ASSIGNMENT 4

Tarun Kumar- 220101103

TASK 1:

Introduction to ZFS and EXT4 Filesystems

ZFS:
Initially developed for the Solaris operating system, ZFS serves both as a **file system** and a **volume manager**, enabling seamless storage management. Its architecture is centered on **data integrity and reliability**, with features designed to ensure optimized file and disk management. Unlike traditional setups where the file system and volume manager operate separately, ZFS integrates both functions, enhancing compatibility and performance. One of ZFS's standout features is **deduplication**, which we will explore further in this comparison.

EXT4:
EXT4, the fourth-generation Extended File System, is focused on **performance and scalability**. It introduces **extents-based allocation**, where files are described by contiguous ranges (extents) on the disk rather than individual fixed-size blocks. This reduces fragmentation and enables more efficient storage by minimizing the number of pointers needed to track file locations. EXT4 also utilizes **delayed allocation**, which defers data writes to allocate larger, contiguous memory segments, further improving performance.

Key Feature Comparison: Deduplication and Large File Creation

1. Deduplication

Deduplication eliminates redundant data, saving significant disk space, especially in environments where duplicate content (with or without minor variations) is common. However, deduplication requires **high computational overhead**, making it suitable only for specific use cases. The process involves:

- **Hashing:** A secure hash function (e.g., SHA-256) generates unique signatures for data segments.
- **Hash Table Lookup:** New data signatures are checked against existing ones. If a match is found, the duplicate is not stored again.

Deduplication can operate at different levels:

- **File-level:** Compares entire files.
- **Block-level:** Compares chunks of data within files.
- **Byte-level:** Detects redundancy at the byte level but incurs the highest overhead.

Additionally, deduplication can be:

- **Synchronous:** Happens as data is written.
- **Asynchronous:** Occurs later when system resources are available.

ZFS supports **block-level synchronous deduplication**, optimizing space savings at the cost of higher CPU usage. On the other hand, **EXT4** lacks built-in deduplication support.

---

2. Large File Creation

The ability to efficiently manage **large files** is essential for modern file systems, and both ZFS and EXT4 have their strengths and limitations in this regard.

- **EXT4** supports a maximum volume size of **1 EiB** (2^60 bytes) and **16 TiB** (2^44 bytes) per file with 48-bit block addressing, using **4 KiB** block sizes. In contrast, its predecessor EXT3 could only handle volumes up to **16 TiB** and individual files up to **2 TiB**.
- **ZFS**, by comparison, supports a maximum file size of **16 TiB**.

EXT4 excels in handling large files due to its **extents-based allocation**. This feature optimizes storage by grouping consecutive blocks under a single extent, reducing the overhead of maintaining large block maps. To further improve performance, EXT4 leverages:

- **Multiblock Allocation:** Allocates multiple blocks in a single call, minimizing overhead and ensuring contiguous block allocation.
- **Delayed Allocation:** Postpones writing to disk until larger, contiguous memory segments are available, improving performance and reducing fragmentation.

These optimizations enable EXT4 to efficiently handle large file creation and management, making it ideal for scenarios where high write performance and large data files are common.

Conclusion

In summary, both ZFS and EXT4 have distinct strengths. **ZFS's deduplication** feature makes it valuable in environments where storage efficiency is critical, although this comes with increased computational overhead. **EXT4**, however, shines in scenarios requiring fast and efficient **large file creation and management**, thanks to its extent-based allocation and delayed write techniques. Choosing between these two filesystems depends on the specific needs—whether the priority is space-saving through deduplication (ZFS) or high performance in handling large files (EXT4).

Deduplication Comparison: ZFS vs. EXT4

Experiment Setup

1. **Enabling Deduplication on ZFS**
    - We activated **deduplication** on the ZFS pool (referred to as `zfs_pool`).

```
udbhav514@IdeaPad:~$ sudo zfs set dedup=on zfs_pool
```

2. **Workload Description**
    - To compare the space usage between ZFS and EXT4, we created a workload (referred to as **workload1**).

```
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
```

    - **Workload Details:**
        - 450 files are created, each **1 MB** in size, structured within a **nested folder of depth 2 and width 3** (50 * 3 * 3 = 450).
        - Files are sequentially read for 30 seconds to gather statistics, though this step is not essential, as **deduplication occurs during file creation**.
        - **Deduplication unit size (dedupunit):** 1 MB (matching the size of each file).
        - **Deduplication ratio (dedupratio):** Set to 2, meaning half the files are duplicates of the other half.
        - ZFS compares new data in **1 MB chunks** against existing data blocks and replaces duplicates with pointers to the original blocks.
3. **Running the Workload on ZFS and EXT4**

- The **anchor** for the workload was set to the root directory of the ZFS pool.

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

- The same workload was then run on the **EXT4 drive**, with its anchor set to the corresponding EXT4 directory.

```
~/vdbench$ sudo ./vdbench -f workload1 anchor=/mnt/54561fbe-141d-4334-a55f-92cd1c8b489e
```

Results

ZFS Results:

1. **Before Workload:**
   - The empty ZFS directory initially occupied **142 KB** of space.

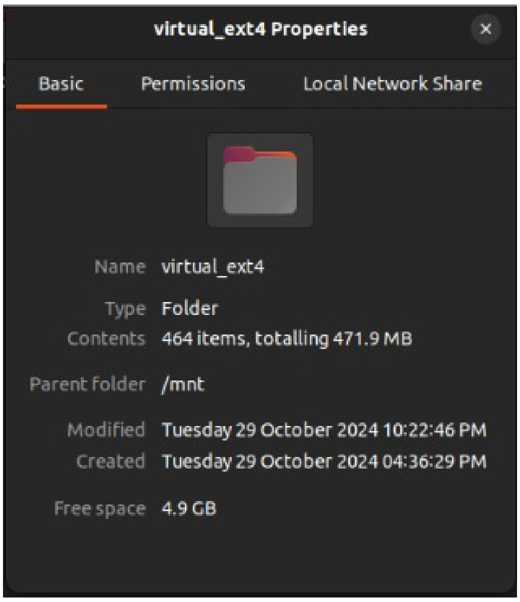
```
tanush@tanush-Precision-Tower-3620:~/lab4/vdbench$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool  4.50G   142K  4.50G        -         -     0%     0%  1.00x    ONLINE  -
```

2. **After Workload:**
   - The ZFS directory size increased to **226 MB**.
   - The deduplication ratio was exactly **2.00x**, as expected.
   - Despite the total intended space usage being **450 MB** (1 MB per file * 450 files), only **226 MB** of additional space was consumed.
     - This reduction was achieved by ZFS using **pointers** to reference duplicate data blocks instead of storing them redundantly.

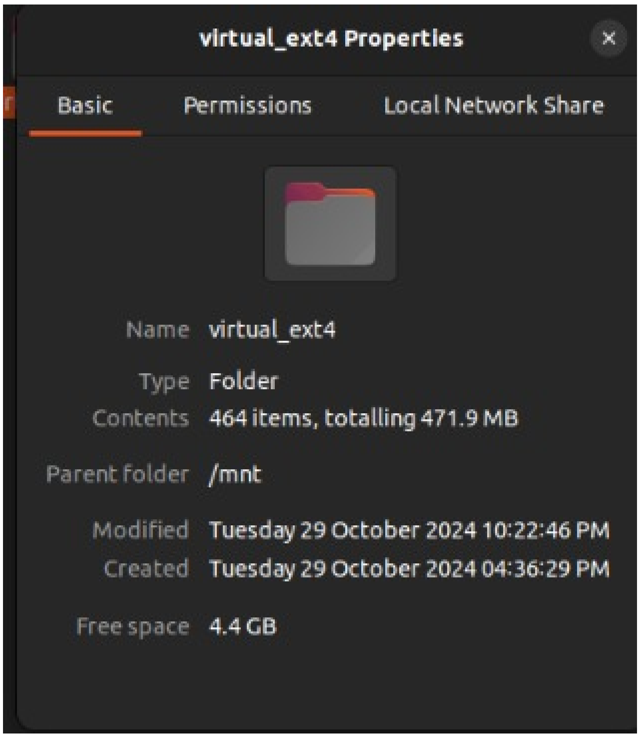
```
tanush@tanush-Precision-Tower-3620:~/lab4/vdbench$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool  4.50G   226M  4.28G        -         -     0%     4%  2.00x    ONLINE  -
```

EXT4 Results:

1. **Before Workload:**
   - The EXT4 directory initially occupied **4.9 GB** of free space.



2. **After Workload:**
   - After running the workload, the free space decreased to **4.4 GB**.
   - The new files consumed **500 MB** of space, slightly more than the expected **450 MB** due to **metadata overhead**.
     - EXT4, lacking a built-in deduplication mechanism, stores each file in full, leading to higher space consumption.



Conclusion

This experiment highlights the efficiency of ZFS's **deduplication** feature. While both file systems stored the same set of files, ZFS required significantly less space—**226 MB vs. 500 MB**—thanks to its ability to detect and eliminate duplicate data. In contrast, EXT4 stored every file independently, resulting in higher storage usage due to the absence of deduplication capabilities.

Large File Creation: ZFS vs. EXT4 Performance Comparison

Experiment Setup

To evaluate the performance of **large file creation** on ZFS and EXT4, we designed a simple workload (**workload2**) focused on creating large files.

1. **Workload Details:**
    - Two **1 GB files** are created in a single folder using the **"create"** operation, as we are specifically testing file creation speed.

```
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. **Execution on Filesystems:**
    - The **ZFS pool directory** was used as the anchor point to test the performance on ZFS.

```
:~/vdbench$ sudo ./vdbench -f workload2 anchor=/zfs_pool
```

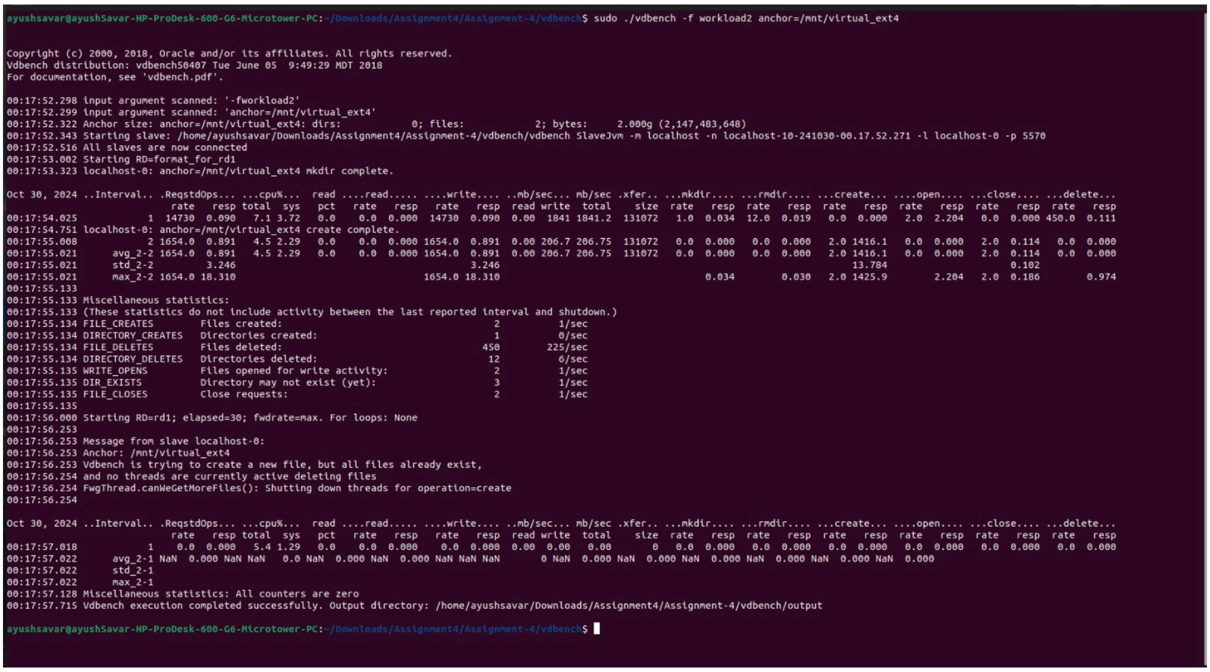
   - Similarly, the **EXT4 drive directory** was set as the anchor point to test EXT4's performance.

```
~/vdbench$ sudo ./vdbench -f workload2 anchor=/mnt/54561fbe-141d-4334-a55f-92cd1c8b489e
```

Results

EXT4 Performance:

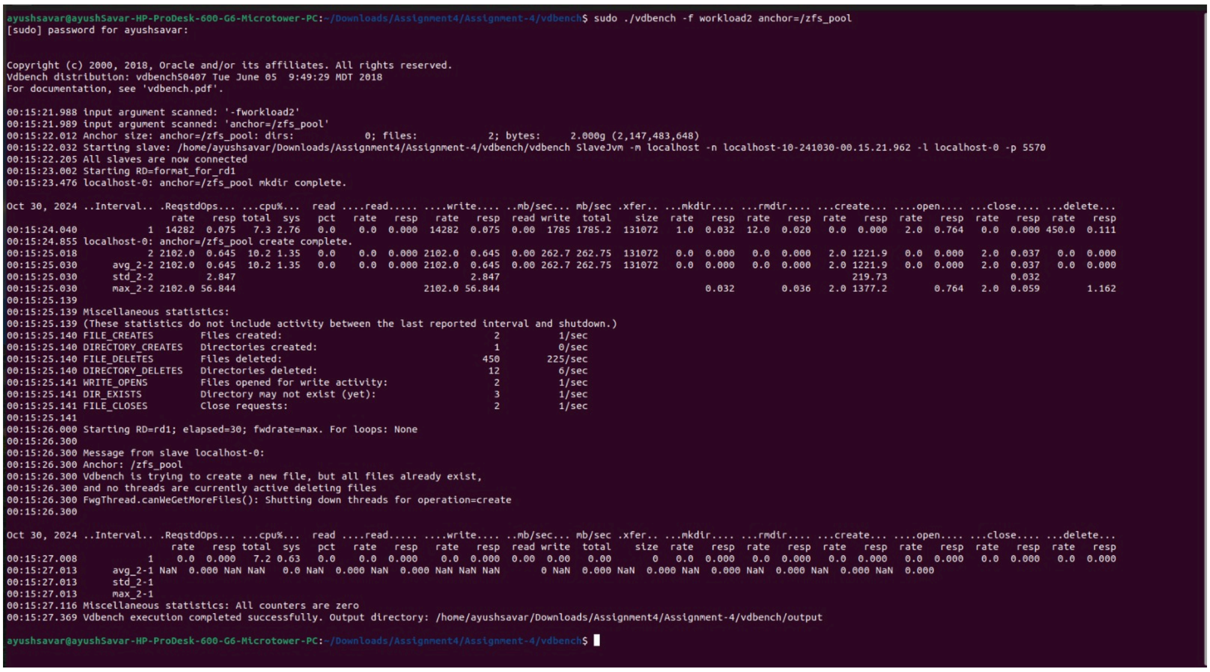
1. **File Creation Time:**
   - Both 1 GB files were created in just **5 seconds**.
2. **Average Write Speed:**
   - The write speed was measured at **409 MB/s**.



```
ayushsavar@ayushSavar-HP-ProDesk-600-G6-Microtower-PC:~/Downloads/Assignment4/Assignment-4/vdbench$ sudo ./vdbench -f workload2 anchor=/mnt/virtual_ext4


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'.

00:17:52.298 input argument scanned: '-fworkload2'
00:17:52.299 input argument scanned: 'anchor=/mnt/virtual_ext4'
00:17:52.322 Anchor size: anchor=/mnt/virtual_ext4: dirs:            0; files:            2; bytes:     2.000g (2,147,483,648)
00:17:52.343 Starting slave: /home/ayushsavar/Downloads/Assignment4/Assignment-4/vdbench/vdbench SlaveJvm -m localhost -n localhost-10-241030-00.17.52.271 -l localhost-0 -p 5570
00:17:52.516 All slaves are now connected
00:17:53.002 Starting RD=format_for_rd1
00:17:53.323 localhost-0: anchor=/mnt/virtual_ext4 mkdir complete.

Oct 30, 2024 ..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
00:17:54.025           1  14730  0.090   7.1 3.72   0.0    0.0  0.000  14730  0.090  0.00  1841 1841.2  131072   1.0  0.034  12.0  0.019   0.0  0.000   2.0  2.204   0.0  0.000 450.0  0.111
00:17:54.751 localhost-0: anchor=/mnt/virtual_ext4 create complete.
00:17:55.008           2 1654.0  0.891   4.5 2.29   0.0    0.0  0.000 1654.0  0.891  0.00 206.7 206.75  131072   0.0  0.000   0.0  0.000   2.0 1416.1   0.0  0.000   2.0  0.114   0.0  0.000
00:17:55.021     avg_2-2 1654.0  0.891   4.5 2.29   0.0    0.0  0.000 1654.0  0.891  0.00 206.7 206.75  131072   0.0  0.000   0.0  0.000   2.0 1416.1   0.0  0.000   2.0  0.114   0.0  0.000
00:17:55.021     std_2-2         3.246                                    3.246                                                         13.784                     0.102
00:17:55.021     max_2-2 1654.0 18.310                             1654.0 18.310                                0.034         0.030   2.0 1425.9          2.204   2.0  0.186          0.974
00:17:55.133
00:17:55.133 Miscellaneous statistics:
00:17:55.133 (These statistics do not include activity between the last reported interval and shutdown.)
00:17:55.134 FILE_CREATES        Files created:                              2          1/sec
00:17:55.134 DIRECTORY_CREATES   Directories created:                        1          0/sec
00:17:55.134 FILE_DELETES        Files deleted:                            450        225/sec
00:17:55.134 DIRECTORY_DELETES   Directories deleted:                       12          6/sec
00:17:55.135 WRITE_OPENS         Files opened for write activity:            2          1/sec
00:17:55.135 DIR_EXISTS          Directory may not exist (yet):              3          1/sec
00:17:55.135 FILE_CLOSES         Close requests:                             2          1/sec
00:17:55.135
00:17:56.000 Starting RD=rd1; elapsed=30; fwdrate=max. For loops: None
00:17:56.253
00:17:56.253 Message from slave localhost-0:
00:17:56.253 Anchor: /mnt/virtual_ext4
00:17:56.253 Vdbench is trying to create a new file, but all files already exist,
00:17:56.254 and no threads are currently active deleting files
00:17:56.254 FwgThread.canWeGetMoreFiles(): Shutting down threads for operation=create
00:17:56.254

Oct 30, 2024 ..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
00:17:57.018           1    0.0  0.000   5.4 1.29   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
00:17:57.022     avg_2-1 NaN  0.000 NaN NaN   0.0 NaN  0.000 NaN  0.000 NaN NaN NaN       0 NaN  0.000 NaN  0.000 NaN  0.000 NaN  0.000 NaN  0.000 NaN  0.000
00:17:57.022     std_2-1
00:17:57.022     max_2-1
00:17:57.128 Miscellaneous statistics: All counters are zero
00:17:57.715 Vdbench execution completed successfully. Output directory: /home/ayushsavar/Downloads/Assignment4/Assignment-4/vdbench/output

ayushsavar@ayushSavar-HP-ProDesk-600-G6-Microtower-PC:~/Downloads/Assignment4/Assignment-4/vdbench$
```

ZFS Performance:

1. **File Creation Time:**
   - It took **22 seconds** to create both 1 GB files.
2. **Average Write Speed:**
   - The write speed was **94 MB/s**.



Conclusion

From this test, it is evident that **EXT4** outperforms **ZFS** when it comes to creating large files. EXT4 managed to complete the task in just 5 seconds, with an impressive **write rate of 409 MB/s**. In comparison, **ZFS** took significantly longer—22 seconds—with a lower average write speed of **94 MB/s**. These results align with the design goals of both filesystems: **EXT4** prioritizes performance and speed, especially with large files, while **ZFS** offers advanced data management features (like deduplication) but at the cost of slower write speeds.

Note: All output folders have been included in the submission for reference.

Disadvantages of Deduplication

1. Impact on Performance

The inclusion of deduplication in ZFS results in a noticeable slowdown in setup time and write speeds compared to EXT4. This can be attributed to the additional processing overhead that deduplication imposes. The large file handling capabilities of EXT4, combined with its simpler structure, help it perform better in scenarios where file operations involve substantial amounts of data. The findings indicate that deduplication introduces a performance trade-off, where the system's processing speed is impacted due to the added workload of identifying and eliminating duplicate data.

2. Increased CPU Utilization

Deduplication in ZFS places a heavier load on the CPU. This increased utilization is mainly due to the additional computational effort required to hash data blocks and compare them with existing blocks. The overhead involved in this deduplication process results in a more resource-intensive workload for ZFS, whereas EXT4, which lacks this feature, operates with a lower CPU demand. This disparity highlights the processing costs associated with deduplication, indicating a trade-off between storage efficiency and CPU resource usage.

Disadvantages of Optimizing Large File Creation in EXT4

1. **Higher Metadata Overhead for Small Files**
   - In **workload1**, the total size of the files created was **450 MB**. However, the actual space used on the EXT4 filesystem was **472 MB**, with **12 MB of overhead**.
   - In contrast, **ZFS incurred minimal overhead**. This overhead in EXT4 comes from maintaining **extent trees**, which are efficient for large files but **consume more space when managing small files**.
2. **Limited Recovery from Data Corruption**
   - EXT4 optimizes large file creation through **delayed allocation, contiguous block allocation, and extents**. However, these optimizations come at the cost of **data recovery capabilities**.
   - The minimal metadata stored for large files across contiguous blocks leaves **little room for data integrity mechanisms, such as checksums**.
   - Even if checksums were implemented, the absence of **individual block pointers** reduces the possibility of effective recovery. Since block pointers and checksums would require a similar amount of space, **choosing not to store block-level metadata sacrifices the ability to repair corrupted data**.

Summary

The comparison reveals key trade-offs between ZFS and EXT4. **ZFS's deduplication** provides space savings but at the expense of performance and higher CPU utilization. On the other hand, **EXT4's large file optimizations** improve speed but introduce overhead for smaller files and limit the ability to recover from data corruption due to minimal metadata storage.

Instructions to run:

Installation Instructions for ZFS

Install ZFS:
a. Begin by installing the ZFS filesystem with the following command:

```
sudo apt install zfsutils-linux -y
```

b. Identify a suitable disk for creating the ZFS pool. **Note:** The disk must be at least **5 GB** in size and **should not** be a system disk (e.g., avoid using `/dev/sda` if it is in use by the OS). Use the following command to list available disks:

```
sudo fdisk -l
```

c. Once you've selected a disk (for example, `/dev/sdb`), create a ZFS pool named `zfs_pool` with the following command:

```
sudo zpool create zfs_pool /dev/sdb
```

d. Enable **deduplication** for the newly created pool to avoid storing duplicate data:

```
sudo zfs set dedup=on zfs_pool
```

1. e. You can now access the ZFS pool at `/zfs_pool`. This directory will serve as the root location for running any workloads.

Setting up the ext4 Filesystem

1. **Overview**
   The ext4 filesystem is pre-installed and serves as the default filesystem on Ubuntu. Follow these steps to format a disk and configure it with the ext4 filesystem.
2. **Steps to Format and Configure ext4**
   a. **Open the "Disks" Application** from the application menu.

   b. **Select a Disk**:
   Choose the disk you want to format. Ensure it has **at least 5 GB of free space**.
   c. **Format the Partition**:
   Click the **gear icon** next to the selected disk and select **"Format Partition"**.
   d. **Set Partition Details**:

   - Assign a **name** to the disk.
   - Select **Ext4** as the filesystem type (it is usually the first option).
   - Enable the **"Erase"** option to securely wipe existing data.
   - Click **Next** to proceed.
3. e. **Confirm and Format**:
   Click **"Format"** to complete the process.
   f. **Mount the Disk**:
   Once the formatting is complete, ensure the disk is mounted. If not:
   - Open **Mount Options** (Gear Icon → **Edit Mount Options**).
   - Disable **"User Session Defaults"**.

- Enable **“Mount at system startup”**.
- Reboot your system to apply the changes.

Finding the Anchors for ZFS and ext4 Partitions

To run workloads on ZFS or ext4 partitions, you need to locate the **anchors**—the mount points corresponding to these partitions. Here's how to find them:

1. **Identifying the ZFS Anchor**:
   a. If we have a ZFS pool (e.g., `zfs_pool`), the anchor will correspond to its mount point.
   b. In this example, the mount point `/zfs_pool` serves as the anchor.
2. **Identifying the ext4 Anchor**:
   a. For an ext4 partition (e.g., mounted from `/dev/sdc`), the anchor will be the path where it is mounted.
   b. In this case, the anchor could look something like:
   `/mnt/54561fbe-141d-4334-a55f-92cd1c8b489e`.
3. **Importance of the Anchor**:
   Finding the correct anchor is essential since the workloads rely on it to run properly. Without knowing the exact mount point, the workloads will not function as expected.

Running Workloads on ZFS and ext4 Filesystems

Follow these steps to run workloads on the ZFS and ext4 partitions:

1. **Prepare the Workloads**
   - Copy both workload files (`workload1` and `workload2`) to your **vdbench** directory.
2. **Navigate to the vdbench Directory**
   - Open a terminal and use the `cd` command to move into the **vdbench** directory.

Execute the Workloads
a. **Run workload1 on the ZFS partition**:
Replace `zfs_pool` with the **anchor** of the ZFS partition.

b. **Run workload1 on the ext4 partition**:
Replace `/mnt/54561fbe-141d-4334-a55f-92cd1c8b489e` with the **anchor** of the ext4 partition.

c. **Run workload2 on either partition**:
Use the same commands as above, but substitute `workload2` for `workload1`.
Example for ZFS:

```
./vdbench -f workload2 -o /zfs_pool
```

Example for ext4:

```
./vdbench -f workload2 -o /mnt/54561fbe-141d-4334-a55f-92cd1c8b489e
```

These commands will ensure the workloads are correctly executed on the designated partitions.

Viewing Statistics

1. **Viewing the Summary for the Last Workload**:
   - After running a workload, you can find a summary of the results in the **`summary.html`** file located in the **Output** folder within your **vdbench** directory.

Monitoring Disk Space Usage Before and After Running Workloads:
a. **For ZFS**:
i. To check the space usage, run the following command:

```
zpool list
```

iii. Run this command both **before** and **after** running the workload to calculate the space used by the files. The difference in allocation will reflect the space consumed by the workload, as described previously.

b. For ext4:

i. Open the **File Manager** in Ubuntu and navigate to the folder where your ext4 partition is mounted (the **anchor**).
- For example, if the anchor is `/mnt/54561fbe-141d-4334-a55f-92cd1c8b489e`, navigate to the `/mnt` directory.

ii. **Right-click** on the anchor folder and select **Properties**.
- In the properties window, you can view the **space usage** for the partition.

iii. We can see the space used **before and after** running the workload to determine the amount of space consumed.