

Installation of filesystems

The filesystems we chose for this lab were: **zfs** and **btrfs**.

- ❖ **ZFS** is a modern filesystem with various new features like deduplication, compression, management of files through data structures etc. It combines a file system with a volume manager.
- ❖ **Btrfs**, an abbreviation for b-tree file system, is a file system based on the copy-on-write (COW) principle.

The first task in installation is to make two partitions of disks/ add disks so that filesystems can be created in those partitions/disks.

So we added two 1.25 GB hard disks(/dev/sdb and /dev/sdc) in the Linux VM system we were working on and gave them a zfs filesystem pool and a btrfs filesystem pool respectively.

Commands for making and mounting a zfs pool:-

1. `sudo apt install zfsutils-linux`
2. `sudo zpool create zfs_pool /dev/sdb`

So this created a zfs pool mounted on /zfs_pool.

Making and mounting a btrfs pool:-

1. `sudo apt install btrfs-progs`
2. `sudo mkfs.btrfs /dev/sdc`
3. `Sudo mkdir btrfs_pool`
4. `Sudo mount /dev/sdc /btrfs_pool`

So this created a btrfs pool mounted on /btrfs_pool.

Now these two filesystem pools will be used for further parts.

Features of interest and their implementation

- ❖ **Data deduplication** is a companion technology to data compression.
 - It removes redundancy from stored data in addition to that which is removed by data compression alone.
 - It removes redundant copies of information by storing multiple copies of a set of data as a single copy along with an index of references to the copy.
 - Data can be deduplicated at the level of files, blocks, or bytes. ZFS provides block-level deduplication because this is the finest granularity that makes sense for a general-purpose storage system.
- ❖ **C-O-W Feature and snapshotting** for backup purposes
 - Btrfs provides a clone operation that atomically creates a copy-on-write snapshot of a file. The actual data blocks are not duplicated; at the same time, due to the copy-on-write (CoW) nature of Btrfs, modifications to any of the cloned files are not visible in the original file and vice versa.
 - For Btrfs Once a writable snapshot is made, it can be treated as an alternate version of the original file system. For example, to roll back to a snapshot, a modified original subvolume needs to be unmounted and the

snapshot needs to be mounted in its place. At that point, the original subvolume may also be deleted.

- In ZFS, Blocks containing active data are never overwritten in place; instead, a new block is allocated, modified data is written to it, then any metadata blocks referencing it are similarly read, reallocated, and written.
- In ZFS, Snapshots are inherently read-only, ensuring they will not be modified after creation, although they should not be relied on as a sole means of backup. Entire snapshots can be restored and also files and directories within snapshots.

Differences in performance observed through vdbench workloads

_____ To run the parameter files use `sudo ./vdbench -f parameterFileName` in linux

❖ Deduplication feature :

The common workload to be used in the filesystems(using the vdbench guide and example files)

`fsd=fsd1,anchor=/pool_name ,depth=2,width=2,files=2,size=2048k`

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

`rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1`

The parameter file will use a directory structure of 4 directories and 10 files in each leaf directory.

The RD parameter 'format=yes' causes the directory structure to be completely created, including initialization of all files to the requested size of 2MB. Larger sized files gave us clearer expected results. 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 will:
- 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.

ZFS provides the deduplication feature in block format as stated above. It can be turned on using this command: `sudo zfs dedup=on zfs_pool`

where `zfs_pool` is the name of the pool created under this file system.

We set the deduplication parameters as follows for running the workload on zfs:

=> `dedupunit=4k,dedupratio=2,dedupsets=50%`

- `dedupunit` : The size of a data block that dedup tries to match with already existing data
- `dedupratio`:Ratio between the original data and the actually written data, e.g. `dedupratio=2` for a 2:1 ratio
- `dedupsets`: How many different sets or groups of duplicate blocks to have. See below.

We use the following commands to find the disk space usage in both the cases.

```
nikunj_heda@nikunj-VirtualBox: ~/Downloads/vdbench
nikunj_heda@nikunj-VirtualBox:~/Downloads/vdbench$ zpool list
NAME      SIZE  ALLOC   FREE CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH
ALTROOT
zfs_pool  1.12G  40.4M  1.09G      -          -         0%    3%  1.99x    ONLINE

nikunj_heda@nikunj-VirtualBox:~/Downloads/vdbench$ sudo btrfs filesystem usage /
btrfs_pool
Overall:
Device size:                1.25GiB
Device allocated:           280.00MiB
Device unallocated:         1000.00MiB
Device missing:             0.00B
Used:                        80.56MiB
Free (estimated):           1.03GiB   (min: 555.94MiB)
Data ratio:                  1.00
Metadata ratio:              2.00
Global reserve:              3.25MiB   (used: 0.00B)

Data,single: Size:136.00MiB, Used:80.06MiB (58.87%)
/dev/sdc      136.00MiB

Metadata,DUP: Size:64.00MiB, Used:240.00KiB (0.37%)
/dev/sdc      128.00MiB
```

We can clearly see the allocated disk space for the zfs_pool is only 40.4MB

The term device in btrfs refers to the block partitions of memory for the pool. We can see the device allocated is a large space of 280MB.

So due to deduplication the allocated memory usage of the ZFS is much lesser than that of BTRFS.

We compare the summary.html files of both of these workloads and we can see the avg cpu usage (%) is higher in the case of ZFS than btrfs. This is a slight drawback of deduplication being employed in the file system. The LHS is the summary of ZFS showing around 18% cpu-usage while the RHS is the summary of Btrfs showing around 12% cpu-usage.

```
file:///home/nikunj_heda/Downloads/vdbench/output/summary.html
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Vdbench summary report, created 11:58:41 Nov 25 2020 IST (06:28:41 Nov 25 2020 UTC)

Link to logfile:      logfile
Run totals:           totals
Vdbench status:       status
Copy of input parameter files:  paramfile
Copy of parameter scan detail:  warnscan
Link to errorlog:     errorlog
Link to flatfile:     flatfile
Link to anchor status: anchors

Link to HOST reports: localhost
Link to FSD reports:  fsd
Link to FwD reports:  format fwt
Link to response time histogram: histogram
Link to workload skew report:  skew
Link to anchor report:  anchors

Link to Run Definitions: format for rd1 For loops: None
rd1 For loops: None

Link to config output: config

11:58:44.003 Starting Rbformat_for_rd1

Nov 25, 2020 ..Interval..ReqstdOps...cpu%... read ....read.....write....mb/sec...mb/sec...xfer...
rate resp total sys pct rate resp rate resp read write total size
11:58:45.170 1 383.0 1.205 50.8 9.13 0.0 0.0 0.000 303.0 1.205 0.00 37.88 37.88 131072
11:58:46.050 2 337.0 26.531 64.0 18.0 0.0 0.0 0.000 337.0 26.531 0.00 42.12 42.12 131072
11:58:46.111 avg 2-2 337.0 26.531 64.0 18.0 0.0 0.0 0.000 337.0 26.531 0.00 42.12 42.12 131072
11:58:46.111 std 2-2 128.47 128.47
11:58:46.111 max 2-2 337.0 787.17 337.0 787.17

11:58:47.002 Starting Rbdr1; elapsed=10; fwrates=100; For loops: None

Nov 25, 2020 ..Interval..ReqstdOps...cpu%... read ....read.....write....mb/sec...mb/sec...xfer...
rate resp total sys pct rate resp rate resp read write total size
11:58:48.038 1 48.0 0.038 24.0 0.51 100.0 48.0 0.038 0.0 0.000 0.19 0.00 0.19 4096
11:58:49.028 2 96.0 0.036 12.5 1.04 100.0 96.0 0.036 0.0 0.000 0.38 0.00 0.38 4096
11:58:50.020 3 180.0 0.037 29.3 0.06 100.0 180.0 0.037 0.0 0.000 0.42 0.00 0.42 4096
11:58:51.016 4 185.0 0.037 34.7 4.00 100.0 185.0 0.037 0.0 0.000 0.41 0.00 0.41 4096
11:58:52.020 5 180.0 0.034 11.2 0.00 100.0 180.0 0.034 0.0 0.000 0.42 0.00 0.42 4096
11:58:53.014 6 110.0 0.035 9.3 0.00 100.0 110.0 0.035 0.0 0.000 0.43 0.00 0.43 4096
11:58:54.012 7 100.0 0.035 7.1 0.00 100.0 100.0 0.035 0.0 0.000 0.39 0.00 0.39 4096
11:58:55.012 8 106.0 0.034 7.1 0.00 100.0 106.0 0.034 0.0 0.000 0.41 0.00 0.41 4096
11:58:56.010 9 100.0 0.035 7.1 1.01 100.0 100.0 0.035 0.0 0.000 0.42 0.00 0.42 4096
11:58:57.013 10 92.0 0.034 43.0 1.02 100.0 92.0 0.034 0.0 0.000 0.36 0.00 0.36 4096
11:58:57.020 avg 2-10 103.7 0.035 18.0 0.79 100.0 103.7 0.035 0.0 0.000 0.40 0.00 0.40 4096
11:58:57.020 std 2-10 6.2 0.014 6.2 0.014
11:58:57.020 max 2-10 110.0 0.141 110.0 0.141
11:58:57.085 Vdbench execution completed successfully
```

```
file:///home/nikunj_heda/Downloads/vdbench/output/summary.html
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Vdbench summary report, created 11:56:34 Nov 25 2020 IST (06:26:34 Nov 25 2020 UTC)

Link to logfile:      logfile
Run totals:           totals
Vdbench status:       status
Copy of input parameter files:  paramfile
Copy of parameter scan detail:  warnscan
Link to errorlog:     errorlog
Link to flatfile:     flatfile
Link to anchor status: anchors

Link to HOST reports: localhost
Link to FSD reports:  fsd
Link to FwD reports:  format fwt
Link to response time histogram: histogram
Link to workload skew report:  skew
Link to anchor report:  anchors

Link to Run Definitions: format for rd1 For loops: None
rd1 For loops: None

Link to config output: config

11:56:37.002 Starting Rbformat_for_rd1

Nov 25, 2020 ..Interval..ReqstdOps...cpu%... read ....read.....write....mb/sec...mb/sec...xfer...
rate resp total sys pct rate resp rate resp read write total size
11:56:38.168 1 640.0 0.536 33.9 3.59 0.0 0.0 0.000 640.0 0.536 0.00 80.00 80.00 131072
11:56:38.290 avg 2-1 NaN 0.000 NaN NaN 0.0 NaN 0.000 NaN 0.000 NaN NaN NaN 0 NaN 0.000 NaN 0.000 NaN
11:56:38.291 std 2-1
11:56:38.291 max 2-1

11:56:39.002 Starting Rbdr1; elapsed=10; fwrates=100; For loops: None

Nov 25, 2020 ..Interval..ReqstdOps...cpu%... read ....read.....write....mb/sec...mb/sec...xfer...
rate resp total sys pct rate resp rate resp read write total size
11:56:40.043 1 97.0 0.011 32.7 2.04 100.0 97.0 0.011 0.0 0.000 0.38 0.00 0.38 4096
11:56:41.038 2 104.0 0.010 12.1 1.01 100.0 104.0 0.010 0.0 0.000 0.41 0.00 0.41 4096
11:56:42.021 3 106.0 0.013 13.5 3.13 100.0 106.0 0.013 0.0 0.000 0.41 0.00 0.41 4096
11:56:43.014 4 97.0 0.011 24.5 2.04 100.0 97.0 0.011 0.0 0.000 0.38 0.00 0.38 4096
11:56:44.013 5 94.0 0.009 24.8 1.06 100.0 94.0 0.009 0.0 0.000 0.37 0.00 0.37 4096
11:56:45.021 6 94.0 0.009 10.2 0.00 100.0 94.0 0.009 0.0 0.000 0.37 0.00 0.37 4096
11:56:46.016 7 121.0 0.009 6.2 0.00 100.0 121.0 0.009 0.0 0.000 0.47 0.00 0.47 4096
11:56:47.016 8 96.0 0.009 5.1 0.00 100.0 96.0 0.009 0.0 0.000 0.38 0.00 0.38 4096
11:56:48.013 9 81.0 0.009 8.1 1.01 100.0 81.0 0.009 0.0 0.000 0.32 0.00 0.32 4096
11:56:49.012 10 102.0 0.009 4.1 0.00 100.0 102.0 0.009 0.0 0.000 0.40 0.00 0.40 4096
11:56:49.031 avg 2-10 99.4 0.010 12.0 0.91 100.0 99.4 0.010 0.0 0.000 0.39 0.00 0.39 4096
11:56:49.032 std 2-10 18.9 0.012 18.9 0.012
11:56:49.032 max 2-10 121.0 0.315 121.0 0.315
11:56:49.763 Vdbench execution completed successfully
```

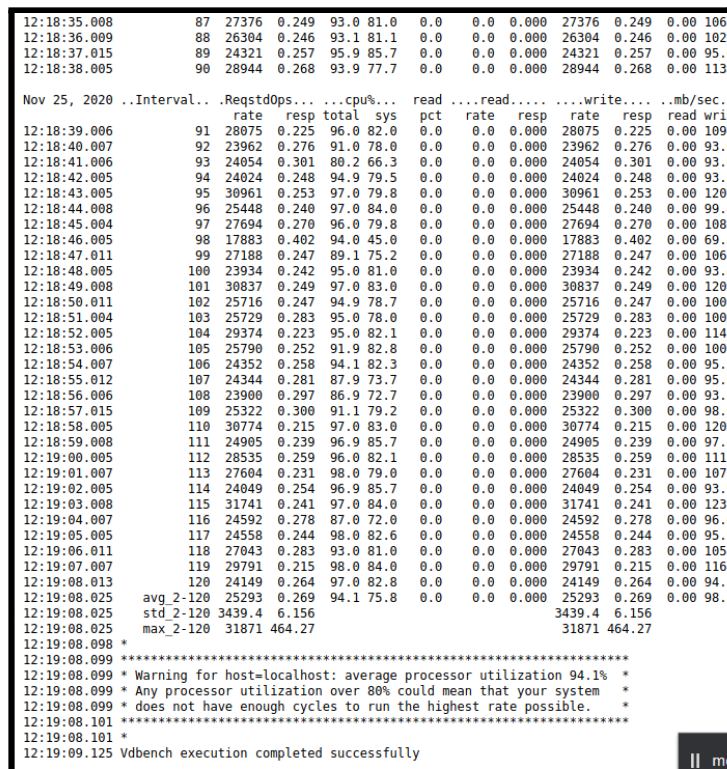

❖ C-O-W Feature and snapshotting

The common workload to be used in the filesystems(using the vdbench guide and example files)

```
fsd=fsd1,anchor=/pool_name ,depth=2,width=2,files=2,size=2048k
fwd=fwd1,fsd=fsd1,operation=write,xfersize=4k,fileio=sequential,
fileselect=random,stopafter=100,threads=8
rd=rd1,fwd=fwd*,fwdrate=max,format=yes,elapsed=120,interval=1
```

The parameter file will use a directory structure of 4 directories and 10 files in each leaf directory. Random writes on randomly selected files. 'format=yes' first creates (depth*width*files) 10 2MB files in each directory as per the structure defined. The test then will have eight threads each randomly select a file and randomly write it. After 'stopafter=100' writes the file is closed and a new file is randomly selected.

After running this workload on both fs we see the write response time for btrfs is less as it uses copy-on-write (0.269 for zfs vs 0.025 for btrfs).



12:18:35.008	87	27376	0.249	93.0	81.0	0.0	0.0	0.000	27376	0.249	0.00	106
12:18:36.009	88	26304	0.246	93.1	81.1	0.0	0.0	0.000	26304	0.246	0.00	102
12:18:37.015	89	24321	0.257	95.9	85.7	0.0	0.0	0.000	24321	0.257	0.00	95
12:18:38.005	90	28944	0.268	93.9	77.7	0.0	0.0	0.000	28944	0.268	0.00	113
Nov 25, 2020 ..Interval.. .ReqstDps... .cpu%... read ...read.... ..write.... .mb/sec..												
12:18:39.006	91	28075	0.225	96.0	82.0	0.0	0.0	0.000	28075	0.225	0.00	109
12:18:40.007	92	23962	0.276	91.0	78.0	0.0	0.0	0.000	23962	0.276	0.00	93
12:18:41.006	93	24054	0.301	80.2	66.3	0.0	0.0	0.000	24054	0.301	0.00	93
12:18:42.005	94	24024	0.248	94.9	79.5	0.0	0.0	0.000	24024	0.248	0.00	93
12:18:43.005	95	30961	0.253	97.0	79.8	0.0	0.0	0.000	30961	0.253	0.00	120
12:18:44.008	96	25448	0.240	97.0	84.0	0.0	0.0	0.000	25448	0.240	0.00	99
12:18:45.004	97	27694	0.270	96.0	79.8	0.0	0.0	0.000	27694	0.270	0.00	108
12:18:46.005	98	17883	0.402	94.0	45.0	0.0	0.0	0.000	17883	0.402	0.00	69
12:18:47.011	99	27188	0.247	89.1	75.2	0.0	0.0	0.000	27188	0.247	0.00	106
12:18:48.005	100	23934	0.242	95.0	81.0	0.0	0.0	0.000	23934	0.242	0.00	93
12:18:49.008	101	30837	0.249	97.0	83.0	0.0	0.0	0.000	30837	0.249	0.00	120
12:18:50.011	102	25716	0.247	94.9	78.7	0.0	0.0	0.000	25716	0.247	0.00	100
12:18:51.004	103	25729	0.283	95.0	78.0	0.0	0.0	0.000	25729	0.283	0.00	100
12:18:52.005	104	29374	0.223	95.0	82.1	0.0	0.0	0.000	29374	0.223	0.00	114
12:18:53.006	105	25790	0.252	91.9	82.8	0.0	0.0	0.000	25790	0.252	0.00	100
12:18:54.007	106	24352	0.258	94.1	82.3	0.0	0.0	0.000	24352	0.258	0.00	95
12:18:55.012	107	24344	0.281	87.9	73.7	0.0	0.0	0.000	24344	0.281	0.00	93
12:18:56.006	108	23900	0.297	86.9	72.7	0.0	0.0	0.000	23900	0.297	0.00	93
12:18:57.015	109	25322	0.300	91.1	79.2	0.0	0.0	0.000	25322	0.300	0.00	98
12:18:58.005	110	30774	0.215	97.0	83.0	0.0	0.0	0.000	30774	0.215	0.00	120
12:18:59.008	111	24905	0.239	96.9	85.7	0.0	0.0	0.000	24905	0.239	0.00	97
12:19:00.005	112	28535	0.259	96.0	82.1	0.0	0.0	0.000	28535	0.259	0.00	111
12:19:01.007	113	27604	0.231	98.0	79.0	0.0	0.0	0.000	27604	0.231	0.00	107
12:19:02.005	114	24049	0.254	96.9	85.7	0.0	0.0	0.000	24049	0.254	0.00	93
12:19:03.008	115	31741	0.241	97.0	84.0	0.0	0.0	0.000	31741	0.241	0.00	123
12:19:04.007	116	24592	0.278	87.0	72.0	0.0	0.0	0.000	24592	0.278	0.00	96
12:19:05.005	117	24558	0.244	98.0	82.6	0.0	0.0	0.000	24558	0.244	0.00	95
12:19:06.011	118	27043	0.283	93.0	81.0	0.0	0.0	0.000	27043	0.283	0.00	105
12:19:07.007	119	29791	0.215	98.0	84.0	0.0	0.0	0.000	29791	0.215	0.00	116
12:19:08.013	120	24149	0.264	97.0	82.8	0.0	0.0	0.000	24149	0.264	0.00	94
12:19:08.025	avg 2-120	25293	0.269	94.1	75.8	0.0	0.0	0.000	25293	0.269	0.00	98
12:19:08.025	std 2-120	3439.4	6.156						3439.4	6.156		
12:19:08.025	max 2-120	31871	464.27						31871	464.27		
12:19:09.125 Vdbench execution completed successfully												