Group C38

OS-344 Assignment-4

Vatsal Gupta 200101105
Sweeya Reddy 200101079
Pranshu Kandoi 200101086

ZFS

As a component of the **Solaris OS**, ZFS was developed. ZFS serves as both a file system and a volume manager, which makes the operations more streamlined and compatible and is used to manage space on mass-storage devices. It was created with security as its top concern, and it makes sure that every action linked to managing files or disk usage is checked for accuracy and optimized — something that standalone volume and file managers can't do.
The ZFS file system and volume manager is characterized by data integrity, high scalability and built-instorage features such as:

- **Replication** - theprocess of makinga replica (a copy) of something.
- **Deduplication** - aprocess that eliminates redundant copies of data and reduces storage overhead.

Target for comparison. <Explained in detail below>

- **Compression** - a reductioninthenumber of bitsneeded to represent data.
- **Snapshots** - a set of reference markers for data at aparticularpoint intime.
- **Clones** - anidentical copy of something.
- **Data protection** - theprocess of safeguardingimportant informationfrom corruption and/or loss.

EXT-4

The performance and capacity of the EXT4 file system are its main priorities. Instead of fixed-size blocks, extents are used in this system to allocate storage. Extents are simply identified by their beginning and ending locations on the hard disc. This method of storing the essential location of data in files avoids fragmentation of the memory allocated by the EXT4 file system and enables the storage of the file's location of data with the use of a few pointers rather than a pointer referring to every block of memory that the file

occupies.
Additionally, it uses delayed allocation, which boosts efficiency and makes it easier for the file system to allocate contiguous blocks of memory because it already knows how much memory it needs to map before allocating any memory.
The following are a few notable features of ext4:

1. File System Size

Target for comparison. <Explained in detail below>

2. Extents

The commonly used file systems of Unix/Linux like ext2 and ext3 use direct, indirect, double indirect, and triple indirect block mapping scheme to map file offsets to on-disk blocks. This is ideal for small files. However, in the case of large files, there will be a large number of mappings which will reduce performance, especially while seeking and deleting data. So extents were introduced to replace the block mapping scheme.

The concept extent means “a contiguous sequence of physical blocks”. Large files are split into several “extents”. The files are allocated to a ‘single extent’ instead of a particular size, thus avoiding the indirect mapping of blocks. Thus extents allow less fragmentation (due to a sequential block allocation) and improves performance.

3. Delayed and Multiblock allocation

In ext4 file systems, multiblock allocation (mballoc) allocates multiple blocks for a file in a single operation, instead of allocating it one by one, as is the case in ext3. This will optimize the allocation of memory.
In delayed allocation, if a function writes data onto a disk instead of allocating it at once, it will get stored in the cache.

The above mentioned techniques will reduce disk fragmentation.

4. Online defragmentation and fsck speed

The fragmentation rate is lesser in ext4 system.

5. Journal check summing

Ext4 uses the checksum of the journal to find out health of the journal blocks. This is used to avoid massive data corruption. You can turn off the journaling mode in ext4, if it causes overhead.

6. Persistent preallocation

Persistent preallocation allows the application to allocate contiguous blocks with a fixed size, before writing the data. This will ensure Lesser fragmentation (because blocks are allocated as contiguously as possible) and ensure that applications have enough space to work.

This feature is appropriate for real time applications, databases and content streaming.

7. Inodes / Timestamps

The ext4 filesystem has a large inode size of 256 bytes by default whereas ext3 has only 128 bytes for inodes.

8. Backward compatibility

Ext3 file systems can be migrated to ext4 easily without formatting or reinstalling the OS, provided the kernel supports the ext4 file system.

Comparison of features

- We compare the files systems ZFS and ext4 using vdbench.
- We created workloads to test 2 features - Deduplication and management of large files - on each file system.

Deduplication

Deduplication is the removal of redundant copies of data. This can free up a significant amount of space on the hard drive and is particularly helpful in some settings where a lot of duplicate data is present, with or without minor changes. However, using this is only advised in exceptional circumstances due to the trade-off of high overhead computations.By hashing (using a secure hash like SHA256) a portion of the data to create an approximate unique signature and storing these in a hash table, deduplication is achieved. Data with a pre-existing signature is considered to be a copy of the data whose signature matches it when new data's signature is compared to values that already exist in the hash table. Depending on the amount of data that needs to be hashed into a signature, deduplication can be implemented at various levels - These are at the file, block, and byte levels. Depending on whether the process occurs as the data is being written or whether the copies are hashed and deleted when the CPU is free, deduplication can also be synchronous or asynchronous. ZFS uses block-level synchronous deduplication for its deduplication feature. Deduplication is not supported by EXT4.These are at the file, block, and byte levels. Depending on whether the process occurs as the data is being written or whether the copies are hashed and deleted when the

CPU is free, deduplication can also be synchronous or asynchronous. ZFS uses block-level synchronous deduplication for its deduplication feature. Deduplication is not supported by EXT4.

Creation and management of large files

Ext4 permits filesystems up to 1 exbibyte (2^60 bytes) size and files up to 16 tebibytes size (16 * 2^40 bytes). Whereas, the ext3 file system supports only a maximum filesystem size of 16 TB and a maximum file size of 2 TB. Another advantage of ext4 is its support for the creation of enormous number of sub directories under a directory. More than 64000 sub directories can be created under a directory(unlimited) in ext4, compared to 32000 in ext3. EXT4 optimizes the creation and handling of very large files very well. This is due to Extents saving a lot of space and time used to save and access huge mapping of blocks of data occupied by large files.

Other features in EXT4 also aid in the proper and effective operation of this new extent-based mapping system. Multiblock allocation is a feature of EXT4 that allows for easy allocation of contiguous blocks of memory while avoiding a significant amount of overhead by allocating multiple blocks in a single call as opposed to one block per call.

This works in conjunction with delayed allocation, which doesn't write to disk during every write operation instead noting the data to be written before using multiblock allocation to write a sizable chunk of data into a contiguous memory segment.

The file system must have the ability to handle large files in order to manage large files.

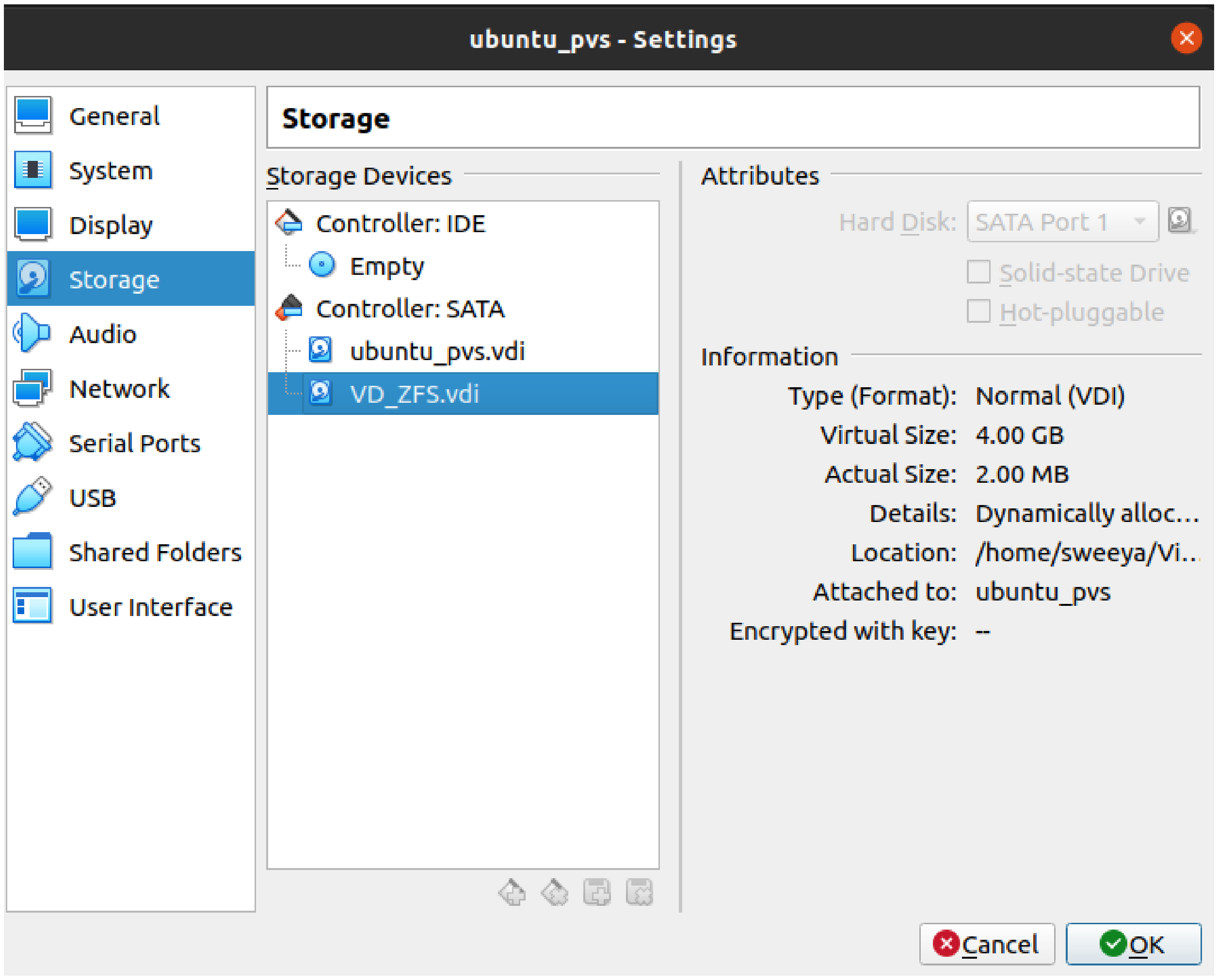
- Ext4 uses extents (as opposed to ext2 and ext3's traditional block mapping scheme), which improves performance when working with large files and reduces metadata overhead, but has a higher metadata overhead than ZFS.
- Ext4 employs internal 48-bit addressing, allowing for the theoretical allocation of files up to 16 TiB on filesystems up to 1,000,000 TiB. (1 EiB).

Creating a ZFS partition

We created a ZFS pool on an VirtualBox(UBUNUTU 20.04 LTS).

1. First, we need to add an extra hard disk to our VM. In the VM settings, under the storage section, add an hard disk in the **Controller: SATA section.**

Here, VD_ZFS.vdi is our added hard disk.



Storage

Storage Devices

- Controller: IDE
  - Empty
- Controller: SATA
  - ubuntu_pvs.vdi
  - VD_ZFS.vdi

Attributes

Hard Disk: SATA Port 1

Solid-state Drive

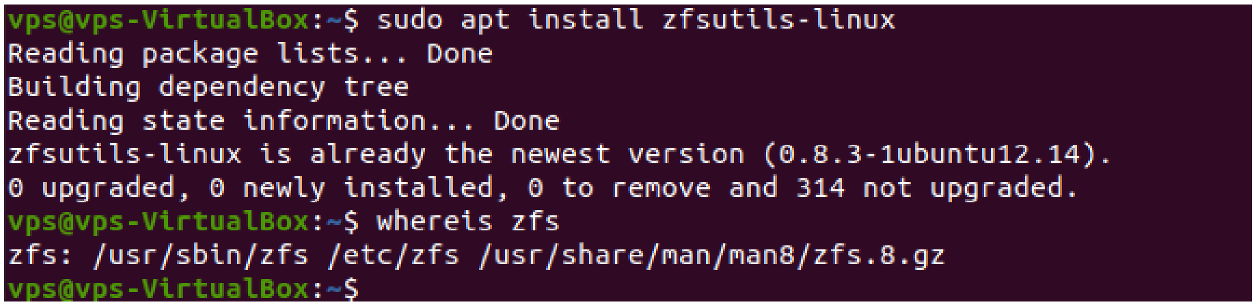
Hot-pluggable

Information

Type (Format):	Normal (VDI)
Virtual Size:	4.00 GB
Actual Size:	2.00 MB
Details:	Dynamically alloc...
Location:	/home/sweeya/Vi...
Attached to:	ubuntu_pvs
Encrypted with key:	--

2. Install ZFS on the VM and check its installation

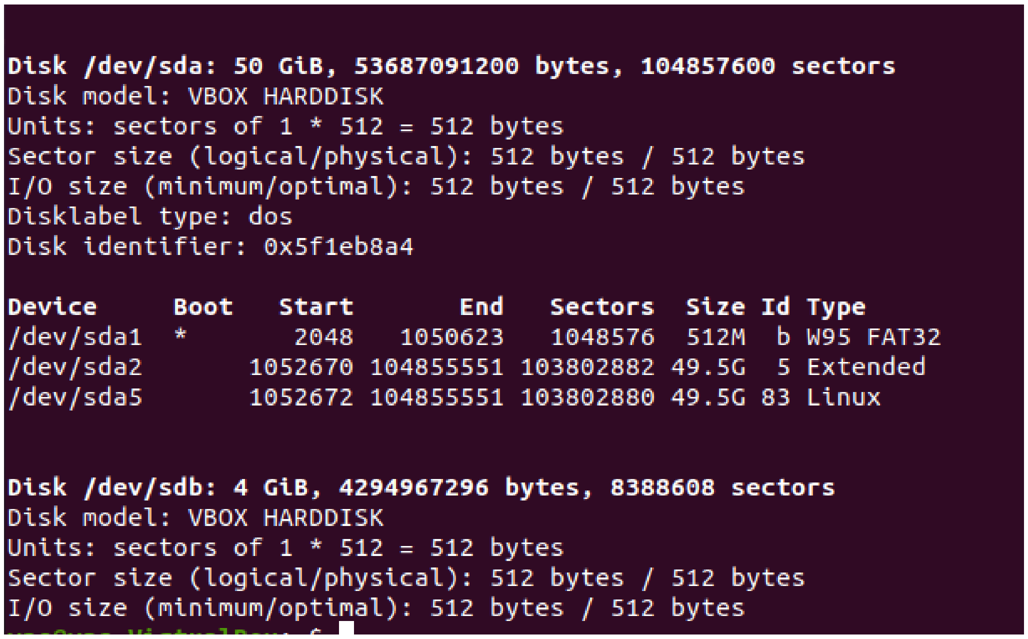
NOTE: we had already installed zfs



```
vps@vps-VirtualBox:~$ sudo apt install zfsutils-linux
Reading package lists... Done
Building dependency tree
Reading state information... Done
zfsutils-linux is already the newest version (0.8.3-1ubuntu12.14).
0 upgraded, 0 newly installed, 0 to remove and 314 not upgraded.
vps@vps-VirtualBox:~$ whereis zfs
zfs: /usr/sbin/zfs /etc/zfs /usr/share/man/man8/zfs.8.gz
vps@vps-VirtualBox:~$
```

3. Check available disks to create the necessary storage pool

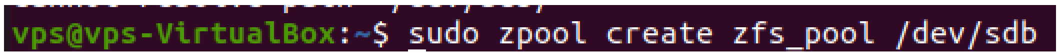
we used the command sudo fdisk -l which lists all the available disks.



```
Disk /dev/sda: 50 GiB, 53687091200 bytes, 104857600 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x5f1eb8a4

Device     Boot   Start       End   Sectors  Size Id Type
/dev/sda1  *       2048   1050623   1048576  512M  b W95 FAT32
/dev/sda2       1052670 104855551 103802882 49.5G  5 Extended
/dev/sda5       1052672 104855551 103802880 49.5G 83 Linux


Disk /dev/sdb: 4 GiB, 4294967296 bytes, 8388608 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

We will use **/dev/sdb** to create our ZFS pool.

4. To create the striped pool, run the following command

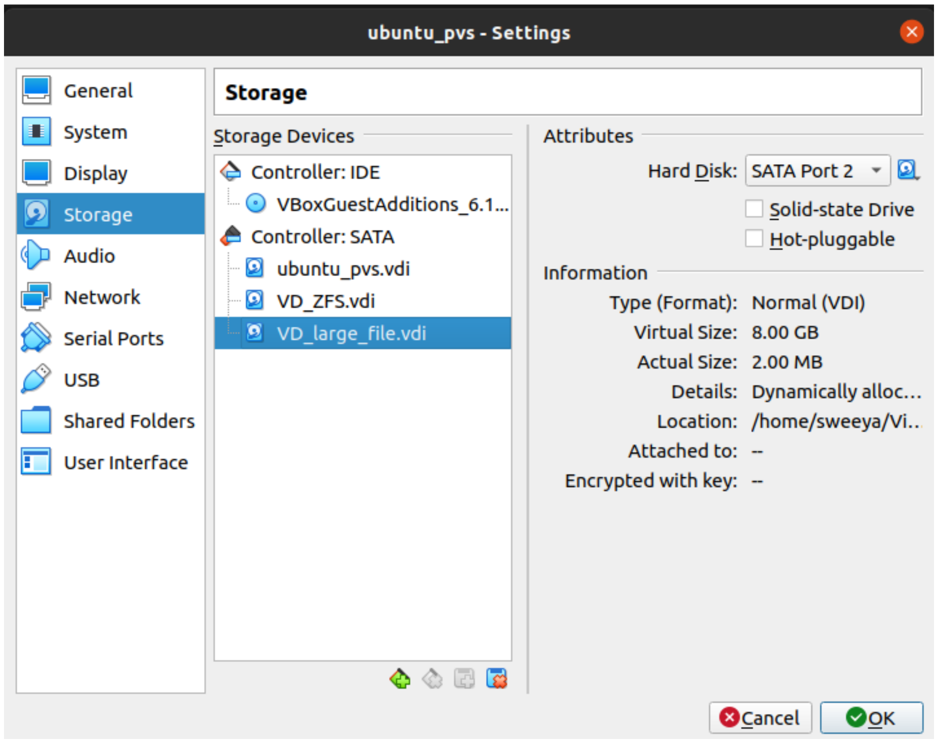


```
vps@vps-VirtualBox:~$ sudo zpool create zfs_pool /dev/sdb
```

Finally our pool is made with the name **zfs_pool**

Creating an EXT4 partition

As before we added a new hard disk with name VD_large_file.vdi

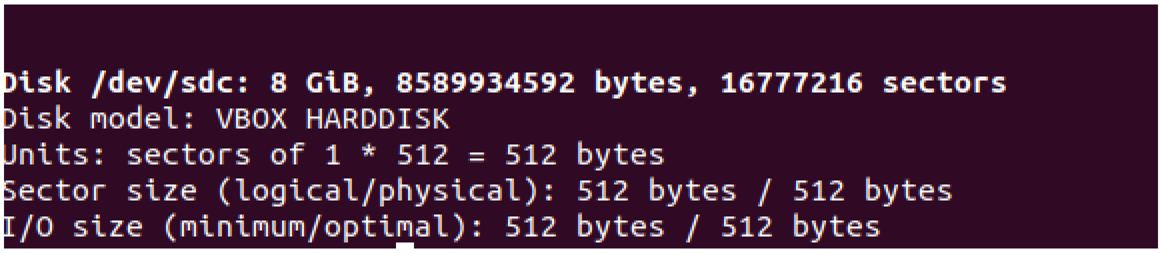


Hard Disk: SATA Port 2
Solid-state Drive
Hot-pluggable

Information
Type (Format): Normal (VDI)
Virtual Size: 8.00 GB
Actual Size: 2.00 MB
Details: Dynamically alloc...
Location: /home/sweeya/Vi..
Attached to: --
Encrypted with key: --

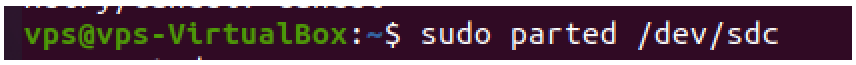
1. We check available disks like before.

We will use /dev/sdc to create our ext4 pool.



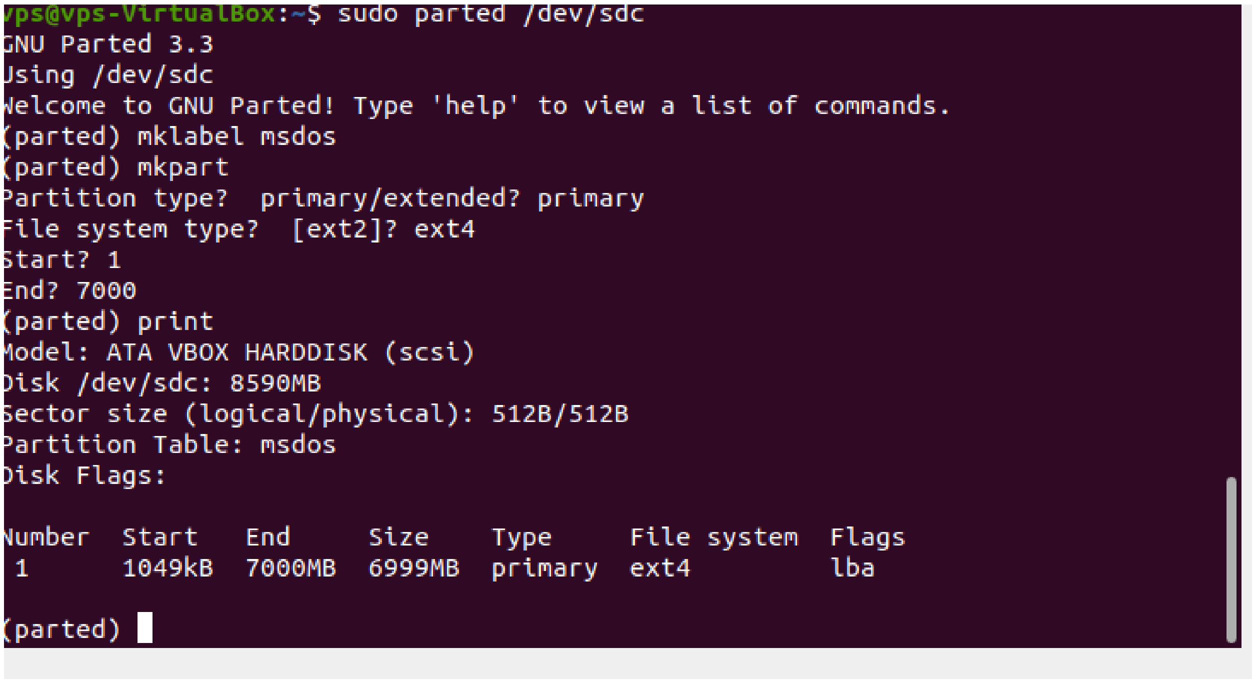
```
Disk /dev/sdc: 8 GiB, 8589934592 bytes, 16777216 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

2. We now use the parted command to configure our selected disk partition:



```
vps@vps-VirtualBox:~$ sudo parted /dev/sdc
```

3. Now we will label the new partition using mklabel in parted and then create a partition using the mkpart command and specify the parameters.

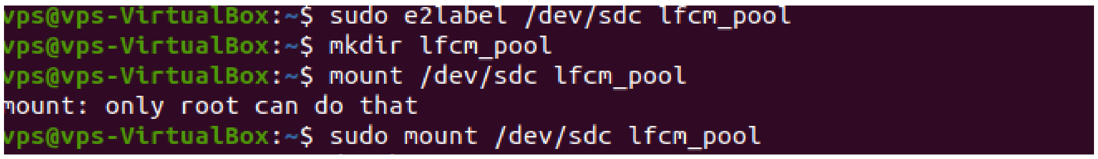


```
vps@vps-VirtualBox:~$ sudo parted /dev/sdc
GNU Parted 3.3
Using /dev/sdc
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel msdos
(parted) mkpart
Partition type?  primary/extended? primary
File system type?  [ext2]? ext4
Start? 1
End? 7000
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdc: 8590MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  7000MB  6999MB  primary  ext4         lba

(parted)
```

4. To format the file system properly with ext4 file system we execute the following command:

mkfs.ext4 /dev/sdcScreenshot from 2022-11-14 22-55-51

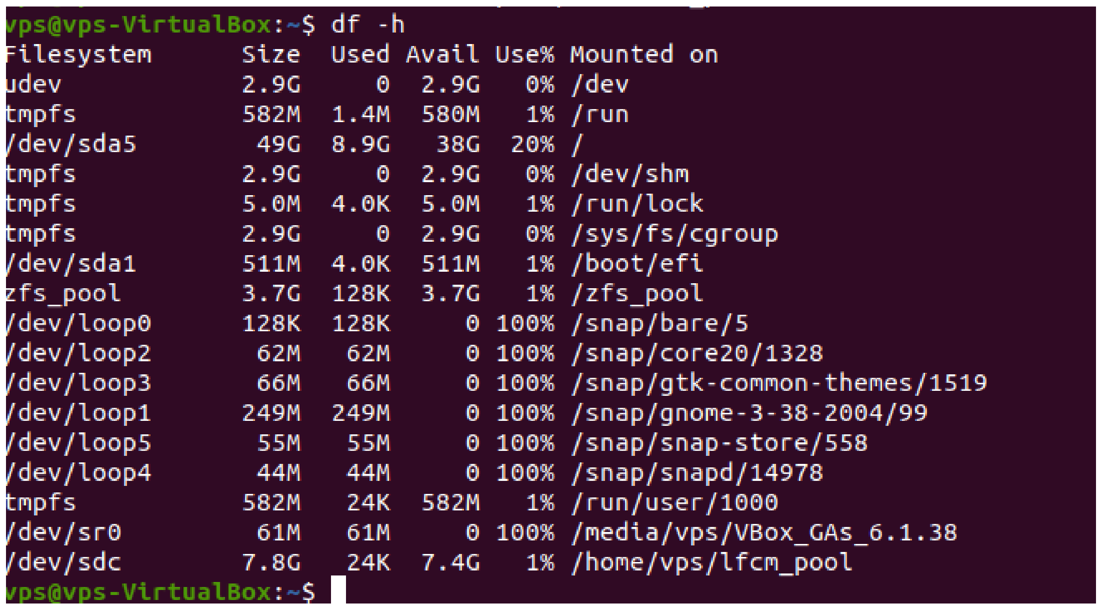
5. We label the partition using e2label command. Also, we create a mount point for our new partition to do so we make a new directory using the mkdir command.Lastly, we will mount our partition to the mount point using the mount command.



```
vps@vps-VirtualBox:~$ sudo e2label /dev/sdc lfcm_pool
vps@vps-VirtualBox:~$ mkdir lfcm_pool
vps@vps-VirtualBox:~$ mount /dev/sdc lfcm_pool
mount: only root can do that
vps@vps-VirtualBox:~$ sudo mount /dev/sdc lfcm_pool
```

6. We have successfully created an ext4 pool named **lfcm_pool**.

NOTE: we can see the filesystem directories that we have created using df -h as below



```
vps@vps-VirtualBox:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            2.9G     0  2.9G   0% /dev
tmpfs           582M  1.4M  580M   1% /run
/dev/sda5        49G  8.9G   38G  20% /
tmpfs           2.9G     0  2.9G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           2.9G     0  2.9G   0% /sys/fs/cgroup
/dev/sda1       511M  4.0K  511M   1% /boot/efi
zfs_pool        3.7G  128K  3.7G   1% /zfs_pool
/dev/loop0      128K  128K     0 100% /snap/bare/5
/dev/loop2       62M   62M     0 100% /snap/core20/1328
/dev/loop3       66M   66M     0 100% /snap/gtk-common-themes/1519
/dev/loop1      249M  249M     0 100% /snap/gnome-3-38-2004/99
/dev/loop5       55M   55M     0 100% /snap/snap-store/558
/dev/loop4       44M   44M     0 100% /snap/snapd/14978
tmpfs           582M   24K  582M   1% /run/user/1000
/dev/sr0         61M   61M     0 100% /media/vps/VBox_GAs_6.1.38
/dev/sdc        7.8G   24K  7.4G   1% /home/vps/lfcm_pool
vps@vps-VirtualBox:~$
```

VDBENCH

Vdbench has three basic definitions in the configuration file.

1. FSD or filesystem definition: This defines what filesystem to use for the testing.
FSD defines the filesystem storage used for the testing. The FSD name should be unique.
FSD parameters include:

- **fsd=name :** Unique name for this File System Definition
- **anchor=/dir :** The name of the directory where the directory structure will be created
- **depth=nn :** How many levels of directories to create under the anchor.
- **files=nn :** How many files to create in the lowest level of directories
- **width=nn :** How many directories to create in each new directory
- **sizes=(nn,nn,…..) :** Specifies the size(s) of the files that will be created.

2. FWD or filesystem workload definition: This defines the workload parameter for the testing.
FWD defines the filesystem workload used for the testing.
The FWD name should be unique.
FWD parameters include:

- **fwd=name :** Unique name for this Filesystem Workload Definition
- **fsd=(xx,….) :** Name(s) of Filesystem Definitions to use.

- **operation=xxxx :** Specifies a single file system operation that must be done for this workload.
- **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.
- **threads=nn :** How many concurrent threads to run for this workload.
- **xfersize=(nn,…) :** Specifies the data transfer size(s) to use for read and write operations.

3. RD or run definition: This defines storage, workload and run duration.RD defines what storage and workload will be run together and for how long. Each run definition name must be unique.
RD parameters include:

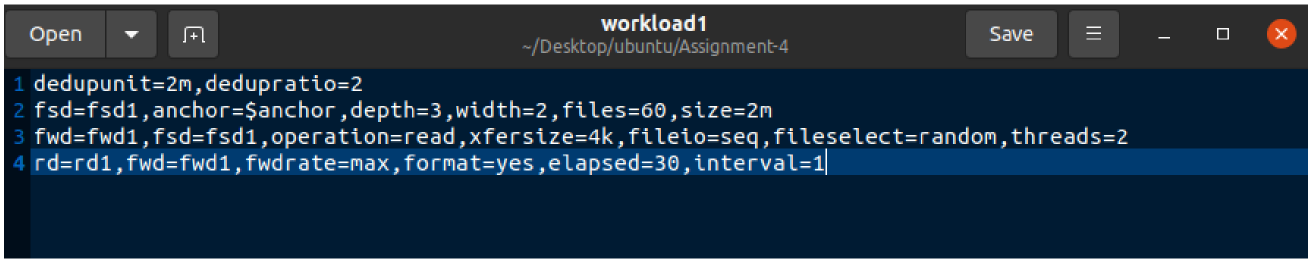
- **fwd=(xx,yy,..) :** Name(s) of Filesystem Workload Definitions to use.
- **format :** 'format=yes' causes the directory structure to be completely created, including initialization of all files to the requested size.
- **elapsed :** How long to run in seconds.
- **interval :** Statistics collection interval in seconds.
- **fwdrate=nn :** Run a workload of nn operations per second

EXPERIMENT

Deduplication - Experiment

Workload explaination

We created the following workload for data deduplication (workload1). We will use this workload to compare the space occupied by the new files in ZFS with the space occupied in ext4.



```
dedupunit=2m,dedupratio=2
fsd=fsd1,anchor=$anchor,depth=3,width=2,files=60,size=2m
fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=seq,fileselect=random,threads=2
rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=30,interval=1
```

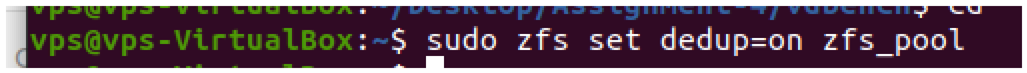
- In a nested folder structure with a depth of 3 and a width of 2, we are essentially producing 480 files ($60*2^3$), each of size 2MB.

Following that, these files are read consecutively for thirty seconds in order to monitor statistics.

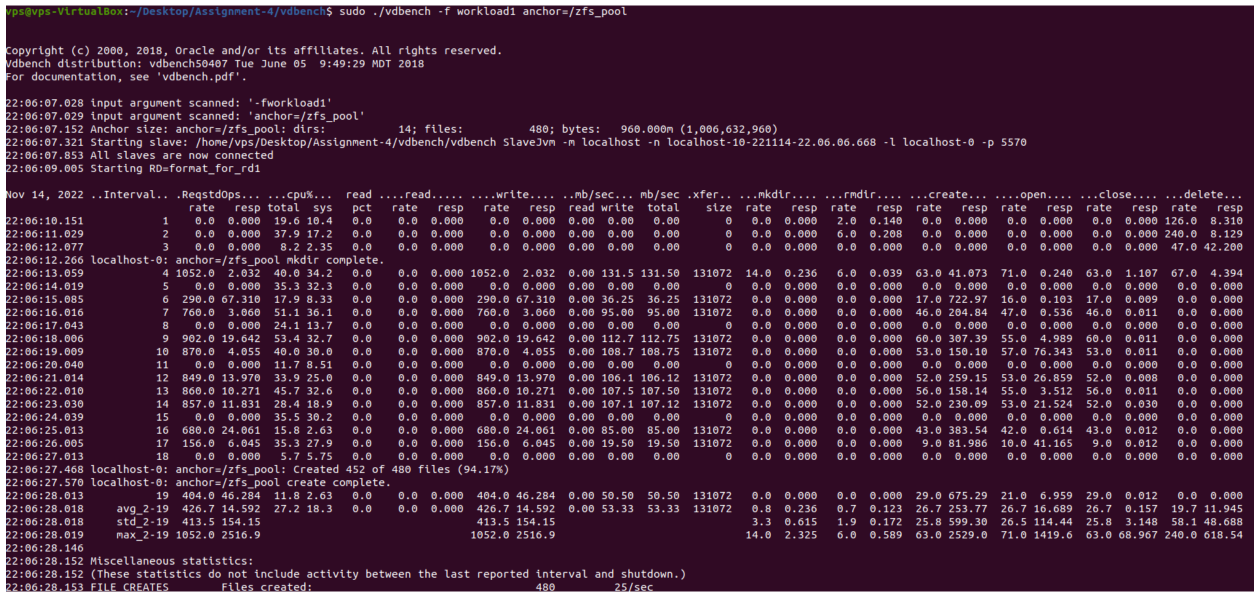
- Dedupratio is set to 2, and dedupunit to 2MB. The ratio of the total number of blocks (of size dedupunit) to the number of blocks containing unique data is known as the dedupratio. The size of the block that will be compared to earlier blocks to look for duplicates is the dedupunit, on the other hand. Due to the size of a single file, we set it to 2MB. Consequently, half of the files will essentially be copies of the other half.

Running and Observation

dedup is turned on



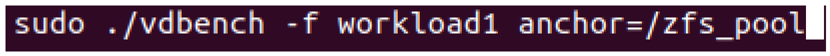
```
vps@vps-VirtualBox:~/Desktop/Assignment-4/vdbench$ cd
vps@vps-VirtualBox:~$ sudo zfs set dedup=on zfs_pool
```



```
vps@vps-VirtualBox:~/Desktop/Assignment-4/vdbench$ sudo ./vdbench -f workload1 anchor=/zfs_pool


Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Vdbench distribution: vdbench50407 Tue June 05  9:49:29 MDT 2018
For documentation, see 'vdbench.pdf'.

22:06:07.028 input argument scanned: '-fworkload1'
22:06:07.029 input argument scanned: 'anchor=/zfs_pool'
22:06:07.152 Anchor size: anchor=/zfs_pool: dirs:           14; files:          480; bytes:   960.000m (1,006,632,960)
22:06:07.321 Starting slave: /home/vps/Desktop/Assignment-4/vdbench/vdbench SlaveJvm -m localhost -n localhost-10-221114-22.06.06.668 -l localhost-0 -p 5570
22:06:07.853 All slaves are now connected
22:06:09.005 Starting RD=format_for_rd1

Nov 14, 2022 ..Interval.. .ReqstdOps... ...cpu%...  read ....read..... ....write.... ..mb/sec... mb/sec .xfer.. ...mkdir.... ...rmdir.... ...create... ....open.... ...close.... ...delete...
                              rate   resp total  sys   pct   rate   resp   rate   resp  read write  total    size  rate   resp  rate   resp  rate   resp  rate   resp  rate   resp  rate   resp
22:06:10.151             1     0.0  0.000  19.6 10.4   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   2.0  0.140   0.0  0.000   0.0  0.000   0.0  0.000 126.0  8.310
22:06:11.029             2     0.0  0.000  37.9 17.2   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   6.0  0.208   0.0  0.000   0.0  0.000   0.0  0.000 240.0  8.129
22:06:12.077             3     0.0  0.000   8.2 2.35   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000  47.0 42.200
22:06:12.266 localhost-0: anchor=/zfs_pool mkdir complete.
22:06:13.059             4  1052.0  2.032  40.0 34.2   0.0    0.0  0.000 1052.0  2.032  0.00 131.5 131.50  131072  14.0  0.236   6.0  0.039  63.0 41.073  71.0  0.240  63.0  1.107  67.0  4.394
22:06:14.019             5     0.0  0.000  35.3 32.3   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000
22:06:15.085             6   290.0 67.310  17.9 8.33   0.0    0.0  0.000  290.0 67.310  0.00 36.25  36.25  131072   0.0  0.000   0.0  0.000  17.0 722.97  16.0  0.103  17.0  0.009   0.0  0.000
22:06:16.016             7   760.0  3.060  51.1 36.1   0.0    0.0  0.000  760.0  3.060  0.00 95.00  95.00  131072   0.0  0.000   0.0  0.000  46.0 204.84  47.0  0.536  46.0  0.011   0.0  0.000
22:06:17.043             8     0.0  0.000  24.1 13.7   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000
22:06:18.006             9   902.0 19.642  53.4 32.7   0.0    0.0  0.000  902.0 19.642  0.00 112.7 112.75  131072   0.0  0.000   0.0  0.000  60.0 307.39  55.0  4.989  60.0  0.011   0.0  0.000
22:06:19.009            10   870.0  4.055  40.0 30.0   0.0    0.0  0.000  870.0  4.055  0.00 108.7 108.75  131072   0.0  0.000   0.0  0.000  53.0 150.10  57.0 76.343  53.0  0.011   0.0  0.000
22:06:20.040            11     0.0  0.000  11.7 8.51   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000
22:06:21.014            12   849.0 13.970  33.9 25.0   0.0    0.0  0.000  849.0 13.970  0.00 106.1 106.12  131072   0.0  0.000   0.0  0.000  52.0 259.15  53.0 26.859  52.0  0.008   0.0  0.000
22:06:22.010            13   860.0 10.271  45.7 32.6   0.0    0.0  0.000  860.0 10.271  0.00 107.5 107.50  131072   0.0  0.000   0.0  0.000  56.0 158.14  55.0  3.512  56.0  0.011   0.0  0.000
22:06:23.030            14   857.0 11.831  28.4 18.9   0.0    0.0  0.000  857.0 11.831  0.00 107.1 107.12  131072   0.0  0.000   0.0  0.000  52.0 230.09  53.0 21.524  52.0  0.030   0.0  0.000
22:06:24.039            15     0.0  0.000  35.5 30.2   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000
22:06:25.013            16   680.0 24.061  15.8 2.63   0.0    0.0  0.000  680.0 24.061  0.00 85.00  85.00  131072   0.0  0.000   0.0  0.000  43.0 383.54  42.0  0.614  43.0  0.012   0.0  0.000
22:06:26.005            17   156.0  6.045  35.3 27.9   0.0    0.0  0.000  156.0  6.045  0.00 19.50  19.50  131072   0.0  0.000   0.0  0.000   9.0 81.986  10.0 41.165   9.0  0.012   0.0  0.000
22:06:27.013            18     0.0  0.000   5.7 5.75   0.0    0.0  0.000    0.0  0.000  0.00  0.00   0.00       0   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000   0.0  0.000
22:06:27.468 localhost-0: anchor=/zfs_pool: Created 452 of 480 files (94.17%)
22:06:27.570 localhost-0: anchor=/zfs_pool create complete.
22:06:28.013            19   404.0 46.284  11.8 2.63   0.0    0.0  0.000  404.0 46.284  0.00 50.50  50.50  131072   0.0  0.000   0.0  0.000  29.0 675.29  21.0  6.959  29.0  0.012   0.0  0.000
22:06:28.018      avg_2-19   426.7 14.592  27.2 18.3   0.0    0.0  0.000  426.7 14.592  0.00 53.33  53.33  131072   0.8  0.236   0.7  0.123  26.7 253.77  26.7 16.689  26.7  0.157  19.7 11.945
22:06:28.018      std_2-19   413.5 154.15                                 413.5 154.15                              3.3  0.615   1.9  0.172  25.8 599.30  26.5 114.44  25.8  3.148  58.1 48.688
22:06:28.019      max_2-19  1052.0 2516.9                                1052.0 2516.9                             14.0  2.325   6.0  0.589  63.0 2529.0  71.0 1419.6  63.0 68.967 240.0 618.54
22:06:28.146
22:06:28.152 Miscellaneous statistics:
22:06:28.152 (These statistics do not include activity between the last reported interval and shutdown.)
22:06:28.153 FILE_CREATES        Files created:                              480         25/sec
```

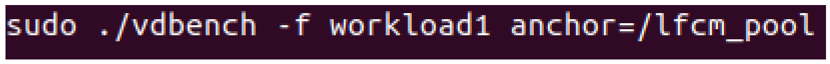
ZFS has a data deduplication feature which we turned on using (zfs_pool is the name of the zfs pool we set up).

- We run this workload on the ZFS file system by setting anchor to the directory of the ZFS Pool (basically the folder pointing to the ZFS Pool)



```
sudo ./vdbench -f workload1 anchor=/zfs_pool
```

- We run this workload on the ext4 file system by setting anchor to the directory of the folder pointing to the ext4 drive



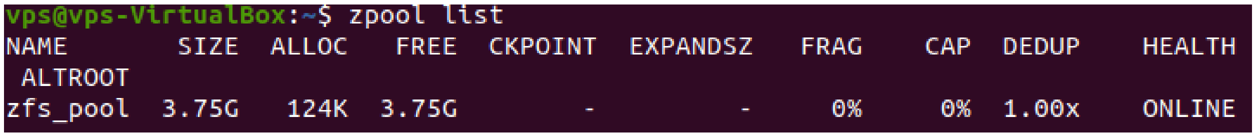
```
sudo ./vdbench -f workload1 anchor=/lfcm_pool
```

- We found the following results:

i. ZFS:

1. Empty ZFS folder had 124 KB of data.
2. After running the workload, the ZFS folder had 483 MB of data when dedup is on and 962 MB when off.
3. We observed a deduplication ratio of 2.00x (which is what we wanted).
4. This means that the new files took 483 MB of space. However, the intended space is 960MB (2MB*480). Hence, using the data deduplication feature, instead of maintaining whole blocks of data,when duplicates are found, ZFS simply makes a pointer point to the old data.

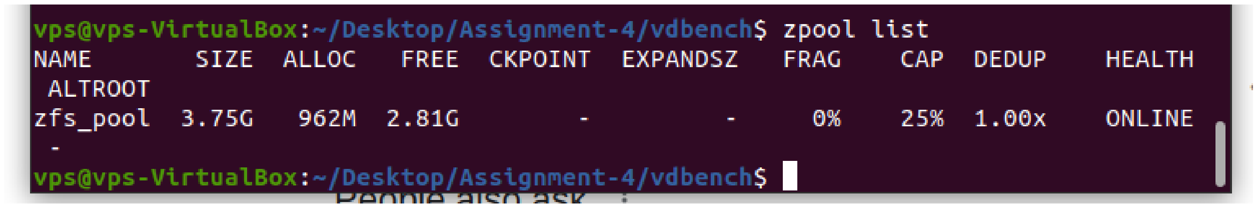
- Before Workload:



```
vps@vps-VirtualBox:~$ zpool list
NAME         SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH
 ALTROOT
zfs_pool  3.75G   124K  3.75G        -         -     0%     0%  1.00x    ONLINE
```

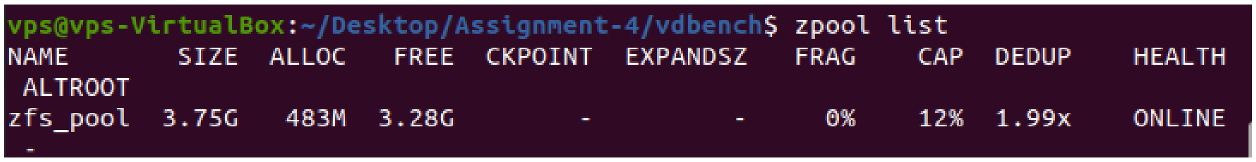
- After Workload:

  Dedup is off



```
vps@vps-VirtualBox:~/Desktop/Assignment-4/vdbench$ zpool list
NAME         SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH
 ALTROOT
zfs_pool  3.75G   962M  2.81G        -         -     0%    25%  1.00x    ONLINE
 -
vps@vps-VirtualBox:~/Desktop/Assignment-4/vdbench$
```

Dedup is on



```
vps@vps-VirtualBox:~/Desktop/Assignment-4/vdbench$ zpool list
NAME         SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH
 ALTROOT
zfs_pool  3.75G   483M  3.28G        -         -     0%    12%  1.99x    ONLINE
 -
```

ii. ext4:

1. Initially the empty ext4 folder had 24k of data.

2. After running the workload, the ext4 folder had 980.9 MB of data.

3. The new files thus took 980.0 MB of space (a little more than

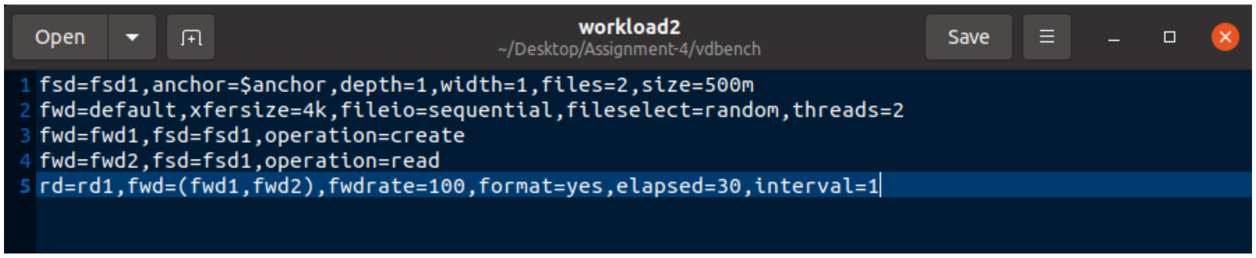
intended because of metadata overhead).

This can be seen using df -h command.

Large file Creation & Management - Experiment

Workload explaination

It is known that ext4 optimises large file creation & management. This can be observed using the following workload



```
fsd=fsd1,anchor=$anchor,depth=1,width=1,files=2,size=500m
fwd=default,xfersize=4k,fileio=sequential,fileselect=random,threads=2
fwd=fwd1,fsd=fsd1,operation=create
fwd=fwd2,fsd=fsd1,operation=read
rd=rd1,fwd=(fwd1,fwd2),fwdrate=100,format=yes,elapsed=30,interval=1
```

- What we are doing here is creating two files of size 1GB in one nested folder(one subdirectory). The operation used is “create” since we are testing file creation. And in the second working directory we are employing read operation.
- Dedupratio is set to 2, and dedupunit to 2MB. The ratio of the total number of blocks (of size dedupunit) to the number of blocks containing unique data is known as the dedupratio. The size of the block that will be compared to earlier blocks to look for duplicates is the dedupunit, on the other hand. Due to the size of a single file, we set it to 2MB. Consequently, half of the files will essentially be copies of the other half.

Running and Observation

Running was done as before changing workload1 to workload2

- We found the following results:

  i. ext4:

  1. Finishes creating files in just 37 seconds! Remember, these are 500MB files!
  2. The average write rate is 49.1 MB/s!

  ii. ZFS:
  1. Takes a whole 54 seconds to create the files.
  2. The average write rate is 50.3 MB/s.

  g. Note: All output folders attached in submission.

Advantages & Disadvantages

Deduplication - Advantages

If we use data deduplication features for ZFS and have a workload which writes similar files onthe disk, then ZFS can significantly reduce the disk space used for storing the data by writing duplicate blocks only once and creating pointers to further copies of duplicate blocks.

Deduplication - Disadvantages

1. **Performance:** In the first workload, the ZFS system set up the file system in 54 seconds whereas ext4 just took 37 seconds. ZFS had an average read speed of 50.3MB/s while ext4 had an average read speed of 49.1 MB/s. In the second workload too, as we have seen above in the 'large file optimisation' section, ext4 was significantly faster. This is partially due to large file optimisation of ext4 and partially due to the deduplication overhead of ZFS. This shows that deduplication harms performance of a file system due to overhead.
2. **CPU Utilisation:** The average CPU utilisation of ZFS was 99.5% during file structure setup (the other half is for read, which has little to do with deduplication), whereas that of ext4 was 94.8%, which is substantially lower.This demonstrates that the deduplication capability consumes much more CPU in both workloads. This is due to the overhead of deduplication.

Large file Creation & Management - Advantages

ext4 can create larger files than ZFS. In this section, we design a workload to generate two files of 500 MB each for both file systems. Because ext4 processes huge files more effectively, ext4 writes the files in 7 intervals of one second each, whereas ZFS takes 24 intervals. As a result, ext4 takes less time than ZFS to produce the same number of files, and hence ext4 has a higher throughput (observation from ext4 largefiles.html and zfs largefiles.html, both generated in the output folder after running vdbench).

Large file Creation & Management - Disadvantages

1. **ext4 stores only the addresses of the first and last blocks of each continuous range of blocks**, **rather than a list of every individual block that makes up the file**. Extents are the continuous ranges of data blocks (and the pairs of numbers that describe them). Extents consume the same amount of space regardless of the size of the range of blocks they represent, hence for smaller files, ext4 has a higher metadata cost than ZFS.

2. **No possible recovery from corruption:** Ext4 optimises large file creation by using delayed and contiguous allocation, and extents. This makes it impossible for any data correction mechanisms to exist since very little

metadata is stored for large files stored in many contiguous blocks. If checksums were to be maintained, we can also maintain individual block pointers since they take around the same amount of space.