Vdbench 5.04.03

Table of Contents

1VDBENCH: DISK I/O WORKLOAD GENERATOR .....................................................11

1.1Introduction.............................................................................................................................11

1.2Objective..................................................................................................................................11

1.3Terminology.............................................................................................................................12

1.4Summary of changes since release 5.03.................................................................................12

1.5Summary of changes since release 5.02.................................................................................13

1.6Installing Vdbench..................................................................................................................14

1.7How to start Vdbench:............................................................................................................14

1.8Execution parameter overview..............................................................................................15

1.8.1Execution Parameters.........................................................................................................15

1.9Parameter File(s).....................................................................................................................17

1.9.1Variable substitution..........................................................................................................17

1.9.2Multi-host parameter replication........................................................................................17

1.9.3include=parmfile................................................................................................................18

1.9.4General Parameters: Overview..........................................................................................18

1.9.5Host Definition (HD) Parameter overview........................................................................19

1.9.6Replay Group (RG) Parameter Overview..........................................................................20

1.9.7Storage Definition (SD) Parameter Overview...................................................................21

1.9.8File system Definition (FSD) Parameter Overview...........................................................22

1.9.9Workload Definition (WD) Parameter Overview..............................................................22

1.9.10File system Workload Definition (FWD) Parameter Overview......................................22

1.9.11Run Definition (RD) Parameter Overview (For raw I/O testing)....................................23

1.10Execution parameter detail..................................................................................................25

1.10.1'-f xxx ': Workload Parameter File(s) ..............................................................................25

1.10.2'-oxxx': Output Directory ................................................................................................26

1.10.3'-v': Activate Data Validation...........................................................................................26

1.10.4'-j': Activate Data Validation and Journaling...................................................................27

1.10.5'-s': Simulate Execution....................................................................................................27

1.10.6'-k': Kstat Statistics on Console........................................................................................27

1.10.7'-m nn': Multi JVM Execution.........................................................................................27

1.10.8‘-t’: Sample Vdbench execution......................................................................................28

1.10.9‘-e nn’ Override elapsed time..........................................................................................28

1.10.10‘-i nn’ Override report interval time..............................................................................28

1.10.11‘-w nn’ Override warmup time......................................................................................28

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

1

Vdbench 5.04.03

1.11 Vdbench utility functions....................................................................................................30

1.11.1./vdbench sds: Generate Vdbench SD parameters...........................................................30

1.11.2./vdbench dvpost: Data Validation post processing.........................................................30

1.11.3./vdbench jstack: Display java execution stacks of active Vdbench runs.......................30

1.11.4./vdbench rsh: Vdbench RSH daemon.............................................................................30

1.11.5./vdbench print: Print any block on any lun or file..........................................................31

1.11.6./vdbench edit: Simple full screen editor, or ‘back to the future’....................................31

1.11.7./vdbench compare: Compare Vdbench test results.........................................................31

1.11.8./vdbench parse: Parse Vdbench flatfile...........................................................................31

1.12General parameter detail.....................................................................................................32

1.12.1‘include=parmfile’ ..........................................................................................................32

1.12.2'data\_errors=xxx': Terminate After Data Validation or I/O errors..................................32

1.12.3''force\_error\_after': Simulate Data Validation Error........................................................33

1.12.4‘swat=xxx’: Create performance charts using Swat........................................................33

1.12.5'startcmd=' and 'endcmd='................................................................................................33

1.12.6'pattern=: Data Pattern to be used....................................................................................35

1.12.7‘compratio=nn’: Set compression for data patterns.........................................................35

1.12.8‘port=nnnn’: Specify port number for Java sockets.........................................................36

1.12.9‘create\_anchors=yes’: Create anchor parent directory....................................................36

1.12.10‘report=’: Generate extra SD reports.............................................................................36

1.12.11‘histogram=’: set bucket count and bucket size for response time histograms..............36

1.12.12‘formatxfersize=nnnn’...................................................................................................37

1.12.13'monitor=', External control of Vdbench termination....................................................37

1.12.14'messagescan=': suppress /var/xxx/messages scan.........................................................38

1.13Replay Group (RG) parameter detail.................................................................................39

1.14Host Definition parameter detail.........................................................................................40

1.14.1‘hd=host\_label’................................................................................................................40

1.14.2‘system=system\_name’....................................................................................................40

1.14.3‘jvms=nnn’.......................................................................................................................40

1.14.4‘vdbench=/vdbench/dir/name’.........................................................................................40

1.14.5‘shell=rsh | ssh | vdbench’................................................................................................40

1.14.6‘user=xxxx’......................................................................................................................41

1.14.7‘mount=xxx’....................................................................................................................41

1.15 Storage Definition parameter detail...................................................................................42

1.15.1'sd=name': Storage Definition Name...............................................................................42

1.15.2'lun=lun\_name': LUN or File Name................................................................................42

1.15.3 ‘host=name’ ...................................................................................................................43

1.15.4‘count=(nn,mm)’..............................................................................................................43

1.15.5'size=nn: Size of LUN or File..........................................................................................43

1.15.6'range=(min,max)': Limit Seek Range.............................................................................44

1.15.7'threads=nn': Maximum Number of Concurrent outstanding I/Os.................................44

1.15.8'hitarea=nn': Storage Size for Cache Hits........................................................................45

1.15.9'journal=name': Directory Name for Journal File............................................................45

1.15.10‘offset=’: Don’t start at byte zero of a LUN..................................................................45

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

2

Vdbench 5.04.03

1.15.11‘align=’: Determine lba boundary for random seeks.....................................................45

1.15.12‘openflags=’: control over open and close of luns or files............................................47

1.15.13streams=: Independent sequential streams.....................................................................49

1.16 Workload Definition parameter detail...............................................................................49

1.16.1'wd=name': Workload Definition Name..........................................................................50

1.16.2‘host=host\_label’.............................................................................................................50

1.16.3'sd=name': SD names used in Workload..........................................................................50

1.16.4'rdpct=nn': Read Percentage.............................................................................................51

1.16.5'rhpct=nn' and 'whpct=nn': Read and Write Hit Percentage............................................51

1.16.6'xfersize=nn': Data Transfer Size.....................................................................................51

1.16.7'skew=nn': Percentage skew ............................................................................................52

1.16.8'seekpct=nn': Percentage of Random Seeks.....................................................................52

1.16.9stride=(min,max): Skip-sequential I/O............................................................................53

1.16.10'range=nn': Limit Seek Range........................................................................................53

1.16.11‘iorate=’ Workload specific I/O rate..............................................................................53

1.16.12‘priority=’ Workload specific I/O priority. ....................................................................54

1.17Run Definition for raw I/O parameter detail.....................................................................55

1.17.1'rd=name': Run Name......................................................................................................56

1.17.2'wd=': Names of Workloads to Run.................................................................................56

1.17.3‘sd=xxx’...........................................................................................................................56

1.17.4'iorate=nn': One or More I/O rates...................................................................................56

1.17.5'curve=nn': Define Data points for Curve........................................................................57

1.17.6'elapsed=nn: Elapsed Time .............................................................................................57

1.17.7'interval=nn': Reporting Interval .....................................................................................57

1.17.8‘warmup=nn’: Warmup period........................................................................................58

1.17.9'maxdata=': stop after nnn bytes.......................................................................................58

1.17.10'distribution=xxx': I/O arrival time distribution.............................................................58

1.17.11'pause=nn': Sleep 'nn' Seconds.......................................................................................59

1.17.12Workload parameter specification in a Run Definition.................................................59

1.17.12.1‘sd=xxx’ Specify SDs to use...................................................................................60

1.17.12.2'(for)xfersize=nn': Create 'for' Loop Using Different Transfer Sizes.....................60

1.17.12.3'(for)threads=nn': Create 'for' Loop Using Different Thread Counts.....................60

1.17.12.4'(for)rdpct=nn': Create 'for' Loop Using Different Read Percentages....................61

1.17.12.5'(for)rhpct=nn': Create 'for' Loop Using Different Read Hit Percentages..............61

1.17.12.6'(for)whpct=nn': Create 'for' Loop Using Different Write Hit Percentages...........61

1.17.12.7'(for)seekpct=nn': Create 'for' Loop Using Different Seek Percentages................62

1.17.12.8'(for)hitarea=nn': Create 'for' Loop Using Different Hit Area Sizes......................62

1.17.12.9'(for)compratio=nn': Create 'for' Loop Using Different compression ratios..........62

1.17.12.10Order of Execution Using 'forxxx' Parameters.....................................................62

1.18Hot banding and SD concatenation:...................................................................................64

1.19 'Hotband=(min,max)': Create a skewed workload over a Limited Seek Range............66

1.20Data Deduplication:..............................................................................................................67

1.20.1Unique blocks..................................................................................................................67

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

3

Vdbench 5.04.03

1.20.2Duplicate blocks: ...........................................................................................................67

1.20.3Vdbench xfersize= limitations.........................................................................................68

1.20.4Rewriting of data blocks..................................................................................................68

1.20.5Use of Data Validation code............................................................................................69

1.20.6Swat/Vdbench Replay with dedup...................................................................................70

1.20.7offset= and align= parameter and dedup..........................................................................70

1.21 Data Validation and Journaling.........................................................................................71

1.22Report files.............................................................................................................................74

1.23Vdbench 'wrappers' or 'how to monitor Vdbench'...........................................................77

1.24Swat Vdbench Trace Replay................................................................................................78

1.25Complete Swat Vdbench Replay Example.........................................................................79

1.26File system testing.................................................................................................................80

1.26.1Directory and file names..................................................................................................81

1.26.2File system sample parameter file....................................................................................81

1.27File System Definition (FSD) parameter overview:...........................................................83

1.28Filesystem Workload Definition (FWD) parameter overview:........................................83

1.29Run Definition (RD) parameters for file systems, overview.............................................84

1.30File System Definition (FSD) parameter detail:.................................................................84

1.30.1‘fsd=name’: Filesystem Storage Definition name...........................................................84

1.30.2‘anchor=’: Directory anchor............................................................................................84

1.30.3‘count=(nn,mm)’ Replicate parameters...........................................................................85

1.30.4‘shared=’ FSD sharing.....................................................................................................85

1.30.5 ‘width=’: Horizontal directory count..............................................................................85

1.30.6‘depth=’: Vertical directory count...................................................................................85

1.30.7‘files=’: File count...........................................................................................................86

1.30.8‘sizes=’: File sizes............................................................................................................86

1.30.9‘openflags=’: Optional file system ‘open’ parameters.....................................................86

1.30.10‘totalsize=’: Create files up to a specific total file size..................................................86

1.30.11‘workingsetsize=nn’ or ‘wss=nn’..................................................................................87

1.31File system Workload Definition (FWD) detail..................................................................88

1.31.1‘fwd=name’: File system Workload Definition name.....................................................88

1.31.2‘fsd=’: which File System Definitions to use..................................................................88

1.31.3‘fileio=’: random or sequential I/O..................................................................................88

1.31.4‘rdpct=’: specify read percentage.....................................................................................88

1.31.5‘stopafter=’: how much I/O?............................................................................................88

1.31.6‘fileselect=’: which files to select?..................................................................................89

1.31.7‘xfersizes=’: data transfer sizes for read and writes.........................................................90

1.31.8‘operation=’: which file system operation to execute......................................................90

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

4

Vdbench 5.04.03

1.31.9‘skew=’: which percentage of the total workload............................................................90

1.31.10‘threads=’: how many concurrent operations for this workload....................................90

1.32Run Definition (RD) parameters for file system testing, detail........................................92

1.32.1‘fwd=’: which File system Workload Definitions to use.................................................92

1.32.2‘fwdrate=’: how many operations per second..................................................................92

1.32.3‘format=’: pre-format the directory and file structure ....................................................93

1.32.4‘operations=’: which file system operations to run.........................................................95

1.32.5‘foroperations=’: create ‘for’ loop using different operations.........................................95

1.32.6‘fordepth=’: create ‘for’ loop using different directory depths........................................96

1.32.7‘forwidth=’: create ‘for’ loop using different directory widths.......................................96

1.32.8‘forfiles=’: create ‘for’ loop using different amount of files...........................................96

1.32.9‘forsizes=’: create ‘for’ loop using different file of sizes................................................96

1.32.10‘fortotal=’: create ‘for’ loop using different total file sizes...........................................97

1.32.11‘forwss=’: ‘for’ loop using working set sizes................................................................97

1.33Multi Threading and file system testing.............................................................................98

1.34Operations counts vs. nfsstat counts:..................................................................................99

1.35Report file examples...........................................................................................................100

1.35.1summary.html................................................................................................................100

1.35.2totals.html; run totals.....................................................................................................101

1.35.3summary.html for file system testing.............................................................................101

1.35.4logfile.html.....................................................................................................................102

1.35.5kstat.html........................................................................................................................102

1.35.6histogram.html...............................................................................................................103

1.35.7nfs3/4.html.....................................................................................................................103

1.35.8 flatfile.html....................................................................................................................104

1.35.9skew.html.......................................................................................................................104

1.36 Sample parameter files......................................................................................................105

1.36.1Example 1: Single run, one raw disk.............................................................................105

1.36.2 Example 2: Single run, two raw disk, two workloads..................................................105

1.36.3 Example 3: Two runs, two raw disks, two workloads..................................................105

1.36.4 Example 4: Complex run, curves with different transfer sizes.....................................106

1.36.5 Example 5: Multi-host..................................................................................................106

1.36.6 Example 6: Swat I/O trace replay.................................................................................107

1.36.7 Example 7: File system test..........................................................................................107

1.37Permanently override Java socket port numbers............................................................108

1.38Java Runtime Environment...............................................................................................108

1.38.1Java and Garbage Collection.........................................................................................108

1.38.2Java and 'unable to create new native thread'................................................................109

1.39 Solaris..................................................................................................................................109

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

5

Vdbench 5.04.03

2VDBENCH FLATFILE SELECTIVE PARSING..........................................................110

3VDBENCH WORKLOAD COMPARE.........................................................................111

4VDBENCH SD PARAMETER GENERATION TOOL.

.......................................................................................................................................112

5VDBENCH DATA VALIDATION POST-PROCESSING TOOL. ................................114

Vdbench

Users Guide

Version: 5.04.03

April 2015

Author: Henk Vandenbergh

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

6

Vdbench 5.04.03

1. Copyright Notice

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

2. Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be

trademarks of their respective owners.

3. License Restrictions Warranty/Consequential Damages Disclaimer

This software and related documentation are provided under a license agreement containing

restrictions on use and disclosure and are protected by intellectual property laws. Except as

expressly permitted in your license agreement or allowed by law, you may not use, copy,

reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform,

publish or display any part, in any form, or by any means. Reverse engineering, disassembly,

or

decompilation of this software, unless required by law for interoperability, is prohibited.

4. Warranty Disclaimer

The information contained herein is subject to change without notice and is not warranted to

be error-free. If you find any errors, please report them to us in writing.

5. Restricted Rights Notice

If this is software or related documentation that is delivered to the U.S. Government or

anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system,

integrated software, any programs installed on the hardware, and/or documentation,

delivered to U.S. Government end users are “commercial computer software” pursuant to the

applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As

such, use, duplication, disclosure, modification, and adaptation of the programs, including

any operating system, integrated software, any programs installed on the hardware, and/or

documentation, shall be subject to license terms and license restrictions applicable to the

programs. No other rights are granted to the U.S. Government.

6. Hazardous Applications Notice

This software or hardware is developed for general use in a variety of information

management applications. It is not developed or intended for use in any inherently

dangerous applications, including applications that may create a risk of personal injury. If

you use this software or hardware in dangerous applications, then you shall be responsible

to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe

use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by

use of this software or hardware in dangerous applications.

7. Third Party Content, Products, and Services Disclaimer

This software or hardware and documentation may provide access to or information on

content, products and services from third parties. Oracle Corporation and its affiliates are not

responsible for and expressly disclaim all warranties of any kind with respect to third-party

content, products, and services. Oracle Corporation and its affiliates will not be responsible

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

7

for any loss, costs, or damages incurred due to your access to or use of third-party content,

products, or services.

Vdbench 5.04.03

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

8

Vdbench 5.04.03

TABLE OF CONTENTS :

1 VDBENCH: DISK I/O WORKLOAD GENERATOR

......................................................

6

1.1 Introduction

..............................................................................................................................

6

1.2 Objective

...................................................................................................................................

6

1.3 Terminology

..............................................................................................................................

7

1.4 Summary of changes since release 5.03

..................................................................................

7

1.5 Summary of changes since release 5.02

..................................................................................

8

1.6 Installing Vdbench

...................................................................................................................

9

1.7 How to start Vdbench:

.............................................................................................................

9

1.8 Execution parameter overview

.............................................................................................

10

1.9 Parameter File(s)

....................................................................................................................

12

1.10 Execution parameter detail

.................................................................................................

20

1.11 Vdbench utility functions.

..................................................................................................

25

1.12 General parameter detail

....................................................................................................

27

1.13 Replay Group (RG) parameter detail

................................................................................

34

1.14 Host Definition parameter detail

........................................................................................

35

1.15 Storage Definition parameter detail

..................................................................................

37

1.16 Workload Definition parameter detail

..............................................................................

44

1.17 Run Definition for raw I/O parameter detail

....................................................................

50

1.18 Hot banding and SD concatenation:

..................................................................................

59

1.19 'Hotband=(min,max)': Create a skewed workload over a Limited Seek Range

...........

61

1.20 Data Deduplication:

.............................................................................................................

62

1.21 Data Validation and Journaling

........................................................................................

66

1.22 Report files

............................................................................................................................

69

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

9

Vdbench 5.04.03

1.23 Vdbench 'wrappers' or 'how to monitor Vdbench'

..........................................................

72

1.24 Swat Vdbench Trace Replay

...............................................................................................

73

1.25 Complete Swat Vdbench Replay Example

........................................................................

74

1.26 File system testing

................................................................................................................

75

1.27 File System Definition (FSD) parameter overview:

..........................................................

78

1.28 Filesystem Workload Definition (FWD) parameter overview:

.......................................

78

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

............................................

79

1.30 File System Definition (FSD) parameter detail:

................................................................

79

1.31 File system Workload Definition (FWD) detail

.................................................................

83

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

.......................................

87

1.33 Multi Threading and file system testing

............................................................................

93

1.34 Operations counts vs. nfsstat counts:

.................................................................................

94

1.35 Report file examples

............................................................................................................

95

1.36 Sample parameter files

.....................................................................................................

100

1.37 Permanently override Java socket port numbers.

..........................................................

103

1.38 Java Runtime Environment

..............................................................................................

103

1.39 Solaris

.................................................................................................................................

104

2 VDBENCH FLATFILE SELECTIVE PARSING

.........................................................

105

3 VDBENCH WORKLOAD COMPARE

........................................................................

106

4 VDBENCH SD PARAMETER GENERATION TOOL.

.......................................................................................................................................

107

5 VDBENCH DATA VALIDATION POST-PROCESSING TOOL.

...............................

109

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

10

Vdbench 5.04.03

1 Vdbench: Disk I/O Workload Generator

Getting started with Vdbench:

•

Installation Instructions

• Execution

• Sample parameter files

1.1 Introduction

Vdbench is a disk 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 Oracle heterogeneous attachment. At

this time I/O has been tested on Solaris Sparc and x86, Windows NT, 2000, 2003, 2008, 2012,

XP and Windows 7+8, HP/UX, AIX, Linux, Mac OS X, zLinux, RaspBerry Pi, 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. I/O 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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

11

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.

Vdbench 5.04.03

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. The parameters include General, Host Definition (HD), Replay

Group (RG), Storage Definition (SD), Workload Definition (WD) and Run Definition (RD)

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.

• File system Workload parameters describe the file system configuration to be used and the

workload to be generated. The parameters include General, Host Definition (HD), File

System Definition (FSD), File system Workload Definition (FWD) and Run Definition

(RD) 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 StorageTekTM Workload Analysis 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 is very good at identifying 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 5.03

1.

Introduction of hot band workloads.

2.

Introduction of SD concatenation.

3. The streams= parameter has been moved from SD to WD and its meaning has changed

slightly.

4. As announced earlier, parameter pattern=use502 has been removed.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

12

Vdbench 5.04.03

1.5 Summary of changes since release 5.02

5. The Vdbench GUI has been removed.

6. Tape specific functionality has been removed. Tape access will work though as long as

you specify single threaded I/O using 'threads=1'.

7. Vdbench allows you to choose the dedup ratio of the data it generates: dedupratio=

8. Compression has been completely rewritten, now using compression ratio instead of

compression percentages: compratio=

9. Replay and data validation can now be done using multiple JVMs.

10. Replay no longer requires all detail data to be kept in memory.

11. Response time histograms now also for raw I/O workloads.

12. Format of the histograms has changed for both file system workloads and raw workloads.

13. Addition of three new columns in raw I/O reports: read and write response time and

active queue depth.

14. Addition of the compression and dedup ratios and above read/write response times in the

flatfile.

15. A color chooser for Vdbench workload compare.

16. Data Validation for raw I/O now supports a mix of data transfer sizes instead of only one

fixed transfer size.

17. Default data patterns generated will guarantee that the data will not compress and is

unique so cannot result in dedup being used. This change was made to assure that

performance results are not accidentally benefiting from a target storage device having

compression and/or dedup available.

18. The following reference under the 'format=' parameter has been removed: . "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." Format WILL be done for each forxxx unless format=once

is specified.

19. Concurrent sequential streams

against the same lun. See the 'streams=' parameter.

20. File sharing. Normally Vdbench File System functionality only allows a single thread to

use a file. Specifying 'fileio=(random,shared)' changes this, allowing a file to be

concurrently used by multiple threads.

21. Variable substitution

: this allows parameter file contents to be overridden from the

command-line, for instance ./vdbench -f parmfile lun=/x/y/z

22. maxdata

=. Instead of controlling the duration of a run using the elapsed= parameter, you

now can specify maxdata=nnn, where Vdbench will terminate after nnn bytes of data

have been read or written. The run will then terminate after the lower of elapsed= or

maxdata=.

23. 'format=limited'

to stop a file system formatting run after elapsed= seconds instead of

waiting for all files to be formatted.

24. 'range=(min,max)

' allows you to wrap around your range from the end to the begin of a

lun.

25. 'skip sequential'

I/O using the stride=(min,max) and the seekpct= parameter.

26. 'xfersize=(min,max,align)

': generate random data transfer size.

27. 'totals.html

': this report only shows run totals.

28. On each report Vdbench always generates html links to the data of each different Run

Definition (RD). With complex runs those links itself could become hard to find. Only

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

13

the last value of the ‘forxx=’ parameters will now be shown, making run output a little

easier to find.

29. distribution=(xxx,variable):

allows for second by second control over I/O rates.

Vdbench 5.04.03

1.6 Installing Vdbench

Vdbench is packaged as a zip file. Unzip the file and you’re ready to go.

The zip file contains everything you need for both Windows and Unix systems.

1.7 How to start Vdbench:

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

•

•

./vdbench –t (for a raw I/O workload)

./vdbench -tf (for a file system workload)

After this, use your favorite web browser to look at /vdbench/output/summary.html and you’ll

see the reports that Vdbench creates.

To start Vdbench:

• Unix:

• Windows:

/home/vdbench/vdbench –f parmfile

c:\vdbench\vdbench.bat –f parmfile

You can find some simple example parameter files here: sample parameter files.

There are many more examples in the ../examples/ directory.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

14

Vdbench 5.04.03

1.8 Execution parameter overview

./vdbench [- fxxx yyy zzz] [-o xxx] [-c x] [-s] [-k] [-e nn] [-i nn] [-w nn ] [-m nn] [-v] [-vr] [-vw]

[-vt] [-vc] [-j] [-jr] [-jm] [-jn] [-jro] [-p nnn] [-t] [-l nnn] [ xxx=yyy,….]

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

‘-vfparmfile’.

or,

./vdbench [compare] [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:

1.8.1 Execution Parameters

See also Execution Parameter Detail.

compare

dvpost

edit

jstack

parse(flat)

print

rsh

sds

Start Vdbench workload compare

Post-processing of output generated by Data Validation

Primitive full screen editor, syntax ‘./vdbench edit file.name’.

Create stack trace. Requires a JDK.

Selective parsing of flatfile.html

Print any block on any disk or disk file

Start Vdbench

RSH daemon

(For multi-host testing)

Start Vdbench

SD parameter generation tool

(Solaris, Linux, Windows)

-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

Run a five second sample workload on a small disk file (for demo).

-tf

Run a five second sample File system workload.

-e nn

-i nn

-w nn

-m nn

-v

-vr

Override ‘elapsed’ parameters in Run Definitions (RD)

Override ‘interval’ parameters in Run Definitions (RD)

Override ‘warmup’ parameters in Run Definitions (RD).

Override the amount of concurrent JVMs to run workload

Activate data validation.

Activate data validation, immediately re-read after each write.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

15

Vdbench 5.04.03

Activate data validation, but don’t read before write.

Activate data validation, keep track of each write timestamp (memory

intensive)

Activate data validation, but continue using the previously created map file.

Activate data validation with journaling.

Recover existing journal, validate data and run workload

Recover existing journal, validate data but do not run requested workload.

Activate journaling, but only write the journal maps.

Activate journaling, but use asynchronous writes to journal.

Simulate execution. Scans parameter files and displays run names.

Solaris only: Report kstat statistics on console.

Clean (delete) existing FSD file system structure at start of run.

Force format=only

Force format=yes

Force format=no

-vw

-vt

-vc

-j

-jr

-jro

-jm

-jn

-s

-k

-c

-co

-cy

-cn

-p nnn

-l nnn

xxx=yyy …..

Override Java socket port number (default 5570).

(lowercase 'L'): After the last run, start over with the first run. Without nnn this

is an endless loop, or loop for a total nnn s/m/h seconds/minutes/hours, e.g. -l

24h

See variable substitution.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

16

Vdbench 5.04.03

1.9 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.

1.9.1 Variable substitution.

Variable substitution allows you to code variables like $lun in your parameter file which then

can be overridden from the command line. For example:

sd=sd1,lun=$lun

$lun must be overridden from the command line: ./vdbench -f parmfile lun=/dev/x.

In case your parameter file is embedded in a shell script, you may also specify a '!' to prevent

accidental substitution by the scripting language, e.g. sd=sd1,lun=!lun

1.9.2 Multi-host parameter replication.

Whenever the constant ‘$host’, ‘!host’ or ‘#host’ is found in an input 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,…..

Just add host=host3,….

Result:

fsd=fsd\_host1,anchor=/dir/host1,…..

fsd=fsd\_host2,anchor=/dir/host2,…..

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

17

Vdbench 5.04.03

1.9.3 include=parmfile

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.

1.9.4 General Parameters: Overview

These parameters must be the first parameters in the parameter file, before any SD or FSD.

See also General Parameter Detail.

General parameters

abort\_failed\_skew=nnn For raw SD/WD workloads only: abort if requested skew is off by more

than nnn%

Specify the compression ratio of the data pattern used for writes.

Create parent directories for FSD anchor.

Terminate after 'nn' read/write/data validation errors (default 50)

Run command or script 'cmd' after first read/write/data validation error,

then terminate.

compratio=nn

create\_anchors=yes

data\_errors=nn

data\_errors=cmd

See also Data Deduplication parameters:

dedupratio=

Expected ratio. Default 1 (all blocks are unique).

dedupunit=

What size of data does Dedup compare?

dedupsets=

How many types of duplicates.

endcmd=cmd

formatsds=

formatxfersize=

Execute command or script at the end of the last run

Force a one-time (pre)format of all SDs

Specify xfersize used when creating, expanding, or (pre)formatting an

SD.

histogram=(default,….) Override defaults for response time histogram.

include=/file/name

Includes /file/name inline. See above.

messagescan=yes/no/

nodisplay

monitor=/file/name

pattern=

port=nn

report=host\_detail

report=slave\_detail

See External control of

Vdbench

termination

Override the default data pattern generation.

Override the Java socket port number.

Specifies which SD detail reports to generate. Default is SD total only.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

18

Vdbench 5.04.03

startcmd=cmd

swat=(xx,yy)

Execute command or script at the beginning of the first run

Call Swat to create performance charts.

Data Validation parameters:

force\_error\_after=(nn,mm)

validate=yes

validate=read\_after\_write

validate=no\_preread

validate=time

Journaling parameters:

journal=yes

journal=recover

journal=only

journal=noflush

Simulate data validation error after nn reads. Simulate ‘mm’ errors

(default 1)

(-vt) Activate Data Validation. Options can be combined:

validate=(x,y,z)

(-vr) Re-reads a data block immediately after it was written.

(-vw) Do not read before rewrite, though this defeats the purpose of

data validation!

(-vt) keep track of each write timestamp (memory intensive)

Activate Data Validation and Journaling:

Recover existing journal, validate data and run workload

Recover existing journal, validate data but do not run requested

workload.

Use asynchronous I/O on journal files

1.9.5 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

hd=localhost

hd=host\_label

system=host\_name

Sets defaults for all HDs that are entered later

Sets values for the current host

Specify a host label.

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

user=xxxx

clients=nn

mount=”mount xxx …”

How to start a

Vdbench

slave

on a remote system.

Userid on remote system Required.

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.

This mount command is issued on the target host after the

possibly needed mount directories have been created.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

19

Vdbench 5.04.03

1.9.6 Replay Group (RG) Parameter Overview

See also Swat and

Vdbench

Replay

.

rg=name

devices=(xxx,yyy,….)

Unique name for this Replay Group (RG).

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. All that's then left to do is specify enough SDs to satisfy the

amount of gigabytes needed.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

20

Vdbench 5.04.03

1.9.7 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/rdsk/cxt0d0s0,threads=8

sd=default

sd=name

host=name

lun=lun\_name

align=nnn

count=(nn,mm)

hitarea=nn

journal=xxx

offset=nnn

openflags=(flag,..)

range=(nn,mm)

replay=(group,..)

replay=(nnn,..)

resetbus=nnn

resetlun=nnn

size=nn

streams=(nn,mm)

threads=nn

Sets defaults for all SDs that are entered later.

Unique name for this Storage Definition (SD).

Name of host where this SD can be found. Default ‘localhost’

Name of raw disk or file system file.

Generate logical byte address in ‘nnn’ byte boundaries, not using default

‘xfersize’ boundaries.

Creates a sequence of SD parameters.

See read hit percentage for an explanation. Default 1m.

Directory name for journal file for data validation

At which offset in a lun to start I/O.

Pass specific flags when opening a lun or file

Use only a subset 'range=nn': Limit Seek Range of this SD.

Replay Group(s) using this SD.

Device number(s) to select for Swat

Vdbench

replay

Issue ioctl (USCSI\_RESET\_ALL) every nnn seconds. Solaris only

Issue ioctl (USCSI\_RESET) every nnn seconds. Solaris only

Size of the raw disk or file to use for workload. Optional unless you

want Vdbench to create a disk file for you.

Create independent sequential streams on the same device.

Maximum number of concurrent outstanding I/O for this SD. Default 8

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

21

Vdbench 5.04.03

1.9.8 File system Definition (FSD) Parameter Overview

See Filesystem Definition (FSD) parameter overview

1.9.9 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

wd=name

sd=xx

Sets defaults for all WDs that are entered later.

Unique name for this Workload Definition (WD)

Name(s) of Storage Definition(s) to use

host=host\_label

iorate=nn

openflags=(flag,..)

priority=nn

range=(nn,nn)

rdpct=nn

rhpct=nn

seekpct=nn

Which host to run this workload on. Default localhost.

Requested fixed I/O rate for this workload.

Pass specific flags when opening a lun or file.

I/O priority to be used for this workload.

Limit seek range to a defined range within an SD.

Read percentage. Default 100.

Read hit percentage. Default 0.

Percentage of random seeks. Default seekpct=100 or

seekpct=random.

Percentage of skew that this workload receives from the total I/O

rate.

stride=(min,max)

To allow for skip-sequential I/O.

threads=nn

Only available during SD concatenation.

whpct=nn

Write hit percentage. Default 0.

xfersize=nn

Data transfer size. Default 4k.

xfersize=(n,m,n,m,..)

Specify a distribution list with percentages.

xfersize=(min,max,align) Generate xfersize as a random value between min and max.

skew=nn

1.9.10

File system Workload Definition (FWD) Parameter

Overview

See Filesystem Workload Definition (FWD) parameter overview

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

22

Vdbench 5.04.03

1.9.11

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

There is a separate list of RD parameters for file system testing.

rd=default

rd=name

wd=xx

sd=xxx

curve=(nn,nn,..)

distribution=(x[,variable]

elapsed=nn

Sets defaults for all RDs that are entered later.

Unique name for this Run Definition (RD).

Workload Definitions to use for this run.

Which SDs to use for this run (Optional).

Data points to generate when creating a performance curve.

I/O inter arrival time calculations: exponential, uniform, or

deterministic. Default exponential.

Elapsed time for this run in seconds. Default 30 seconds.

maxdata=nnn

Stop the run after nnn bytes have been read or written, e.g.

maxdata=200g. I/O will stop at the lower of elapsed= and

maxdata=.

endcmd=cmd

Execute command or script at the end of the last run

(for)compratio=nn

Multiple runs for each compression percentage.

(for)hitarea=nn

Multiple runs for each hit area size.

(for)hpct=nn

Multiple runs for each read hit percentage.

(for)rdpct=nn

Multiple runs for each read percentage.

(for)seekpct=nn

Multiple runs for each seek percentage.

(for)threads=nn

Multiple runs for each thread count.

(for)whpct=nn

Multiple runs for each write hit percentage.

(for)xfersize=nn

Multiple runs for each data transfer size.

Most forxxx parameters may be abbreviated to their regular name, e.g. xfersize=(..,..)

interval=nn

Reporting interval in seconds. Default 'min(elapsed/2,60)'

iorate=(nn,nn,nn,…)

One or more I/O rates.

iorate=curve

Create a performance curve.

iorate=max

Run an uncontrolled workload.

iorate=(nn,ss,…)

nn,ss: pairs of I/O rates and seconds of duration for this I/O

rate. See also 'distribution=variable'.

openflags=xxxx

pause=nn

Pass specific flags when opening a lun or file

Sleep 'nn' seconds before starting next run.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

23

Vdbench 5.04.03

replay=(filename,

split=split\_dir,

repeat=nn)

startcmd=cmd

warmup=nn

-'filename': Replay file name used for Swat

Vdbench

replay

- 'split\_dir': directory used to do the replay file split.

- 'nn': how often to repeat the replay.

Execute command or script at the beginning of the first run

Override warmup period.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

24

Vdbench 5.04.03

1.10 Execution parameter detail

1.10.1

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

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

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 parameters) 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,..)

keyword=(1-10,1)

keyword=(1-64,d)

keyword=(64,1,d)

Individual values

Range, from 1 to 10, incremented by 1

(1,2,3,4,5,6,7,8,9,10)

Doubles: from 1 to 64, each successive value

doubled (1,2,4,8,16,32,64)

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 I/O

output directory you will see exactly what workload was executed -- no more guessing trying to

remember ‘what did I run 6 months ago’.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

25

Vdbench 5.04.03

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

1.10.2

'-oxxx': 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'.

1.10.3

'-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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

26

Vdbench 5.04.03

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.

1.10.4

'-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.

1.10.5

'-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.

1.10.6

'-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/rdsk/cxtxdxsx) 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.

1.10.7

'-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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

27

Vdbench 5.04.03

copy of itself (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, though I decided to keep the default unchanged.

The maximum number of JVMs started this way is limited by the number specified with the '-m

nn' execution parameter or ‘hd=default,jvms=nn’, or the maximum of (# SDs or 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=default,jvms=nn

or hd=hostX,jvms=nn parameter.

Vdbench will display a warning when more than 50,000 iops are done per JVM. This serves as a

warning that you possibly do not have enough active JVMs and you may have to experiment

increasing this value. Note that if a lot of these IOPS result from file system cache hits this

default limit of 50,000 per JVM could be much higher.

1.10.8

‘-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 first create a parameter file.

./vdbench -tf will run a quick File system test.

1.10.9

‘-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.

1.10.10

‘-i nn’ Override report interval time.

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

1.10.11

‘-w nn’ Override warmup time.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

28

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

Vdbench 5.04.03

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

29

Vdbench 5.04.03

1.11 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’.

1.11.1

./vdbench sds: 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,

1.11.2

./vdbench dvpost: Data Validation post processing

Running ‘./vdbench dvpost’ brings up a screen 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. See also Vdbench

Data Validation post-processing tool

.

1.11.3

./vdbench jstack: Display java execution stacks of active

Vdbench runs.

Like all software, there always is a chance that there is a problem or bug. 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; you also must be using a

JDK/SDK, since that includes the java jstack executable.

1.11.4

./vdbench rsh: Vdbench RSH daemon.

Not every OS has an RSH or SSH 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

java socket that will be used to start Vdbench slaves on that host and return its stdout and

stderr output.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

30

Vdbench 5.04.03

1.11.5

./vdbench print: 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:

lba:

xfersize:

any device or file name.

logical byte address. May be prefixed with 0x if this is hexadecimal. You

may also specify k or m for kilobytes or megabytes.

length of block to print.

1.11.6

./vdbench edit: 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.

1.11.7

./vdbench compare: 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

.

There is a much more complete Workload Compare available in Swat.

1.11.8

./vdbench parse: 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

.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

31

Vdbench 5.04.03

1.12 General parameter detail

1.12.1

‘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. 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.

1.12.2

'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

data\_errors="script\_name"

data\_errors=(nn,mm)

data\_errors=remove

Causes an abort after nn validation errors. Default 50

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.

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 period of

time after an error.

For raw (SD/WD) I/O: The failing SD will be removed but

I/O will continue to all others.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

32

Vdbench 5.04.03

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.

1.12.3

''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).

1.12.4

‘swat=xxx’: Create performance charts using Swat.

This is only available for SD/WD raw Vdbench workloads.

Swat (Sun StorageTek™ Workload Analysis Tool) has a batch utility that allows you to

automatically create a set of JPG files containing 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.

Note: newer versions of Swat after Swat302 are no longer made available, so this option above

will be removed soon.

1.12.5

'startcmd=' and 'endcmd='

Until Vdbench 503 these parameters were known as 'start\_cmd' and 'end\_cmd' and can continue

to be used. Double quotes must be used if any of the commands include a blank.

You may specify one or more commands: startcmd=(cmd1,cmd2,….), each of course using

parentheses if needed.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

33

Vdbench 5.04.03

You may also add 'cons', 'sum', or 'log' as an extra parameter, with 'log' being the default. This

parameter determines where the output of these commands will be sent: the console,

summary.html, or to logfile.html. Example: startcmd=("echo hello world", cons)

(Similar syntax for both startcmd and endcmd)

startcmd=command1

Run 'command1' on he first slave on a host, sedning

output to file '\*stdout.html'

Run 'xxx yyy' sending output to the console.

Run 'cmd' on the master

Send output to summary.html

Send output to logfile.html (default)

startcmd=("xxx yyy",cons)

startcmd=(cmd,master)

startcmd=(cmd,sum)

startcmd=(cmd,log)

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, 'startcmd=cmd' will be executed right before the first run;

'endcmd=cmd' will be executed immediately after the last run is finished or has failed. Whether

'endcmd' will run after a failure of course depends on the type of failure. The use of CTRL-C

precludes 'endcmd' from being run.

When used as a Run Definition parameter, 'startcmd=cmd' will be executed right before each

run; 'endcmd=cmd' will be executed immediately after each run is finished.

There is one extra parameter available: 'startcmd=(master,xxx)". By default the command will be

run on the first slave on each client, but when you're running on a dozen clients that became a

little too expensive. The 'master' parameter will now move the execution to the master JVM, of

which there of course is just one.

Note that if you start a long running command you need to add '&' at the end of the command to

allow asynchronous execution.

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

34

Vdbench 5.04.03

1.12.6

'pattern=: Data Pattern to be used

By default Vdbench writes a random non-compressible, non dedupable data pattern. See also the

'compratio=' and 'dedupratio=' parameters.

pattern=/file/name

The data found in this file is copied as often as possible into the data

pattern buffer. The first four eight in the data buffer will be replaced with a

unique value before each write operation, this to prevent this block to

become a duplicate, unless of course dedupratio= has been requested.

This is how the data pattern is generated:

1. A one-megabyte minimum data pattern buffer is filled with random numbers. This is done

only once.

2. A number of random 32-bit words will be replaced with zeroes, this depending on the

compratio= parameter used. The number of zeros is based on experiments done using the

LZJB compression algorithm; compression ratios 1:1 through 25:1 are the only ones

implemented; any ratio larger than 25:1 will be set to 25.

3. Data starting at the pattern buffer + (remainder of the Logical Byte Address (lba) divided by

the length of the pattern buffer) is copied into the data buffer. The copy wraps around to the

beginning of the pattern buffer if needed. This takes care of the requested compression ratio.

4. The first 8 bytes of each 4k portion of the data buffer is overlaid by a combination of lba, 4k

offset, file handle, and the lower four bytes of the system's High Resolution Timer (HRT).

This is done to make sure that the data can not be deduped. See also the dedupratio=

parameter.

1.12.7

‘compratio=nn’: Set compression for data patterns

See also pattern= above.

By default Vdbench will write an uncompressible random data pattern. 'compratio=nn' generates

a data pattern that results in a nn:1 ratio.

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 ‘LZJB'

compression algorithm, and with that in mind the accuracy for this implementation is plus or

minus 5%.

This parameter may be overridden using the ‘forcompratio=’ parameter.

Note: earlier versions of Vdbench used 'compression=nn' and 'forcompression=nn'. Though for

now these parameters are still accepted and internally translated to 'compratio=(100/nn)', future

versions may no longer do so.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

35

Vdbench 5.04.03

1.12.8

‘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 eight times before it gives up. The end result

is that it will try to use ports 5570 through 5578 (default).

Starting Vdbench50402 however the connection port will be released as soon as the master and

all of its slaves are connected making the port available again for other copies of Vdbench. The

chance that you will ever need more than 8 ports therefore is very small.

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.

1.12.9

‘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.

1.12.10

‘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.

1.12.11

‘histogram=’: set bucket count and bucket size for

response time histograms.

Response time histograms allow you to get a more detailed understanding of I/O response times.

Vdbench already reports averages and maximums, but at times it can be very useful to know

what variation there is in response times. The histograms for instance can give you some

indication as to how much of the I/O was handled from storage system cache and which ones had

to really come from spinning disk. Note of course that the response time can very depending on

the data transfer size and the queue depth.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

36

Vdbench 5.04.03

The 'histogram=(default,nn,…,nn)' parameter defines how many histogram buckets there are and

what the range of the buckets is. The default is:

histogram=(default,20,40,60,80,100,200,400,600,800,1m,2m,4m,6m,8m,10m,20m,40m,60m,80

m,100m,200m,400m,600m,800m,1s,2s) (All in one line).

Values are in microseconds, you may also specify u(micro), m(milli) or s(seconds). You may

specify a maximum of 64 buckets).

You may permanently override the default by creating file 'histogram.txt' with a valid histogram

parameter inside of your Vdbench installation directory.

For raw I/O workloads (SD/WD), histograms will include all reporting intervals, including

warmup. For File System workloads (FSD/FWD) the warmup interval(s) will be excluded.

1.12.12

‘formatxfersize=nnnn’

When Vdbench creates new or expands existing disk files specified using SD parameters the

default is xfersize=128k. (xfersize=512 for files smaller than 128k). To override this default,

specify formatxfersize=nnn.)

1.12.13

'monitor=', External control of Vdbench termination

Vdbench runs terminate after 'elapsed=nnn' seconds, or, when 'seekpct=eof' is used after

'elapsed=nnn' seconds or when EOF is reached on the last SD, whichever is first.

The monitor=/monitor/file/name parameter allows you to terminate the currently active run, or

even the complete Vdbench execution.

Vdbench at the end of each reporting interval will look at the monitor file name, and depending

on its contents will either terminate the current run, or will terminate Vdbench.

Vdbench will always start with erasing the monitor file, but after that will check the file contents

each interval:

•

•

'end\_rd'

will terminate the current run AFTER the next reporting interval.

'end\_vdbench' will terminate Vdbench AFTER the next reporting interval, skipping any

still remaining runs. (No quotes in the file please).

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

37

Vdbench 5.04.03

1.12.14

'messagescan=': suppress /var/xxx/messages scan

By default on Solaris /var/adm/messages, and on Linux /var/log/message is scanned every five

seconds to look for error messages that possibly could be related to the workload you are

running.

The 'messagescan=' parameter give you some control over this.

messagescan=no

messagescan=yes

messagescan=nodisplay

messagescan=1000

suppresses the scan

Keeps the default

Does the scan, but does not display the messages found on stdout.

Stop scanning after 1000 messages. (default)

• messagescan=no

suppresses the scan

• messagescan=yes

Keeps the default

• messagescan=nodisplay

Does the scan, but does not display the messages found on

stdout.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

38

Vdbench 5.04.03

1.13 Replay Group (RG) parameter detail

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/rdsk/cxtxdxsx,replay=group1

sd=sd2,lun=/dev/rdsk/cytydysy,replay=group1

sd=sd3,lun=/dev/rdsk/cztzdzsz,replay=group2

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

A Vdbench replay parameter file can also be created by Swat Trace Facility (STF using the ‘File’

‘Create replay parameter file’ menu option.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

39

Vdbench 5.04.03

1.14 Host Definition parameter detail

1.14.1

‘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, the host where the Vdbench

master JVM is running.

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.

1.14.2

‘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.

1.14.3

‘jvms=nnn’

This parameter tells Vdbench how many JVMs to use on this host. See ‘Multi JVM Execution.

1.14.4

‘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.

1.14.5

‘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 SSH 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’.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

40

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.

Vdbench 5.04.03

1.14.6

‘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.

1.14.7

‘mount=xxx’

This parameter is mainly useful when doing some serious multi-host file system 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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

41

Vdbench 5.04.03

1.15 Storage Definition parameter detail

1.15.1

'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.

1.15.2

'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 J. 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/rdsk/cxt0d0s0

Name of raw Solaris disk to be used for this SD.

lun=/dev/vx/rdsk/cxt0d0s0 Name of a raw VXVM volume

lun=/home/dir/filename

Solaris file system filename. No control over physical I/O

guaranteed since the file system may use system cache.

Name of raw mounted Windows disk to be used for this SD.

Physical number of raw Windows disk to be used for this SD.

Windows: file system file name.

Windows: raw access to file.

lun=\\.\d:

lun=\\.\PhysicalDrive1

lun=c:\temp\filename

lun=\\.\c:\temp\filename

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 on a raw volume 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: 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:

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

42

Vdbench 5.04.03

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).

You may also specify formatsds=yes as a General Parameter, but remember, this will cause the

format to be done each time the same parameter file is used.

Note: When an SD is recognized to be a raw device (lun name starts with "\\" on Windows or

"/dev" on Unix), Vdbench will refuse to write to block zero, this to avoid overwriting a volume

label.

1.15.3

‘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/rdsk/a on hosta but it is named /dev/rdsk/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/rdsk/a,host=hosta,lun=/dev/rdsk/b,host=hostb

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

1.15.4

‘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 a 'printf' mask, e.g.

sd=sd,lun=/dev/rmt/%dcbn,count=(0,2) This will result in devices 0cbn and 1cbn being selected.

Note: I have been asked numerous times to also support using something like lun=/dev/rdsk/c0\*.

I consider this far too dangerous though and therefore decided against it. Once upon a time about

40 years ago I accidentally erased all disk drives on a production system (05:00 am). You don't

ever want to do that .

1.15.5

'size=nn: Size of LUN or File

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

43

Vdbench 5.04.03

'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/rdsk/ 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 .

You can also use size= to give Vdbench access to only a portion of your volume, though for that

you can also use 'range=' below.

1.15.6

'range=(min,max)': 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 max value is larger than 100 but smaller than 200, Vdbench will consider this a wrap

across the end of your volume. For instance with range=(90,110) , Vdbench will generate an I/O

workload using the last 10% and the first 10% of your volume.

When the values are greater than 200, the values will be considered given in bytes instead of in

percentages; e.g., ‘range=(1g,2g)’.

Note: the 'range=' parameter may not be used during Sd concatenation.

1.15.7

'threads=nn': Maximum Number of Concurrent

outstanding I/Os

'threads=nn' specifies the maximum number of concurrent I/O that can be outstanding for this

SD. Be aware that depending on the storage subsystem, some of these I/Os may be queued inside

of the operating system (wait queue). (On Solaris, check file kstat.html).

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

44

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.

Vdbench 5.04.03

Note: This SD parameter will be ignored when using SD concatenation.

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

1.15.8

'hitarea=nn': Storage Size for Cache Hits

See read hit percentage for an explanation. Default value is 1MB.

1.15.9

'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 allows you to specify a RAW device, e.g. journal=/dev/xxxx, bypassing the

possibly faulty file system code.

1.15.10

‘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 raw volume. This has been done to make sure

that it never overwrites a volume label and/or vtoc.

1.15.11

‘align=’: Determine lba boundary for random seeks.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

45

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.

Vdbench 5.04.03

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

46

Vdbench 5.04.03

1.15.12

‘openflags=’: control over open and close of luns or files

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:

o\_dsync

o\_rsync

o\_sync

0x……

fsync

directio

directio\_off

clear\_cache

Linux:

o\_direct or

directio

o\_dsync

o\_rsync

o\_sync

fsync

(xx\_SYNC descriptions found in man open)

Write I/O operations on the file descriptor complete as defined by

synchronized I/O data integrity completion

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.

Write I/O operations on the file descriptor complete as defined by

synchronized I/O file integrity completion.

Any hex value, to be passed to the open() function.

Call fsync() before the file is closed.

Calls the directio() function after the file is opened, using ‘DIRECTIO\_ON’

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.

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.

Gives you raw access to a full volume (no partition). This parameter is

required when using /dev/xxx volumes

These three all pass ‘0x01000’ to the Linux open() function.

I highly suggest you check /usr/include/bits/fcntl.h since not all flavors of

Linux use the same bits. Then if needed code openflags=0x…. instead.

Call fsync() before the file is closed.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

47

0x……

Windows:

directio

AIX

o\_dsync

o\_rsync

o\_sync

o\_direct or

directio

0x……

Vdbench 5.04.03

Any hex value, to be passed to the open() function.

Opens a file using the ' FILE\_FLAG\_NO\_BUFFERING' flag

Passes 0x00400000 to open().

Passes 0x00200000 to open().

Passes 0x00000010 to open().

Passes 0x08000000 to open().

Any hex value, to be passed to the open() function.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

48

Vdbench 5.04.03

1.15.13

streams=: Independent sequential streams.

Note: In Vdbench 503 this was an SD parameter; it has been moved to WD, and its meaning has

slightly changed (the 'stream size' subparameter has been removed).

The streams=nn parameter overrides the default processing done when reading or writing

sequential data. By default Vdbench just reads or writes the whole lun or file (SD, or

concatenated SD), or only a portion of it when the range= parameter is used, but the streams=

parameter allows multiple concurrent sequential streams to be active.

streams=stream\_count. This parameter works together with the threads= parameter. The file or

lun is split into 'stream\_count' pieces.

Example: wd=wd1,…..,streams=10

Each SD (or the concatenated SD) is split into 10 equally sized smaller pieces. The requested

threads are spread out over the streams, with as a possible result that some streams may get more

threads than others. With 10 streams and 16 threads there will be six streams with two threads,

and four with just one.

Note that, since this is sequential I/O, a stream may run on only ONE JVM, this to avoid for

instances reading blocks 1,1,2,2,3,3,4,4, etc.

Note: The stream count must be a multiple of the amount of JVMs used, this to make sure that

the requested stream count can be equally spread around these JVMs. See Multi JVM execution.

Note: rhpct= and whpct= are ignored when streams are requested. However, if you really want to

run cache hits with streams, just code size='small' to force all I/O to cache.

Note: Only ONE Workload Definition (WD) may be active when using streams.

1.16 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

wd=name

Sets defaults for all WDs that are entered later.

Unique name for this Workload Definition (WD)

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

49

Vdbench 5.04.03

host=host\_label

iorate=nn

openflags=

priority=

range=(nn,nn)

rdpct=nn

rhpct=nn

sd=xx

seekpct=nn

Specify here which host you want this to run on.

Workload specific I/O rate

See openflags=

Workload specific I/O priority.

Limit seek range to a defined range within an SD.

Read percentage. Default 100.

Read hit percentage. Default 0.

Name(s) of Storage Definition(s) to use

Percentage of random seeks. Default seekpct=100 or

seekpct=random.

Percentage of skew that this workload receives from the total

I/O rate.

stride=(min,max)

To allow for skip-sequential I/O.

whpct=nn

Write hit percentage. Default 0.

xfersize=nn

Data transfer size. Default 4k.

xfersize=(nn,%%,…)

Data transfer size distribution.

xfersize=(min,max,align) Generate xfersize as a random value between min and max.

skew=nn

1.16.1

'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.

1.16.2

‘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,…

1.16.3

'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.

Note: When using an SD range, leading zeros are not allowed, e.g. (sd01-sd09).

You may specify SDs also as an RD= subparameter.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

50

Vdbench 5.04.03

1.16.4

'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.

1.16.5

'rhpct=nn' and 'whpct=nn': Read and Write Hit Percentage

'rhpct=' and 'whpct=' 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 'forcedirectio'

(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=' 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 'forrhpct=' or ‘forwhpct=’ parameters.

1.16.6

'xfersize=nn': Data Transfer Size

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

(m)ega bytes. This parameter may be overridden using the ‘forxfersize’ parameter.

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)

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

51

A third option uses three values: xfersize=(min,max,align). This causes a random value between

min and max, with a multiple of align to be generated. This also requires the used of the SD

align= parameter. This parameter may not be used when dedupratio= is specified.

Vdbench 5.04.03

1.16.7

'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 its 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.

Note: see also the ' abort\_failed\_skew=nn' parameter.

To accommodate users that still would like to run an uncontrolled workload see 'iorate=max'.

1.16.8

'seekpct=nn': Percentage of Random Seeks

'seekpct=' specifies how often a seek to a random lba will be generated. See also the stride=

parameter for skip sequential processing.

seekpct=100

or

seekpct=random

seekpct=0

or

seekpct=sequential

Every I/O will go to a different random seek address.

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

52

Vdbench 5.04.03

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.

See stride= below for skip sequential I/O.

seekpct=-1

or

seekpct=eof

A negative one value causes a sequential workload to terminate as

soon as end-of-file is reached. Vdbench will continue until all

workloads 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 (WD). For sequential processing, I/O for raw devices 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.

1.16.9

stride=(min,max): Skip-sequential I/O.

The stride= parameter changes the behavior when a new lba has to be generated because of the

use of the seekpct= parameter. Instead of generating a brand new random lba, I/O skips 'n'

blocks, 'n' being a random value between min and max, with a multiple of the current selected

data transfer size. When end of volume is reached we start again at the beginning.

1.16.10

'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 max value is larger than 100 but smaller than 200, Vdbench will consider this a wrap

across the end of your volume. For instance with range=(90,110) , Vdbench will generate an

I/O workload using the last 10% and the first 10% of your volume.

When the values are greater than 200, the values will be considered given in bytes instead of in

percentages; e.g., ‘range=(1g,2g)’.

1.16.11

‘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..

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

53

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?”.

Vdbench 5.04.03

1.16.12

‘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 .

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

54

Vdbench 5.04.03

1.17 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 runs, depending on the parameters used.

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

rd=default

rd=name

wd=xx

sd=xxx

curve=(nn,nn,..)

distribution=(x[,variable]

elapsed=nn

Sets defaults for all RDs that are entered later.

Unique name for this Run Definition (RD).

Workload Definitions to use for this run.

Which SDs to use for this run (Optional).

Data points to generate when creating a performance curve.

I/O inter arrival time calculations: exponential, uniform, or

deterministic. Default exponential.

Elapsed time for this run in seconds. Default 30 seconds.

maxdata=nnn

Stop the run after nnn bytes have been read or written, e.g.

maxdata=200g. Vdbench will stop at the lower of elapsed=

and maxdata=.

endcmd=cmd

Execute command or script at the end of the last run

(for)compratio=nn

Multiple runs for each compression percentage.

(for)hitarea=nn

Multiple runs for each hit area size.

(for)hpct=nn

Multiple runs for each read hit percentage.

(for)rdpct=nn

Multiple runs for each read percentage.

(for)seekpct=nn

Multiple runs for each seek percentage.

(for)threads=nn

Multiple runs for each read thread count.

(for)whpct=nn

Multiple runs for each write hit percentage.

(for)xfersize=nn

Multiple runs for each data transfer size.

Most forxxx parameters can just be abbreviated to their regular name, e.g. xfersize=(..,..)

interval=nn

Reporting interval in seconds. Default 'min(elapsed/2,60)'

iorate=(nn,nn,nn,…)

One or more I/O rates.

iorate=curve

Create a performance curve.

iorate=max

Run an uncontrolled workload.

iorate=(nn,ss,nn,ss,…)

nn,ss: pairs of I/O rates and seconds of duration for this I/O

rate. See also 'distribution=variable'.

openflags=xxxx

pause=nn

Pass specific flags when opening a lun or file

Sleep 'nn' seconds before starting next run.

replay=(filename,…)

File name used for Swat I/O replay (file 'flatfile.bin' from

Swat)

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

55

Vdbench 5.04.03

startcmd=cmd

warmup=nn

Execute command or script at the beginning of the first run

Override warmup period.

1.17.1

'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.

1.17.2

'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.

Note: When using a WD range, leading zeros are not allowed, e.g. (wd01-wd09)

1.17.3

‘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.

1.17.4

'iorate=nn': One or More I/O rates

iorate=100

iorate=(100,200,…)

iorate=(100-1000,100)

iorate=curve

iorate=max

iorate=(nn,ss,nn,ss,…)

Run a workload of 100 I/Os per second

Run a workload of 100 I/Os per second, then 200, etc.

Run workloads with I/O rates from 100 to 1000, incremented by

100.

Run a performance curve. See below.

Run the maximum uncontrolled I/O rate possible. See below.

nn,ss: pairs of I/O rates and seconds of duration for this I/O rate.

See also 'distribution=variable'.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

56

Vdbench 5.04.03

'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. This is called an uncontrolled max workload.

• 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 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.

1.17.5

'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!

1.17.6

'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 (or warmup=nn

duration). The requirement to have at least 2 intervals allows us to have at least 1 reporting

interval included in the workload total.

See also: maxdata and 'format=limited'.

1.17.7

'interval=nn': Reporting Interval

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

57

Vdbench 5.04.03

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 possible

performance problems.

1.17.8

‘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’.

1.17.9

'maxdata=': stop after nnn bytes.

Normally a run terminates after elapsed= seconds. maxdata= will terminate the run after the

shorter of elapsed= or maxdata= bytes have been read or written.

Note that the byte count starts AFTER warmup= seconds, and that maxdata is only checked at

the end of a reporting interval.

This parameter is available both for both raw and file system workloads, with the following

difference: for raw workloads, any specified value less than 100 will be used to multiply this

value with the total size of all currently used SDs. For instance two SDs, each of 10gb, using

maxdata=5 will stop this run after 2\*10gb\*5=100gb worth of data has been accessed AFTER the

warmup period.

Remember though: your elapsed time must be long enough to get this far.

1.17.10

'distribution=xxx': I/O arrival time distribution

distribution=([exponential][uniform][deterministic] ,[variable] ,[spike] )

distribution=([e][u][d] ,[variable] ,[spike] )

• Exponential: exponential distribution. Default. 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.

• Uniform: uniform distribution

• Deterministic: deterministic: all I/O is evenly spread out using fixed inter arrival times.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

58

Vdbench 5.04.03

• Variable: changes the meaning of the iorate= parameter, defining a set of pairs, first in an

I/O rate, then the amount of seconds to use that I/O rate. This causes I/O to generate a

variable I/O rate.

• Spike: the I/O for the above mentioned variable I/O rates will be all started together at the

beginning of each second instead of them being spread out using the requested

exponential or uniform distribution.

Example:

rd=rd1,…,iorate=(100,10,1000,5,50,10),dist=variable

Result: 100 iops for ten seconds, 1000 iops for five seconds, 50 iops for ten seconds and then

again 100 iops. Make sure your elapsed time covers your requested amount of seconds.

The total amount of seconds specified may not be more than 3600 seconds.

1.17.11

'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.

1.17.12

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 xfersize=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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

59

Vdbench 5.04.03

1.17.12.1

‘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.

Three little tricks: sd=single, sd=range and sd=setsofN.

•

sd=single: Let's say you have 10 devices you want to test. Just code an SD for each of

them, sd1-sd10. Instead of having to specify one Run Definition for each, just code

sd=single, and Vdbench will repeat the current RD once for each SD.

sd=range: To do a test for sd1, then sd1-sd2, then sd1-sd3 etc. code sd=range and

Vdbench will take care of it.

sd=setsofN: Vdbench will create a Run Definition for each set of 'N' randomly selected

SDs. If you have 512 drives, sd=single just is too slow. Example: sd=setsof4

•

•

1.17.12.2

'(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

forxfersize=(4k,8k,12k,16k)

forxfersize=(4k-32k,4k)

forxfersize=(1k-128k,d)

forxfersize=(128k,1k,d)

One run with 4k transfer size.

One run each for 4k, 8k, etc.

One run each from 4k to 32k in increments of 4k.

One run each from 1k to 128k, each time doubling the

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

'Reverse' double: (128k,64k,32k,etc)

See Order of Execution for information on the execution order of this parameter.

1.17.12.3

'(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.

Note: the meaning of this parameter changes when using SD concatenation. During SD

concatenation this parameter specified the total amount of threads that will be shared by all

storage definitions.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

60

Vdbench 5.04.03

forthreads=32

forthreads=(1,2,3,4)

forthreads=(1-5,1)

forthreads=(1-64,d)

forthreads=(64-1,d)

Do one run with 32 threads for each SD.

Do one run each with 1, 2, 3 and 4 threads.

Do one run each from 1 to 5 threads in increments of one.

Do one run each from 1 to 64 threads, each time doubling the

thread count.

'Reverse' double: (64,32,16,etc)

See Order of Execution for information on the execution order of this parameter.

1.17.12.4

'(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

forrdpct=(10-100,10)

Do one run with 50% reads

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.

1.17.12.5

'(for)rhpct=nn': Create 'for' Loop Using Different Read Hit

Percentages

The 'forrhpct=' parameter is an override for all workload specific rhpct parameters, and allows

multiple automatic executions of a workload with different read hit percentages.

forrhpct=50

forrhpct=(10-100,10)

Do one run with 50% read hits

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.

1.17.12.6

'(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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

61

Vdbench 5.04.03

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.

1.17.12.7

'(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

forseekpct =(10-100,10)

Do one run with 50% seek

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.

1.17.12.8

'(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

forhitarea =(10m-100m,10m)

Do one run with 4MB hit area

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.

1.17.12.9

'(for)compratio=nn': Create 'for' Loop Using Different

compression ratios.

The 'forcompratio=' parameter is an override for the compratio= parameter, and allows multiple

automatic executions of a workload with different compression ratios.

1.17.12.10 Order of Execution Using 'forxxx' Parameters

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

62

Vdbench 5.04.03

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

63

Vdbench 5.04.03

1.18 Hot banding and SD concatenation:

Vdbench 504 introduces two main new pieces of functionality for raw I/O: "hot band" or

"hotband" workloads and SD concatenation.

Hotband parameter, e.g. wd=wd1,sd=\*,hotband=(10,20)

hotband=(start,end)

Specify the starting and ending point of a hotband.

Can be specified in either percentages 'hotband=(10,20)', or in bytes,

e.g. 'hotband=(30g,60g)'

Hot Band workloads: Vdbench until now has basically had three types of workloads when doing

raw I/O: pure sequential, pure random, and a mix of random and sequential. Sequential speaks

for itself, but it is the 'random' that started becoming a problem. Random for I/O has always

meant that for each I/O a new random seek address over the whole volume (SD) was generated.

We all know of course, that it is highly unlikely that there are any real workloads that access data

in a pure random fashion. Typically there is some locality of reference going on, where a big

portion of the I/O is done against a smaller subset of the amount of storage that is available.

Of course, doing only pure random I/O against all available storage also does not make for a very

nice test for storage devices that have one or more levels of cache.

To resolve this problem a new type of random workload has been created, a workload called

'hot-banding'. The new “hot band” I/O workload provides a workload that considers the

contribution of read caching. This workload allows skewed access across one or more subsets

of the available storage. This skewed access tends to hold data in cache and creates "cache hits"

for improved throughput and performance.

Introducing this new hot banding workload created a different problem though: until now, any

workload given to Vdbench was executed identically on each single volume given to Vdbench.

With an objective to create hot bands forcing identical hot bands to every single volume of

course is a huge contradiction.

This in turn required the second large change to Vdbench, something that we call "SD

concatenation", with SD of course being a Vdbench Storage Definition, any

volume/lun/disk/drive/slice/partition/file that you tell Vdbench to use.

SD concatenation basically is a simple volume manager, where all volumes (SDs) given to a

workload are treated as if they are one large concatenated storage device. The hot band

workloads then are executed against these concatenated volumes.

This of course was not all: normally, by using the 'threads=' parameter you tell Vdbench what the

maximum amount of concurrent outstanding I/O may be against each SD. With hot band

workloads dedicating threads to an SD that possibly is not or rarely used of course does not make

sense, so, when defining hot bands, the amount of threads to be used will be given to the

concatenated SD instead. Be careful though, the default thread count is still just eight, so make

sure you give the workloads what they need.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

64

Vdbench 5.04.03

How many threads do you need? If you are running just one single workload it may not be too

difficult to figure that out, but if you run a dozen different workloads concurrently things can get

ugly fast. So for that, you can, for SD concatenation only, specify a thread count for a Run

Definition (RD), and all threads will be shared between all SDs, and all workloads. Note though

that you may end up having 500 threads and 500 volumes, but all 500 threads will be used for the

slowest volume.

In normal Vdbench operation the Run Definition 'threads=' or 'forthreads=' parameter overrides

the threads= value for each individual SD. With SD concatenation active however this is the

TOTAL amount of threads that will be shared.

SD concatenation and sequential I/O: Of course, no good deed goes unpunished. When you have

twelve drives and treat them as one large concatenation, and are doing sequential I/O on that

concatenated SD, you get only ONE active drive. That of course then results in slow (one drive

only) throughput. So for this, either don't use concatenation, or ask Vdbench to start doing

independent sequential streams using the streams= parameter.

New or changed parameters:

•

concatenate=yes

Must be specified BEFORE your first SD parameter. Note that you

can not have a mix of concatenated and not concatenated SDs.

• wd=xxx,sd=(sd1,sd2,….) With SD concatenation active all these SDs will be treated

as ONE large concatenated SD.

• wd=xxx,sd=(…..),……,threads=nn This specifies the total amount of threads to be

shared by this workload. Can only be used with SD concatenation.

rd=xxx,……,threads=nn With SD concatenation active this is the amount of threads

shared between all SDs and workloads.

sd=xxx,……,streams=nn

This parameter has been moved and now is a Workload

Definition (WD) parameter, e.g. wd=xxx,….,streams=nn.

sd=xxx,…,streams=

This parameter has been moved to the Workload Definition

and had its meaning slightly changed.

•

•

•

Note: Dedup, Data Validation and Journaling are all not allowed with SD concatenation.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

65

Vdbench 5.04.03

1.19 'Hotband=(min,max)': Create a skewed workload over a

Limited Seek Range

By default, the whole SD or concatenated SD will be used to generate a random LBA. To limit

the lba range for a workload, specify the starting and ending range of the SD: 'hotband=(40,60)'

will limit I/O activity starting at 40% into the SD and ending at 60% into the SD. If the max

value is larger than 100 but smaller than 200, Vdbench will consider this a wrap across the end

of your SD. For instance with hotband=(90,110) , Vdbench will generate an I/O workload using

the last 10% and the first 10% of your volume. When the values are greater than 200, the values

will be considered given in bytes instead of in percentages; e.g., ‘Hotband=(1g,2g)’.

Pb of access

LBA range

Hotband has a skewed workload across the defined range. The probability (Pb) accessing a

block in the range is biased toward lower LBAs It is intended to simulate something similar to

database index. This access will allow for systems with a small cache to have a small cache hit.

Systems with larger caches, have larger cache hit rates. The cache rate does not increase linearly

as system cache size increases. Here is a sample scatter plot of four hotbands and the intensity of

access is skewed to lower LBA. The X axis is the storage LBA range, the Y axis is time.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

66

Vdbench 5.04.03

1.20 Data Deduplication:

Data Deduplication is built into Vdbench with the understanding that the dedup logic included in

the target storage device looks at each n-byte data block to see if a block with identical content

already exists. When there is a match the block no longer needs to be written to storage and a

pointer to the already existing block is stored instead.

Since it is possible for dedup and data compression algorithms to be used at the same time,

dedup by default generates data patterns that do not compress.

'dedupratio' is one of the General Parameters and must be specified at the top of your

parameter file BEFORE the first Host Definition (HD) or if HD is not used, before the

first SD or FSD.

dedupratio=n

compratio=n

dedupunit=nn

dedupsets=nn

Ratio between the original data and the actually written data, e.g.

dedupratio=2 for a 2:1 ratio. Default: no dedup, or dedupratio=1

Ratio between the original data and the actually written data, e.g.

compratio=2 for a 2:1 ratio. Default: compratio=1

The size of a data block that dedup tries to match with already

existing data. Default dedupunit=128k

How many different sets or groups of duplicate blocks to have. See

below. Default: dedupsets=5% (You can also just code a numeric

value, e.g. dedupsets=100)

For a Storage Definition (SD) dedup is controlled on an SD level; For a File System Definition

(FSD) dedup is controlled on an FSD level, so not on a file level.

There are two different dedup data patterns that I/O creates:

1.20.1

Unique blocks

Unique blocks: These blocks are unique and will always be unique, even when they are

rewritten. In other words, a unique block will be rewritten with a different content than all its

previous versions.

1.20.2

Duplicate blocks:

With dedupratio=1 there of course will not be any duplicate blocks.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

67

Vdbench 5.04.03

Duplicate blocks as the name indicates are duplicates of each other. They are not all duplicates of

one single block though. That would have been too easy. There are ‘nn’ sets or groups of

duplicate blocks. All blocks within a set are duplicates of each other. How many sets are there? I

have set the default to dedupsets=5%, or 5% of the estimated total amount of dedupunit=nn

blocks.

Example: a 128m SD file, for 1024 128k blocks. There will be (5% of 1024) 51 sets of duplicate

blocks.

Dedupratio=2 ultimately will result in wanting 512 data blocks to be written to disk and 512

blocks that are duplicates of other blocks.

‘512-51 = 461 unique blocks’ + ‘512+51=563 duplicate blocks’ = 1024 blocks.

The 461 unique blocks and the 51 sets make for a total of 512 different data blocks that are

written to disk. 1024 / 512 = 2:1 dedup ratio. (The real numbers will be slightly different because

of integer rounding and/or truncation).

1.20.3

Vdbench xfersize= limitations.

Since the accuracy of dedup all revolves around the dedupunit= parameter, all read and write

operations must be a multiple of that dedupunit= size. This means that if you use dedupunit=8k,

all data transfer sizes used must me multiples of that: 8k, 16k, 24k, etc. Vdbench will fail if it

finds transfer sizes that do not follow these rules.

Technically of course (unless you are running with data validation) there is no need for read

requests to follow these rules. I thought it best though to follow the same rules for both reads and

writes.

1.20.4

Rewriting of data blocks.

As mentioned above, the unique block’s data contents will change each time they are written.

The data pattern includes the current time of day in microseconds, together with the SD or FSD

name. This makes the content pretty unique unless of course the same block to the same SD is

written more than once within the same microsecond.

For the duplicate blocks that was a whole different trail of discovery. Initially I had planned to

never change these blocks until I realized that if I do not change them there will never be an

other physical disk write because there always will be an already existing copy of each duplicate

block. That makes for great benchmark numbers, but that never is my objective. Honesty is

always the only way to go.

But changing the contents of these blocks for each write operation then causes a new problem: I

won’t get my expected dedupratio. Catch 22.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

68

Vdbench 5.04.03

That’s when I decided that yes, I will change the contents, but only once. And the next time that

this block is written it will be changed back to its original content, (flip flop). It still means that

my expected dedupratio can be a little off, this because within a set of duplicates there now can

be two different data contents, but it stays close. I typically see 1.87:1 instead of the requested

2:1, which is close enough.

If there is a better way, let me know. This is the best that I could come up with at this time. I do

understand that once all dedup sets have blocks in both the ‘flip’ and ‘flop’ state all physical

write activity will seize as long as there is a minimum of one block in each state. Suggestions for

improvement are always welcome.

1.20.5

Use of Data Validation code.

So how to keep track of what the current content is of a duplicate data block?

Data Validation already had everything that is needed; it knows exactly what is written where.

Using this ability was a very easy decision to make. Of course, unless specifically requested the

actual contents of a block after a read operation will not be validated.

So now, when dedup is used, Data Validation instead of keeping track of 126 different data

patterns per block now keeps track of only two different data patterns to support the flip-flop

mentioned above.

One more problem needed to be resolved: how to pass on the information about each block’s

current content between Vdbench runs? Of course there is Journaling, but journaling is very

expensive and is only needed to allow for some serious testing around possible data integrity

issues and that is when optimum performance is not a 100% requirement.

I therefore decided against the use of Journaling, but instead moved the in-memory Data

Validation maps from Java heap space to a memory mapped (mmap) disk file. Unless your

operating system goes down, memory mapping assures that the information of what is written

where is preserved on disk.

To ask Vdbench to reuse the existing information, code ‘validate=continue’. By default

Vdbench will create a brand new map, but validate=continue reuses the existing contents.

Validate=continue is ONLY allowed with journaling though.

These memory mapped files will be created in your current system TEMP directory and will

include the current process-id. Since the SD or FSD name is part of the file name, e.g.

'vdbench.sd1.mmap', having more than one Vdbench test running with the same SD or FSD

name caused these tests to step on each others toes (mmap files) reporting false data corruptions.

By now including the process-id (vdbench.pid29101.sd1.mmap) that problem has been

eliminated. Stale mmap files in the temp directory will be removed the next time Data Validation

is run. Using the 'journal=' parameter however puts you in complete control of where the mmap

files will be.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

69

Vdbench 5.04.03

1.20.6

Swat/Vdbench Replay with dedup.

One of the great features when you combine Swat and Vdbench is the fact that you can take any

customer I/O trace (Solaris and RedHat/OEL), and replay the exact I/O workload whenever and

wherever (any OS) you want.

Of course, the originally traced I/O workload likely does not properly follow the above-

mentioned requirements of all data transfer sizes and therefore lba’s being a multiple of the

dedupunit= size.

For Replay Vdbench adjusts all data transfer sizes to its nearest multiple of the required size and

lba, and then also reports the average difference between the original and modified size.

1.20.7

offset= and align= parameter and dedup.

Because of the requirement for everything to be properly aligned the offset= and align=

parameters of course cannot be used with dedup.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

70

Vdbench 5.04.03

1.21 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 the same 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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

71

Vdbench 5.04.03

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.

Note: I witnessed one scenario where the journal file was properly maintained but the file system

structure used for the journal files was invalid after a system outage. I therefore allow now the

use of raw devices for journal files to get around this problem.

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.

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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

72

Vdbench 5.04.03

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 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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

73

1.22 Report files

Vdbench 5.04.03

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

74

summary.html

totals.html

hostx.summary.html

hostx-n.summary.html

logfile.html

hostx\_n.stdout.html

parmfile.html

parmscan.html

sdname.html

hostx.sdname.html

hostx\_n.sdname.html

kstat.html

hostx.kstat.html

host\_x.instance.html

nfs

3/4

.html

flatfile.html

errorlog.html

Vdbench 5.04.03

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 &’.

Reports only run totals, allowing you to get a quick overview of run

totals instead of having to scan through page after page of numbers.

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.

Summary for one specific slave.

Contains a copy of most messages displayed on the console window,

including several messages needed for debugging.

Contains logfile-type information for one specific slave.

Contains a copy of the parameter file(s) from the ‘-f parmfile ‘

execution parameter.

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.

Contains performance data for each defined Storage Definition. See

summary.html for a description.

Identical to sdname.html, but containing results for only one specific

host. This report will be identical to sdname.html when not used in a

multi-host environment. This report is only created when the

‘report=host\_detail’ parameter is used.

SD report for one specific slave. . This report is only created when the

‘report=slave\_detail’ parameter is used.

Contains Kstat summery performance data for Solaris

Kstat summary report for one specific host. This report will be identical

to kstat.html when not used in a multi-host environment.

Contains Kstat device detailed performance data for each Kstat

‘instance’.

Solaris only: Detailed NFS statistics per interval similar to the nfsstat

command output.

A file containing detail statistics to be used for extraction and input for

other reporting tools. See also Parse

Vdbench

flatfile

Any I/O errors or Data Validation errors will be written here. This file

serves as input to the ‘./vdbench dvpost’ post-processing utility.

swat\_mon.txt

swat\_mon\_total.txt

This file can be imported into the Swat Performance Monitor allowing

you to display performance charts of a Vdbench run.

Similar to swat\_mon.txt, but allows Swat to display only run totals.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

75

Vdbench 5.04.03

swat\_mon.bin

messages.html

Similar to swat\_mon.txt above, but for File System workload data.

For Solaris and Linux only. At the end of a run the last 500 lines

from /var/adm/messages or /var/log/messages are copied here. These

messages can be useful when certain I/O errors or timeout messages

have been displayed.

fwdx.html

wdx.html

histogram.html

sdx.histogram.html

wdx.histogram

fsdx.histogram.html

fwdx.histogram

skew.html

A detailed report for each File system Workload Definition (FWD).

A separate workload report is generated for each Workload Definition

(WD) when more than one workload has been specified.

For file system workloads only. A response time histogram reporting

response time details of all requested FWD operations.

A response time histogram for each SD.

A response time histogram for each WD. Only generated when there is

more than one WD.

A response time histogram for each FSD.

A response time histogram for each FWD. Only generated when there

is more than one FWD.

A workload skew report for raw (SD/WD) workloads.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

76

Vdbench 5.04.03

1.23 Vdbench 'wrappers' or 'how to monitor Vdbench'

Vdbench is now almost 15 years old and through the years several users, to fit their own needs,

have created a wrapper around Vdbench.

Wrappers doing things like "if X happens with Vdbench, do Y".

And then things get scary, because some 'if X' scenarios depend on something someone may

have found over the years in one of the many Vdbench output files.

That then means that if I decide to change something these wrappers all of a sudden no longer

work. Ouch #1.

And that then prevents these teams from using the latest versions of Vdbench. Ouch #2.

Several months ago I created what we can call an "Official Vdbench Status API", where I,

though some mechanism, keep users informed about the current 'state' of a Vdbench test.

This API then will be fixed, allowing for stability for whoever decides to write a wrapper around

Vdbench.

This is it: feedback is always welcome on the Oracle Vdbench Forum.

<pre>

\* Vdbench status

\* The objective of this file is to contain easily parseable information about

the current state of Vdbench.

\* This then can serve as an 'official' interface for any software monitoring

the state of Vdbench.

\* Each line of output will be immediately flushed to the file system, making

its content accessible by any monitoring program

12/08/2014-10:38:34-MST Starting slaves

12/08/2014-10:38:34-MST Slaves connected

12/08/2014-10:38:34-MST Query host configuration started

12/08/2014-10:38:34-MST Query host configuration completed

12/08/2014-10:38:36-MST Starting rd=rd1 For loops: iorate=10

12/08/2014-10:38:38-MST Warmup done rd=rd1 For loops: iorate=10

12/08/2014-10:38:38-MST Workload done rd=rd1 For loops: iorate=10

12/08/2014-10:38:38-MST Slaves done rd=rd1 For loops: iorate=10

12/08/2014-10:38:39-MST Starting rd=rd1 For loops: iorate=20

12/08/2014-10:38:41-MST Warmup done rd=rd1 For loops: iorate=20

12/08/2014-10:38:41-MST Workload done rd=rd1 For loops: iorate=20

12/08/2014-10:38:41-MST Slaves done rd=rd1 For loops: iorate=20

12/08/2014-10:38:41-MST Shutting down slaves

12/08/2014-10:38:42-MST Vdbench complete

All 'columns' are tab-delimited so that should help you parse stuff.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

77

Vdbench 5.04.03

1.24 Swat Vdbench Trace Replay

Vdbench, in cooperation with the Sun StorageTekTM 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.gz which contains one record for each I/O operation identified by Swat.

See Example 6: Swat Vdbench Trace Replay.

There are two ways to do a replay:

1.

2.

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.

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.

Vdbench/Swat trace replay has been completely rewritten for Vdbench503. Two of the original

requirements of Replay became a hindrance: the fact that all of the to-be-replayed I/O

information had to be loaded into memory, and the problem that Replay could only run inside of

just one JVM (Replaying a high IOPS SSD workload just wouldn’t work this way).

At the start of a replay run the replay file (flatfile.bin.gz) will be split into one separate file per

replayed device. This splitting will be done only once, unless subsequent runs change the amount

of devices that will be replayed. These ‘split’ files then will be read during the replay instead of

needing to have all the data in memory. (BTW: if you want to remove unused devices from your

replay file even before you give it to Vdbench, use the ‘File’ Filter Replay file’ menu option.)

You can now request the Replay to be done ‘repeat=nn’ times instead of stopping after the last

I/O is complete.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

78

Vdbench 5.04.03

As I mentioned above, Replay can now be run using multiple JVMs.

Parameter change to the RD parameter:

replay=(/xx/flatfile.bin.gz, split=split\_directory,repeat=nnn), where ‘nnn’ is the amount of times

that Vdbench needs to do the Replay, and split\_directory is the target directory where the ‘split’

files are written (instead of in the same directory as flatfile.bin.gz).

Note about 'nnn', how often to do the Replay: realize that even though you can repeat the replay

'nnn' times, the likelihood that previously accessed data will be cached somewhere is great if you

do not have a very long trace or when your cache is very large. The same is valid for any

consecutive replay, whether by repeating replay 'nnn' times using above parameter, or by a new

start of Vdbench.

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!

1.25 Complete Swat Vdbench 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. (You can also create a trace using the

'swat\_trace' script provided.

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’

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

79

Vdbench 5.04.03

1.26 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 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. (32 million

when running 32bit Java, 128 million with 64bit Java. Note that you will need to make

sure your java heap size is large enough. Check your ./swat script for your -Xmx value.

Also look for GcTracker information in this document.

• A File System Operation. File system operations are directory create/delete, file

create/delete, file read/write, file open/close, access, 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 using directory

searches.

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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

80

Vdbench 5.04.03

1.26.1

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 current 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!

1.26.2

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).

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

81

Vdbench 5.04.03

• Each thread:

o Randomly selects a file (fileselect=random)

o Opens this file for read (operation=read)

o Sequentially reads 4k blocks (xfersize=4k) until end of file (size=128k)

o 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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

82

Vdbench 5.04.03

1.27 File System Definition (FSD) parameter overview:

fsd=name

fsd=default

anchor=/dir/

count=(nn,mm)

depth=nn

distribution=all

Unique name for this File System Definition.

All parameters used will serve as default for all the following fsd’s.

The name of the directory where the directory structure will be

created.

Creates a sequence of FSD parameters.

How many levels of directories to create under the anchor.

Default ‘bottom’, creates files only in the lowest directories. ‘all’

creates files in all directories.

files=nn

How many files to create in the lowest level of directories.

openflags=(flag,..) Pass extra flags to (Solaris) file system open request (See: man open)

shared=yes/no

Default ‘no’: See FSD sharing

sizes=(nn,nn,…..)

Specifies the size(s) of the files that will be created.

totalsize=nnn

Stop after a total of ‘nnn’ bytes of files have been created.

width=nn

How many directories to create in each new directory.

workingsetsize=nn

Causes Vdbench to only use a subset of the total amount of files

wss=nn

defined in the file structure. See workingsetsize.

journal=dir

Where to store your Data Validation journal files.

1.28 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

openflags=

Pass extra flags to (Solaris) file system open request (See: man open)

fileio=(random.shared) Allows multiple threads to use the same file.

fileio=(seq,delete)

Sequential I/O: When opening for writes, first delete the file

fileio=random

How file I/O will be done: random or sequential

fileio=sequential

How file I/O will be done: random or sequential

fileselect=random/seq How to select file names or directory names for processing.

host=host\_label

Which host this workload to run on.

operation=xxxx

Specifies a single file system operation that must be done for this

workload.

For operation=read and operation=write only. This allows a mix and

read and writes against a single file.

The percentage of the total amount of work for this FWD

For random I/O: stop and close file after ‘nnn’ reads or writes. Default

'size=' bytes for random I/O.

How many concurrent threads to run for this workload. (Make sure

you have at least one file for each thread).

Specifies the data transfer size(s) to use for read and write operations.

skew=nn

stopafter=nnn

rdpct=nn

threads=nn

xfersize=(nn,…)

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

83

Vdbench 5.04.03

1.29 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 ‘forrhpct=’ are not

supported for file system testing.

fwd=(xx,yy,..)

fwdrate=nn

format=yes/no/only/

restart/clean/once/

directories

operations=xx

foroperations=xx

fordepth=xx

forwidth=xx

forfiles=xx

forsizes=xx

fortotal=xx

Name(s) of Filesystem Workload Definitions to use.

How many file system operations per second

During this run, if needed, create the complete file structure.

Overrides the operation specified on all selected FWDs.

Multiple runs for each specified operation.

Multiple runs for each specified directory depth

Multiple runs for each specified directory width

Multiple runs for each specified amount of files

Multiple runs for each specified file size

Multiple runs for each specified total file size

1.30 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, 128 million when running 64bit java. About 64 bytes of

Java heapspace is needed per file, possibly causing memory problems. You may have to update

the ‘Xmx’ parameter in your ./vdbench script.

1.30.1

‘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.

1.30.2

‘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/

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

84

1.30.3

‘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

Vdbench 5.04.03

1.30.4

‘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 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.

If you're sure you will want to delete an existing file structure each time you run, you can of

course also code startcmd="rm -rf /file/anchor" which will do the delete for you. Be careful

though; Vdbench only deletes its own files, while 'rm -rf /root' deletes anything it finds.

1.30.5

‘width=’: Horizontal directory count

This parameter specifies how many directories to create in each new directory. See above for an

example.

1.30.6

‘depth=’: Vertical directory count

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

85

Vdbench 5.04.03

This parameter specifies how many levels of directories to create under the anchor. See above for

an example.

1.30.7

‘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.

Vdbench for each directory or file keeps track of a lot of information. That requires about 100

bytes per directory/file in the java heap. I suggest that you plan for about 200 bytes worth of java

heap size though, this to accommodate very fast selection and switching of different files, and

then avoid possible Garbage Collection issues. See 'GcTracker' information in thism document.

1.30.8

‘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 used:

•

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.

•

1.30.9

‘openflags=’: Optional file system ‘open’ parameters

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

Flags currently supported: O\_DSYNC, O\_RSYNC, O\_SYNC.

See also ‘openflags=’: Control over open() system call.

1.30.10

‘totalsize=’: 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: totalsize=100g

See also 'format=limited'

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

86

Vdbench 5.04.03

1.30.11

‘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’.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

87

1.31 File system Workload Definition (FWD) detail

Vdbench 5.04.03

1.31.1

‘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.

1.31.2

‘fsd=’: which File System Definitions to use

This parameter specifies which FSDs to use for this workload.

Example: fsd=(fsd1,fsd2)

1.31.3

‘fileio=’: random or sequential I/O

Default: fileio=sequential

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, one thread per file only.

-

fileio=(random,shared)

do random I/O, but allow multiple threads to share

the same file.

do sequential I/O (default)

delete the file before writing.

fileio=sequential:

fileio=(seq,delete):

-

-

1.31.4

‘rdpct=’: 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.

1.31.5

‘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% of the

requested file size is processed.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

88

Vdbench 5.04.03

For random I/O Vdbench by default will stop after ‘file size’ bytes are read or written. This

default prevents you from accidentally doing 100 million 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.

1.31.6

‘fileselect=’: which files to select?

fileselect=random

fileselect=sequential

fileselect=(xxx,once)

fileselect=(poisson,nn)

Default

Stop after ANY FSD has all of its files referenced

Poisson distribution is designed to skew the workload across the

file access. Poisson takes one parameter to increase the skew

when dealing with very large numbers of files. See distribution

below. Default for 'nn' is 3.

(For those that recevied a very early copy of this code, this

equates to 'fileselect=(skewed,midpoint=3)'

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.

Note though that when you use fileio=(random,shared) with as one of the objectives the setting

of your active working set size, fileselect=random may not be the correct thing to do. You may

end up with multiple threads using the same file. The total working set size therefore may not be

what you expect.

You may also specify fileselect=(xxxxx,once). This allows each file to be used only ONCE, and

Vdbench will terminate after the last file has been processed. of any FSD.

As usual, make sure that your elapsed time is long enough to get this far.

Note: fileselect=once stops after the FIRST FSD has accessed it's last file. This means that other

FSDs may not have referenced its last file yet.

Example of Poisson distribution: This shows that when you specify 'fileselect=(poisson,3)' 50%

of the file accesses are made within the first 2,000 of 10,000 files.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

89

Vdbench 5.04.03

Poisson Distribution ove r 10,000 fi le s

50% of f ile selection is over 1-2000 f iles

=3

=5

=10

10%

20%

30%

40%

50%

60%

70%

80%

90% 100%

Cum %

n

o

i

t

c

e

l

e

s

e

l

i

F

10,000

9,000

8,000

7,000

6,000

5,000

4,000

3,000

2,000

1,000

0

0%

1.31.7

‘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). Percentages of course must add up to 100.

1.31.8

‘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, access, 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.

1.31.9

‘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.

1.31.10

‘threads=’: how many concurrent operations for this

workload

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

90

Vdbench 5.04.03

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. Unless overridden using the fileio=(random,shared)

parameter 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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

91

Vdbench 5.04.03

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

1.32.1

‘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\*’.

1.32.2

‘fwdrate=’: how many operations per second.

fwdrate=100

fwdrate=(100,200,…)

fwdrate=(100-1000,100)

fwdrate=curve

fwdrate=max

Run a workload of 100 operations per second

Run a workload of 100 operations per second, then 200, etc.

Run workloads with operations per second from 100 to 1000,

incremented by 100.

Create a performance curve.

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, access, getattr and setattr. These operations are all metadata

operations which are therefore not considered I/O operations.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

92

Vdbench 5.04.03

1.32.3

‘format=’: pre-format the 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.

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 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.

You may however plan for growth of your file system. The directory and file structure specified

will be the maximum; you can use both totalsize= and workingsetsize= to use subsets of this

maximum.

A format implies that first all the directories are created. After this all files will be sequentially

formatted using 128k as a transfer size.

When specifying ‘format=yes’ for a file system workload Vdbench automatically inserts an extra

workload and Run Definition to do the formatting. Defaults for this workload are

threads=8,xfersize=128k.

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.

no

yes

restart

format=

When using more than one option use parenthesis: format=(yes,restart)

Default: format=no

No format required, though the existing file structure must match the structure

defined for this FSD.

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.

Vdbench will create only files that have not been created and will also expand files

that have not reached their proper size. (This is where totalsize and workingsetsize

can come into play).

only

The same as ‘yes’, but Vdbench will NOT execute the current RD.

dir(ectories) The same as ‘yes’, but it will only create the directories.

clean

Vdbench will only delete the current file structure and NOT execute the current

RD.

This overrides the default behavior that a format is done for each forxxx parameter

loop done.

The format will terminate after 'elapsed= seconds instead of after all files or files

selected for totalsize= have been formatted.

limited

once

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

93

complete

May only be used with 'format=no', and will tell Vdbench that the format has been

completed, but that Vdbench should not try to verify the status of each directory

and file by doing directory searches. Results are unpredictable of course if one or

more directories or files are missing or files have not reached their expected size.

VERY dangerous when deleting or creating directories or files during your test.

Vdbench 5.04.03

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

94

Vdbench 5.04.03

1.32.4

‘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, access, 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)’. For sequential write operations a create is done if the

file does not exist.

1.32.5

‘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

foroperations=(read,write,delete,rmdir)

Only do read operations

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

95

Vdbench 5.04.03

1.32.6

‘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

fordepth =(5-10,1)

One run using depth=5

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.

1.32.7

‘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

forwidth =(5-10,1)

One run using width=5

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.

1.32.8

‘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

forfiles =(5-10,1)

One run using files=5

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.

1.32.9

‘forsizes=’: create ‘for’ loop using different file of sizes

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

96

Vdbench 5.04.03

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

forsizes =(5-10,1)

One run using sizes=5

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.

1.32.10

‘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 ‘totalsize=’ parameter.

fortotal=5g

fortotal=(5g,10g)

One run using fortotal=5g

One run with totalsize=5g, and then one run with totalsize=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).

1.32.11

‘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

forwss=(16g,32g)

Uses only a 16g subset of the files specified in the FSD

Two runs: one for 16g and one for 32g.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

97

Vdbench 5.04.03

1.33 Multi Threading and file system testing

By default, 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.

This behavior can be overridden when specifying 'fileio=(random,shared)'.

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 a few 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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

98

1.34 Operations counts vs. nfsstat counts:

Vdbench 5.04.03

The operations that you can specify are: mkdir, rmdir, create, delete, open, close, read, write,

access, 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 non

NFS file system.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

99

Vdbench 5.04.03

1.35 Report file examples

1.35.1

summary.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.

Starting RD=rd1; I/O rate: 1000; elapsed=3; For loops: xfersize=1k

interval i/o MB/sec bytes read resp read write resp resp queue cpu% cpu%

rate 1024\*\*2 i/o pct time resp resp max stddev depth sys+u sys

1 833.00 0.81 1024 0.00 0.007 0.000 0.007 0.035 0.003 0.0 12.5 0.4

2 991.00 0.97 1024 0.00 0.007 0.000 0.007 0.032 0.004 0.0 31.6 0.0

3 1001.00 0.98 1024 0.00 0.007 0.000 0.007 0.086 0.004 0.0 7.4 0.6

avg\_2-3 996.00 0.97 1024 0.00 0.007 0.000 0.007 0.086 0.004 0.0 19.4 0.3

interval

I/O rate

MB sec

bytes I/O

read pct

resp time

read resp

write resp

resp max

resp stddev

queue depth

cpu% sys+usr

cpu% sys

Reporting interval sequence number. See 'interval=nn' parameter.

Average observed I/O rate per second.

Average number of megabytes of data transferred.

Average data transfer size.

Average percentage of reads.

Average response time measured as the duration of the read/write

request. All I/O times are in milliseconds.

Average response time for reads

Average response time for writes

Maximum response time observed in this interval. The last line contains

total max.

Standard deviation for response time.

Average I/O queue depth calculated by Vdbench. There may be slight

differences with the Kstat results, this due to at what time during the

I/O process the calculations are made.

Realize also that Kstat reports on TWO queues: the host wait queue and

the device active queue.

Processor busy = 100 - (system + user time) (Solaris, Windows, Linux)

Processor utilization; system time.

Note that Vdbench will display a warning if average cpu utilization

during a test reaches 80%. This warns that you may not have enough

cpu cycles available to properly run at the highest workload possible.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

100

Vdbench 5.04.03

1.35.2

totals.html; run totals

The run totals report allows you to get a quick overview of all the totals without the need to

scroll through page after page of detailed interval results.

Starting RD=rd1; I/O rate: 1000; elapsed=3; For loops: xfersize=1k

interval i/o MB/sec bytes read resp read write resp resp queue cpu% cpu%

rate 1024\*\*2 i/o pct time resp resp max stddev depth sys+u sys

avg\_2-3 996.00 0.97 1024 0.00 0.007 0.000 0.007 0.086 0.004 0.0 19.4 0.3

Starting RD=rd1; I/O rate: 1000; elapsed=3; For loops: xfersize=2k

avg\_2-3 1000.00 1.95 2048 0.00 0.007 0.000 0.007 0.054 0.004 0.0 7.6 0.2

Starting RD=rd1; I/O rate: 1000; elapsed=3; For loops: xfersize=3k

avg\_2-3 1000.00 2.93 3072 0.00 0.007 0.000 0.007 0.056 0.004 0.0 9.2 0.1

1.35.3

summary.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:

ReqstdOps

cpu% total

cpu% sys

read

write

mb/sec

xfer

…..

Reporting interval sequence number. See 'interval=nn' parameter.

The total amount of requested 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.

Processor busy = 100 - (system + user time) (Solaris, windows, Linux)

Processor utilization; system time

Total reads and average response time

Total writes and average response time.

Mb per second for reads, writes, and the sum of reads and writes.

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

101

Vdbench 5.04.03

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 .

1.35.4

logfile.html

'logfile.html' contains a copy of each line of information that has been written by the Java code to

the terminal. 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 anyway is this tar or zip file. I can usually answer 99%

of your questions when I have the output directory available.

1.35.5

kstat.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=run1; 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:

resp time:

wait time:

service time:

MB/sec:

read pct:

busy pct:

avg #I/O waiting:

avg #I/O active:

bytes per I/O:

cpu% sys+usr

cpu% sys

I/O rate per second over the duration of the reporting interval

Response time (the sum of wait time and service time). All I/O

times are in milliseconds.

Average time each I/O spent queued on the host

Average time the I/O was being processed

Average data transfer rate per second

Average percentage of total I/O that was read

Average device busy percentage

Average number of I/Os queued on the host

Average number of I/Os active

Average number of bytes transferred per I/O

Processor busy = 100 - (system + user time)

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

102

Vdbench 5.04.03

1.35.6

histogram.html

This report shows the distribution of response times for both reads and writes combined, for

reads, and for writes. When only reads or only writes are done there will of course be only one

report. A histogram is generated for each SD and FSD and for each WD and FWD if there is

more than one specified.

Note that this file can be directly read into Excel as a tab-delimited file.

Reads and writes:

min(ms) < max(ms) count %% cum%% '+': Individual%; '+-': Cumulative%

0.000 < 0.020 91 25.8523 25.8523 ++++++++++++

0.020 < 0.040 2 0.5682 26.4205 ------------

0.040 < 0.060 0 0.0000 26.4205 -------------

0.060 < 0.080 1 0.2841 26.7045 -------------

0.080 < 0.100 0 0.0000 26.7045 -------------

0.100 < 0.200 0 0.0000 26.7045 -------------

0.200 < 0.400 50 14.2045 40.9091 +++++++-------------

0.400 < 0.600 40 11.3636 52.2727 +++++--------------------

0.600 < 0.800 14 3.9773 56.2500 +--------------------------

0.800 < 1.000 4 1.1364 57.3864 ----------------------------

1.000 < 2.000 7 1.9886 59.3750 ----------------------------

2.000 < 4.000 30 8.5227 67.8977 ++++-----------------------------

4.000 < 6.000 26 7.3864 75.2841 +++---------------------------------

6.000 < 8.000 15 4.2614 79.5455 ++-------------------------------------

8.000 < 10.000 31 8.8068 88.3523 ++++---------------------------------------

10.000 < 20.000 29 8.2386 96.5909 ++++--------------------------------------------

20.000 < 40.000 11 3.1250 99.7159 +------------------------------------------------

40.000 < 60.000 1 0.2841 100.0000 -------------------------------------------------

60.000 < 80.000 0 0.0000 100.0000 --------------------------------------------------

80.000 < 100.000 0 0.0000 100.0000 --------------------------------------------------

100.000 < 200.000 0 0.0000 100.0000 --------------------------------------------------

200.000 < 400.000 0 0.0000 100.0000 --------------------------------------------------

400.000 < 600.000 0 0.0000 100.0000 --------------------------------------------------

600.000 < 800.000 0 0.0000 100.0000 --------------------------------------------------

800.000 < 1000.000 0 0.0000 100.0000 --------------------------------------------------

1000.000 < 2000.000 0 0.0000 100.0000 --------------------------------------------------

2000.000 < max 0 0.0000 100.0000 --------------------------------------------------

1.35.7

nfs3/4.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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

103

Vdbench 5.04.03

1.35.8

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 both for the original RAW I/O Vdbench functionality (SD/WD/RD)

and for file system testing using FSD/FWD/RD parameters.

1.35.9

skew.html

This report is generated for raw (SD/WD) workloads.

One of the many objectives of Vdbench is to allow users to run multiple concurrent workloads,

with each workload getting a skew=nn percentage of the workload.

This report will give you information about how successful the skew has been, or better yet, that

the requested skew has been met. There are a few scenarios where, because of contradicting user

parameters, the skew percentage may not be reached, for instance a multi-jvm/slave run with one

workload doing 100% sequential and an other workload doing random I/O. The sequential

workload may run on only ONE slave, while the random workload may run on all available

slaves. Workload skew will ONLY work properly when all workloads are allowed to run on

all slaves.

Note that when using SD Concatenation Vdbench makes sure that this scenario can not happen,

it is 'legacy' Vdbench workloads where this still may occur.

See also the ' abort\_failed\_skew=nn' parameter.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

104

Vdbench 5.04.03

1.36 Sample parameter files

When running ./vdbench -t Vdbench will run a small hard coded raw I/O function test.

When running ./vdbench -tf Vdbench will run a small hard coded file system function test.

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:

1.36.1

Example 1: Single run, one raw disk

\*SD:

Storage Definition

\*WD: Workload Definition

\*RD:

Run Definition

\*

sd=sd1,lun=/dev/rdsk/cxt0d0sx

wd=wd1,sd=sd1,xfersize=4096,rdpct=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

1.36.2

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

sd=sd1,lun=/dev/rdsk/cxt0d0sx

sd=sd2,lun=/dev/rdsk/cxt0d1sx

wd=wd1,sd=sd1,xfersize=4k,rdpct=80,skew=40

wd=wd2,sd=sd2,xfersize=8k,rdpct=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.

1.36.3

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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

105

Vdbench 5.04.03

sd=sd1,lun=/dev/rdsk/cxt0d0sx

sd=sd2,lun=/dev/rdsk/cxt0d1sx

wd=wd1,sd=(sd1,sd2),xfersize=4k,rdpct=75

wd=wd2,sd=(sd1,sd2),xfersize=8k,rdpct=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=wd\*,iorate=(100,200),elapsed=10,interval=1

1.36.4

sizes

Example 4: Complex run, curves with different transfer

sd=sd1,lun=/dev/rdsk/cxt0d0sx

wd=wd1,sd=sd1,rdpct=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 –f parmfile –s'.

1.36.5

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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

106

Vdbench 5.04.03

1.36.6

Example 6: Swat I/O trace replay

\*Example 6: Swat I/O trace replay

rg=group1,devices=(123,456,789)

sd=sd1,lun=/dev/rdsk/cxt0d0sx,replay=group1

sd=sd2,lun=/dev/rdsk/cyt0d0sx,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/rdsk/cxt0d0sx and /dev/rdsk/cyt0d0sx

1.36.7

Example 7: File system test

\*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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

107

\* dir1/vdb1\_2.dir/vdb2\_2.dir/vdb\_f0002.file

\*

Vdbench 5.04.03

1.37 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

rshdeamonport=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.

1.38 Java Runtime Environment

It is expected that the Java Runtime Environment (JRE) or Java Development Kit (JDK) already

has been installed. Vdbench expects Java 1.6 or higher.

See the following web pages:

http://www.oracle.com

for Solaris, Windows, and Linux.

•

http://www-106.ibm.com/developerworks/java/jdk/index.html

for Aix.

•

Follow the vendor's installation instructions.

It is OK to install java in your own private directory; there is no need to override the existing

version of java that is already present. Modify either your standard search path, or change the

vdbench or vdbench.bat script changing 'java=java' to point to the proper java executable, which

normally is /some/thing/bin/java(.exe)

1.38.1

Java and Garbage Collection.

The current default of -Xmx1024m of java heapsize available for each Vdbench slave/JVM

likety will cover 95+% of all Vdbench executions. However, if you are running a huge amount of

threads and/or a huge amount of files and directories, this limit either may not be enough causing

Vdbench to abort, or may dramatically slow down the speed with which Vdbench runs because

of java garbage collection.

To help you with this Vdbench50403 has added some new functionality that reports when it sees

that garbage collection has been done. This is reported on a slave's stdout.html file.

GcTracker:

cum: 0 intv: 0 ms: 0 mss: 0.00% Heap\_MB max: 455 curr: 245 used: 22 free: 223

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

108

Vdbench 5.04.03

cum:

intv:

ms:

mss:

Cumulative count of how many times GC was done.

How many GC's were done since the last time GC data was collected

Number of milliseconds that GC was active since the last time GC data was

collected.

A few milliseconds is normal, and this will be done asynchronously. If you start

running out of java heap space though the amount of time GC is active can become

large, and sometimes this will run synchronous, causing Vdbench to temporarily

come to a halt. No indication however is given when GC is done synchronously.

Milliseconds per second: ms / 'time between the last two sets of GC data'. I am not

sure if this field is accurate or useful.

Heap max: Maximum amount of memory that the Java virtual machine will attempt to use.

This usually equates to the -Xmx value specified minus internal java overhead.

The total amount of memory in the Java virtual machine.

'curr - free'

The amount of free memory in the Java Virtual Machine

curr:

used:

free:

1.38.2

Java and 'unable to create new native thread'

This message may be displayed by java when it is creating so many threads that the OS refuses

to start new threads. I do not understand the complexities of this, but a usual solution to this

problem is to lower the specified or default -Xss value for SlaveJvm in the vdbench script. I have

not really been able to find very clear information about this, but usually changing the current

default -Xss value (it would be nice btw if java would display what the default is), lowering it to

still be above the required minimum -Xss value (which is also not displayed), this change usually

helps.

What some times will also help is to increase the amount of slaves/JVMs which will then split

the amount of requested threads over more OS processes/JVMs.

I'll leave this over to the experts.

1.39 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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

109

Vdbench 5.04.03

2 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 parseflat -i flatfile.html -o output.csv [-c col1 col2 ..]

[-a] [-f col1 value1 col2 value2 .. ..]

input flatfile, e.g. output/flatfile.html

output CSV file name (default stdout)

which column to write to CSV. Columns are written in the order specified.

filters: 'if (colX == valueX) ... ...' (Alphabetic compare)

include only the 'avg' data. Default: include only non-avg data.

-i

-o

-c

-f

-a

Example:

./vdbench parse -i output\flatfile.html -c run interval rate resp -f run rd1 -o out.csv

This will give you file out.csv which can be directly read into Excel or StarOffice:

run,interval,rate,resp

rd1,1,104.0000,3.3225

rd1,2,119.0000,2.2137

rd1,3,98.0000,3.6711

rd1,4,104.0000,3.0038

rd1,5,99.0000,2.5214

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

110

Vdbench 5.04.03

3 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'

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

111

Vdbench 5.04.03

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/rdsk/$%

Where:

•

# is replaced by a sequence number, starting from 1

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

112

Vdbench 5.04.03

•

$ 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 on the screen, or by adding it to file

‘build\_sds.txt’ in your Vdbench installtion 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/c1t0d0s6 \

/dev/dsk/c1t1d0s6

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/rdsk/$%

\* 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/$%>

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

113

Vdbench 5.04.03

5 Vdbench Data Validation post-processing tool.

Please note that dvpost needs some extra work. I just noticed that it does not know about

DEDUP.

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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

114

Vdbench 5.04.03

‘./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 program 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

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

115

Vdbench 5.04.03

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.

Copyright © 2000, 2015, Oracle and/or its affiliates. All rights reserved.

116