CS - 344 (OS LAB)

Assignment - 4

Group 1

Members :

1. Parth Agarwal, CSE, 220101074
2. Shashwat Shankar, CSE, 220101092
3. Shobhit Gupta, CSE, 220101093
4. Shubhranshu Pandey, CSE, 220101094

Part - 1:-

We have selected the **ZFS** and **ext4** file systems and the two features we would be comparing are **data deduplication** and **large file creation.** Following is some detail about the file systems selected for evaluation.

ZFS File system

- **Developed by Sun Microsystems**: Built to handle large-scale data with reliability.
- **Data Integrity**: Utilizes checksumming to prevent data corruption and ensure integrity.
- **Scalability**: Suitable for large storage environments, handling data across multiple disks.
- **Built-in Volume Management**: Functions as both a filesystem and a logical volume manager.
- **Use Cases:** Ideal for environments that require data integrity, like databases, backups, or scientific data processing. Widely used in data centers and cloud services due to its robust storage management and reliability.
- **Advanced Features**: Includes deduplication, copy-on-write, compression, and snapshot capabilities.

ext4 Filesystem

- **Widely Used on Linux**: Standard on many Linux distributions.
- **Optimized for Speed and Efficiency**: Lightweight, with a focus on performance for everyday tasks.
- **Supports Large Files and Partitions**: Extends capabilities beyond its predecessors (ext2/ext3).
- **Use Cases:** Suitable for standard desktop, server, and embedded systems where performance and reliability are key. Often used in environments with high demands for large file handling, such as media storage and software development, due to its efficient large file creation.
- **Features like Journaling and Delayed Allocation**: Enhances data consistency and allocation efficiency.

- **No Deduplication**: Optimized for standard use cases without significant CPU/memory overhead.

Here is a description of both the features we have picked to compare for this assignment:

Feature 1: Deduplication in ZFS

- **Definition**: Deduplication is a storage optimization technique that eliminates duplicate copies of identical data. It ensures only unique data blocks are stored on disk, thus reducing storage requirements.
- **How It Works**: In ZFS, each data block is hashed (typically with a checksum). When a new block is written, ZFS checks if the hash matches any existing block. If a match is found, the new data block isn't stored again; instead, it points to the existing block.
- **Benefits**: Deduplication is particularly useful in environments with redundant data (e.g., virtual machines with similar configurations, repetitive datasets) because it significantly reduces the amount of disk space required.
- **Limitations**: This process requires additional CPU and memory resources, as calculating and managing hashes for each block can be demanding.
- **Not Available in ext4**: ext4 does not have a built-in deduplication feature, meaning it cannot eliminate redundant data. Consequently, identical files or blocks will consume additional space, which can lead to inefficient storage usage in scenarios with repetitive data.

Feature 2: Large File Creation in ext4

- **Optimized Allocation Techniques**: Uses delayed and multiblock allocation to reduce fragmentation.
- **Maximum File Size**: ext4 can support a maximum file size of 16 TiB (Tebibytes) on systems with larger block sizes (typically 4 KiB).
- **Enhanced Write Performance**: Deferred allocation allows for more efficient data placement.
- **Reduces Fragmentation**: Minimizes fragmentation when writing large files, improving read/write speed.

- **Ideal for Large Files:** Suitable for multimedia, databases, and other large-file applications.
- **Balanced Resource Usage:** Provides performance benefits without significantly increasing CPU or memory load.
- **Not available in ZFS:** Although ZFS can handle very large files (theoretically up to **16 Exabytes**), it lacks ext4's specific optimizations for large file creation. This may lead to slower performance and higher fragmentation when writing very large files without additional configuration.

ZFS Installation:

- First, install the **ZFS filesystem** with the following command:

  sudo apt install zfsutils-linux -y

- Select a disk from the installed disks (Note: This disk must be at least 5 GB in size and should not be one currently in use by the system, such as sda). You can list available disks with:

  sudo fdisk -l

- Once you've selected the disk (here we used an external device, i.e a SanDisk pendrive, which is /dev/sdb1) create a **ZFS pool** named *zfs_pool* by running:

  sudo zpool create zfs_pool /dev/sdb1

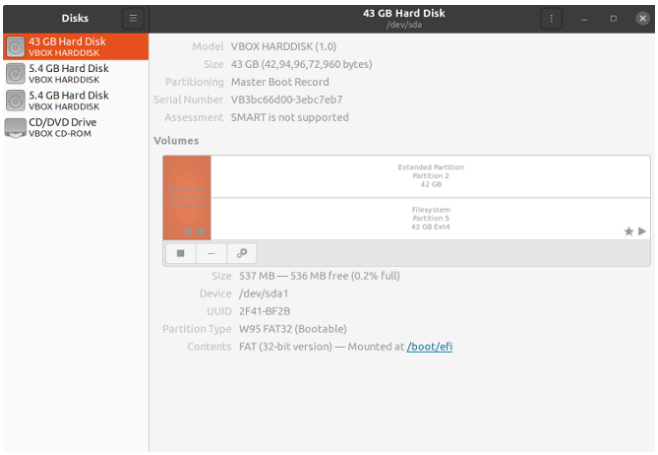
- Enable deduplication for the newly created ZFS pool:

  sudo zfs set dedup=on zfs_pool

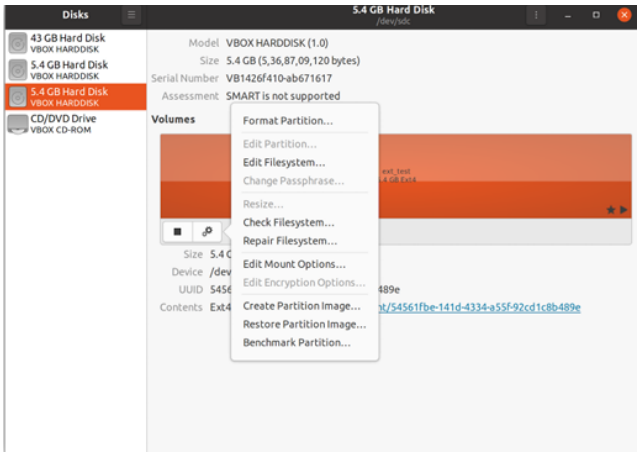
- You should now see a directory named /zfs_pool in the root directory. This directory will serve as the base location for running your workloads.

Ext4 Installation:

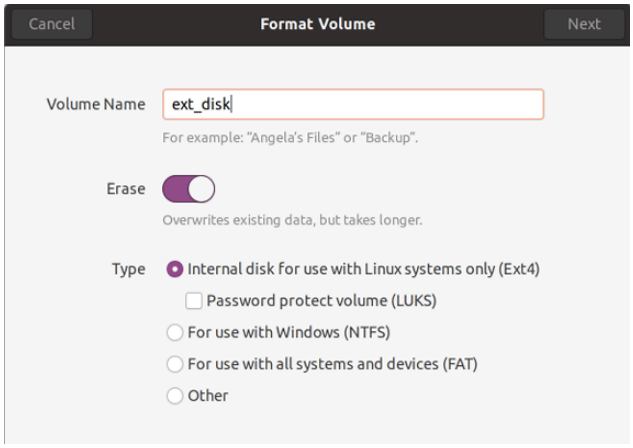
- **ext4 Setup**: ext4 is pre installed as the default filesystem on Ubuntu. Follow these steps to format a disk and set up ext4 on it.
- **Open the “Disks” Application**.



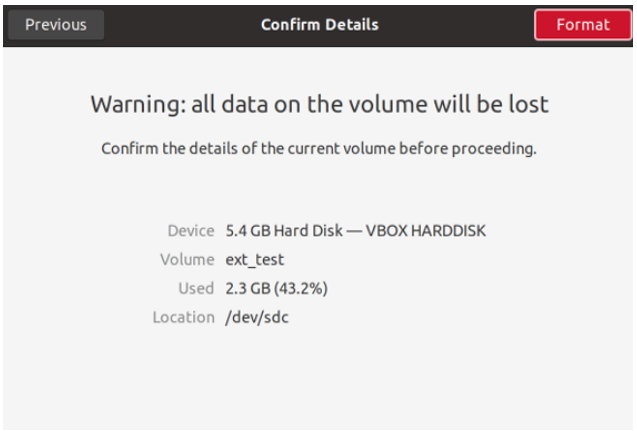
- Select the disk you wish to use (must have at least 5 GB of space, in our case we have used an external 8GB pendrive) and click the “Gear” icon. Then, choose “Format Partition”.



- Choose a name for the new disk, select the “Ext4” option, and enable the “**Erase**” switch. Click “Next” to proceed.



- Click "**Format**" to finalize the setup.



- After formatting, ensure the disk is mounted. If it's not, go to "Mount Options" (Gear Icon -> Edit Mount Options), uncheck "User Session Defaults," and enable "Mount on system startup." Then, reboot the system.

Finding the anchors:

- Identifying the anchor is crucial, as our workloads cannot function without it.
- To run workloads on ZFS or ext4 partitions, you need to identify the corresponding anchors for those partitions

- For a ZFS pool named zfs_pool, you can find the anchor as the highlighted section (/zfs_pool) represents the anchor.

```
shubhranshu@shubhranshu-HP-ProDesk-600-G6-Microtower-PC:~$ mount | column -t | grep zfs_pool
zfs_pool                                  on  /zfs_pool                                  type  zfs             (rw,xattr,noacl)
```

- In my case, the ext4 drive was mounted as "/dev/sdc". Here, "/mnt/usb-SanDisk_Cruzer_Blade_4C530001120618118530-0:0-part1 " serves as the anchor.

```
shubhranshu@shubhranshu-HP-ProDesk-600-G6-Microtower-PC:~$ mount | column -t | grep sdb1
/dev/sdb1                                 on  /mnt/usb-SanDisk_Cruzer_Blade_4C530001120618118530-0:0-part1  type  ext4          (rw,nosuid,nodev,relatime,x-gvfs-show)
```

Viewing the stats:

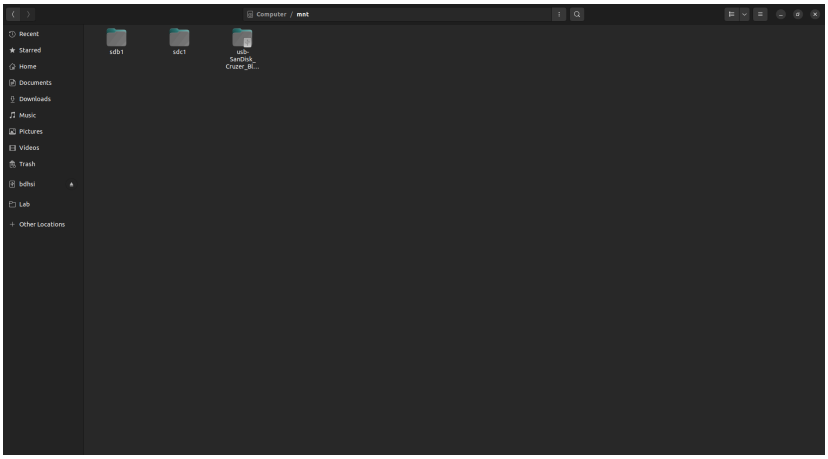
- **ZFS:**
  - Run the following command: *zpool list*
  - The sample output will look like this:

```
shubhranshu@shubhranshu-HP-ProDesk-600-G6-Microtower-PC:~$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool     7G   118K  7.00G        -         -     0%     0%  1.00x    ONLINE  -
```

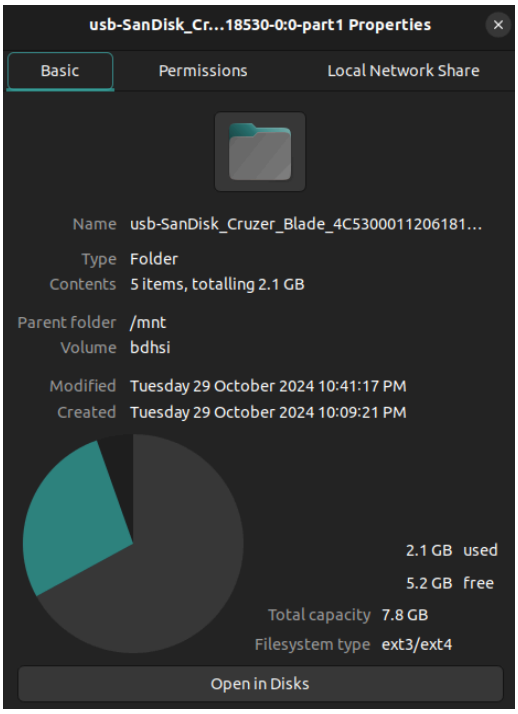
  - We need to run this before and after running the workload to calculate the space taken by the files after the workload.

- **Ext4**

  - Navigate to the folder where the ext4 anchor is present in the GUI file manager for Ubuntu.



- Then right click and view the properties of the anchor folder. Here you can see the space taken:



Part - 2:-

a) Data Deduplication

- Workload creation

1. We developed a workload (workload1) specifically for testing data deduplication. Using this workload, we will compare the space utilized by new files in ZFS against the space used in ext4.

workload1.txt

```
dedupunit=1m,dedupratio=2
fsd=fsd1,anchor=$anchor,depth=2,width=3,files=50,size=1m
fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2
rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=30,interval=1

```

2. We are generating **450 files** (50 sets of 3x3) with each file sized at 1MB, organized in a nested folder structure with a depth of 2 and width of 3.

3. After creation, the files are read sequentially for thirty seconds to collect statistics (though this part is secondary since deduplication occurs during file creation).
4. The **deduplication unit** (dedupunit) is set to 1MB, and the **deduplication ratio** (dedupratio) is set to 2.
5. **Deduplication Ratio** (dedupratio) defines the ratio of total blocks (each of size dedupunit) to the number of blocks containing unique data.
6. **Deduplication Unit** (dedupunit) represents the block size used for comparison against pre-existing blocks to identify duplicates. Setting it to 1MB aligns with our file size, allowing efficient deduplication.

- **ZFS:**
  1. Firstly we will turn on the data deduplication feature using the following command. (**zfs pool** is the name of the zfs pool that we are setting up)
  2. Now run the workload on the zfs file system using the following command (here we set up the anchor to the zfs pool directory)

```
k-600-G6-Microtower-PC:~/vdbench$ sudo ./vdbench -f workload1 anchor=/zfs_pool
```

  3. We found the following results
     - Initially, the empty ZFS directory occupied **118 KB** of space.
     - After executing the workload, the ZFS directory occupied **129 MB.**
     - We achieved a deduplication ratio of **2.06x**, as expected.
     - This indicates that the new files consumed **128 MB** of space, whereas the intended space usage was **450 MB** (**1MB x 450 files**).
     - By leveraging data deduplication, ZFS avoided storing duplicate data blocks and instead used pointers to reference the existing data.

Before Workload:

```
shubhranshu@shubhranshu-HP-ProDesk-600-G6-Microtower-PC:~$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool     7G   118K  7.00G        -         -     0%     0%  1.00x    ONLINE  -
```

After Workload:

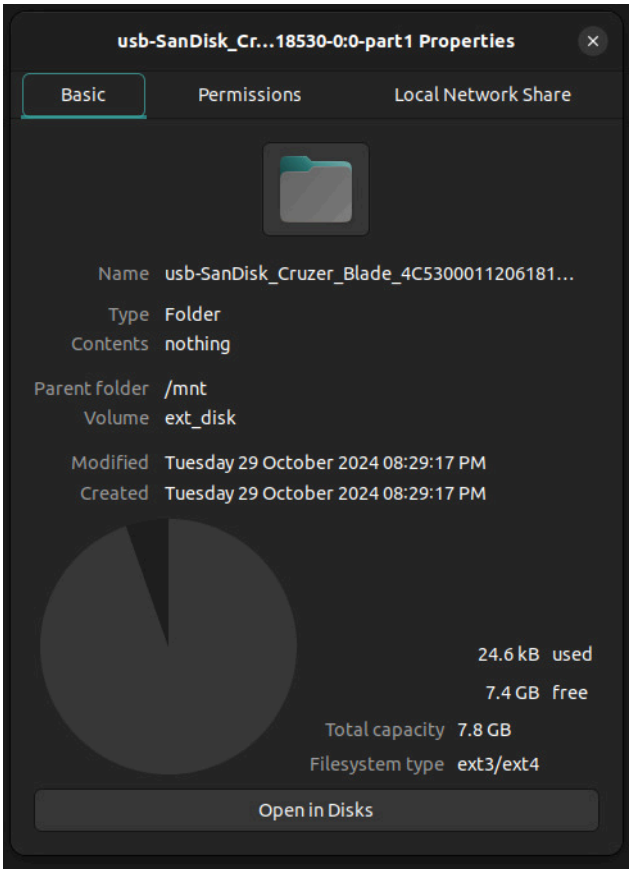
```
shubhranshu@shubhranshu-HP-ProDesk-600-G6-Microtower-PC:~/vdbench$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool     7G   129M  6.87G        -         -     0%     1%  2.06x    ONLINE  -
```

- **ext4**
    1. We will run the workload on the ext4 file system by setting anchor to the directory of the folder pointing to the ext4 drive:

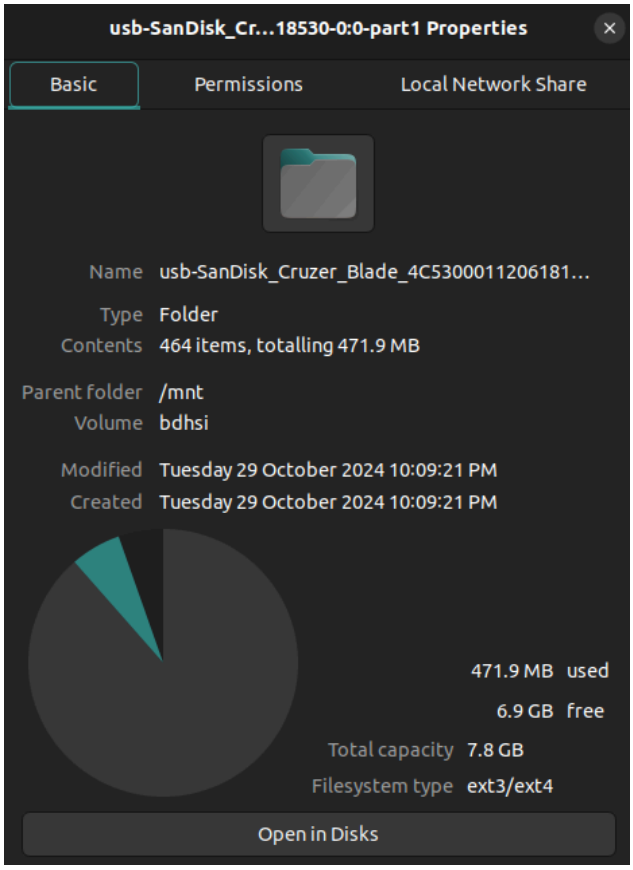
```
icrotower-PC:~$ sudo ./vdbench -f workload1 anchor=/mnt/usb-SanDisk_Cruze_Blade_4C530001120618118530-0:0-part1
```

    2. We found the following results:
        - Initially the empty ext4 folder had **24.6KB** of data.
        - After running the workload, the ext4 folder had **471.9 MB**.
        - The new files thus took **471 MB** of data. of space (a little more than intended because of metadata overhead).

Before Workload:



After Workload:



b) Large File Creation:

- **Workload for data deduplication:**
    1. Firstly, we developed a workload (workload2) specifically for testing large file creation. Using this workload, we will compare the space utilized by large files in ext4 against the space used in ZFS.

workload2.txt

```
fsd=fsd1,anchor=$anchor,depth=0,width=1,files=2,size=1G
fwd=fwd1,fsd=fsd1,operation=create,fileio=sequential,fileselect=random,threads=2
rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=30,interval=1
```

    2. Here we are creating two large files of size 1GB in one folder. We used the “create” operation since we are testing file creation.

- **Ext4:**
    1. We will run the workload on the ext4 file system by setting anchor to the directory of the folder pointing to the ext4 drive

```
G6-Microtower-PC:~$ sudo ./vdbench -f workload2 anchor=/mnt/usb-SanDisk_Cruze_Blade_4C530001120618118530-0:0-part1
```

    2. We found the following results:
        - Finishes creating the two 1GB files in just 7.13 seconds.
        - Average write rate = 2048/7.13 = 287 MB/s.

- **ZFS:**
    1. We run this on the ZFS file system by setting anchor equal to the directory pointing to the ZFS pool:
    2. We found the following results:
        - Finishes creating the two 1GB files in 10 minutes.

- Average write rate = 2048/600 = 3.4 MB/s

```
shubhranshu@shubhranshu-HP-ProDesk-600-G6-Microtower-PC:~/vdbench$ sudo ./vdbench -f workload2 anchor=/zfs_pool
```

- Clearly the file creation takes much less time in ext4 as compared to ZFS.

Part - 3:-

Disadvantages of Deduplication:

- **Performance:**
  - In the first workload, ZFS set up the file system in 5 seconds, while ext4 completed it in just 3 seconds. ZFS achieved an average write speed of 77.75 MB/s, whereas ext4 reached a much higher speed of 157.75 MB/s.
  - In the second workload, Ext4 was again significantly faster, as highlighted in the 'large file optimization' section.
  - The performance difference is due to ext4's large file optimization and the deduplication overhead present in ZFS.
  - This demonstrates that deduplication can reduce file system performance due to the added overhead.
- **CPU Utilization:**
  - During the first workload, ZFS recorded an average CPU utilization of 76.1%, while ext4's was significantly lower at 53.4%.
  - In the second workload, ZFS's CPU utilization rose to 81.3%, compared to ext4's 71.1%.
  - These results illustrate that the deduplication feature leads to significantly higher CPU usage in both workloads, primarily due to the overhead associated with deduplication.

Disadvantages of Large File Optimization:

1. **Higher Metadata Overhead for Small Files:**
    - In workload 1, only 450 MB was needed by the files; however, the additional space used afterward was 471 MB, reflecting a 11 MB overhead.
    - Ext4 incurs significant extra space for managing extent trees, particularly with small files.
    - By contrast, ZFS exhibited minimal overhead in this scenario.
2. **No Recovery from Corruption:**
    - Ext4 optimizes large file creation through delayed and contiguous allocation as well as the use of extents.
    - This optimization reduces metadata storage for large files spread across contiguous blocks, leaving no room for data correction mechanisms.
    - Storing checksums without also storing individual block pointers—since they use similar space—limits the ability to address corruption effectively.