CS-344 | OS LAB | ASSIGNMENT-4 | GROUP-5 |
DRISTIRON SAIKIA, KRISHNA PANDE, NIKUNJ HEDA, NIYATI CHAUDHARY |
180101022, 180101038, 180101049, 18101053

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
```

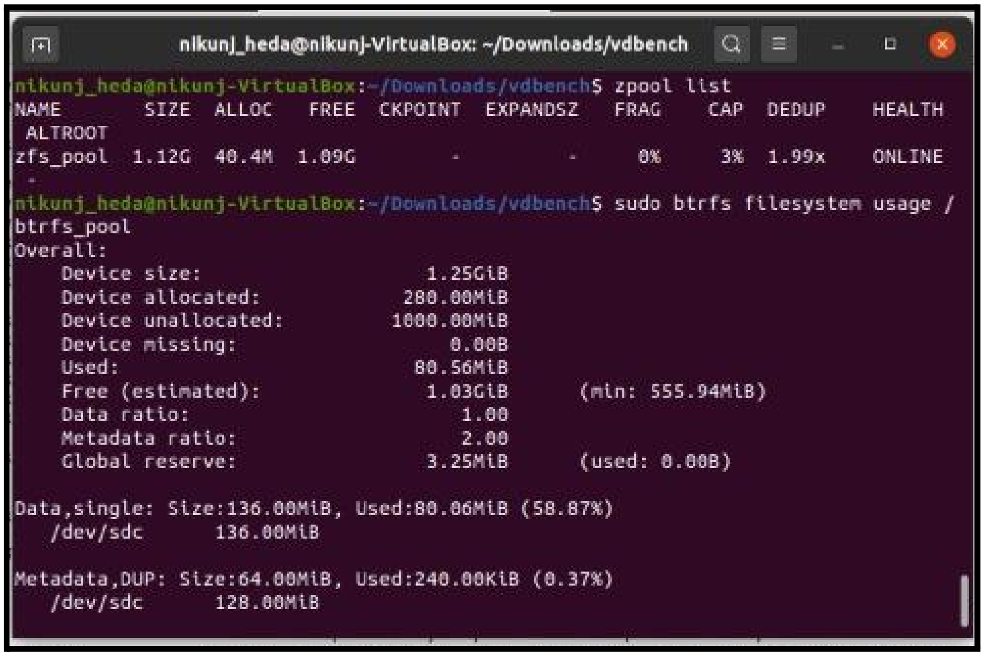
The parameter file will use a directory structure of 4 directories and 10 files in each leaf directory.
The RD parameter 'format=yes' causes the directory structure to be completely created, including initialization of all files to the requested size of 2MB. Larger sized files gave us clearer expected 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.

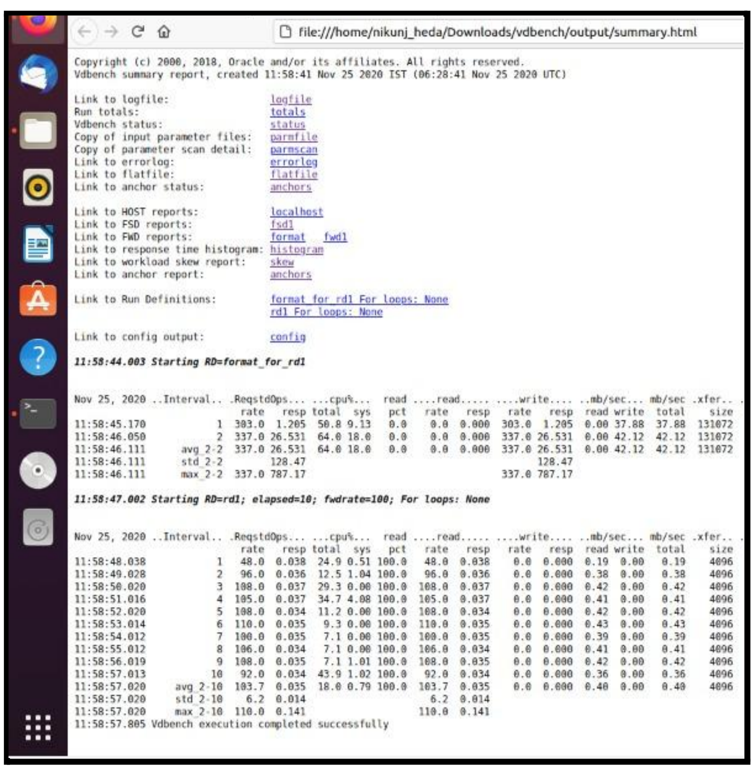


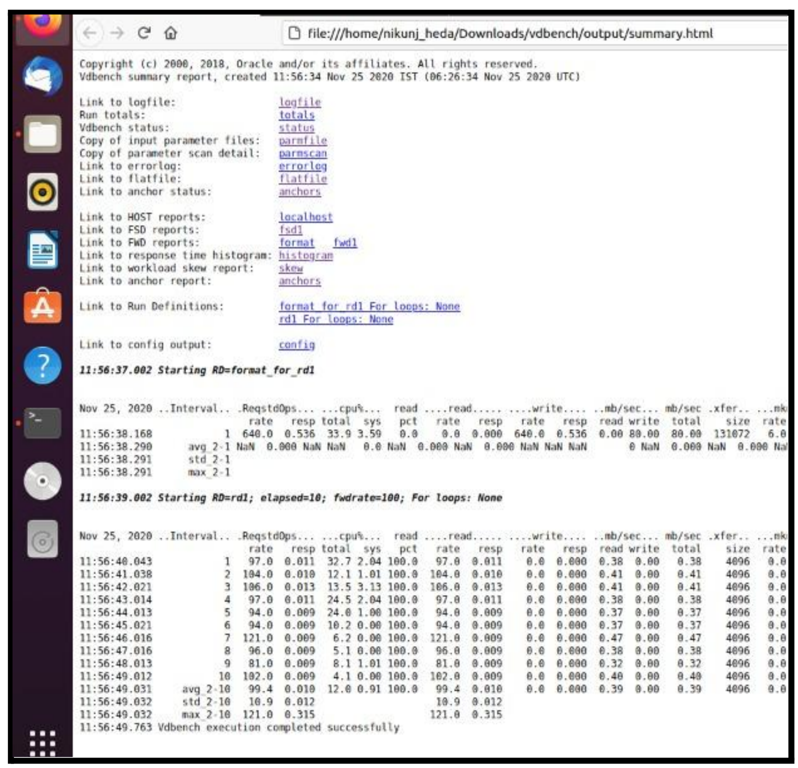
We can clearly see the allocated disk space for the zfs_pool is only 40.4MB

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.





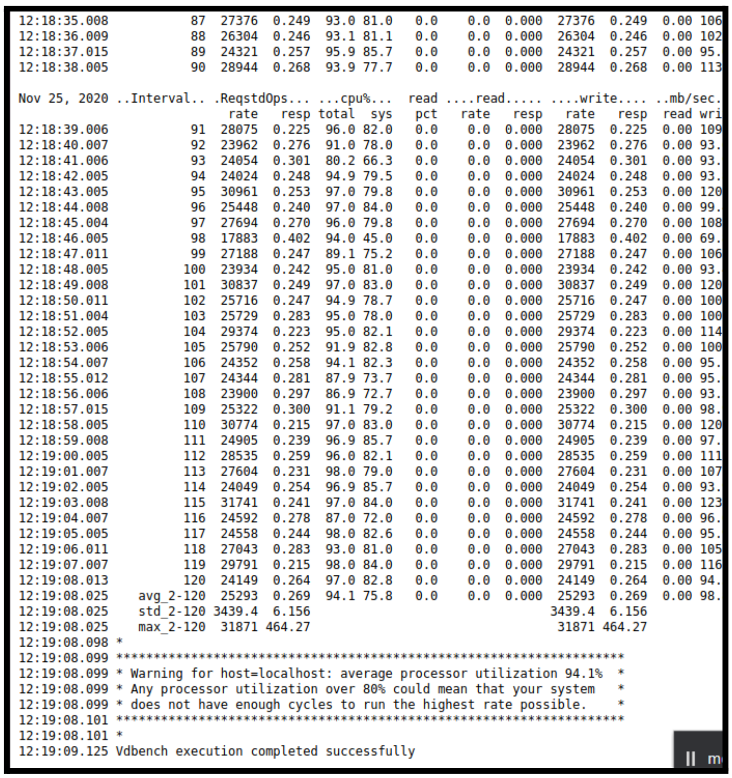
❖ C-O-W Feature and snapshotting

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).**



Btrfs pollutes the file system namespace by keeping snapshots and file systems in the same location. A snapshot appears as a "copy-on-write" (and writable!) directory which is a sibling of the directory containing the subvolume you just snapshotted.
In contrast, ZFS intelligently separates snaps which makes it possible for ZFS not to list them by default, or to list them separately from file systems. The name of the snapshot is distinct and separate from the name of the file system that was snapshotted. This lack of clutter makes ZFS more efficacious for you to manage large numbers of snapshots with large numbers of file systems, and less likely for you to touch data you snapshotted for backup purposes.