

# CS 344 Lab Assignment – 4

## Group-M12

200123030-Kura Priyanka

200123054-Siddam Shetty Sahithi Shresta

200123070-Vemulapati Sai Lakshmi Swarupa

## PART-1

We will be comparing **Deduplication** and **Large File Creation** in the **ZFS** and **EXT4** filesystems.

An introduction to the two filesystems:

### **ZFS:**

**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.

One of the features provided by the **ZFS** system is data deduplication, about which we will discuss later.

### **EXT4:**

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.

## Deduplication:

**Deduplication** is the process of eliminating duplicate copies of data. This can save a lot of space on the hard drive, and is especially useful in some environments where a lot of duplicate data is encountered, with or without minor changes. However, this also comes with a tradeoff of high overhead computations, and is thus only recommended to be used in rare scenarios. The deduplication is achieved by hashing(using a secure hash like SHA256) a portion of data to a unique(approximately) signature, and storing these in a hash table. The signature of new data is compared to pre-existing values in the hash table, and data with pre-existing signature is deemed to be a copy of the data whose signature matches with it. Deduplication can be implemented in different levels, depending on the size of data that gets hashed to a signature, with an increasing amount of tradeoff between overhead computations and space saved due to redundant data not being copied. These are file-level, block-level and byte-level. Deduplication can also be synchronous or asynchronous, depending on whether the process happens as the data is being written, or whether the copies are hashed and deleted when the CPU is free. **ZFS** has the deduplication feature, and it uses block-level synchronous deduplication. **EXT4** does not support deduplication.

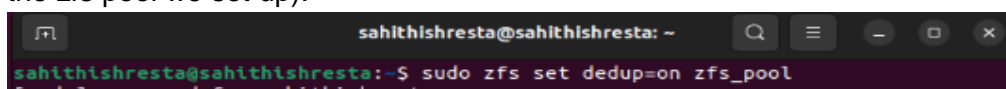
## Large File Creation:

The **EXT4** file system supports a maximum volume of **1 EiB**(ExbiByte)= $2^{60}$  Bytes, and a maximum file size of **16 TiB**(TebiBytes)= $2^{48}$  Bytes with the standard **4KiB** blocks, with **48** bit block addressing. By comparison, **EXT3** only supports file system size of **16TiB** and **2 TiB** file size. **ZFS** supports **16 TiB** file system size. **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. For this new mapping system using extents to function properly and efficiently, other features in **EXT4** also help. **EXT4** features multiblock allocation, which allocates many blocks in a single call, instead of a single block per call, avoiding a lot of overhead, and being able to easily allocate contiguous blocks of memory. This works in tandem with delayed allocation, where it doesn't write to disk on every write operation, but notes the data to be written, and then writes a big chunk of data into a contiguous memory segment using multiblock allocation.

## Experimental Section:

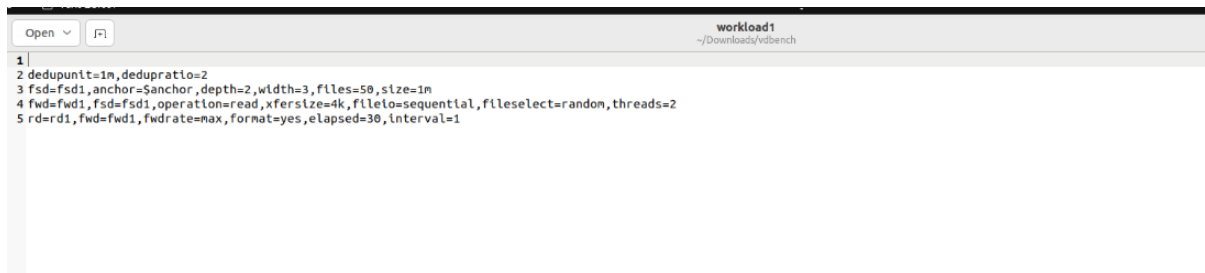
### Deduplication:

1: ZFS has a data deduplication feature which we turned on using (zfs\_pool is the name of the zfs pool we set up):

A terminal window with a dark background. The prompt is 'sahithishresta@sahithishresta: ~'. The command entered is 'sudo zfs set dedup=on zfs\_pool'. The output shows a password prompt: 'sudo: password for sahithishresta:'.

```
sahithishresta@sahithishresta: ~  
sahithishresta@sahithishresta:~$ sudo zfs set dedup=on zfs_pool  
[sudo] password for sahithishresta:
```

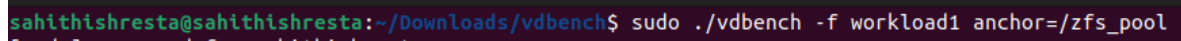
2 : We created the following workload for data deduplication (**workload1**). We will use this workload to compare the space occupied by the new files in ZFS with the space occupied in ext4



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

3: Basically, we are creating **450 files (50\*3\*3) each of size 1MB** in a nested folder structure of depth 2 and width 3. Then these files are being read sequentially for thirty seconds to monitor statistics (although this part is not important since the deduplication is done during file creation).

4 : **dedupunit** is set to **1MB** and **dedupratio** is set to **2**. **dedupratio** is the ratio of the total number of blocks (of size **dedupunit**) with the number of blocks containing unique data. **dedupunit** on the other hand is the size of the block which will be compared with pre-existing blocks to check for duplicates. **We set it to 1MB because this is the size of one file**. So basically, half of the files will be duplicates of the other half.



```
sahithishresta@sahithishresta:~/Downloads/vdbench$ sudo ./vdbench -f workload1 anchor=/zfs_pool
```

5 : 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):

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

7: We found the following results:

i. **ZFS:**

1. Initially, the empty ZFS folder had **7.6 MB** of data.
2. After running the workload, the ZFS folder had **231 MB** of data.
3. We observed a deduplication ratio of 2.00x (which is what we wanted).

This means that the new files took **223 MB** of space. However, the intended space is **450 MB** (1 MB\*450). 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.

## After Workload:

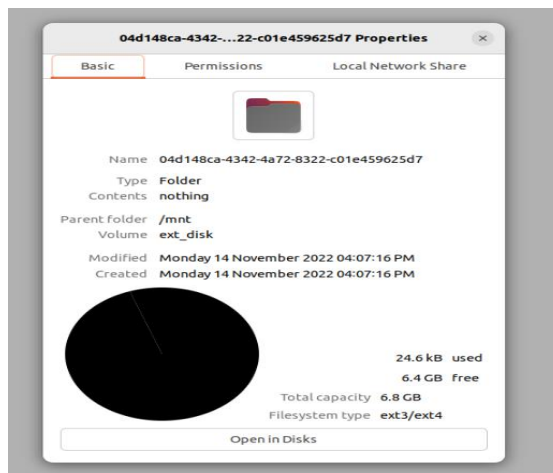
```
sahithishresta@sahithishresta:~/Downloads/vdbench$ zpool list
NAME      SIZE  ALLOC   FREE CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool  6.50G   226M   6.28G      -        -       0%    3%   2.00x    ONLINE   -
```

## ii. ext4:

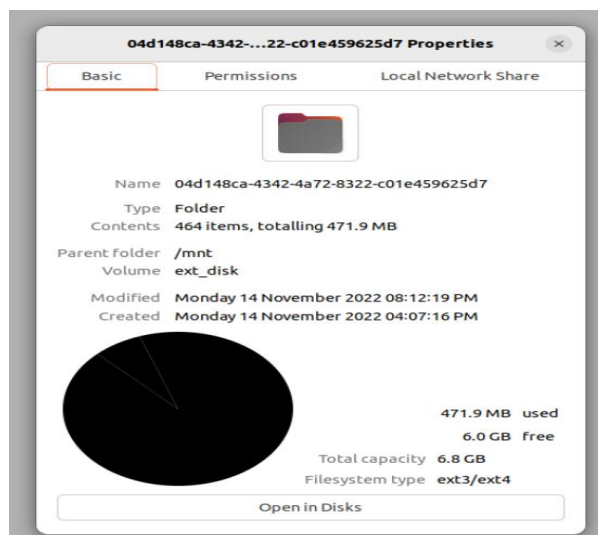
1. Initially the empty ext4 folder had **24.6 kB** of data.
2. After running the workload, the ext4 folder had **471.9 MB** of data.

The new files thus took **471 MB** of space (a little more than intended because of metadata overhead).

## Before Workload

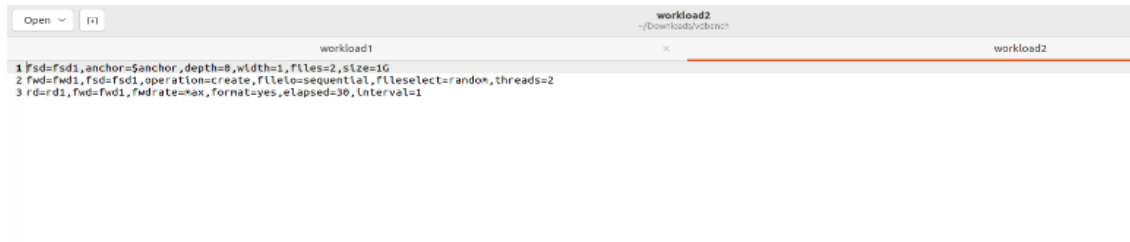


## After Workload



## 2. Large File Creation:

- a. We know that **ext4** optimises large file creation better than **ZFS** does as we can clearly see using our workload.
- b. We created the following workload for testing large file creation (**workload2**):



```

1 fsd=fsd1,anchor=$anchor,depth=8,width=1,files=2,size=1G
2 fwd=fwd1,fsd=fsd1,operation=create,fileio=sequential,fileselect=round,threads=2
3 rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=30,interval=1

```

- c. What we are doing here is creating **two files of size 1GB** in one folder. The operation used is “create” since we are testing file creation.
- d. We run this on the ZFS file system by setting anchor equal to the directory pointing to the **ZFS** pool similar to workload1.
- e. We run this workload on the ext4 file system by setting anchor equal to the directory pointing to the ext4 drive similar to workload1.
- f. We found the following results:
  - i. **ext4:**
    1. Finishes creating files in just **4 seconds!** Remember, these are 1GB files!
    2. The average write rate is **511 MB/s!**
  - ii. **ZFS:**
    1. Takes a whole **16 seconds** to create the files.
    2. The average write rate is **125.9 MB/s.**
- g. **Note: All output folders attached in submission**

#### Disadvantages of deduplication:

1. **Performance:** In the first workload, the ZFS system set up the file system in **5 seconds** whereas ext4 just took **3 seconds**. ZFS had an average write speed of **77.75 MB/s** while ext4 had an average write speed of **157.75 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:** In the first workload, **during file structure setup** (the other section is for read which doesn’t have much to do with deduplication), the average CPU

utilisation of ZFS was **76.1%** while that of ext4 was **53.4%** which is significantly lower. In the second workload too, the CPU utilisation of ZFS was **81.3%** while that of ext4 was **71.1%**. This shows that the deduplication feature has significantly higher CPU usage in both workloads. This is due to deduplication overhead.

**Note: Overhead described at the beginning of report when describing deduplication.**

### **Disadvantages of Optimising Large File Creation:**

1. **Greater metadata overhead for small files:** When running workload1, only **450 MB** was required by the files. But, additional **used space after running workload1 was 472 MB** (12 MB of overhead). In ZFS however, the overhead was very small. A lot of **additional space is used in maintaining the extent trees** compared to the actual data **(for small files)**.
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, why not also maintain individual block pointers since they take around the same amount of space?