Assignment 4

Group - 31

Chandrabhushan Reddy	200101027
Pramodh Billa	200101025
Sathvika Kalangi	200101048

Part 1:

We will compare the files systems ZFS and ext4 using vdbench. We created workloads to test the following two features in both the file systems:

1. Data Deduplication
2. Large file management

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.

Description of the features:

Deduplication:

Deduplication automatically avoids writing the same data twice on our drive by detecting duplicate data blocks,files or bytes(depends on the file system) and keeping track of the multiple places where these are needed. This can save space and unnecessary I/O operations which can also improve performance.

Implementation in ZFS file system:

- ZFS implements block level deduplication. Blocks are checksummed by doing ZFS's 256 bit block checksums with some secure hash like SHA256. A table maps the block's checksum to its storage location and reference count. When we store another copy of an existing block , instead of allocating a new block on the disk, zfs increments the reference count on the existing block.
- ZFS deduplication is synchronous. That means duplicates are eliminated as they appear instead of being stored and being eliminated at leisure.
- In case of hash collisions i.e the checksums are the same but the blocks aren't , zfs also supports a 'verify' option. We normally do set dedup=on but to add verification set dedup=verify can be used. This option verifies that the incoming block and the alleged duplicate are indeed the same.
- There is no limit to data to which dedup can be applied in zfs. It's just that as data size increases , the size of the dedup table gets larger and hence it may not be possible after some limit to keep in its memory and it will have to be loaded from L2ARC or disk.

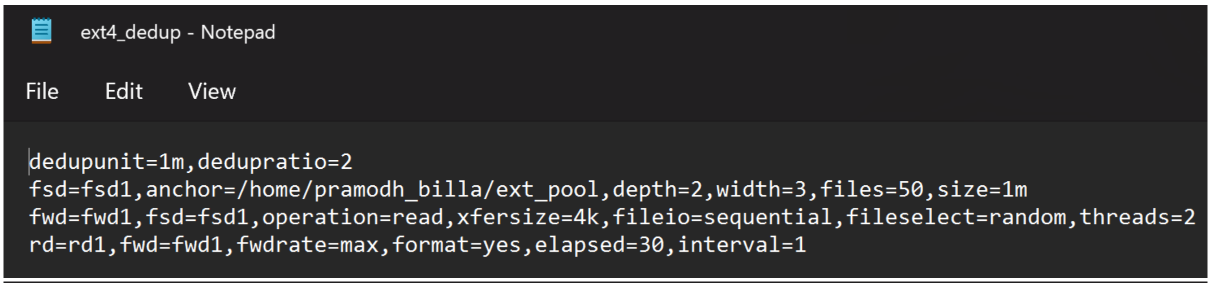
Large File Creation:

The EXT4 file system supports a maximum volume of 1 EiB(ExbiByte)=260 Bytes, and a maximum file size of 16 TiB(TebiBytes)=244 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.

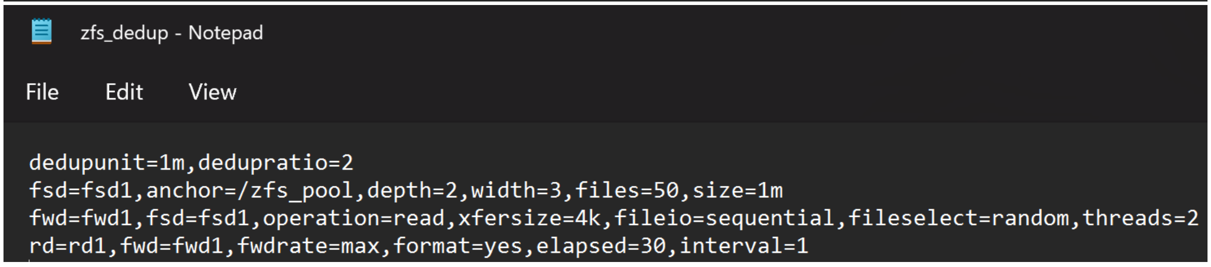
Part 2:

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):
   sudo zfs set dedup=on zfs_pool
2. We created the following workload for data deduplication. We will use this workload to compare the space occupied by the new files in ZFS with the space occupied in ext4.



```
dedupunit=1m,dedupratio=2
fsd=fsd1,anchor=/home/pramodh_billa/ext_pool,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
```



```
dedupunit=1m,dedupratio=2
fsd=fsd1,anchor=/zfs_pool,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. 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

5. After running them we got the following results:

a) ZFS:

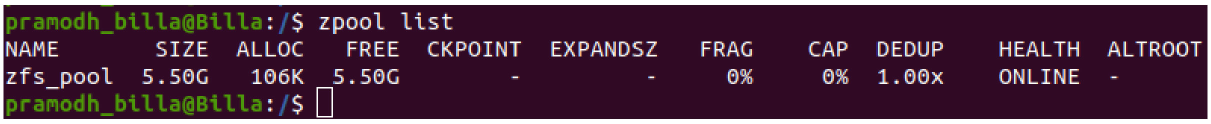
I. Initially, the empty ZFS folder had 0.106 MB of data.

II. After running the workload, the ZFS folder had 226 MB of data

III. We observed a deduplication ratio of 2.00x (which is what we wanted).

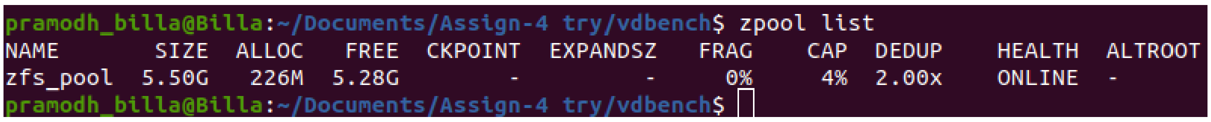
IV. This means that the new files took 226 MB of space. However, the intended space is 450MB (1MB*450). Hence, using the data deduplication feature, instead of maintaining whole blocks of data, when duplicates are found, ZFS simply makes a pointer point to the old data.

Before workload:



```
pramodh_billa@Billa:/$ zpool list
NAME        SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool   5.50G   106K  5.50G        -         -     0%     0%  1.00x    ONLINE  -
pramodh_billa@Billa:/$ 
```

After workload:



```
pramodh_billa@Billa:~/Documents/Assign-4 try/vdbench$ zpool list
NAME        SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs_pool   5.50G   226M  5.28G        -         -     0%     4%  2.00x    ONLINE  -
pramodh_billa@Billa:~/Documents/Assign-4 try/vdbench$ 
```

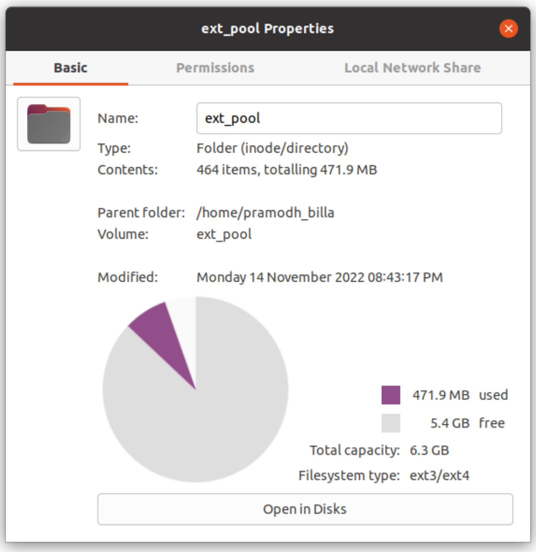
b) Ext4:

I. Initially the empty ext4 folder had 24.6 kB of data.

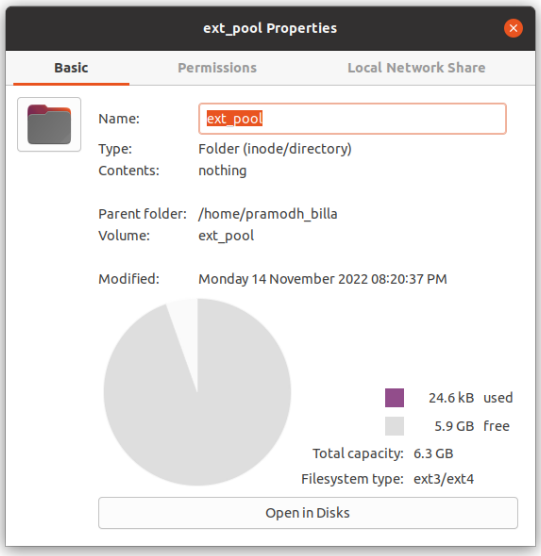
II. After running the workload, the ext4 folder had 471.9 MB of data.

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

After workload: Before workload:



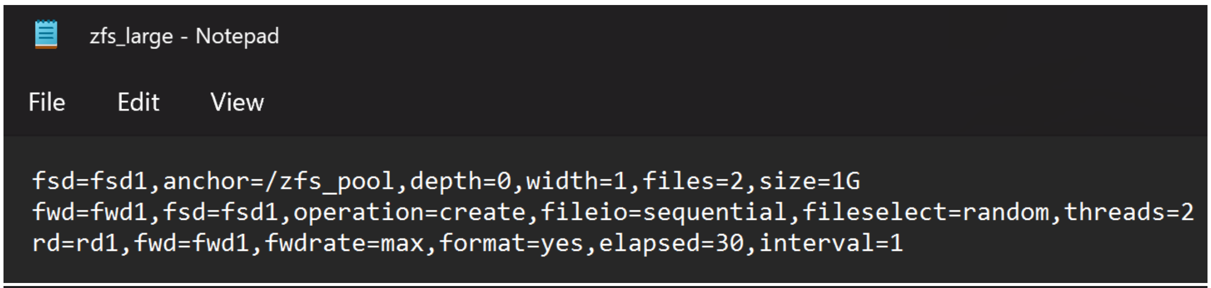
Name: ext_pool
Type: Folder (inode/directory)
Contents: 464 items, totalling 471.9 MB
Parent folder: /home/pramodh_billa
Volume: ext_pool
Modified: Monday 14 November 2022 08:43:17 PM
471.9 MB used
5.4 GB free
Total capacity: 6.3 GB
Filesystem type: ext3/ext4



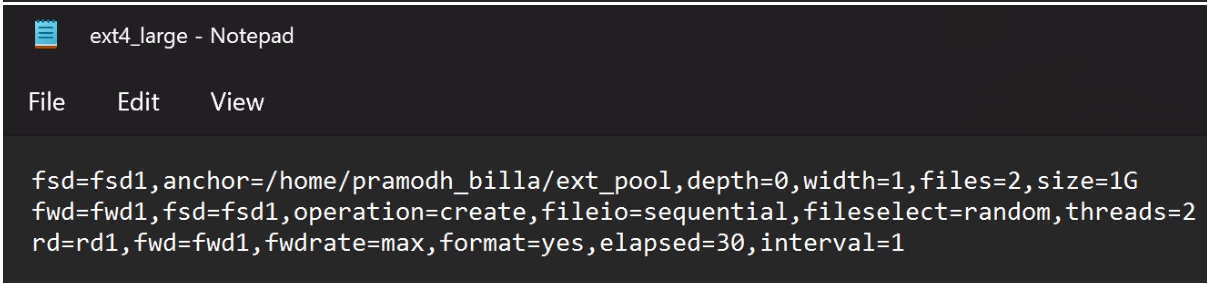
Name: ext_pool
Type: Folder (inode/directory)
Contents: nothing
Parent folder: /home/pramodh_billa
Volume: ext_pool
Modified: Monday 14 November 2022 08:20:37 PM
24.6 kB used
5.9 GB free
Total capacity: 6.3 GB
Filesystem type: ext3/ext4

Large File Creation:

1. We know that ext4 optimises large file creation better than ZFS does as we can clearly see using our workload
2. We created the following workload for testing large file creation



```
fsd=fsd1,anchor=/zfs_pool,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
```



```
fsd=fsd1,anchor=/home/pramodh_billa/ext_pool,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
```

3. 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
4. After running them we got the following results:
   a) Ext4:
      I. Finishes creating files in just 9 seconds.
      II. The average write rate is 197.53 MB/s

b) ZFS:

   I. Takes a whole 18 seconds to create the files
   II. The average write rate is 107.89 MB/s

Part 3:

Disadvantages of deduplication:

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 70.31 MB/s while ext4 had an average write speed of 168.06 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.

Disadvantages of Optimising Large File Creation:

Greater metadata overhead for small files: When running deduplication, only 450 MB was required by the files. But, additional used space after running workload1 was 471 MB (11 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).