CS-344

Assignment 4

Group Number : 2

Tejas Khairnar	180101081
Pooja Bhagat	180101057
Parth Bakare	180101056
Param Aryan Singh	180101055

25th November, 2020

Comparing Features of ZFS and ext4

- We compare the files systems ZFS and ext4 using vdbench.
- We created workloads to test 2 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.

How compression works?

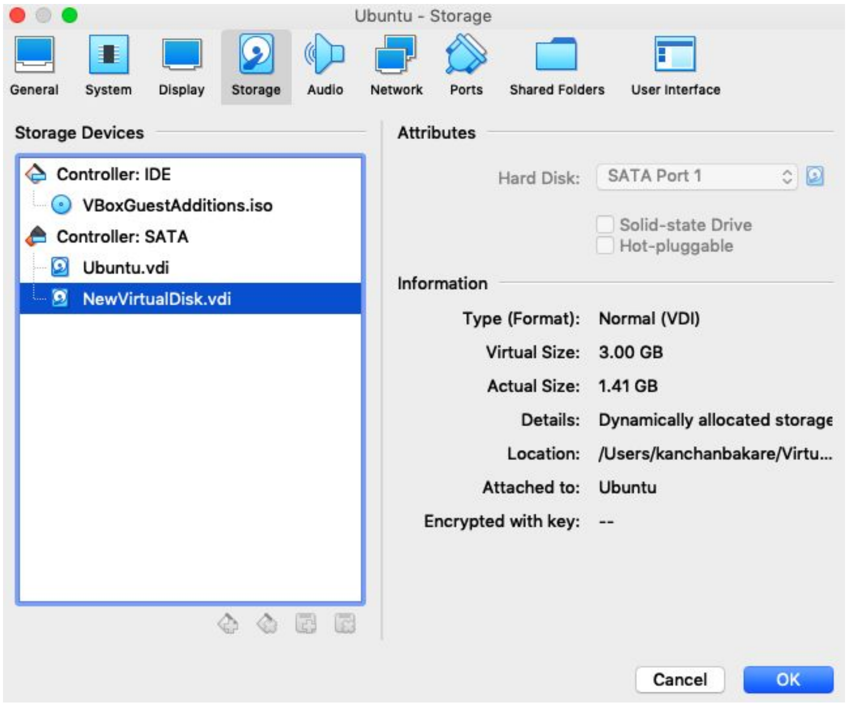
- Compression is performed by a program that uses a formula or algorithm to determine how to **shrink the size of the data**.
- For instance, an algorithm may represent a string of bits -- or 0s and 1s -- with a smaller string of 0s and 1s by using a dictionary for the conversion between them, or the 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.



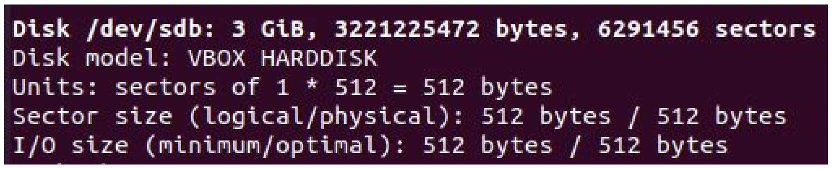
Storage Devices
- Controller: IDE
  - VBoxGuestAdditions.iso
- Controller: SATA
  - Ubuntu.vdi
  - NewVirtualDisk.vdi

Attributes
Hard Disk: SATA Port 1
Solid-state Drive
Hot-pluggable

Information
Type (Format): Normal (VDI)
Virtual Size: 3.00 GB
Actual Size: 1.41 GB
Details: Dynamically allocated storage
Location: /Users/kanchanbakare/Virtu...
Attached to: Ubuntu
Encrypted with key: --

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`

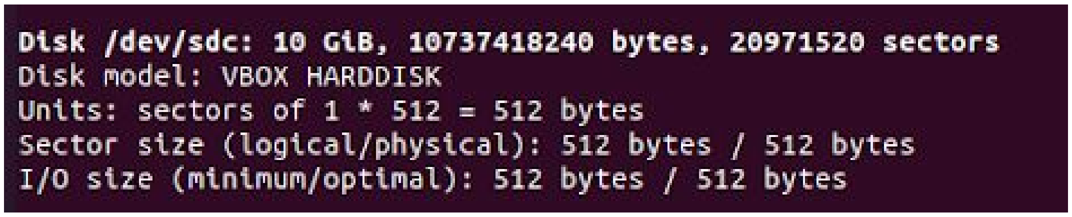
We will use **/dev/sdb** to create our ZFS pool.



```
Disk /dev/sdb: 3 GiB, 3221225472 bytes, 6291456 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
```

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

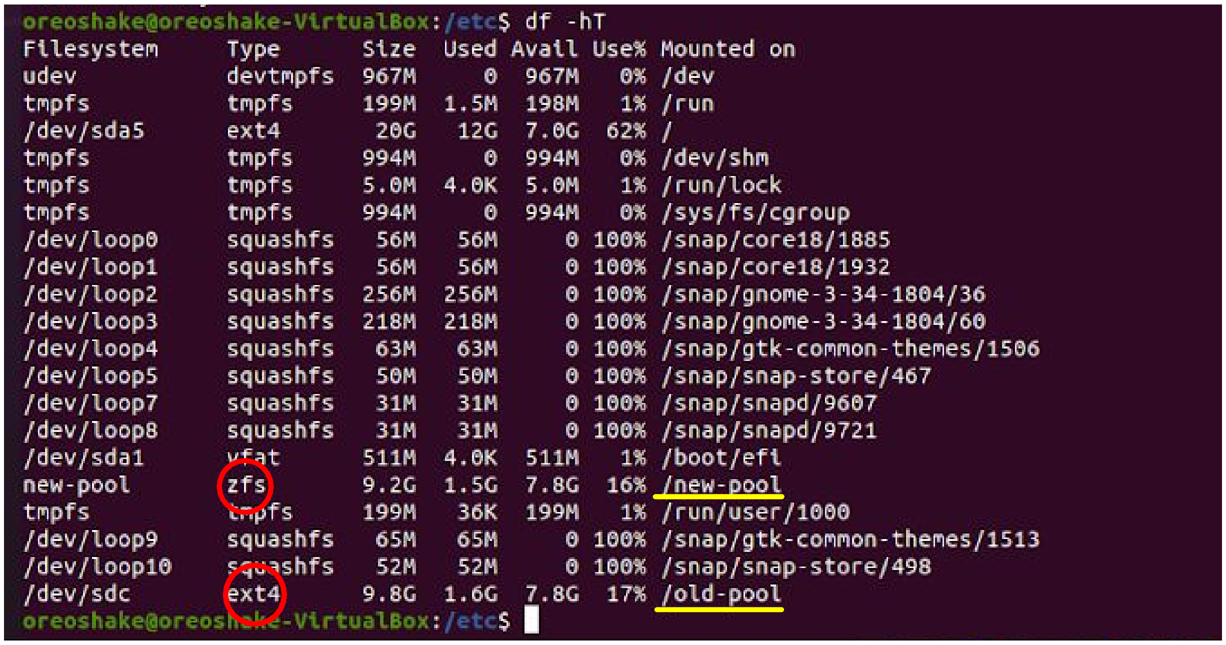
Creating an ext4 Partition

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



```
oreoshake@oreoshake-VirtualBox:/etc$ df -hT
Filesystem     Type      Size  Used Avail Use% Mounted on
udev           devtmpfs  967M     0  967M   0% /dev
tmpfs          tmpfs     199M  1.5M  198M   1% /run
/dev/sda5      ext4       20G   12G  7.0G  62% /
tmpfs          tmpfs     994M     0  994M   0% /dev/shm
tmpfs          tmpfs     5.0M  4.0K  5.0M   1% /run/lock
tmpfs          tmpfs     994M     0  994M   0% /sys/fs/cgroup
/dev/loop0     squashfs   56M   56M     0 100% /snap/core18/1885
/dev/loop1     squashfs   56M   56M     0 100% /snap/core18/1932
/dev/loop2     squashfs  256M  256M     0 100% /snap/gnome-3-34-1804/36
/dev/loop3     squashfs  218M  218M     0 100% /snap/gnome-3-34-1804/60
/dev/loop4     squashfs   63M   63M     0 100% /snap/gtk-common-themes/1506
/dev/loop5     squashfs   50M   50M     0 100% /snap/snap-store/467
/dev/loop7     squashfs   31M   31M     0 100% /snap/snapd/9607
/dev/loop8     squashfs   31M   31M     0 100% /snap/snapd/9721
/dev/sda1      vfat      511M  4.0K  511M   1% /boot/efi
new-pool       zfs       9.2G  1.5G  7.8G  16% /new-pool
tmpfs          tmpfs     199M   36K  199M   1% /run/user/1000
/dev/loop9     squashfs   65M   65M     0 100% /snap/gtk-common-themes/1513
/dev/loop10    squashfs   52M   52M     0 100% /snap/snap-store/498
/dev/sdc       ext4      9.8G  1.6G  7.8G  17% /old-pool
oreoshake@oreoshake-VirtualBox:/etc$
```

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

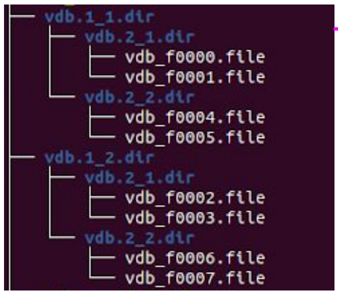
Line 2 fsd=fsd1,anchor=/dir,depth=2,width=2,files=2,size=100m

Line 3 fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random, threads=2

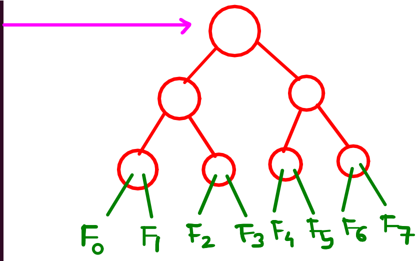
Line 4 rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1

- **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** are 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



```
├── 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
```



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

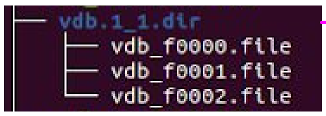
Line3 fwd=fwd1,fsd=fsd1,operation=read

Line4 fwd=fwd2,fsd=fsd1,operation=write

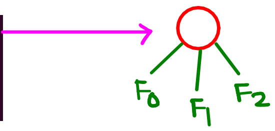
Line5 rd=rd1,fwd=(fwd1,fwd2),fwdrate=100,format=yes,elapsed=10,interval=1

- **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**

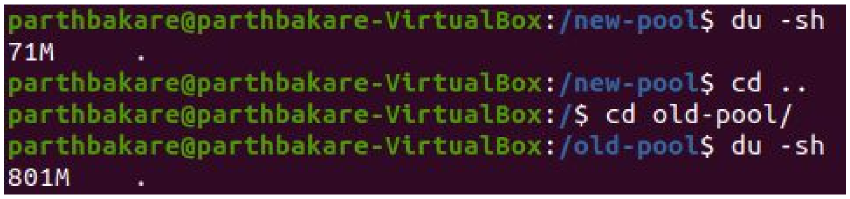
Directory structure



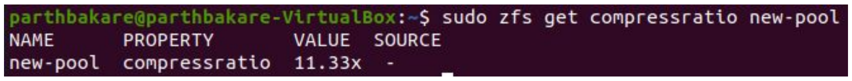
```
├── vdb.1_1.dir
│   ├── vdb_f0000.file
│   ├── vdb_f0001.file
│   └── vdb_f0002.file
```



Observations for Data Compression



```
parthbakare@parthbakare-VirtualBox:/new-pool$ du -sh
71M     .
parthbakare@parthbakare-VirtualBox:/new-pool$ cd ..
parthbakare@parthbakare-VirtualBox:/$ cd old-pool/
parthbakare@parthbakare-VirtualBox:/old-pool$ du -sh
801M    .
```



```
parthbakare@parthbakare-VirtualBox:~$ sudo zfs get compressratio new-pool
NAME      PROPERTY       VALUE   SOURCE
new-pool  compressratio  11.33x  -
```

Advantage:

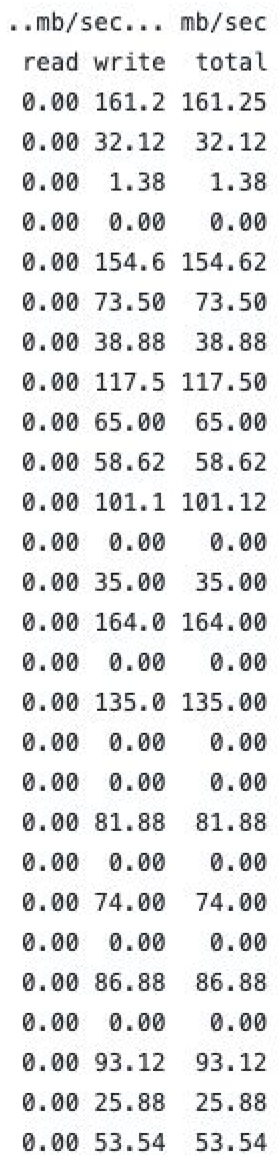
ZFS has a data compression feature, which can be turned on using the following command:
`sudo zfs compression=on new-pool`
Using an identical workload to create 8 files, each of size **100 MB**, we see that **new-pool** occupies only **71 MB** whereas **old-pool** occupies the full **801 MB** space. 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:

ZFS has a very high average CPU utilisation (greater than **80%**) whereas **ext4** has a moderate average CPU utilisation (around **55%**). CPU usage is extensive because of file compression (results produced in **zfs_compression.html** and **ext4_compression.html** files).

Observations for Management of Large Files

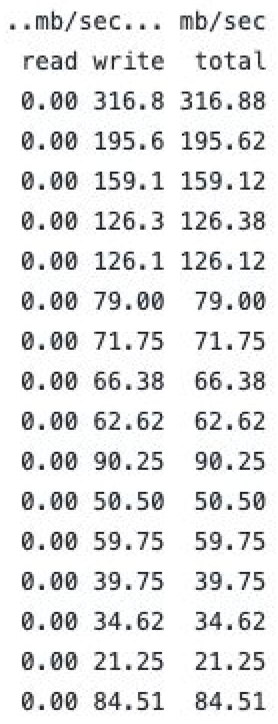
26 intervals



```
..mb/sec... mb/sec
 read write  total
 0.00 161.2 161.25
 0.00 32.12  32.12
 0.00  1.38   1.38
 0.00  0.00   0.00
 0.00 154.6 154.62
 0.00 73.50  73.50
 0.00 38.88  38.88
 0.00 117.5 117.50
 0.00 65.00  65.00
 0.00 58.62  58.62
 0.00 101.1 101.12
 0.00  0.00   0.00
 0.00 35.00  35.00
 0.00 164.0 164.00
 0.00  0.00   0.00
 0.00 135.0 135.00
 0.00  0.00   0.00
 0.00  0.00   0.00
 0.00 81.88  81.88
 0.00  0.00   0.00
 0.00 74.00  74.00
 0.00  0.00   0.00
 0.00 86.88  86.88
 0.00  0.00   0.00
 0.00 93.12  93.12
 0.00 25.88  25.88
 0.00 53.54  53.54
```

zfs

15 intervals



```
..mb/sec... mb/sec
 read write  total
 0.00 316.8 316.88
 0.00 195.6 195.62
 0.00 159.1 159.12
 0.00 126.3 126.38
 0.00 126.1 126.12
 0.00 79.00  79.00
 0.00 71.75  71.75
 0.00 66.38  66.38
 0.00 62.62  62.62
 0.00 90.25  90.25
 0.00 50.50  50.50
 0.00 59.75  59.75
 0.00 39.75  39.75
 0.00 34.62  34.62
 0.00 21.25  21.25
 0.00 84.51  84.51
```

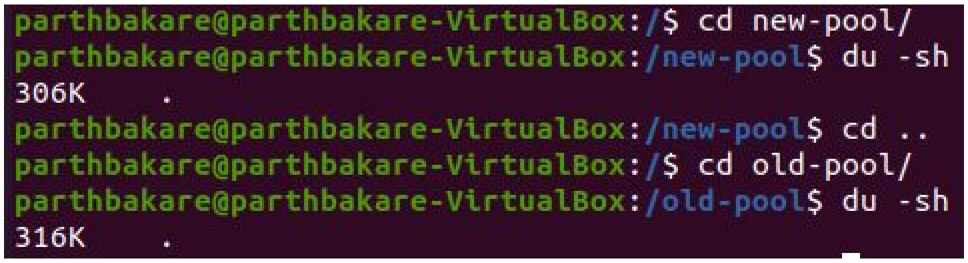
ext4

Advantage:

ext4 is better equipped to create large files than **ZFS**. Here we design a workload to create 3 files, each of size **500 MB**, for both the file systems. Since **ext4** handles large files more efficiently, **ext4** takes only **15 intervals** of one second each to write the files and **ZFS** takes as long as **26 intervals**. So, for writing the same amount of data, **ext4** takes less time than **ZFS** and thus **ext4** has a better throughput (observation from **ext4_largefiles.html** and **zfs_largefiles.html**, created in the **output** folder after running vdbench).

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**.



```
parthbakare@parthbakare-VirtualBox:/$ cd new-pool/
parthbakare@parthbakare-VirtualBox:/new-pool$ du -sh
306K    .
parthbakare@parthbakare-VirtualBox:/new-pool$ cd ..
parthbakare@parthbakare-VirtualBox:/$ cd old-pool/
parthbakare@parthbakare-VirtualBox:/old-pool$ du -sh
316K    .
```

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.