer size causes the in-memory maps to be cleared with the result that there is no validation of data written in an earlier run.
- For file system testing however different data transfer sizes are allowed, as long as they are all multiples of each other. If for instance you use a 1k, 4k and 8k data transfer size, data validation will internally use the 1k value as the 'data validation key block size', with therefore a 4k block occupying 4 smaller data validation key blocks.

Note: when you do a data validation test against a large amount of disk space it may take quite a while for a random block to be accessed for the second time. (Remember, Vdbench can only compare the data contents when it knows what is there). This means that a relative short run may appear successful while in fact no blocks have been re-read and validated. Vdbench therefore since Vdbench 5.00 keeps track of how many blocks were actually read and validated. If the amount of blocks validated at the end of a run is zero, Vdbench will abort.

Example: For a one TB lun running 100 iops of 8k blocks it will take 744 hours or 31 days for each random block to be accessed at least twice!

Note: since any re-write of a block when running data validation implies a pre-read of that block I suggest that when you specify a read percentage (rdpct=) you specify rdpct=0. This prevents you, especially at the beginning of a test, from reading blocks that Vdbench has not written (yet) and therefore is not able to compare, wasting precious IOPS and bandwidth. In these runs (unless you forcibly request an immediate re-read) you'll see that the run starts with a zero read percentage, but then slowly climbs to 50% read once Vdbench starts writing (and therefore pre-reading) blocks that Vdbench has written before.

1.20 Swat I/O Trace Replay

Vdbench, in cooperation with the Sun StorageTek™ Workload Analysis Tool (Swat) Trace Facility (STF) allows you to replay the I/O workload of a trace created using Swat.

A trace file created and processed by Swat using the ‘Create Replay File’ option creates file 'flatfile.bin' (flatfile.bin.gz for vdbench403 and up) which contains one record for each I/O operation identified by Swat.

See [Example 6: Swat I/O Trace Replay](#).

You can find Swat on the Sun download website, look for Sun StorageTek Workload Analysis Tool.

There are two ways to do a replay:

1. If you want precise control over which device is replayed on which SD, specify the device number in the SD: `sd=sd1,lun=xx,replay=(123,456,789)`. You cannot replay larger devices on a smaller target SD.
2. If you want Vdbench to decide what goes where, create a Replay Group:
`rg=group1,devices=(123,456,789)`
`sd=sd1,lun=xxx,replay=group1`
`sd=sd2,lun=yyy,replay=group1`
 Using this method, Vdbench will act like his own volume manager
 Swat will even help you with the creation of this replay parameter file. Select ‘File’
 ‘Create replay parameter file’. Just add enough SDs and some flour.

You can create a mix of both methods if you want partial control over what is replayed where.

Add the Swat replay file name to the Run Definition parameters, and set an elapsed time at least larger than the duration of the original trace: `'rd=rd1,...,elapsed=60m,replay=flatfile.bin(.gz)'`.

The I/O rate by default is set to the I/O rate as it is observed in the trace input. If a different I/O rate is requested, the inter-arrival times of all the I/Os found in the trace will be adjusted to allow for the requested change. The run will terminate as soon as the last I/O has completed.

With Swat replay, there is no longer a need to have an application installed to do performance testing. All you need is a one-time trace of the application's I/O workload and from that moment on, you can replay that workload as often as you want without having to go to the effort and expense of installing and/or purchasing the application and/or data base package and copying the customer's production data onto your system.

You can replay the workload on a different type of storage device to see what the performance will be, you can increase the workload to see what happens with performance, and with Veritas VxVm raw volumes (Veritas VxVm I/O is also traced) you can even modify the underlying Veritas VxVm structure to see what the performance will be when, for instance, you change from RAID 1 to RAID 5 or change stripe size!

1.21 Complete Swat I/O Replay Example

Note: See Swat documentation for further information.

1. Log on as root (this example is for Solaris; for Windows you need to log on as Administrator)
2. `cd /swat`
3. `./swat` This starts the Swat graphical user interface.
4. Select 'Swat Trace Facility (STF).
5. Use the 'Create I/O trace' tab to start a trace.
6. Wait for the trace to complete.
7. You may leave root now.
8. Using Swat, run 'Extract'. Wait for completion.
9. Run Analyze, with the 'Create Replay File' checkbox selected. Wait for completion.
10. Use 'Reporter'. Select the devices that you want replayed.
11. Select 'File' 'Create replay parameter file'.
12. Copy the sample replay parameter file and paste it into any new parameter file.
13. Add enough SDs to be at least equal to the total amount of gigabytes needed.
14. Run '*vdbench -f parameter.file*'

1.22 File system testing

The basic functionality of Vdbench has been created to test and report the performance of one or more raw devices, and optionally one or more large file system files.

Starting release 405 a second type of performance workload has been added to assist with the testing of file systems.

Vdbench file system workloads revolve around two key sets of new parameters:

- A *File System Anchor*, consisting of a directory name, and a directory and file structure that will be created under that anchor. Structure information consists of directory depth, directory width, number of files, and file sizes. Multiple file anchors may be defined and used concurrently. A maximum of 32 million files per anchor are supported.
- A *File System Operation*. File system operations are directory create/delete, file create/delete, file read/write, file open/close, setattr and getattr.

Parameter structure:

- File System Definition (FSD): This parameter describes the directory and file structure that will be created.
- File system Workload Definition (FWD) is used to specify the FSD(s) to be used and specifies miscellaneous workload parameters.
- Run Definition (RD) has a set of parameters that controls the file system workloads that will be executed.

Each time Vdbench starts it needs to know the current status of all the files, unless of course `format=yes` is specified. When you have loads of files, querying the directories can take quite a while. To save time, each time when Vdbench terminates normally the current status of all the files is stored in file `'vdb_control.file'` in the anchor directory. This control file is then read at the start of the next run to eliminate the need to re-establish the current file status.

When using `'shared=yes'` as an FSD parameter this control file however will not be maintained.

1.22.1 Directory and file names

Directory names are generated as follows: `vdb_x_y.dir`, e.g. `vdb1_1.dir`

Where 'x' represents the depth of this directory (as in `depth=nn`), and 'y' represents the width (as in `width=nn`).

File names are generated as follows: `vdb_fnnnn.file`

Where `'nnnn'` represents a sequence number from 1 to `'files=nnnn'`

Example: `fsd=fsd1,anchor=dir1,depth=2,width=2,files=2`

```
find dir1 | grep file
dir1/vdb_control.file
dir1/vdb1_1.dir/vdb2_1.dir/vdb_f0001.file
dir1/vdb1_1.dir/vdb2_1.dir/vdb_f0002.file
dir1/vdb1_1.dir/vdb2_2.dir/vdb_f0001.file
```

```

dir1/vdb1_1.dir/vdb2_2.dir/vdb_f0002.file
dir1/vdb1_2.dir/vdb2_1.dir/vdb_f0001.file
dir1/vdb1_2.dir/vdb2_1.dir/vdb_f0002.file
dir1/vdb1_2.dir/vdb2_2.dir/vdb_f0001.file
dir1/vdb1_2.dir/vdb2_2.dir/vdb_f0002.file

```

File ‘vdb_control.file’ contains a description of the current directory and file structure. This file is there to allow consecutive Vdbench tests that use an *existing* structure to make sure that the existing structure matches the parameter settings.

During a cleanup of an existing directory structure Vdbench only deletes files and directories that contain this naming pattern. No other files will be deleted. So rest assured that if you specify /root as your anchor directory you won’t lose your system ☺

Files by default are created in the lowest directory level. When specifying ‘distribution=all’, files will be created in every directory I expect to build more detailed file structures in the future.

FEEDBACK NEEDED!

1.22.2 File system sample parameter file

```

fsd=fsd1,anchor=/dir1,depth=2,width=2,files=2,size=128k
fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2
rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=10,interval=1

```

This parameter file will use a directory structure of 4 directories and 8 files (see above for file names). The RD parameter ‘format=yes’ causes the directory structure to be completely created (after deleting any existing structure), including initialization of all files to the requested size of 128k.

After the format completes the following will happen for 10 seconds at a rate of 100 reads per second:

- Start two threads (threads=2; 1 thread is default).
- Each thread:
 - Randomly selects a file (fileselect=random)
 - Opens this file for read (operation=read)
 - Sequentially reads 4k blocks (xfersize=4k) until end of file (size=128k)
 - Closes the file and randomly selects another file.

This is a very simple example. Much more complex scenarios are possible when you use the complete set of Vdbench parameters. Be aware though that complexity comes at a price. For instance, you can’t read or write before a file is created and you can’t create a file before its parent directory is created. The ‘format=yes’ parameter can be very helpful here, even though it is possible to do your own format using mkdir, create and write operations. See also [Multi Threading and file system testing](#)

1.23 File System Definition (FSD) parameter overview:

fsd=name	Unique name for this File System Definition.
fsd=default	All parameters used will serve as default for all the following fsd's.
count=(nn,mm)	Creates a sequence of FSD parameters.
anchor=/dir/	The name of the directory where the directory structure will be created.
shared=yes/no	Default 'no': See FSD sharing
width=nn	How many directories to create in each new directory.
depth=nn	How many levels of directories to create under the anchor.
files=nn	How many files to create in the lowest level of directories.
sizes=(nn,nn,....)	Specifies the size(s) of the files that will be created.
distribution=all	Default 'bottom', creates files only in the lowest directories. 'all' creates files in all directories.
openflags=(flag,...)	Pass extra flags to (Solaris) file system open request (See: man open)
total_size=nnn	Stop after a total of 'nnn' bytes of files have been created.
workingsetsize=nn wss=nn	Causes Vdbench to only use a subset of the total amount of files defined in the file structure. See workingsetsize.

1.24 Filesystem Workload Definition (FWD) parameter overview:

fwd=name	Unique name for this Filesystem Workload Definition.
fwd=default	All parameters used will serve as default for all the following fwd's.
fsd=(xx,...)	Name(s) of Filesystem Definitions to use
host=host label	Which host this workload to run on.
fileio=random	How file I/O will be done: random or sequential
fileio=sequential	How file I/O will be done: random or sequential
fileio=(seq,delete)	Sequential I/O: When opening for writes, first delete the file
stopafter=nnn	For random I/O: stop and close file after 'nnn' reads or writes. Default 100 for random I/O.
fileselect=random/ sequential	How to select file names or directory names for processing.
xfersizes=nn	Specifies the data transfer size(s) to use for read and write operations.
operation=xxxx	Specifies a single file system operation that must be done for this workload.
rdpct=nn	For 'read' and 'write' only. This allows a mix and read and writes against a single file.
skew=nn	The percentage of the total amount of work for this FWD
threads=nn	How many concurrent threads to run for this workload. (Make sure you have at least one file for each thread).

1.24.1 Multi-host parameter replication.

Whenever the constant '\$host', '!host' or '#host' is found in a line in a parameter file, this line is automatically repeated once for each host label that has been defined using the Host Definition (HD) parameters. Some times when you run tests against multiple different hosts, directing file system workloads towards specific target hosts can become mighty complex. The \$host parameter is there to make life a little easier. A simple example:

hd=host1,... hd=host2,... fsd=fsd \$host,anchor=/dir/\$host,.....	Result: fsd=fsd_host1,anchor=/dir/host1,..... fsd=fsd_host2,anchor=/dir/host2,.....
Just add host=host3,....	fsd=fsd_host1,anchor=/dir/host1,..... fsd=fsd_host2,anchor=/dir/host2,..... fsd=fsd_host3,anchor=/dir/host3,.....

Note that this only works on one single line in the parameter file, not if the parameters are split over multiple lines, for instance using above example, one line for fsd=fsd\$host, and then anchor= on the next line.

'\$host' and '!host' are replaced with the host label. '!host' is there to prevent problems when you include your parameter file inside of a ksh/csh script that is trying to interpret \$host too early. '#host' is replaced with the current relative host, 0,1,2, etc.

1.25 Run Definition (RD) parameters for file systems, overview

These parameters are file system specific parameters. More RD parameters can be found at [Run Definition Parameter Overview](#). Be aware, that some of those parameters like 'forrhpc=' are not supported for file system testing.

fwd=(xx,yy,...)	Name(s) of Filesystem Workload Definitions to use.
fwdrate=nn	How many file system operations per second
format=yes/no/only/ restart/clean/ directories	During this run, if needed, create the complete file structure .
operations=xx	Overrides the operation specified on all selected FWDs.
foroperations=xx	Multiple runs for each specified operation.
fordepth=xx	Multiple runs for each specified directory depth
forwidth=xx	Multiple runs for each specified directory width
forfiles=xx	Multiple runs for each specified amount of files
forsizes=xx	Multiple runs for each specified file size
fortotal=xx	Multiple runs for each specified total file size

1.26 File System Definition (FSD) parameter detail:

Warning: specifying a directory and file structure is easy. However, it is also very easy to make it too large. Width=5,depth=5,files=5 results in 3905 directories and 15625 files! Vdbench allows 32 million files per FSD. About 64 bytes of Java heap space is needed per file, possibly causing memory problems. You may have to update the 'Xmx' parameter in your Vdbench script.

1.26.1 'fsd=name': Filesystem Storage Definition name

'fsd=' uniquely identifies each File System Definition. The FSD name is used by the Filesystem Workload Definition (FWD) parameter to identify which FSD(s) to use for this workload.

When you specify 'default' as the FSD name, the values entered will be used as default for all FSD parameters that follow.

1.26.2 'anchor=': Directory anchor

The name of the directory where the directory structure will be created. This anchor may not be a parent or child directory of an anchor defined in a different FSD. If this anchor directory is the same as an anchor directory in a different FSD the directory structure (width, depth etc) must be identical. If this directory does not exist, Vdbench will create it for you. If you also want Vdbench to create this directory's parent directories, specify ['create_anchor=yes'](#).
Example: anchor=/file/system/

1.26.3 'count=(nn,mm)' Replicate parameters

This parameter allows you to quickly create a sequence of FSDs, e.g.
fsd=fsd,anchor=/dir,count=(1,5) results in fsd1-fsd5 for /dir1 through /dir5

1.26.4 'shared=' FSD sharing.

With Vdbench running multiple slaves and optionally multiple hosts, communications between slaves and hosts about a file's status becomes difficult. The overhead involved to have all these slaves communicate with each other about what they are doing with the files just becomes too expensive. You don't want one slave to delete a file that a different slave is currently reading or writing. Vdbench therefore does not allow you to share FSDs across slaves and hosts.

That of course all sounds great until you start working with huge file systems. You just filled up 500 terabytes of disk files and you then decide that you want to share that data with one or more remote hosts. Recreating this whole file structure from scratch just takes too long. What to do?

When specifying ‘shared=yes’, Vdbench will allow you to share a File System Definition (FSD). It does this by allowing each slave to use only every ‘nth’ file as is defined in the FSD file structure, where ‘n’ equals the amount of slaves.

This means that the different hosts won’t step on each other’s toes, with one exception: When you specify ‘format=yes’, Vdbench first deletes an already existing file structure. Since this can be an old file structure, Vdbench cannot spread the file deletes around, letting each slave delete his ‘nth’ file. Each slave therefore tries to delete ALL files, but will not generate an error message if a delete fails (because a different slave just deleted it). These failed deletes will be counted and reported however in the [‘Miscellaneous statistics’](#), under the ‘FILE_DELETE_SHARED’ counter. The ‘FILE_DELETES’ counter however can and will contain a count higher than the amount of files that existed. I have seen situations where multiple slaves were able to delete the same file at the same time without the OS passing any errors to Java.

1.26.5 ‘width=’: Horizontal directory count

This parameter specifies how many directories to create in each new directory. [See above for an example.](#)

1.26.6 ‘depth=’: Vertical directory count

This parameter specifies how many levels of directories to create under the anchor. [See above for an example.](#)

1.26.7 ‘files=’: File count

This parameter specifies how many files to create in the lowest level of directories. [See above for an example.](#) Note that you need at least one file per ‘fwd=xxx,threads=’ parameter specified. If there are not enough files, a thread may try to find an available file up to 10,000 times before it gives up.

1.26.8 ‘sizes=’: File sizes

This parameter specifies the size of the files. Either specify a single file size, or a set of pairs, where the first number in a pair represents file size, and the second number represents the percentage of the files that must be of this size. E.g. sizes=(32k,50,64k,50)

When you specify ‘sizes=(nnn,0)’, Vdbench will create files with an average size of ‘nnn’ bytes. There are some rules though related to the file size that is ultimately created:

- If size > 10m, size will be a multiple of 1m
- If size > 1m, size will be multiple of 100k
- If size > 100k, size will be multiple of 10k
- If size < 100k, size will be multiple of 1k.

1.26.9 'openflags=': Optional file system 'open' parameters

Use this parameter to pass extra flags to (Solaris) file system open request (See: man open(2))

Flags currently supported: O_DSYNC, O_RSYNC, O_SYNC.

See also ['openflags=': synchronous control over file system writes.](#)

1.26.10 'total_size=': Create files up to a specific total file size.

This parameter stops the creation of new files after the requested total amount of file space is reached. Be aware that the 'depth=', 'width=', 'files=' and 'sizes=' parameter values must be large enough to accommodate this request. See also the RD 'fortotal=' parameter.

Example: total_size=100g

1.26.11 'workingsetsize=nn' or 'wss=nn'

While the depth, width, files and sizes parameters define the maximum possible file structure, 'totalsize=' if used specifies the amount of files and file space to create.

'workingsetsize=' creates a subset of the file structure of those files that will be used during this run. If for instance you have 200g worth of files, and 32g of file system cache, you can specify 'wss=32g' to make sure that after a warmup period, all your file space fits in file system cache. Can also be used with 'forworkingsetsize' or 'forwss'.

1.27 File system Workload Definition (FWD) detail

1.27.1 'fwd=name': File system Workload Definition name

'fwd=' uniquely identifies each File system Workload Definition. The FWD name is used by the Run Definition (RD) parameter to identify which FWDs to use for this workload.

When you specify 'default' as the FWD name, the values entered will be used as default for all FWD parameters that follow.

1.27.2 'fsd=': which File System Definitions to use

This parameter specifies which FSDs to use for this workload.

Example: fsd=(fsd1,fsd2)

1.27.3 'fileio=': random or sequential I/O

This parameter specifies the type of I/O that needs to be done on each file, either random or sequential. A random LBA will be generated on a data transfer size boundary.

- fileio=random: do random I/O.
- fileio=sequential: do sequential I/O.
- fileio=(seq,delete): delete the file before writing.

1.27.4 'rdpct=nn': specify read percentage

This parameter allows you to mix reads and writes. Using operation=read only allows you to do reads, operation=write allows you to only do writes. Specify rdpct= however, and you will be able to mix reads and writes within the same selected file. Note that for sequential this won't make much sense. You could end with read block1, write block2, read block3, etc. For random I/O however this makes perfect sense.

1.27.5 'stopafter=': how much I/O?

This parameter lets Vdbench know how much reads or writes to do against each file. You may specify 'stopafter=nn' which will cause Vdbench to stop using this file after 'nn' blocks, or you may specify 'stopafter=nn%' which will stop processing after nn% if the requested file size is processed.

For random I/O Vdbench by default will stop after 100 blocks (stopafter=100), or after 'file size' bytes are read or written. This default prevents you from accidentally doing 100 8k random reads or writes against a single 8k file.

For sequential I/O Vdbench by default will always read or write the complete file. When you specify 'stopafter=' though, Vdbench will only read or write the amount of data requested. The next time this file is used for sequential I/O however it will continue after the last block previously used.

This can be used to simulate a file 'append' when writing to a file.

1.27.6'fileselect=': which files to select?

This parameter allows you to select directories and files for processing either sequentially in the order in which they have been specified using the depth=, width=, and files= FSD parameters, or whether they should be selected randomly. See also [Directory and file names](#).

1.27.7'xfersizes=': data transfer sizes for read and writes

This parameter specifies the data transfer size(s) to use for read and write operations. Either specify a single xfersize, or a set of pairs, where the first number in a pair represents xfersize, and the second number represents the percentage of the I/O requests that must use this size. E.g. xfersizes=(8k,50,16k,30,2k,20)

1.27.8'operation=': which file system operation to execute

Specifies a single file system operation that must be executed for this workload: Choose one: mkdir, rmdir, create, delete, open, close, read, write, getattr and setattr. If you need more than one operation specify 'operations=' in the Run Definition.

To allow for mixed read and write operations against the same file, specify fwd=xxx,rdpct=nn.

1.27.9'skew=': which percentage of the total workload

The percentage of the total amount of work (specified by the fwdrate= parameter in the Run Definition) assigned to this workload. By default all the work is evenly distributed among all workloads.

1.27.10'threads=': how many concurrent operations for this workload

Specifies how many concurrent threads to run for this workload. It should be clear that this does not mean that 'n' threads are running against each file, but instead it means that there will be 'n' concurrent files running this same workload. *All file operations for a specific directory or file are single threaded.* See [Multi Threading and file system testing](#).

Make sure you always have at least one file for each thread. If not, one or more threads continue trying to find an available file, but Vdbench gives up after 10,000 consecutive failed attempts.

1.28 Run Definition (RD) parameters for file system testing, detail

1.28.1 'fwd=': which File system Workload Definitions to use

This parameter tells Vdbench which FWDs to use during this run. Specify a single workload as 'fwd=fwd1' or multiple workloads either by entering them individually 'fwd=(fwd1,fwd2,fwd3)', a range 'fwd=(fwd1-fwd3)' or by using a wildcard character 'fwd=fwd*'.

1.28.2 'fwdrate=': how many operations per second.

- fwdrate=100	Run a workload of 100 operations per second
- fwdrate=(100,200,...)	Run a workload of 100 operations per second, then 200, etc.
- fwdrate=(100-1000,100)	Run workloads with operations per second from 100 to 1000, incremented by 100.
- fwdrate=curve	Create a performance curve.
- fwdrate=max	Run the maximum uncontrolled operations per second.

This parameter specifies the combined rate per second to generate for all requested file system operations.

See also ['iorate=nn': One or More I/O Rates](#).

There is a specific reason why the label 'fwdrate' was chosen compared to 'iorate' for raw I/O workload. Though usually most of the operations executed against file systems will be reads and writes, and therefore I/O operations, Vdbench also allows for several other operations: mkdir, rmdir, create, delete, open, close, getattr and setattr. These operations are all metadata operations which are therefore not considered I/O operations.

1.28.3 'format=': pre-format the requested directory and file structure

When specifying `format=yes`, this parameter requests that at the start of each run any old directory structure first is deleted and then the new one recreated. When one or more '`forxxx=`' parameters are specified the delete and recreation in between runs is only done when the directory and file structure changes, for instance because of using the '`fordepth=`' parameter.

Any format request (unless `format=restart`) will delete every file and directory that follows the directory and file naming that Vdbench generates. Don't worry; Vdbench won't accidentally delete your root directory. See also [Directory and file names](#).

Be careful though with `format`: you may just have spent 48 hours creating a file structure. You don't want to accidentally leave '`format=yes`' in your parameter file when you only want to reuse the just created file structure.

Also understand that if you change the file structure a format run is required. Vdbench keeps track of what the previous file structure was and will refuse to continue if it has been changed.

A format implies that first all the directories are created. After this all files will be sequentially formatted using 64k as a maximum `xfersize`.

When specifying '`format=yes`' for a file system workload Vdbench automatically inserts an extra workload and run to do the formatting. Defaults for this run are `threads=8,xfersize=64k`.

To override this, add `fwd=format,threads=nn,xfersize=nn`. You can also specify '`openflags=xxx`'. All other parameters used in `fwd=format` will be ignored.

<code>format=</code>	When using more than one option use parenthesis: <code>format=(yes,restart)</code> Note: Unless ' <code>no</code> ' is used anywhere, ' <code>yes</code> ' is implied.
<code>no</code>	No format required, though the existing file structure must match the structure defined for this FSD.
<code>yes</code>	Vdbench will first delete the current file structure and then will create the file structure again. It will then execute the run you requested in the current RD.
<code>restart</code>	Vdbench will create only files that have not been created and will also expand files that have not reached their proper size.
<code>only</code>	The same as ' <code>yes</code> ', but Vdbench will NOT execute the current RD.
<code>dir(ectories)</code>	The same as ' <code>yes</code> ', but it will only create the directories.
<code>clean</code>	Vdbench will only delete the current file structure and NOT execute the current RD.

1.28.4 'operations=': which file system operations to run

Specifies one or more of the available file system operations: mkdir, rmdir, create, delete, open, close, read, write, getattr and setattr. This overrides the 'fwd=xxx,operations=' parameter. E.g. operations=mkdir or operations=(read,getattr)

This can get tricky, but Vdbench will be able to handle it all. If for instance you do not have an existing file structure, and you ask for operations=read, Vdbench will fail because there are no files available. Code operations=(create,read) and Vdbench will still fail because there still are no directories available. Code operations=(mkdir,create,read) will also fail because even though the files exist, they are still empty. With operations=(mkdir,create,write,read) things should work just fine.

There's one 'gotcha' here though: once all directories and files have been created the threads for those operations are terminated because there no longer is anything for them to do. This means that if you have specified for instance fwdrate=1000 the remaining threads for 'read' and 'write' will continue doing their requested portion of the total amount of work, and that is 250 operations per second each for a total of fwdrate=500.

A different way to do your own formatting of the file structure is run with 'foroperations=(mkdir,create,write,read)'.

1.28.5 'foroperations=': create 'for' loop using different operations

The 'foroperations=' parameter is an override for all workload specific *operations* parameters, and allows multiple automatic executions of a workload with different operations.

While the 'operations=' parameter above does one run with all requested operations running at the same time, 'foroperations=' does one run per operation.

- foroperations=read	Only do read operations
- foroperations=(read,write,delete,rmdir)	Does one run each first reading all files, then writing, and then deletes all directories and files (A test like this requires the directory structure to first have been created by for instance using 'format=yes')

See [Order of Execution](#) for information on the execution order of this parameter.

1.28.6 'fordepth=': create 'for' loop using different directory depths

The 'fordepth=' parameter is an override for all FSD specific *depth* parameters, and allows multiple automatic executions of a directory structure with different depth values..

- fordepth =5	One run using depth=5
- fordepth =(5-10,1)	Does one run each with different depth values ranging from five to ten, incrementing the directory depth by one each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.28.7'forwidth=': create 'for' loop using different directory widths

The 'forwidth=' parameter is an override for all FSD specific *width* parameters, and allows multiple automatic executions of a directory structure with different width values.

- forwidth =5	One run using width=5
- forwidth =(5-10,1)	Does one run each with different width values ranging from five to ten, incrementing the directory width by one each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.28.8'forfiles=': create 'for' loop using different amount of files

The 'forfiles=' parameter is an override for all FSD specific *files* parameters, and allows multiple automatic executions of a directory structure with different files values.

- forfiles =5	One run using files=5
- forfiles =(5-10,1)	Does one run each with different files= values ranging from five to ten, incrementing the amount of files by one each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.28.9'forsizes=': create 'for' loop using different file of sizes

The 'forsizes=' parameter is an override for all FSD specific *sizes* parameters, and allows multiple automatic executions of a directory structure with different file sizes.

When you use this parameter you cannot specify a distribution of file sizes as you can do using the [FSD definitions](#).

- forsizes =5	One run using sizes=5
- forsizes =(5-10,1)	Does one run each with different sizes= values ranging from five to ten, incrementing the amount of files by one each time.

See [Order of Execution](#) for information on the execution order of this parameter.

1.28.10 'fortotal=': create 'for' loop using different total file sizes

This parameter is an override for the FSD 'files=' parameter. It allows you to create enough files to fill up the required amount of total file sizes, e.g. fortotal=(10g,20g). These values must be incremental. See also the [total_size=](#) parameter.

- fortotal=5g	One run using fortotal=5g
- fortotal=(5g,10g)	One run with total_size=5g, and then one run with total_size=10g

Note that this results in multiple format runs being done if requested. Since you do not want the second format to first delete the previous file structure you may specify format=(yes,restart).

1.28.11 'forwss=': 'for' loop using working set sizes.

This parameter overrides the FSD [workingsetsize=](#) parameter forcing Vdbench to use only a subset of the file structure defined with the FSD.

- forwss=16g	Uses only a 16g subset of the files specified in the FSD
- forwss=(16g,32g)	Two runs: one for 16g and one for 32g.

1.29 Multi Threading and file system testing

As mentioned earlier, multi threading for file system testing does not mean that multiple threads will be concurrently using the same file. All individual file operations are done single threaded. Other threads however can be active with different files.

Considering the complexity allowing directory creates, file creates, file reads and file writes against the same directory structure happening concurrently there are some pretty interesting scenarios that Vdbench has to deal with. Some of them:

- Creating a file before its parent directory or directories exist.
- Reading or writing a file that does not exist yet.
- Reading a file that has not been written yet.
- Deleting a file that is currently being read or written.
- Reading a file that does not exist while there are no new files being created.

When these things happen Vdbench will analyze the situation. For instance, if he wants to write to a file that does not exist, the code will check to see if any new files will be created during this run. If so, the current thread goes to sleep for 200 microseconds, selects the next directory or file and tries again. If there are no file creates pending Vdbench will abort.

At the end of each run numerous statistics related to these issues will be reported in logfile.html and on stdout, with a brief explanation and with a count.

To identify deadlocks (which is an error situation and should be reported to me) Vdbench will abort after 10000 consecutive sleeps without a successful operation.

Note: there currently is a known deadlock situation where there are more threads than files. If you for instance specify 12 threads but only 8 files, 4 of the threads will continually be in the 'try and sleep' loop, ultimately when the run is long enough hitting the 10000 count.

Miscellaneous statistics example:

```
13:28:35.183 Miscellaneous statistics:
13:28:35.183 DIRECTORY_CREATES    Directories creates:          7810
13:28:35.183 FILE_CREATES                File creates:                625000
13:28:35.183 WRITE_OPENS                 Files opened for write activity: 625000
13:28:35.184 DIR_EXISTS          Directory may not exist (yet):  33874
13:28:35.184 FILE_MAY_NOT_EXIST          File may not exist (yet):      94
13:28:35.184 MISSING_PARENT              Parent directory does not exist (yet): 758
13:28:35.184 PARENT_DIR_BUSY             Parent directory busy, waiting: 25510
```

1.30 Operations counts vs. nfsstat counts:

The operations that you can specify are: mkdir, rmdir, create, delete, open, close, read, write, getattr and setattr. These are the operations that Vdbench will execute. After a run against an NFS directory if you look at the nfs3/4.html files (they are linked to from kstat.html) you'll see the nfsstat reported counts. *These counts do not include the operations that were either handled from file system cache or from inode cache.* Even if you mount the file systems for instance with *forcedirectio* and *noac* there is no guarantee that the nfsstat counts match one-for-one the work done by Vdbench. For instance, one single stat() C function request translates into four NFS getattr requests.

The only way for the Vdbench and nfsstat counts to possibly match is if Vdbench would use native NFS code. This is not within the scope of Vdbench.

Also, nfsstat shows a total of **all** NFS operations, not only of what Vdbench is running against your specific file system. If you have a dedicated system for testing then you can control how much other NFS work there is going on. To make sure that Vdbench reporting does not generate extra NFS activity use the Vdbench '-output' parameter to send the Vdbench output to a local file system.

1.31 Report file examples

1.31.1summary.html

'summary.html' reports the total workload generated for each run per reporting interval, and the weighted average for all intervals *except* the first (used to be first and last, report examples have not been updated).

Note: the first interval will be ignored for the run totals unless the [warmup= parameter](#) is used in which case you can ask Vdbench to ignore more than one interval.

	interval	I/O rate	MB/sec 1024**2	bytes i/o	read pct	resp time	resp max	resp stddev	cpu% sys+usr	cpu% sys
11:56:00.056	1	5012.98	4.90	1024	100.00	0.966	35.943	0.295	5.9	2.4
11:56:04.120	2	4998.36	4.88	1024	100.00	0.956	7.767	0.171	2.2	0.5
11:56:08.048	3	4999.57	4.88	1024	100.00	0.963	1.675	0.157	1.4	0.2
11:56:12.035	4	5002.67	4.89	1024	100.00	0.948	24.736	0.232	1.4	0.4
11:56:16.069	5	5001.08	4.88	1024	100.00	0.936	2.706	0.166	1.3	0.4
11:56:16.124	avg_2-5	5000.42	4.88	1024	100.00	0.951	24.736	0.184	1.6	0.4

11:56:18.342 Vdbench execution completed successfully

- interval:	Reporting interval sequence number. See 'interval=nn' parameter.
- I/O rate:	Average observed I/O rate per second.
- MB sec:	Average number of megabytes of data transferred.
- bytes I/O:	Average data transfer size.
- read pct:	Average percentage of reads.
- resp time:	Average response time measured as the duration of the read/write request. All Vdbench times are in milliseconds.
- resp max:	Maximum response time observed in this interval. The last line contains total max.
- resp stddev:	Standard deviation for response time.
- cpu% sys+usr:	Processor busy = 100 - (system + user time) (Solaris, windows, Linux)
- cpu% sys:	Processor utilization; system time

1.31.2summary.html for file system testing

This sample report has been truncated. Three columns exist for each file system operation.

```
.Interval. .ReqstdOps.. ...cpu%... ....read.... ...write.... ..mb/sec... mb/sec .xfer. etc.etc
          rate  resp total  sys   rate  resp   rate  resp   read write  total  size
1      93.0  4.81   2.7 1.00   93.0  4.81   0.0  0.00  0.05  0.00  0.05  512
2      98.0  2.16   5.3 1.27   98.0  2.16   0.0  0.00  0.05  0.00  0.05  512
3     100.0  1.33   2.3 0.25  100.0  1.33   0.0  0.00  0.05  0.00  0.05  512
4      99.0  0.58   2.2 0.75   99.0  0.58   0.0  0.00  0.05  0.00  0.05  512
5      99.0  0.86   1.5 0.75   99.0  0.86   0.0  0.00  0.05  0.00  0.05  512
avg_2-5  99.0  1.23   2.8 0.75   99.0  1.23   0.0  0.00  0.05  0.00  0.05  512
```

Interval:	Reporting interval sequence number. See ' interval=nn ' parameter.
ReqstdOps	The total amount of <i>requested</i> operations. Though when asking for operation=read requires an open operation, this open has not been specifically requested and is therefore not included in this count. This open however IS reported in the 'open' column. For a format run this count includes all write operations.
cpu% total	Processor busy = 100 - (system + user time) (Solaris, windows, Linux)
cpu% sys	Processor utilization; system time
read	Total reads and average response time
write	Total writes and average response time.
mb/sec	Mb per second for reads, writes, and the sum of reads and writes.
xfer	Average transfer size for read and write operations.
.....	Two columns each for all remaining operations.

Each operation, and also ReqstdOps have two columns: the amount of operations and the average response time. There are also columns for average xfersize and megabytes per second.

Note: due to the large amount of columns that are displayed here the precision of the displayed data may vary. For instance, a rate of 23.4 per second will be displayed using decimals, but a rate of 2345.6 will be displayed without decimals as 2345. I like my columns to line up ☺.

1.31.3 SD_name.html

'SD_name.html' contains performance information for each Storage Definition used. For report descriptions, see 'summary.html' above.

1.31.4logfile.html

'logfile.html' contains a copy of each line of information that has been written by the Java code to the console window. Logfile.html is primarily used for debugging purposes. If ever you have a problem or a question about a Vdbench run, always add a tar or zip file of the complete Vdbench output directory in your email. Especially when crossing multiple time zones this can save a lot of time because usually the first thing I'll ask for is this tar or zip file.

1.31.5kstat.html

'kstat.html' contains Kstat statistics *for Solaris only*:

```

interval  KSTAT_i/o    resp    wait    service  MB/sec    read    busy  avg_i/o  avg_i/o  bytes    cpu%    cpu%
          rate    time     time     time  1024**2   pct    pct  waiting  active  per_io  sys+usr  sys
11:55:51.035 Starting RD=runl; I/O rate: 5000; Elapsed: 20 seconds. For loops: threads=8
11:56:00.023      1  4998.10    0.89    0.02    0.87    4.88 100.00  67.7    0.11    4.33    1024    5.9    2.4
11:56:04.087      2  5000.06    0.88    0.02    0.86    4.88 100.00  67.7    0.11    4.30    1024    2.2    0.5
11:56:08.024      3  5000.07    0.89    0.02    0.87    4.88 100.00  67.6    0.11    4.35    1024    1.4    0.2
11:56:12.013      4  4999.95    0.87    0.02    0.85    4.88 100.00  67.7    0.11    4.25    1024    1.4    0.4
11:56:16.013      5  4999.83    0.86    0.02    0.84    4.88 100.00  67.7    0.11    4.21    1024    1.3    0.4
11:56:16.101 avg_2-5  4999.98    0.88    0.02    0.86    4.88 100.00  67.7    0.11    4.28    1024    1.6    0.4

```

- I/O rate:	I/O rate per second over the duration of the reporting interval
- resp time:	Response time (the sum of wait time and service time). All Vdbench times are in milliseconds.
- wait time:	Average time each I/O spent queued on the host
- service time:	Average time the I/O was being processed
- MB/sec:	Average data transfer rate per second
- read pct:	Average percentage of total I/O that was read
- busy pct:	Average device busy percentage
- avg #I/O waiting:	Average number of I/Os queued on the host
- avg #I/O active:	Average number of I/Os active
- bytes per I/O:	Average number of bytes transferred per I/O
- cpu% sys+usr	Processor busy = 100 - (system + user time)
- cpu% sys	Processor utilization; system time

Warning: Vdbench reports the sum of the service time and wait time correctly as **response time**. *iostat* reports the same value as **service time**. The terminology used by *iostat* is **wrong**.

1.31.7nfs3/4.html

This report is created on Solaris if any of the workloads created use NFS mounted filesystems. This sample report has been truncated. One column exists for each NFS operation.

See also [Operations counts vs. nfsstat counts](#):

	interval	getattr	setattr	lookup	access	read	write	commit	create	mkdir	etc.etc.
		rate	rate	rate	rate	rate	rate	rate	rate	rate	
10:02:55.038	1	1.4	0.0	2.9	1.4	1.4	1.4	0.0	0.0	0.0	
10:02:56.476	2	5.5	1.4	11.7	3.4	2.8	1.4	0.0	0.0	6.9	
10:02:57.041	3	1.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
10:02:58.029	4	8.0	0.0	16.0	8.0	0.0	0.0	0.0	8.0	0.0	
10:02:59.071	5	10.0	0.0	12.0	17.0	2.0	2.0	0.0	0.0	0.0	
10:03:00.028	6	10.0	0.0	0.0	0.0	0.0	8.0	8.0	0.0	0.0	
10:03:01.019	7	2.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	

Note: due to the large amount of columns that are displayed here the precision of the displayed data may vary. For instance, a rate of 23.4 per second may be displayed using decimals, but a rate of 2345.6 will be displayed without decimals as 2345.

1.31.8 flatfile.html

'flatfile.html' contains Vdbench generated information in a column by column ASCII format. The first line in the file contains a one word 'column header name'; the rest of the file contains data that belongs to each column. The objective of this file format is to allow easy transfer of information to a spreadsheet and therefore the creation of performance charts.

See '[Selective flatfile parsing](#)'.

This format has been chosen to allow backward compatibility with future changes. Specifically, by making data selection column header-dependent and not column number-dependent, we can assure that modifications to the column order will not cause problems with existing data selection programs. On Solaris only, storage performance data extracted from Kstat is written to the flat file, along with CPU utilization information like user, kernel, wait, and idle times. Flatfile.html data is written for the original RAW I/O Vdbench functionality (SD/WD/RD) and for file system testing using FSD/FWD/RD parameters.

1.32 Sample parameter files

These example parameter files can also be found in the installation directory.

There is a larger set of sample parameter files in the /examples/ directory inside your Vdbench install directory.

- [Example 1](#): Single run, one raw disk
- [Example 2](#): Single run, two raw disk, two workloads.
- [Example 3](#): Two runs, two concatenated raw disks, two workloads.
- [Example 4](#): Complex run, including curves with different transfer sizes
- [Example 5](#): Multi-host.
- [Example 6](#): Swat trace replay.
- [Example 7](#): File system test. See also [Sample parameter file](#):

1.32.1 Example 1: Single run, one raw disk

```
*SD:    Storage Definition
*WD:    Workload Definition
*RD:    Run Definition
*
sd=sd1,lun=/dev/rdisk/c0t0d0s0,size=4m
wd=wd1,sd=sd1,xfersize=4096,readpct=100
rd=run1,wd=wd1,iorate=100,elapsed=10,interval=1
```

Single raw disk, 100% random read of 4KB blocks at I/O rate of 100 for 10 seconds

1.32.2 Example 2: Single run, two raw disk, two workloads.

```
sd=sd1,lun=/dev/rdisk/c0t0d0s0
sd=sd2,lun=/dev/rdisk/c0t0d1s0
wd=wd1,sd=sd1,xfersize=4k,readpct=80,skew=40
wd=wd2,sd=sd2,xfersize=8k,readpct=0
rd=run1,wd=wd*,iorate=200,elapsed=10,interval=1
```

Two raw disks: sd1 does 80 I/O's per second, read-to-write ratio 4:1, 4KB blocks. sd2 does 120 I/Os per second, 100% write at 8KB blocks.

1.32.3 Example 3: Two runs, two concatenated raw disks, two workloads.

```
sd=sd1,lun=/dev/rdisk/c0t0d0s0
sd=sd2,lun=/dev/rdisk/c0t0d1s0
wd=wd1,sd=(sd1,sd2),xfersize=4k,readpct=75
wd=wd2,sd=(sd1,sd2),xfersize=8k,readpct=100
```

```
rd=default,elapsed=10,interval=1
rd=run1,wd=(wd1,wd2),iorate=100
rd=run2,wd=(wd1,wd2),iorate=200
```

Run1: Two concatenated raw disks with a combined workload of 50 4KB I/Os per second, r/w ratio of 3:1, and a workload of 50 8KB reads per second.

Run2: same with twice the I/O rate.

This can also be run as:

```
rd=run1,wd=(wd1,wd2),iorate=(100,200),elapsed=10,interval=1
```

1.32.4 Example 4: Complex run, curves with different transfer sizes

```
sd=sd1,lun=/dev/rdisk/c0t0d0s0
wd=wd1,sd=sd1,readpct=100
rd=run1,wd=wd1,io=curve,el=10,in=1,forx=(1k-64k,d)
```

This generates 49 workload executions: 7 curve runs (one to determine max I/O rate and 6 data points for 10, 50, 70, 80, 90, 100%) for 7 different transfer sizes each. First 7 runs for 1KB, then 7 runs for 2KB, etc.

Add 'forthreads=(1-64,d)', and we go to $7 * 49 = 343$ workload executions. This is why it is helpful doing a simulated run first by adding '-s' to your execution: 'vdbench -fparmfile -s'.

1.32.5 Example 5: Multi-host

- * This test does a three second 4k read test from two hosts against the same file.
- * The 'vdbench=' parameter is only needed when Vdbench resides in a different directory on the remote system.
- * You yourself are responsible for setting up RSH (default) or SSH access to your remote system. If your remote system does NOT have an RSH daemon, you may use the [Vdbench RSH daemon](#) by starting './vdbench rsh' once on your target system.

```
hd=default,vdbench=/home/user/vdbench,user=user
hd=one,system=systema
hd=two,system=systemb
sd=sd1,host=*,lun=/home/user/junk/vdbench_test,size=10m
```

```
wd=wd1,sd=sd*,rdpct=100,xf=4k
```

```
rd=rd1,wd=wd1,el=3,in=1,io=10
```

1.32.6 Example 6: Swat I/O trace replay

*Example 6: Swat I/O trace replay

```
rg=group1,devices=(123,456,789)
sd=sd1,lun=/dev/rdisk/c0t0d0s0,replay=group1
sd=sd2,lun=/dev/rdisk/c1t0d0s0,replay=group1
wd=wd1,sd=sd1
rd=run1,wd=wd1,elapsed=9999,interval=10,replay=/tmp/flatfile.bin.gz
```

- * Replay the workload of device numbers 123, 456 and 789 from the Swat
- * flatfile.bin.gz file on luns /dev/rdisk/c0t0d0s0 and /dev/rdisk/c1t0d0s0

1.32.7 Example 7: File system test

See also file [file system sample parameter file](#):

*Example 7: File system testing

```
fsd=fsd1,anchor=/dir1,depth=2,width=2,files=2,size=128k

fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2

rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=10,interval=1
```

*

- * This parameter file will use a directory structure of 4 directories and 8 files
- * The RD parameter 'format=yes' causes the directory structure to be completely
- * created, including initialization of all files to the requested size of 128k.
- * After the format completes the following will happen for 10 seconds at a rate
- * of 100 reads per second:
 - * Start two threads (threads=2; 1 thread is default).
 - * Each thread:
 - * Randomly selects a file (fileselect=random)
 - * Opens this file for read (operation=read)
 - * Sequentially reads 4k blocks (xfersize=4k) until end of file (size=128k)
 - * Closes the file and randomly selects another file.

*

* Directory structure:

*

- * find dir1 | grep file
- * dir1/vdb_control.file
- * dir1/vdb1_1.dir/vdb2_1.dir/vdb_f0001.file
- * dir1/vdb1_1.dir/vdb2_1.dir/vdb_f0002.file
- * dir1/vdb1_1.dir/vdb2_2.dir/vdb_f0001.file

```
* dir1/vdb1_1.dir/vdb2_2.dir/vdb_f0002.file
* dir1/vdb1_2.dir/vdb2_1.dir/vdb_f0001.file
* dir1/vdb1_2.dir/vdb2_1.dir/vdb_f0002.file
* dir1/vdb1_2.dir/vdb2_2.dir/vdb_f0001.file
* dir1/vdb1_2.dir/vdb2_2.dir/vdb_f0002.file
*
.
```

1.33 Permanently override Java socket port numbers.

You can temporarily override the port numbers used by Vdbench to communicate between the master and the slaves (5570), or the port numbers used for Vdbench's own RSH 'daemon' (5560).

To do this you must create file 'portnumbers.txt' in the Vdbench installation directory, or if you run Vdbench multi-host, in each Vdbench installation directory.

Content of this file:

```
masterslaveport=nnnn
rshdaemonport=nnnn (Yes, this is a hard coded 'daemon' spelling error )
```

Make sure that if you have some firewall software installed that Java is allowed to use these ports.

1.34 Java Runtime Environment

It is expected that the Java Runtime Environment (JRE) or Java Development Kit (JDK) already has been installed. See the following web pages:

- <http://java.sun.com/javase/downloads/index.jsp> for Solaris, Windows, and Linux.
- <http://www-106.ibm.com/developerworks/java/jdk/index.html> for Aix, Linux, Windows
- <http://www.hp.com/products1/unix/java/> for HP/UX

Follow the vendor's installation instructions.

1.35 Solaris

When not running MAX I/O rates, Vdbench uses Solaris 'sleep' functions. Because the default granularity of the clock timers is one 'clock tick' every 10 milliseconds, it is recommended to add 'set hires_tick=1' to */etc/system* and reboot.

This allows I/O's to be started about 10 milliseconds closer to their expected start time.

1.36 HP/UX

To get the best performance possible on an HP/UX system, some work needs to be done after Java has been installed. JavaOOB (out-of-box) must be installed. It is available from <http://www.hp.com/products1/unix/java/>.

This information is about 7-8 years old, so it may no longer be valid.

2 Vdbench Workload Compare

This tool compares two sets of Vdbench output directories and shows the delta iorate or fwdrate and response time and optionally the data rate in 9 different colors: light green is good, dark green is better, red is bad, etc. Look at sample screen below.

You may give the tool either two Vdbench output directories, e.g. /run1 and /run2, or the parents of several Vdbench output directories, e.g. /test1 and /test2, where test1 and test2 have one or more subdirectories, e.g. /test1/run1, /test1/run2, etc.

To run Vdbench workload compare, enter './vdbench compare'

Vdbench Workload Comparator. Old: C:\junk\sbm-4000\R1T3_regT3_32k_36G10rpm_43t_tseekd - New: C:\junk\sbm-4000\R1T...												
File Options												
Old directory	New directory	Compare	Exit	+4.0%+	+3.0%	+2.0%	+1.0%	0.0%	-1.0%	-2.0%	-3.0%	-4.0%+
Subdirect...	Run	xfersize	Old iorate	New iorate	Old resp	New resp	Delta resp	Old iops	New iops	Delta iops		
/	run1_seek_sys(50%)	524288	curve	curve	1783.3	1828.1	-2.5%	141.4	138.8	-1.8%		
/	run1_seek_sys(50%)(10%)	524288	20.0	20.0	25.7	16.9	+34.0%	19.8	19.5	-1.7%		
/	run1_seek_sys(50%)(50%)	524288	70.0	70.0	59.0	39.6	+32.8%	69.4	70.3	+1.3%		
/	run1_seek_sys(50%)(75%)	524288	110.0	110.0	137.0	83.2	+39.3%	111.8	110.8	-.9%		
/	run1_seek_sys(50%)(85%)	524288	120.0	120.0	185.3	117.3	+36.7%	121.2	119.2	-1.6%		
/	run1_seek_sys(50%)	16384	curve	curve	211.0	228.3	-8.2%	1218.3	1085.1	-10.9%		
/	run1_seek_sys(50%)(10%)	16384	130.0	110.0	4.1	3.8	+7.2%	129.4	108.3	-16.3%		
/	run1_seek_sys(50%)(50%)	16384	610.0	550.0	13.1	11.8	+10.1%	615.8	546.0	-11.3%		
/	run1_seek_sys(50%)(75%)	16384	920.0	820.0	28.2	24.5	+13.1%	917.6	819.5	-10.7%		
/	run1_seek_sys(50%)(85%)	16384	1100.0	930.0	55.4	36.5	+34.1%	1095.7	932.6	-14.9%		
/	run1_seek_sys(50%)	8192	curve	curve	216.3	215.2	+5%	1190.0	1131.9	-4.9%		
/	run1_seek_sys(50%)(10%)	8192	120.0	120.0	3.5	3.7	-4.5%	119.7	120.4	+6%		
/	run1_seek_sys(50%)(50%)	8192	600.0	570.0	10.9	11.0	-.9%	602.6	572.9	-4.9%		
/	run1_seek_sys(50%)(75%)	8192	900.0	850.0	24.0	22.4	+6.9%	895.5	848.9	-5.2%		
/	run1_seek_sys(50%)(85%)	8192	1100.0	970.0	52.1	37.6	+27.9%	1100.4	968.5	-12.0%		
/	run1_seek_sys(50%)	512	curve	curve	185.7	178.4	+3.9%	1384.7	1365.4	-1.4%		
/	run1_seek_sys(50%)(10%)	512	140.0	140.0	3.1	3.3	-6.1%	139.1	140.6	+1.1%		
/	run1_seek_sys(50%)(50%)	512	700.0	690.0	10.2	11.2	-9.3%	702.9	688.6	-2.0%		
/	run1_seek_sys(50%)(75%)	512	1100.0	1100.0	24.9	28.4	-14.0%	1100.5	1105.5	+5%		
/	run1_seek_sys(50%)(85%)	512	1200.0	1200.0	35.0	43.8	-25.0%	1202.4	1198.6	-.3%		

3 Vdbench flatfile selective parsing

It took me about 7 years, but I finally made some time to create a simple program that takes the flatfile, picks out the columns and rows that the user wants, and then writes it to a tab delimited file.

Usage: `./vdbench parse input output 'filter pairs' show 'output columns'`

- 'input': Input file name, e.g. `output/flatfile.html`
- 'output': Name of the tab delimited file, or '-' for stdout.
- 'Filter pair': Contains a column header and a filter value that is used to select data from flatfile. There can be as many filter pairs as you need.
 Note: The 'avg_' row by default is never written to the output file. To get ONLY the 'avg_' row, specify 'interval avg' as a filter pair.
- 'show': A separator between filter pairs and output columns.
- 'output columns': Specifies the column header(s) from the flatfile that need to be written to the tab delimited file. You can specify as many columns as you need.

Example:

`./vdbench parse output/flatfile.html - show tod rate resp`

(Tab delimited) results:

tod	rate	resp
11:29:11.046	100.0000	0.8873
11:29:12.015	100.0000	0.2297
11:29:13.015	100.0000	0.2301

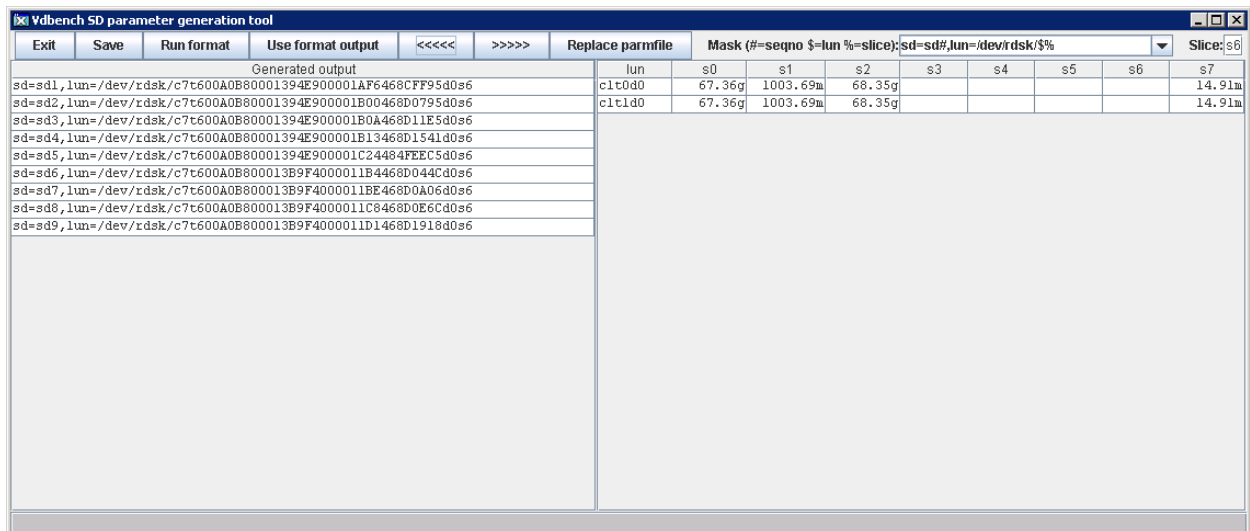
4 Vdbench SD parameter generation tool.

The creation of SD parameters can become quite cumbersome on Solaris since it uses very long hexadecimal target numbers as part of their device names.

This tool is also available for Linux and Windows, though the '50 or more hexadecimal characters per device name' problem does not exist there.

The SD parameter generation tool will assist you in the selection of the proper device names and then the creation of a set of SD parameters (or other parameters, see below).

The program either takes in a file containing the output of 'format << EOF', or runs the command itself. The 'prtvtoc' command is run when available for each device found so that it can display the partition sizes for partitions 0 through 7.



Click and select one or more of the device names on the right side of the window, then click on the '<<<<<' button and the selected device(s) will be added to the list of SDs. A double click will immediately move the selected device. Click 'Save' to then save the selected SDs into a file. The 'Replace parmfile' button will read an existing parameter file, and replace the existing SDs within that file with the new SD parameters just created.

Note: since the new SDs are all labeled *sd1* through *sdn*, SD parameters in this parameter file that use different SD names can no longer be referenced, e.g. *wd=wd1,sd=disk1*.

You can also use this program to use the selected device names for anything else. For that the program takes as input a *mask*. By default the mask contains:

```
sd=sd#, lun=/dev/rdisk/%s
```

Where:

- # is replaced by a sequence number, starting from 1
- \$ is replaced by the selected device name.

- % is replaced by the entered partition/slice number.

You can modify the mask either by directly entering it in the GUI, or by adding it to file 'build_sds.txt' in your Vdbench installation directory.

Any mask containing '<' and '>' will be split in two, with the objective of the left mask (until '<') being used for the first disk, and the right side of the mask (inside '<' and '>') being used for all other disks. This allows the creation of a single command with multiple disks and command continuation characters ('\').

A mask containing the '<' and '>' characters allows you to create a multi-line command, for instance:

`newfs </dev/dsk/%>` used for two devices will create (after 'Save') a file containing:

```
newfs /dev/dsk/clt0d0s6 \
/dev/dsk/clt1d0s6
```

Below are the currently defined masks in file 'build_sds.txt'. If you have any other ideas for things to add let me know.

```
* This is the hardcoded default:
sd=sd#,lun=/dev/rdisk/%

* Placed in a script this should label this disk
printf "label\nyes\nquit\n" | format -d $

* This results in only a list of disks:
$

* Create a file system for one disk:
printf "yes\n" | newfs /dev/dsk/%

* Create a file system for multiple disks:
newfs </dev/dsk/%>

* Testing:
ls -l </dev/dsk/%>
```

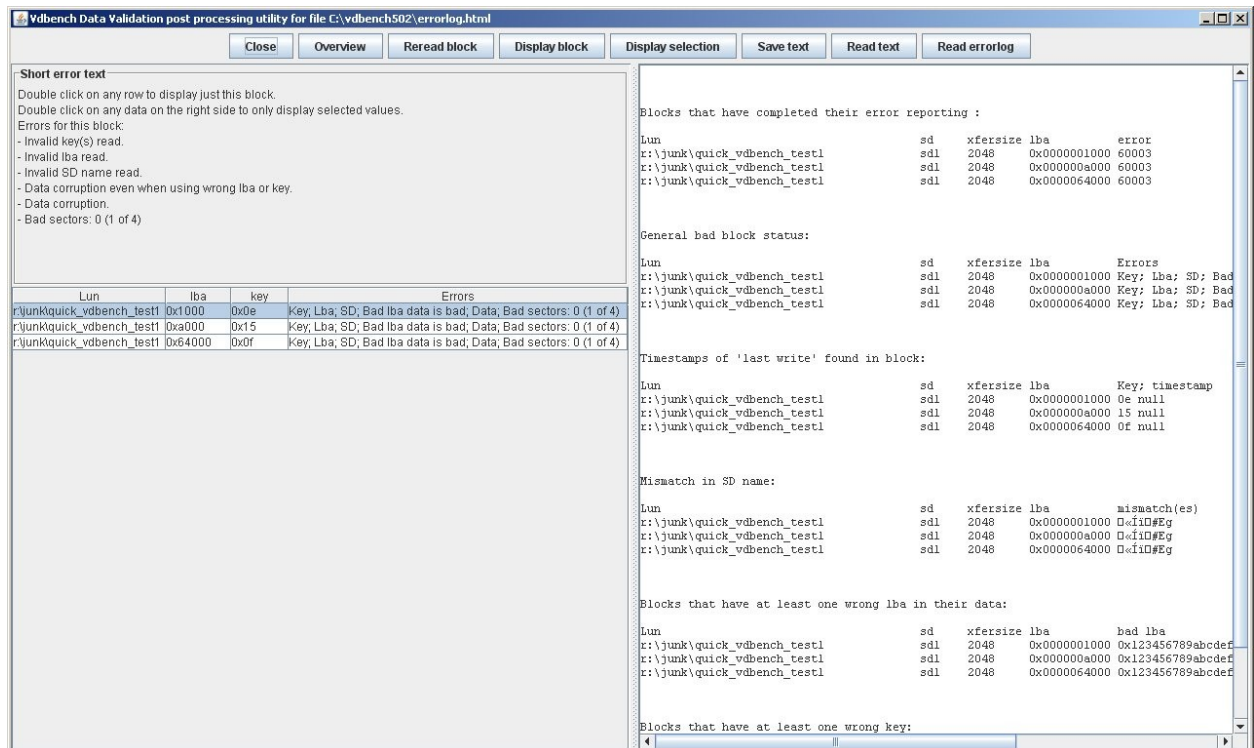
5 Vdbench Data Validation post-processing tool.

When Data Validation does find a data corruption problem, file errorlog.html will contain all the gory detail about the data that Vdbench expects, and the data that Vdbench has found on the data block it just read and compared.

Things get ugly when scrolling through hundreds and some times thousands of lines of output, so for that I wrote a primitive tool that allows you to quickly look at all the output trying to help you understand what's going on.

Run './vdbench dvpost' or './vdbench /output/errorlog.html' and Vdbench brings up the following window (sorry, it's hard to read here in the doc).

The errorlog.html file shown here is included in the /examples directory. It's probably easier for you to just run './vdbench dvpost examples/errorlog.html'.



Buttons:

- Close: closes the tool.
- Overview: shows an overview of the errors that Vdbench has found.
- Reread block: Some times data corruptions are intermittent. For instance, the corruption happened in file system or storage cache. Select a row from the list of failed data blocks on the left, click 'Reread block', Vdbench then will re-read the block (using the './vdbench print' function) and displays it on the right. Before you do this however, make sure that the status of the storage in question has not changed.

- **Display Block:** this button allows you to display all available information related to the currently selected 'failed data block' on the right side of the window. Since Data Validation is multi-threaded, having multiple blocks fail at the same time can cause errorlog.html to have numerous different errors all intermixed. This GUI can help you filter out just those pieces of information that you need.
- **Display selection:** On the right side of your screen, highlight any piece of information that you want, and then click this button (or just double click on any value). Vdbench will then display only those values that you selected. The button then will change to Reset selection, which you click if you want to clear your selection.
- **Save text:** this allows you to save the currently displayed contents of the right side of your window.
- **Read text:** this allows you to read and display any disk file.
- **Read errorlog:** this allows you to display the complete contents of errorlog.html.

There are of course numerous reasons for data corruptions. A few that I can think of right now: Block never arrived at the storage; block was written in the wrong place; the block was overwritten because a different write ended up on the wrong place; the wrong block was read; only a piece of the block (for instance 4k) is misplaced or overlaid; after the data buffer was filled data was not copied from processor cache to memory; device drivers picking up the wrong memory pages; corruptions due to any kind of transmission error anywhere; loose cables; block partially written to storage due to a power failure; indeed just a bad disk; etc, etc.

Short error text: This shows the errors found for the currently selected data block.

This gives you a list of errors that may be displayed for a data block in the list of failed data blocks.

- **Invalid key(s) read:** check the Vdbench documentation for the meaning of Data Validation keys. This tells you that the key value that Vdbench expected was not found in the data block.
- **Invalid lba read:** each 512-byte sector contains the logical byte address of that sector. If the value there does not match something clearly is wrong.
- **Invalid SD or FSD name read:** the SD or FSD name is also written in each sector. If you want block1 of sd1, but get block1 of sd2 the lba will match, but the SD name won't.
- **Data corruption even when using wrong lba or key:** this is there to answer the question "if I read the wrong block, are the contents of that wrong block even good or bad?". The data pattern store in each 512-byte sector is generated using Linear Feedback Shift Register logic, which uses as seed the lba, key, and SD or FSD name. Using these values from the block that was read (not from what was requested), Vdbench validates the data in the block for a second time. It gets confusing, but reporting that the 'bad' block is 'good' can be useful.
- **Data corruption:** Each sector contains a 32-byte header and then 480 bytes of the LFSR data pattern. Any error in these 480 bytes will be reported as Data corruption.
- **Bad sectors:** this will tell you for a data block how many 512-byte sectors had errors. For instance, a 1mb block consists of 2048 sectors. If they're all good you won't see this block, but if only some of them are bad you'll have a partial data corruption for this block. Some times you can also see 'incomplete' here. Depending on the maximum

amount of data errors you allow (data_errors=) and how many concurrent threads you are running it can happen that Vdbench aborts before Vdbench is able to report each individual sector. If you see 'bad sectors, 2048 of 2048' you know that all sectors are bad, however, if you see 'bad sectors 8 of 2048 (incomplete)' there may be more bad sectors than that Vdbench had the chance to report. For this, use the 'Reread block' above.

- Not all sectors have been reported: see Bad sectors above.
- At least one single bit error: this is a quick warning that there was only one single bit difference between what we expected and what was read.

6 Vdbench GUI Documentation

6.1 Overview

Note: The GUI is only meant to help you quickly become familiar with the Vdbench SD, WD, and RD parameters. The GUI only uses a very small subset of the options available to Vdbench.

The Vdbench GUI application is a user-friendly application designed to provide a basic subset of the storage performance measurement and benchmarking capabilities of Vdbench to novice users.

The GUI is started by entering ‘vdbench gui’ or ‘vdbench.bat gui’

Specifically, the Vdbench GUI allows users to construct a parameter file, which is then used to run Vdbench transparently, with resulting output, displayed in a clear, easily perusable, graphical format. This has the added benefit of allowing the user to observe how correctly formatted Vdbench parameter files are constructed, thereby helping him to subsequently create more complex parameter files for use with the full, command line version of Vdbench, and hopefully reducing the learning curve associated with the command line version of this powerful tool.

This chapter illustrates the features and functionality provided by the Vdbench GUI, and is divided into the following sections:

- Entering Operating Parameters
- Running Vdbench
- Application Menus

6.2 Entering Operating Parameters

As visible in Fig. 1, the Vdbench GUI consists of two main panels. The first, or topmost, of these is entitled *Run Options*. It is here that the user specifies the device(s) or file(s) of interest, and workload execution parameters.

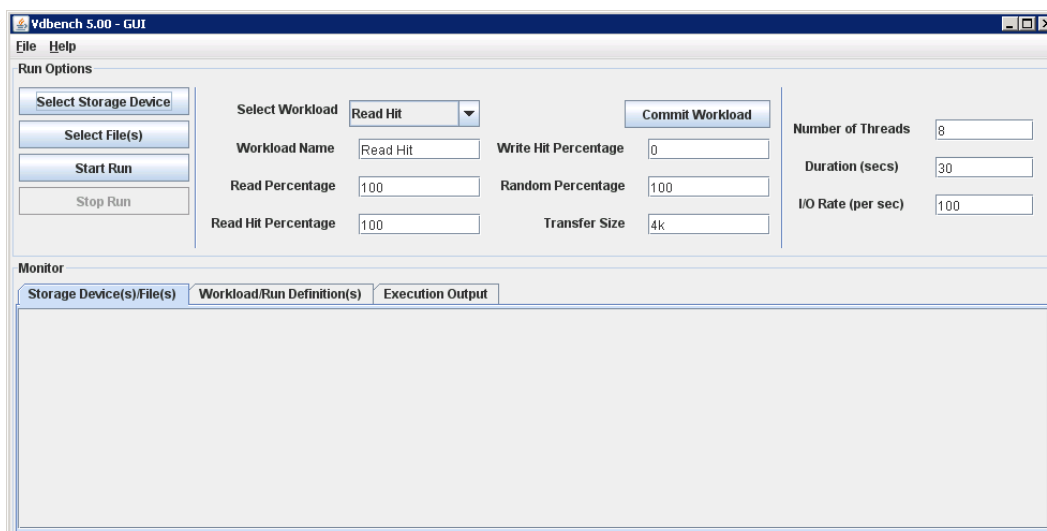


Fig. 1 The Vdbench GUI, as seen at startup.

6.3 Storage Device Selection

The top button on the left-hand side of the *Run Options* panel, labeled *Select Storage Device*, allows the user to specify the storage device(s) on which Vdbench is to run. When clicked, a Storage Device Selector window appears, as seen in Fig. 2.

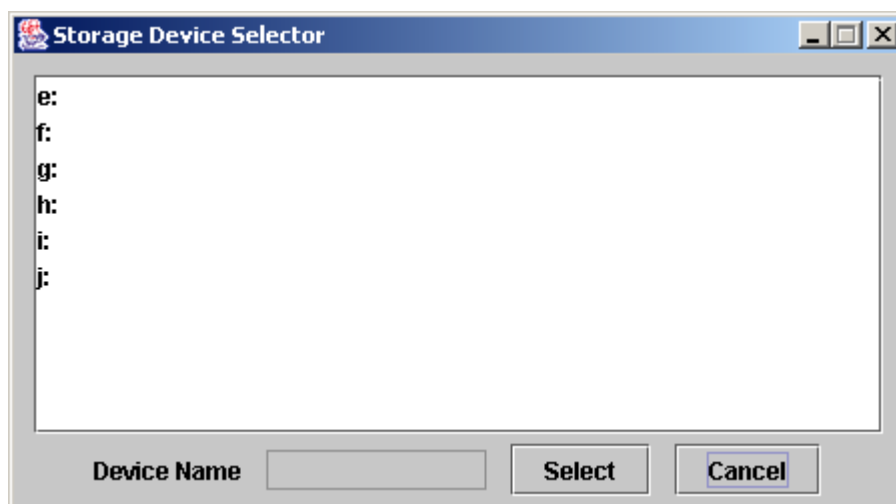


Fig. 2 Vdbench GUI Storage Device Selection Window.

Visible in Fig. 2 are a series of storage devices, specified as appropriate for the operating system on which the Vdbench GUI is running (Microsoft Windows 2000, in the example shown). (Note that, by default, any drive designated with the letter “C” is not shown. Due to the fact that Vdbench can, if used carelessly, result in the corruption of stored data, this has been done to protect the user from accidentally selecting the disk drive which is commonly the user’s “local” drive.)

A single drive on the list may be selected by left clicking on the corresponding entry. The entry then appears in the *Device Name* field at the bottom of the window, and the user may conclude the device selection operation by clicking on the *Select* button at the bottom of the window.

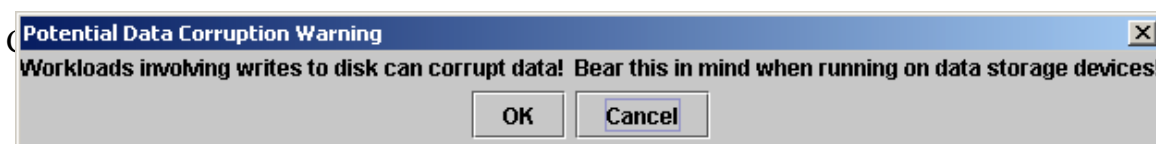


Fig. 3 Data corruption warning after device selection.

At this time, as shown in Fig. 3, a dialog window appears, warning the user that workloads involving write operations can corrupt user data. This is simply to make the novice user aware of the potential dangers inherent in the use of this tool. If the user clicks *Cancel* to dismiss the warning, the selected device will not be added to the list of chosen devices. Clicking *OK* results in the addition of the selected device to the *Storage Device(s)/File(s)* tab of the *Monitor* section of the application, as shown below in Fig 5. Note that if the user wishes to add multiple devices, he must repeat this process for each device he wishes to add.

6.4 File Selection

The second button on the left-hand side of the *Run Options* panel, labeled *Select File(s)*, allows the user to specify the file(s) on which Vdbench is to run. Note that this option is primarily for demonstration purposes, since file I/O is usually served from system cache. When clicked, a File Selection window appears, as seen in Fig. 4.

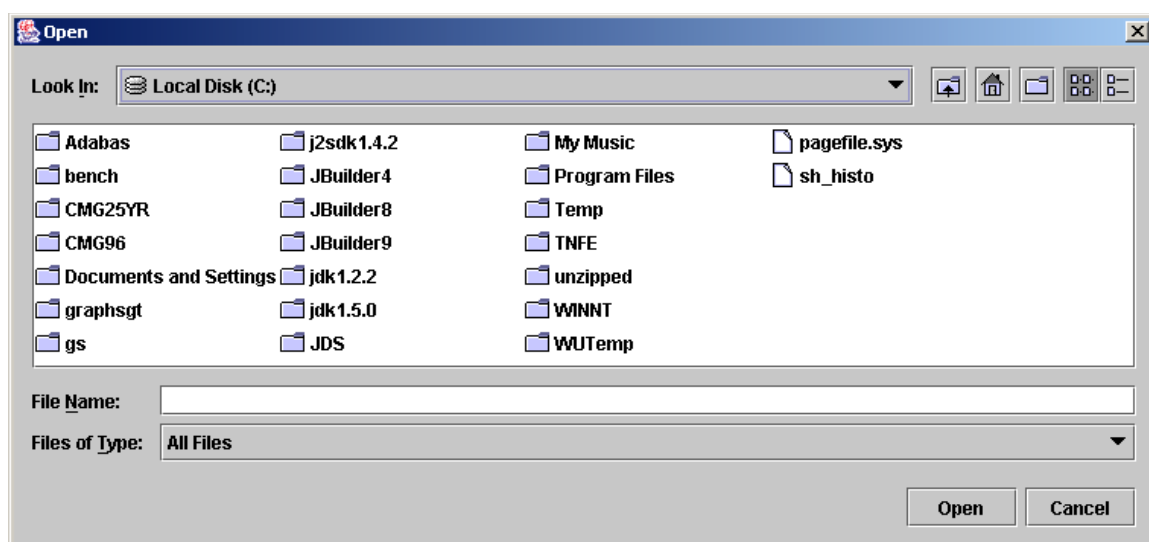


Fig. 4 File Selection window.

From this window, one or more files may be selected. Multiple selection is accomplished by use of the *Control* or *Shift* keys, allowing selection of multiple, non-consecutive entries, and multiple, consecutive entries, respectively. Clicking *Open* results in the addition of the selected file(s) to the *Storage Device(s)/File(s)* tab of the *Monitor* section of the application.

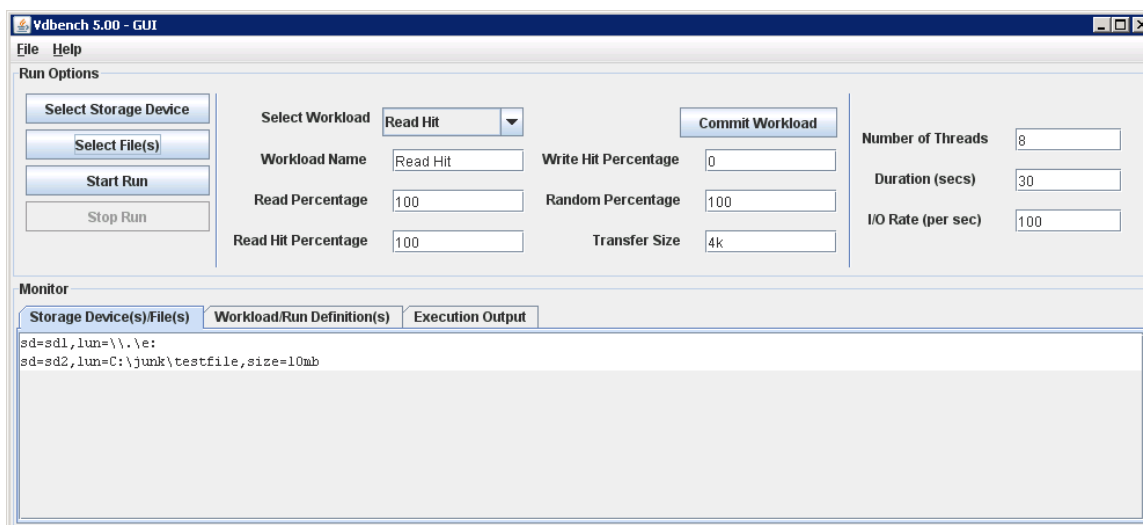


Fig. 5 The Vdbench GUI, showing user-selected storage devices and file.

6.5 Storage Device and File Definition Deletion

Note that storage device or file entries, which have been added to the Monitor panel, may be subsequently deleted. An entry may be deleted by placing the mouse pointer on an entry in the *Storage Device(s)/File(s)* tab of the *Monitor* panel and right clicking on it. A popup window will then appear, asking the user if he wishes to delete the selected entry. Once the user confirms his intent, the desired entry is deleted.

6.6 Workload Definition Specification

The middle portion of the Run Options panel, as seen in Fig. 1, is the Vdbench GUI workload definition specification area. Each of the available workload parameters is described below:

- **Select Workload List** This drop-down list contains six predefined workloads for user convenience. They include *Read Hit*, *Read Miss*, *Write Hit*, *Write Miss*, and *Sequential Read* and *Sequential Write* workloads. Selection of a particular workload from this list specifies a set of values for the six remaining workload parameter fields. However, once a choice has been made, any of the workload parameter fields, including *Workload Name*, may be modified by the user.
- **Workload Name** Specifies the name for the set of workload parameters as currently defined in the other workload parameter fields.
- **Read Percentage** Specifies the percentage of the workload corresponding to read activity.
- **Read Hit Percentage** Specifies the percentage of the workload corresponding to read activity resulting in an attempted cache read hit.
- **Write Hit Percentage** Specifies the percentage of the workload corresponding to write activity resulting in a cache write hit.
- **Random Percentage** Specifies the percentage of the workload corresponding to non-sequential (hence random) activity. In this case, random refers to activity which involves the disk read/write head seeking to a new location on the disk. Hence, such activity involves more performance-impairing overhead, in the form of mechanical latencies, than sequential activity.

- **Transfer Size (KB)** Specifies the block transfer size in kilobytes for the workload. Note that multiple values may be specified, separated by commas or spaces. Units are not to be included.

After a workload has been specified, it must be committed by clicking on the *Commit Workload* button. This adds the corresponding workload definition to the *Workload/Run Definitions* tab of the application's *Monitor* panel, shown below, with the corresponding Run Definition added to the same panel, but below it.

Note that if a new workload is created which shares the name of a previously committed workload, the new workload of the same name will overwrite its predecessor. Thus, if it is desired to create a number of read hit workloads, for example, the user will have to distinctly name each such workload. This is shown in Fig. 6, where the user has defined three "read hit" workloads, each with a different read percentage value.

6.7 Workload Definition Modification

Once a workload has been committed, it may be modified. This is accomplished by left clicking on the desired entry on the *Workload/Run Definition(s)* tab of the *Monitor* panel. This action causes the workload parameter fields of the *Run Options* panel to be populated with the corresponding parameters of the selected workload. The parameters may then be modified as the user wishes and set by clicking on the *Commit Workload* button. However, note that if the **name** of the selected workload is changed, a new workload with the new name and current parameters will be appended to the list of workloads present on the *Workload/Run Definition(s)* tab of the *Monitor* panel.

6.8 Workload Definition Deletion

In a fashion identical to Storage Device and File Definition deletion, Workload Definitions may also be removed. As before, right clicking on the desired entry in the *Workload/Run Definition(s)* tab of the *Monitor* panel elicits a popup window, asking the user if he wishes to delete the selected entry. Once the user confirms his intent, the desired entry is deleted. However, in this case, since a corresponding Run Definition exists for each Workload Definition, deletion of a given Workload Definition means that the corresponding Run Definition is also deleted.

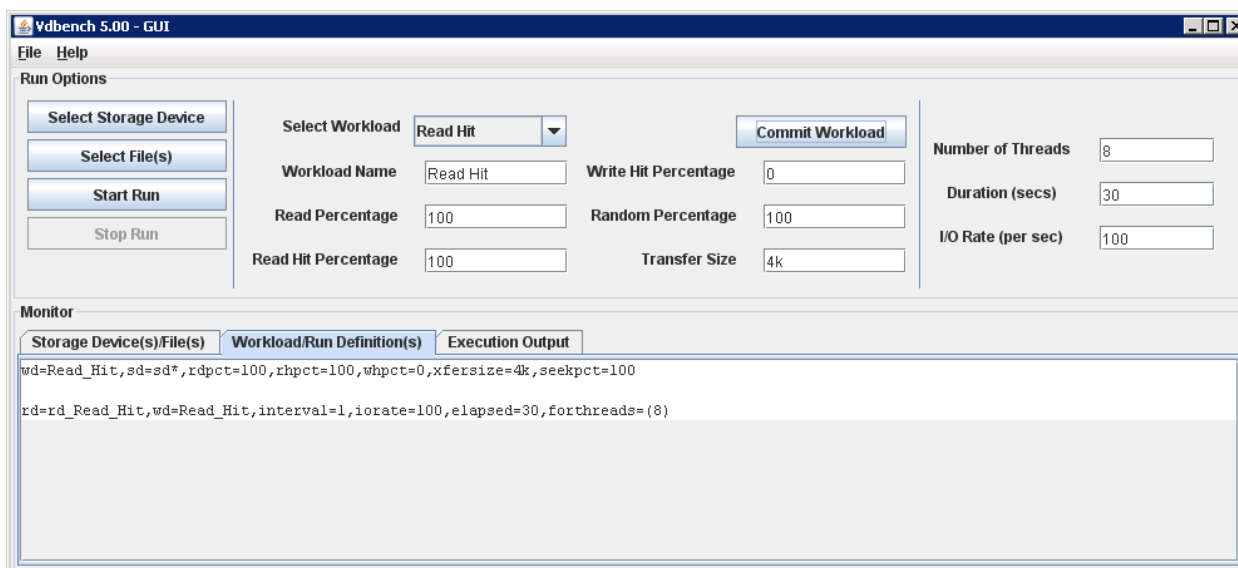


Fig. 6 The GUI, showing several Workload Definitions, along with their corresponding Run Definitions.

6.9 Run Definition Specification

The right-hand portion of the Run Options panel, as seen in Fig. 1, is the Vdbench GUI Run Definition specification area. Each of the available Run Definition parameters is described below.

- **Number of Threads** Specifies the number of concurrent threads of execution. One or more numeric values may be entered, separated by commas or spaces.
- **Duration (secs)** Determines the duration of each individual workload, in seconds, listed as an entry in the *Workload/Run Definitions* tab of the application's *Monitor* panel. Only a single numeric value may be entered.
- **I/O Rate (per sec)** Specifies the number of I/Os per second for the workload with which it is associated. This parameter may be a single numeric value, or may consist of the words "MAX" or "CURVE", as described in the Vdbench documentation. Note: Use of "MAX" can result in very high data rates and CPU utilization.

Note that a given set of Run Definition parameters will apply to **every** workload definition committed. Furthermore, if any of the Run Definition parameters are changed prior to clicking on the *Commit Workload* button, any pre-existing Run Definitions will be modified to reflect any such change. **This will occur even if no Workload Definition parameters have been modified.**

6.10 Running Vdbench

Once one or more storage devices and or files have been selected, and at least one workload has been committed, the Vdbench GUI is ready to run.

6.11 Starting a Run

At the time the *Start Run* button is clicked, a standard Vdbench parameter file, containing the user-specified Storage, Workload, and Run Definitions is written to the user's official temporary directory, where it is read by Vdbench. Then, all subsequent Vdbench output is presented on the *Execution Output* tab of the *Monitor* Panel. This output is identical to that produced by Vdbench when it is run from the command line.

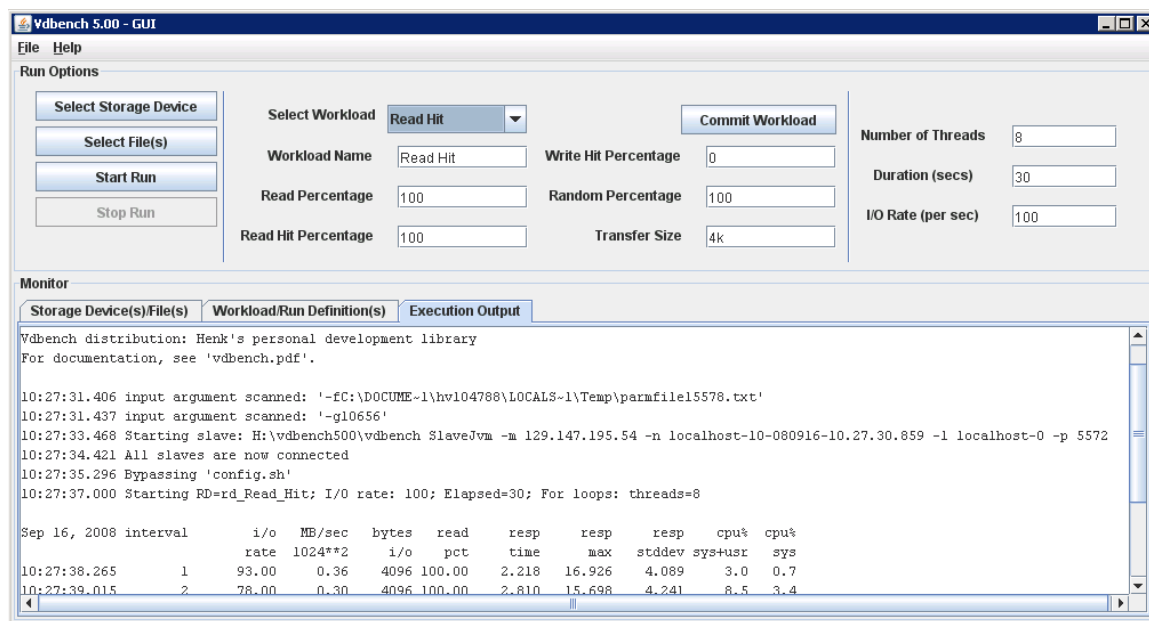


Fig. 7 The Vdbench GUI, showing Vdbench output in progress.

6.12 Stopping a Run

At this time, the *Stop Run* button of the *Run Options* panel becomes enabled to allow the user to gracefully terminate a Vdbench run in progress, if he desires. This option does not prevent the user from subsequently modifying his current run and restarting it.

6.13 Application Menus

In addition to the functionality available through its console buttons, the Vdbench GUI also offers additional functionality through its *File* and *Help* menus.

6.13.1 File Menu

The *File* menu allows the user to name and save the parameter files produced by the Vdbench GUI, and also to open a previously saved parameter file to run. When a previously saved parameter file is opened, the Storage Device/File and Workload/Run Definition(s) which it contains are displayed on the *Storage Device(s)/File(s)* and *Workload/Run Definition(s)* tabs of the *Monitor* panel.

Note, however, that parameter files created by the Vdbench GUI make use of only a limited subset of the commands available when using the command line version of Vdbench. Therefore, it is essential that users do not, as a rule, open parameter files written by hand for use with the Vdbench GUI. For this reason, files created by the Vdbench GUI contain a specific header which, if not present, causes the Vdbench GUI to alert the user that the intended parameter file may have been written by hand, and may run with potentially disastrous results.

The *File* menu also allows the user to exit the application. Note that this may also be accomplished by simply clicking on the application's "close" box in the upper right-hand corner of the application frame.

6.13.2 Help Menu

The *Help* menu provides the user with very basic information about the Vdbench GUI. The *About...* item provides the user with version information for the application, as well as the location of the user's Java Runtime Environment, and a recommendation that the user run the application with Java version 1.4.2, or higher.