

Assignment 4 - Quantitative comparison of filesystems

Group C25

Anindya Vijayvargeeya - 200101015

Gunjan Dhanuka - 200101038

Pranjal Singh - 200101084

Part 1

For this lab, we have chosen the **Deduplication** feature of the ZFS file system and the **Large File Creation** of the EXT4 file system.

ZFS

ZFS began as part of the Sun Microsystems Solaris operating system, and later published under an Open source license. Management of stored data generally involves 2 aspects: managing the physical volume of multiple storage devices, and their organization into logical blocks visible to an operating system. ZFS unifies both of these roles and acts as both the volume manager and the file system. Therefore, it has complete knowledge of both the physical disks and volumes, and also of all the files stored on them. ZFS' complete control of the storage system is used to ensure that every step, whether related to file management or disk management, is verified, confirmed, corrected if needed, and optimized, in a way that storage controller cards and separate volume and file managers cannot achieve.

Deduplication in ZFS

- Deduplication is the process of eliminating duplicate copies of data. It comes with high overhead computations but is highly useful in scenarios where large data duplication is encountered.
- Cryptographic hash functions (like SHA256) are used to attach a portion of data to a unique signature which is stored in a table. Signatures of new data are compared to pre-existing values to ensure no duplicate data is being stored.
- Deduplication can be implemented on 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.

EXT4

The EXT4 filesystem primarily improves performance, reliability, and capacity. Here, data allocation was changed from fixed blocks to extents. An extent is described by its starting and ending place on the hard drive. This makes it possible to describe very long, physically contiguous files in a single inode pointer entry, which can significantly reduce the number of pointers required to describe the location of all the data in larger files. Other allocation strategies have been implemented in EXT4 to further reduce fragmentation. 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

Large File Creation in EXT4

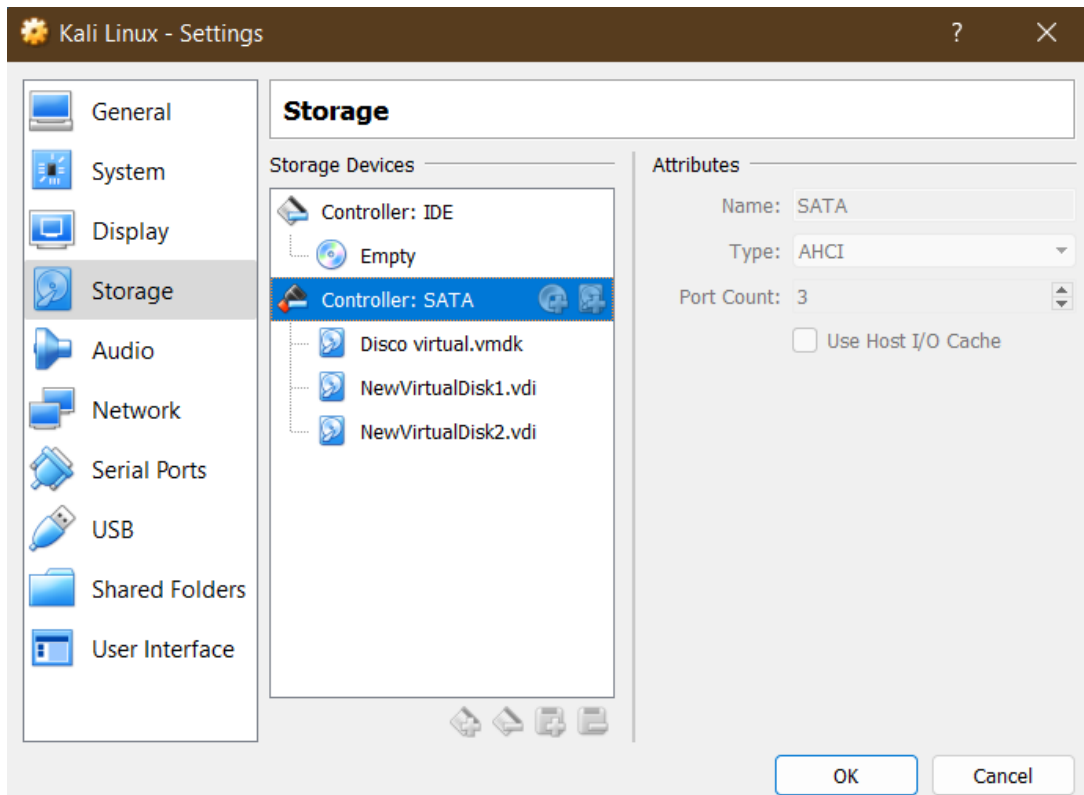
- The EXT4 file system uses 48-bit addressing, and thus is able to support a maximum volume of 1 EiB (2^{60} Bytes), and a maximum file size of 16 TiB (2^{44} Bytes) with the standard 4KiB blocks. In comparison to this, ZFS supports 16 TiB file system size.
- EXT4 optimizes the creation and handling 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.
- 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.

Installing File Systems

Creating a ZFS Partition

1. We have used Kali Linux in VirtualBox and created two hard disks – one for ZFS and one for EXT4 respectively.
2. Firstly, add a new disk to the Virtual Machine. In the Storage section, add a hard in the SATA section. Here we have added NewVirtualDisk1.vdi and NewVirtualDisk2.vdi as the

new disks.



3. Run the Virtual Machine and install ZFS using `sudo apt install zfsutils-linux`
4. Verify the installation using the `whereis zfs` command.
5. Now we first check the available disks using the `sudo fdisk -l` which lists all the disks. We will use the `/dev/sdb` disk to create the pool for ZFS.

```
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
Disklabel type: gpt
Disk identifier: F1421486-2DBC-1B44-BC4B-DB96EE2F1414
```

6. We create a striped pool using the command `sudo zpool create new-pool /dev/sdc`

Creating an ext4 Partition

1. We will use another hard disk of the two we added in the ZFS Partition creation time.
2. Run the Virtual Machine and execute the following commands to create ext4 Partition.
3. ext4 comes pre-installed as it is the default file system for Debian-based Linux distros, so no need to install anything here.

4. Check the list of disks using the `sudo fdisk -l` command. We will be using the `/dev/sdc` disk to create our ext4 pool.

```
Disk /dev/sdc: 8 GiB, 8589934592 bytes, 16777216 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. Then use the command `mkfs.ext4 /dev/sdc` to create an ext4 file system from the disk partition.

```
root@kali:~# mkfs.ext4 /dev/sdc
mke2fs 1.46.5 (30-Dec-2021)
/dev/sdc contains a ext4 file system
    created on Sun Nov 13 04:48:04 2022
Proceed anyway? (y,N) y
Creating filesystem with 2097152 4k blocks and 524288 inodes
Filesystem UUID: e0cle264-3c36-447e-be30-7ac6f9ab53c3
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

6. Use the `e2label /dev/sdc old-pool` to give a label to the newly created partition.
7. Then we create a mount point for the partition and mount it using the commands:
`mkdir old-pool`
`mount /dev/sdc old-pool`

Running the workloads

Filesystem Definition (FSD)

1. `fsd=name`: Unique name for this FSD
2. `anchor=/dir`: The name of the directory where the directory structure will be created.
3. `depth=num`: How many levels of directories to create under the anchor.
4. `files=num`: How many files or create in the lowest level of directories.
5. `width=num`: How many directories to create in each new directory.
6. `sizes=(num, num,...)`: Size of the files that need to be created

Filesystem Workload Definition (FWD)

1. `fwd=name`: Unique name for this FWD
2. `fsd=(xx, ...)`: Name of the FSD to use
3. `operation=xxxx`: Specifies a single file system operation that must be done for this workload.

4. fileio=sequential: FileIO can be of two types – random or sequential
5. fileselect=random/seq: How to select file names for processing
6. threads=num: How many thread to run for this workload
7. xfersize=(num,...): Specifies the data transfer size to use for read and write operations

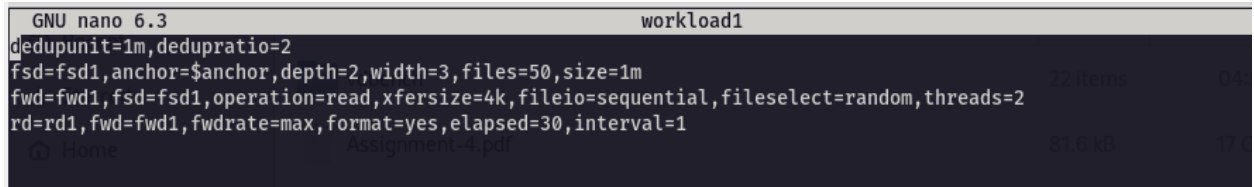
Run Definition(RD)

RD binds together storage and workload will be run together and for how long. Each RD name has to be unique.

1. fwd=(xx, yy, ...): Name of the FWD to use
2. format: 'yes' causes the directory to be created from scratch, including initialization of all files
3. elapsed: How long to run (in seconds)
4. interval : Statistics collection interval in seconds
5. fwdrate=num : Run workload of num operation per second

Deduplication

1. We turn on the data deduplication option in the ZFS pool by using the command **sudo zfs set dedup=on new-pool**
2. Then we create a new workload called **workload1** which we will use to compare the space occupied by new files in ZFS with the space occupied in ext4.



```
GNU nano 6.3 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
```

3. Explanation of the workload:
 - a. We create $3*3*30 = 450$ files each of size 1 MB in a nested directory structure which has **depth=2 and width=3**.
 - b. Then we read the files sequentially for 30 seconds to monitor the statistics.
 - c. **dedupunit=1MB** is the size of the block that will be complicated with pre-existing blocks to check for duplicates.
 - d. **dedupratio=2** is the ratio of total number of blocks (of size 1 MB) to the total number of blocks having unique data.
 - e. Since the size of a file is also 1MB, half of the files will be the duplicates of the other half.
 - f. We then read the files to validate them, using 2 threads of the CPU and a data transfer unit of 4kB. The file IO is done sequentially while the files are selected randomly.

4. Then we run this workload on the ZFS file system by using the following command.

```
root@kali:~/vdbench# sudo ./vdbench -f workload1 anchor=/new-pool

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'.

05:02:49.674 Created output directory '/root/vdbench/output'
05:02:49.731 input argument scanned: '-fworkload1'
05:02:49.731 input argument scanned: 'anchor=/new-pool'
05:02:49.830 Anchor size: anchor=/new-pool: dirs:          12; files:          450; bytes:    450.000m (471,859,200)
05:02:49.938 Starting slave: /root/vdbench/vdbench SlaveJvm -m localhost -n localhost-10-221113-05.02.49.639 -l local
host-0 -p 5570
05:02:50.511 All slaves are now connected
05:02:52.003 Starting RD=format_for_rd1
05:02:52.302 localhost-0: anchor=/new-pool mkdir complete.
```

We also run this workload on the ext4 file system:

```
root@kali:~/vdbench# sudo ./vdbench -f workload1 anchor=/root/old-pool

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'.

05:12:02.133 input argument scanned: '-fworkload1'
05:12:02.136 input argument scanned: 'anchor=/root/old-pool'
05:12:02.257 Anchor size: anchor=/root/old-pool: dirs:          12; files:          450; bytes:    450.000m (471,859,
200)
05:12:02.380 Starting slave: /root/vdbench/vdbench SlaveJvm -m localhost -n localhost-10-221113-05.12.02.011 -l local
host-0 -p 5570
05:12:02.965 All slaves are now connected
05:12:05.002 Starting RD=format_for_rd1
```

5. Results of running the workload:

a. ZFS:

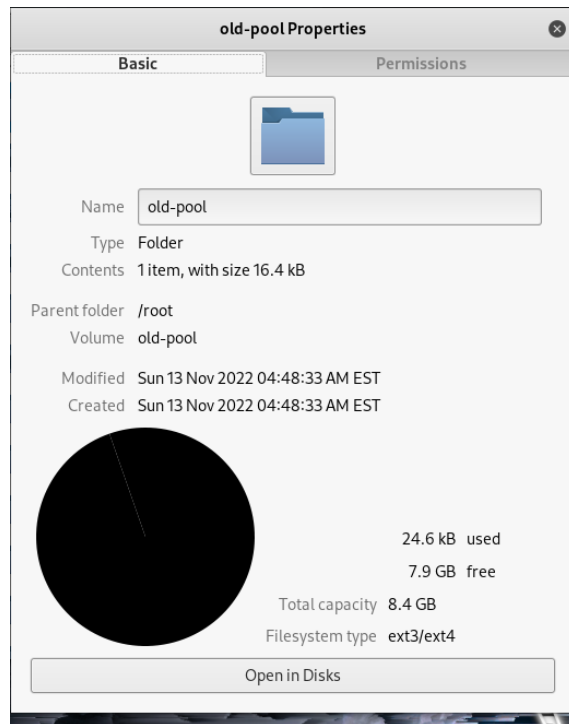
- Initially the ZFS folder was empty.
- After the workload, we see that there is 226 MB of data.
- Hence the deduplication ratio is 2 (which is what we expected).
- So the **new files have taken a space of 226 MB** while the expected space should have been 450 MB. Hence the data deduplication feature simply makes a pointer point to the old data when a duplicate is found, instead of maintaining whole blocks of data.

After running the workload:

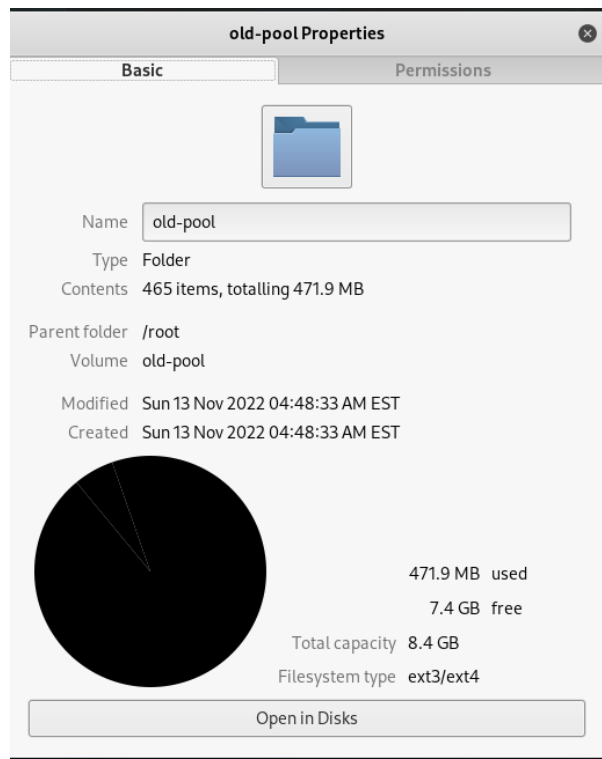
```
root@kali:~/vdbench# zpool list
NAME      SIZE  ALLOC   FREE CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
new-pool  2.75G  226M   2.53G      -          -     0%    8%   2.00x  ONLINE  -
```

b. Ext4:

- i. Initially the empty ext4 folder had 16.4 kB of data.



- ii. Then we run the workload1 as shown above. After running the workload:



- iii. Hence we see that the created files occupied nearly 472 MB of space.

Large File Creation

1. We will compare the large file creation abilities of ext4 and ZFS in this workload called **workload2**.

```
GNU nano 6.3 workload2
fsd=fsd1,anchor=$anchor,depth=0,width=1,files=3,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. In the above workload, we are creating **2 files of size 1GB each** in one folder. Here we are using the operation *create* since we need to test file creation.
 - a. We do the fileIO sequentially using 2 threads of the CPU but the files are selected randomly.
 - b. We make depth=0 and the width=1 so that the files are created within the same directory.
 - c. format='yes' will clean the directory before running the workload.
3. Then again, we run this workload on the ZFS and the ext4 file systems.
 - a. **ZFS:**

00:00:14.26
hr min sec


```
root@kali:~/vdbench# zpool list
NAME      SIZE  ALLOC   FREE CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
new-pool  2.75G   100K   2.75G      -          -        0%    0%   1.00x    ONLINE   -
```

```
root@kali:~/vdbench# zpool list
NAME      SIZE  ALLOC   FREE CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
new-pool  2.75G   2.00G   767M      -          -        0%   72%   1.00x    ONLINE   -
```

- b. **ext4:**

old-pool Properties

BasicPermissions



Name

old-pool

Type

Folder

Contents

465 items, totalling 471.9 MB

Parent folder

/root

Volume

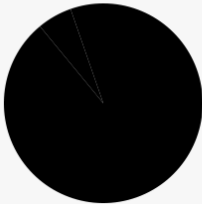
old-pool

Modified

Sun 13 Nov 2022 04:48:33 AM EST

Created

Sun 13 Nov 2022 04:48:33 AM EST



471.9 MB used

7.4 GB free

Total capacity 8.4 GB

Filesystem type ext3/ext4


Open in Disks

00:00:11.76

hrminsec

old-pool Properties

BasicPermissions



Name

old-pool

Type

Folder

Contents

6 items, totalling 2.1 GB

Parent folder

/root

Volume

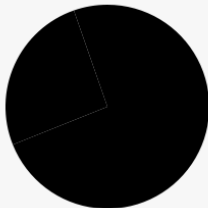
old-pool

Modified

Sun 13 Nov 2022 04:48:33 AM EST

Created

Sun 13 Nov 2022 04:48:33 AM EST



2.1 GB used

5.8 GB free

Total capacity 8.4 GB

Filesystem type ext3/ext4

Open in Disks

Part 3

Disadvantages of deduplication

1. **Performance:** ZFS took approximately 4 seconds while ext4 took 3 seconds to create the files in the first workload. ZFS could only achieve an average write speed of 102.50 MBps while ext4 had average write speeds of about 147.31 MBps. In the large-file optimization section, we notice that ext4 was able to get better speeds. This can be attributed to the deduplication overhead of ZFS and the large file optimization of the ext4 system.
2. **CPU Utilization:** In the first workload, we notice the average CPU utilization for ZFS is 75.3% while for ext4 it is 57%. In the second workload also, the CPU utilization of ZFS was nearly 69% while that for ext4 was 66%.

From this we can conclude that deduplication has significantly higher CPU usage in both workloads.

Disadvantages of optimizing large file creation

1. **Greater space wastage due to metadata:** While running workload1, the file size was 450 MB but ext4 used up 472 MB of space (22 MB of overhead). On the other hand, in ZFS, the overhead is very small. This is because the extra space is utilized for maintaining the extent trees compared to the actual files which were quite small.
 2. **No recovery:** ext4 uses delayed and contiguous allocation to speed up large file creation. This makes it nearly impossible for any mechanism to correct data due to the less amount of metadata stored for large files.
-