CS344: Operating System Lab
Assignment-4

Group 24:

Name: **Dhruv Tarsadiya** Roll No: **200101123**

Name: **Vikram Singla** Roll No: **200101117**

Name: **Harshwardhan Bhakkad** Roll No: **200101040**

Comparing File Systems

ZFS was created as a part of the Solaris OS. ZFS acts as both a file system and a volume manager(used to allocate space on mass-storage devices), which helps with making the processes more compatible and smooth. It was designed with security as the highest priority, and ensures that every step related to file management or disk management is verified and optimized, which separate volume and file managers can't achieve.

The EXT4 file system primarily focuses on performance and capacity. In this system, data allocation is in the form of extents, instead of fixed size blocks. Extents are described by just their starting and ending places on the hard drive. This form of storing the necessary location of the data in files makes use of the reduces fragmentation of the memory allocated by the EXT4 file system, and thus helps to store the location of data of the file with the help of a small number of pointers, instead of using a pointer pointing to all the blocks of memory occupied by the file. It also makes use of delayed allocation, which helps improve the performance, as well as helps the file system allocate contiguous blocks of memory, as it already knows how much memory it has to map before it starts allocating any memory.

We compare the file systems ZFS and ETX4 by using vdbench. We create workloads to test these two features: **Data Compression** and **Management of Large Files** - on each file system.

Data Compression

Data compression is a reduction in the number of bits needed to represent data. Compressing data can save storage capacity, speed up file transfer, and decrease costs for storage hardware and network bandwidth. Compression is performed by a program that uses a formula or algorithm to determine how to shrink the size of the data. For instance, an algorithm may represent a string of bits -- or 0s and 1s -- with a smaller string of 0s and 1s by using a dictionary for the conversion between them, or the formula may insert a reference or pointer to a string of 0s and 1s that the program has already seen. Text compression can be as simple as removing all unneeded characters, inserting a single repeat character to indicate a string of repeated characters and substituting a smaller bit string for a frequently occurring bit string. Data compression can reduce a text file to 50% or a significantly higher percentage of its original size. Data compression can be performed on the data content or on the entire transmission unit, including header data. When information is sent or received via the internet, larger files, either singly or with others as part of an archive file, may be transmitted in a ZIP, GZIP or other compressed format.

Management of Large Files

To manage large files the file system must have a support to handle large files. Ext4 uses extents (as opposed to the traditional block mapping scheme used by ext2 and ext3), which improves performance when using large files and reduces metadata overhead for large files but has a larger metadata overhead compared to ZFS. Ext4 uses 48-bit internal addressing, making it theoretically possible to allocate files up to 16 TiB on filesystems up to 1,000,000 TiB (1 EiB).

Creating a ZFS partition

1. We created a ZFS pool on an Ubuntu Virtual Machine.
2. 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, NewVirtualDisk.vdi is our added hard disk.
3. Start the VM, and then open the terminal.
4. To install ZFS, run the following command:

    sudo apt install zfsutils-linux

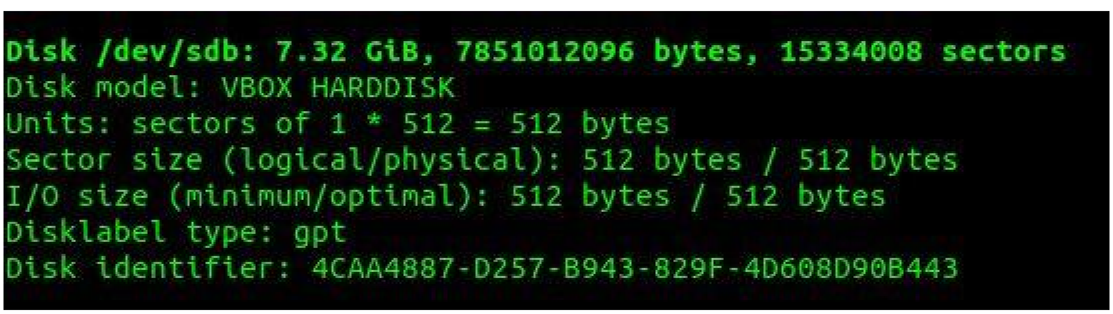
5. To check if ZFS has been successfully installed, run the following command:

    whereis zfs

6. To create the storage pool, first check the availables disks using the following command:

    sudo fdisk -l

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



```
Disk /dev/sdb: 7.32 GiB, 7851012096 bytes, 15334008 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: gpt
Disk identifier: 4CAA4887-D257-B943-829F-4D608D90B443
```

7. To create the striped pool, run the following command:

    sudo zpool create new-pool /dev/sdb/

Here, new-pool is the name of our ZFS pool

8. We have now successfully created a ZFS pool

Creating an ext4 Partition

1. We created an ext4 pool on an Ubuntu Virtual Machine.
2. First, we need to add an extra hard disk to our VM. Refer step 2 of creating a zfs

partition.

3. Start the VM, and then open the terminal.
4. As ext4 is the default file system of Ubuntu, we don't need to install it explicitly. To create the storage pool, first check the availables disks using the following command:

    sudo fdisk -l

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

```
Disk /dev/sdc: 10 GiB, 10737418240 bytes, 20971520 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
```

5. We now use the parted command to configure our selected disk partition:
parted /dev/sdc
6. Now we will label the new partition using mklabel in parted and then create a partition using the mkpart command and specify the parameters.
7. To format the file system properly with ext4 file system we execute the following command:

    mkfs.ext4 /dev/sdc

8. We label the partition using e2label command:

    e2label /dev/sdc old-pool

9. To create a mount point for our new partition, we create a new directory using the mkdir command:

    mkdir old-pool

10. We will mount our partition to the mount point using the mount command:

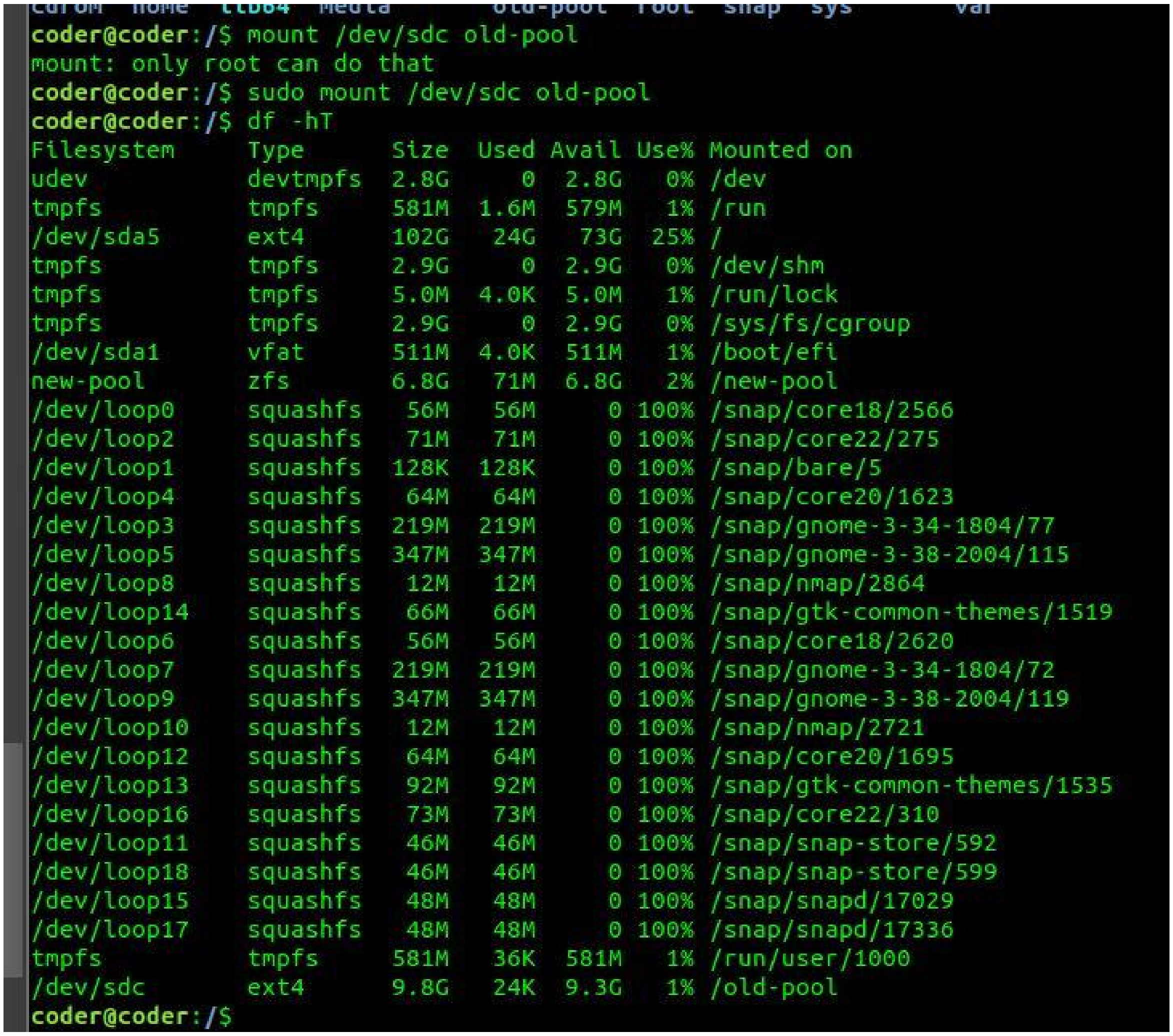
    mount /dev/sdc old-pool

11. We have successfully created an ext4 pool.

Checking the file system in the directories

We have mounted :

1. ZFS filesystem on the new-pool directory
2. ext4 filesystem on the old-pool directory



```
coder@coder:/$ mount /dev/sdc old-pool
mount: only root can do that
coder@coder:/$ sudo mount /dev/sdc old-pool
coder@coder:/$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
udev           devtmpfs  2.8G     0  2.8G   0% /dev
tmpfs          tmpfs     581M  1.6M  579M   1% /run
/dev/sda5      ext4      102G   24G   73G  25% /
tmpfs          tmpfs     2.9G     0  2.9G   0% /dev/shm
tmpfs          tmpfs     5.0M  4.0K  5.0M   1% /run/lock
tmpfs          tmpfs     2.9G     0  2.9G   0% /sys/fs/cgroup
/dev/sda1      vfat      511M  4.0K  511M   1% /boot/efi
new-pool       zfs       6.8G   71M  6.8G   2% /new-pool
/dev/loop0     squashfs   56M   56M     0 100% /snap/core18/2566
/dev/loop2     squashfs   71M   71M     0 100% /snap/core22/275
/dev/loop1     squashfs  128K  128K     0 100% /snap/bare/5
/dev/loop4     squashfs   64M   64M     0 100% /snap/core20/1623
/dev/loop3     squashfs  219M  219M     0 100% /snap/gnome-3-34-1804/77
/dev/loop5     squashfs  347M  347M     0 100% /snap/gnome-3-38-2004/115
/dev/loop8     squashfs   12M   12M     0 100% /snap/nmap/2864
/dev/loop14    squashfs   66M   66M     0 100% /snap/gtk-common-themes/1519
/dev/loop6     squashfs   56M   56M     0 100% /snap/core18/2620
/dev/loop7     squashfs  219M  219M     0 100% /snap/gnome-3-34-1804/72
/dev/loop9     squashfs  347M  347M     0 100% /snap/gnome-3-38-2004/119
/dev/loop10    squashfs   12M   12M     0 100% /snap/nmap/2721
/dev/loop12    squashfs   64M   64M     0 100% /snap/core20/1695
/dev/loop13    squashfs   92M   92M     0 100% /snap/gtk-common-themes/1535
/dev/loop16    squashfs   73M   73M     0 100% /snap/core22/310
/dev/loop11    squashfs   46M   46M     0 100% /snap/snap-store/592
/dev/loop18    squashfs   46M   46M     0 100% /snap/snap-store/599
/dev/loop15    squashfs   48M   48M     0 100% /snap/snapd/17029
/dev/loop17    squashfs   48M   48M     0 100% /snap/snapd/17336
tmpfs          tmpfs     581M   36K  581M   1% /run/user/1000
/dev/sdc       ext4      9.8G   24K  9.3G   1% /old-pool
coder@coder:/$
```

Setting up vdbench

Vdbench has three basic definitions in the configuration file.

1. FSD or filesystem definition: This defines what filesystem to use for the testing.
2. FWD or filesystem workload definition: This defines the workload parameter for the testing.
3. RD or run definition: This defines storage, workload and run duration.

Filesystem Definition

FSD defines the filesystem storage used for the testing.

The FSD name should be unique.

FSD parameters include:

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

Filesystem Workload Definition

FWD defines the filesystem workload used for the testing.

The FWD name should be unique.

FWD parameters include:

1. fwd=name : Unique name for this Filesystem Workload Definition
2. fsd=(xx,....) : Name(s) of Filesystem Definitions to use.
3. operation=xxxx : Specifies a single file system operation that must be done for this workload.
4. fileio=sequential : How file I/O will be done: random or sequential
5. fileselect=random/seq : How to select file names or directory names for processing.
6. threads=nn : How many concurrent threads to run for this workload.
7. xfersize=(nn,...) : Specifies the data transfer size(s) to use for read and write operations.

Run Definition

RD defines what storage and workload will be run together and for how long.

Each run definition name must be unique.

RD parameters include:

1. fwd=(xx,yy,..) : Name(s) of Filesystem Workload Definitions to use.
2. format : 'format=yes' causes the directory structure to be completely created, including

initialization of all files to the requested size.
3. elapsed : How long to run in seconds.
4. interval : Statistics collection interval in seconds.
5. fwdrate=nn : Run a workload of nn operations per second

Data Compression

vdbench code

Line 1

```
compratio=10
```

Line 2

```
fsd=fsd1,anchor=/dir,depth=2,width=2,files=2,size=100m
```

Line 3

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

Line 4

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

Line 1 defines the compression ratio i.e the amount by which the files are going to be compressed in this case the files would be compressed by 10 times.
Line 2 is the FSD i.e the filesystem definition.
Line 3 is the FWD i.e the filesystem workload definition.
Line 4 is the RD i.e the run definition.
Here **dir** is ‘new-pool’ for zfs and ‘old-pool’ for ext4

Directory Structure:

```
coder@coder:/old-pool$ tree
.
├── no_dismount.txt
├── vdb.1_1.dir
│   ├── vdb.2_1.dir
│   │   ├── vdb_f0000.file
│   │   └── vdb_f0001.file
│   └── vdb.2_2.dir
│       ├── vdb_f0004.file
│       └── vdb_f0005.file
├── vdb.1_2.dir
│   ├── vdb.2_1.dir
│   │   ├── vdb_f0002.file
│   │   └── vdb_f0003.file
│   └── vdb.2_2.dir
│       ├── vdb_f0006.file
│       └── vdb_f0007.file
└── vdb_control.file

6 directories, 10 files
```

Management of Large Files

vdbench code

Line1

fsd=fsd1,anchor=/dir,depth=1,width=1,files=3,size=500m

Line2

fwd=default,xfersize=4k,fileio=sequential,fileselect=seq,threads=2

Line3

fwd=fwd1,fsd=fsd1,operation=read

Line4

fwd=fwd2,fsd=fsd1,operation=write

Line5

rd=rd1,fwd=(fwd1,fwd2),fwdrate=100,format=yes,elapsed=10,interval=1

Line 1 is the FSD i.e the filesystem definition.

Lines 2,3,4 are the FWD i.e the filesystem workload definition.

Line 5 is the RD i.e the run definition.

Here dir is new-pool for zfs and old-pool for ext4

Directory Structure:

```
coder@coder:/old-pool$ tree
.
├── no_dismount.txt
├── vdb.1_1.dir
│   ├── vdb_f0000.file
│   ├── vdb_f0001.file
│   └── vdb_f0002.file
└── vdb_control.file

1 directory, 5 files
coder@coder:/old-pool$ tree
```

Observations for Data Compression

```
coder@coder:/old-pool$ du -sh
801M    .
```

```
coder@coder:/new-pool$ du -sh
71M     .
coder@coder:/new-pool$ sudo zfs get compressratio new-pool
NAME      PROPERTY       VALUE   SOURCE
new-pool  compressratio  11.32x  -
coder@coder:/new-pool$
```

Advantage:

ZFS has a data compression feature, which can be turned on using the following command: sudo zfs compression=on new-pool

Using an identical workload to create 8 files, each of size 100 MB, we see that new-pool occupies only 71 MB whereas old-pool occupies the full 801 MB space. 8 files, each of size 100 MB, will normally occupy 800 MB plus some overhead, which is what happens in ext4.

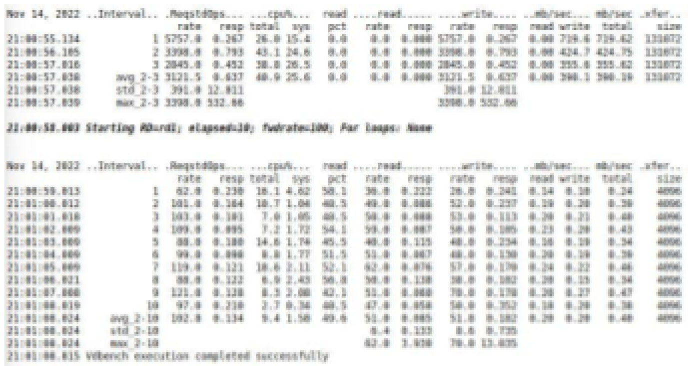
However, with compression feature turned on and compression ratio set to 10, ZFS occupies a total size which is roughly one-tenth of the expected size.

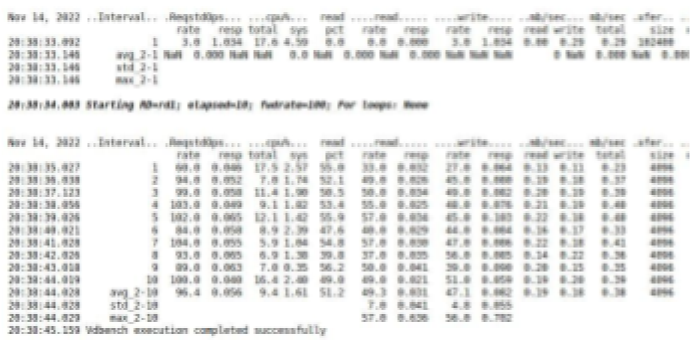
Disadvantage:

It takes more time to read and write with compression in ZFS. While writing, the data has to be compressed by the LZ4 algorithm. Some additional time gets utilized for running the algorithm. While reading the data gets uncompressed which takes some time. Hence, Average response time for ZFS is 105.2 ms while the same for EXT4 is 99.6 ms. Average response time being higher for ZFS. (Results can be seen in .html files produced)
We also note that the Average Write rate is much higher for ext4 compared to Average Write Rate for ZFS. This shows that Data Compression harms performance of file systems.

Observations for Management of Large Files

The top image is for ZFS.





Bottom image for EXT4.

Ext4 is better equipped to create large files than ZFS. We designed a workload to create 3 files, each of 500MB, for both the file systems. Since ext4 handles large files efficiently; ext4 takes only 1 interval of one second to write the files (note that total size was 1.5GB!) while ZFS takes 3 intervals of one second each. So for writing the same amount of data, ext4 takes less time than ZFS and thus ext4 has better throughput.
We made these observations by looking at html files in the output.

Disadvantage:
ext4 instead of storing a list of every individual block which makes up the file, the idea is to store just the address of the first and last block of each continuous range of blocks. These continuous ranges of data blocks (and the pairs of numbers which represent them) are called extents. Extents take a constant amount of space regardless of how big a range of blocks it describes and hence for smaller files ext4 has a larger overhead of metadata compared to metadata in ZFS.

```
coder@coder:/old-pool$ du -sh
316K    .
```

```
coder@coder:/new-pool$ du -sh
306K    .
```

In the above screenshot we have created 3 files of size 100 KB in both the file systems.
Hence the metadata overhead in case of ext4 is : 316-3*100=16KB
Hence the metadata overhead in case of ZFS is : 306-3*100=6KB
Hence the metadata overhead in case of ext4 for small files is larger than that of ZFS.

Another disadvantage is : No possible recovery from corruption.Ext4 optimizes 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, why not also maintain individual block pointers since they take around the same amount of space?