CS 344 OPERATING SYSTEMS LABORATORY - 04

GROUP 19 ***Drishti Chouhan (180101021)*** ***Mridul Garg(180123028)***
Kousik Rajesh (180101094) ***Eklavya Jain (180123065)***

FILE SYSTEMS CONSIDERED:

1. *ZFS (with different configurations)*
2. *ext4*

COMPARISON OF FILE SYSTEMS:

ZFS	*ext4*
No optimization for large files	Optimizes large file creation and handling
Implements Deduplication	Does not implement data deduplication.
Data compression can be turned on	No compression by default
Provides high data integrity checks in the background	No such data integrity checks present
ZFS offers nearly unlimited capacity for data and metadata storage	ext4 can support a file size no larger than 18 terabytes
ZFS supports advanced file systems and can manage data long term	ext4 is not so flexible

FEATURES :

1. DATA DEDUPLICATION:

Data deduplication controls whether duplicate copies of data are eliminated. Deduplication is synchronous, pool-based, or block-based. Data written with deduplication enabled is entered into the deduplication table indexed by the data checksum. Subsequent writes will identify duplicate data and retain only the existing copy on disk. Deduplication only works between blocks of the same size. If our data doesn't create any duplicates, then data deduplication will only increase CPU overhead. However, if the data does contain duplicates, then this feature shall save space by storing only one copy of the data regardless of how many times it occurs.
ZFS provides block-level deduplication, using SHA256 hashing, and it maps naturally to ZFS's 256-bit block checksums. The deduplication is done inline, with ZFS.

In order to study the effects of deduplication, we took two instances of ZFS, one with dedup set to on and the second with dedup set to off. In order to implement this, we create a ZFS pool and alter the dedup setting as follows.

EXPERIMENT 1.1: *dedup=off*

Step 1: Switch the dedup off with the following command.

```
$sudo zfs set dedup=off zfs_pool
```

Step 2: Now, using vdbench, we create files that occupy 1.5 GB of space. We observe the free memory space available with ***dedup=off.*** Create the files with the following command:

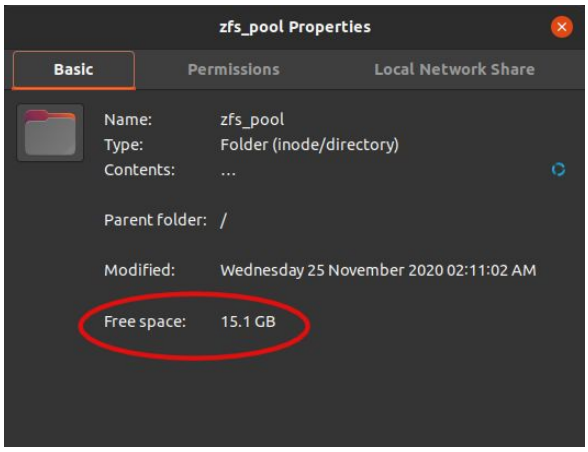
```
$sudo ./vdbench -f ./dedup_off -o output_off
```

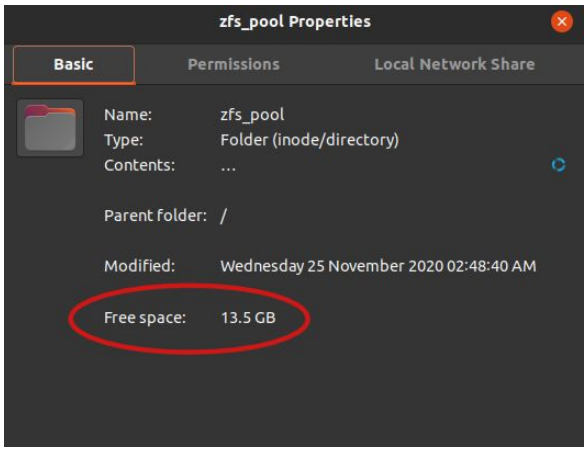
./dedup_off

```
dedupunit=128k
dedupratio=5

fsd=fsd1,anchor=/zfs_pool/dedup_off/,depth=1,width=30,files=2,size=25m

fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2

rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1
```

RESULT OF EXPERIMENT 1.1:

Before the creation of the files, available space in the memory was **15 GB**. And after the creation of files using vdbench, the available space **reduced to 13.5 GB,** as expected.





EXPERIMENT 1.2: *dedup=on*

Step 1: Switch the dedup on with the following command.

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

Step 2: Again we create files that occupy 1.5 GB of space and observe the free memory space available now.

Before the creation of the files, available space in the memory was **13.5 GB**. And **after the creation** of files using vdbench, the available space **reduced to 13.2 GB.** This happens because the ***dedup ratio has been set to 5***, and hence ***files actually occupy only ⅕ of the space.***

Run the following command to create the files:

```
$sudo ./vdbench -f ./dedup_on -o output_on
```

./dedup_on

```
dedupunit=128k
dedupratio=5

fsd=fsd1,anchor=/zfs_pool/dedup_on/,depth=1,width=30,files=2,size=25m

fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2

rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1
```

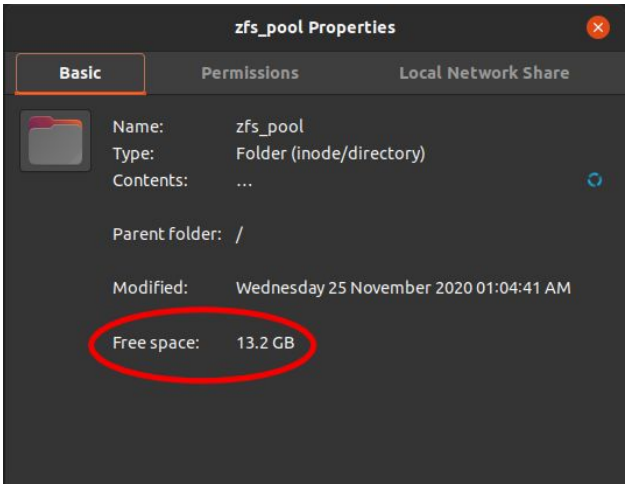
Explanation of the parameters that have been changed are :

1. ***anchor***: This specifies the directory where the files are created
2. ***depth***: This parameter specifies the depth of the tree structure that is created
3. ***width*:** number of immediate subdirectories in any one directory.
4. ***size***: For each file, the size is set as 25MB. Since 60 such files are created, a total space of **25x60 = 1500 MB ~ 1.5GB** is occupied by the files.

Dedup ratio is a way of expressing the amount of data reduction achieved by data deduplication. For example, a 20:1 deduplication ratio means that 20 units of logical data (gigabytes, terabytes, etc.) are stored in 1 unit of physical disk capacity.

RESULT OF EXPERIMENT 1.2:

- We observe that the available free space now is 13.2 GB (as opposed to 13.5 GB - 1.5 GB = 12 GB). This is because the second time, deduplication takes place and the space occupied is just 0.3 GB (one-fifth of 1.5 GB) instead of 1.5 GB.



2. COMPRESSION :

Controls whether data is compressed before being written to disk. This feature allows for greater storage utilization at the cost of increased CPU utilization. This allows for much greater storage utilization at the expense of increased CPU utilization. By default, no compression is done. If the compression does not yield a minimum space savings, it is not committed to disk to avoid unnecessary decompression when reading back the data. For the experiments in this assignment, we chose lz4 as our compression algorithm.
LZ4 only uses a dictionary-matching stage (LZ77), and unlike other common compression algorithms does not combine it with an entropy coding stage

LZ4 compresses approximately 50% faster than LZJB when operating on compressible data, and is over three times faster when operating on incompressible data. LZ4 also decompresses approximately 80% faster than earlier versions like LZJB.

ZFS supports file compression, and there are multiple compression algorithms available. We use the lz4 compression algorithm for comparison. We take two instances of ZFS and alter compression settings.

EXPERIMENT 2.1: *compression=off*

Step 1: The following command **sets the compression flag to off** in the ZFS pool:

```
$sudo zfs set compression=off  zfs_pool
```

Step 2: Now we create a 1 GB file, which occupies the exact same amount of space since compression has been turned off. Run the following command to create the files:

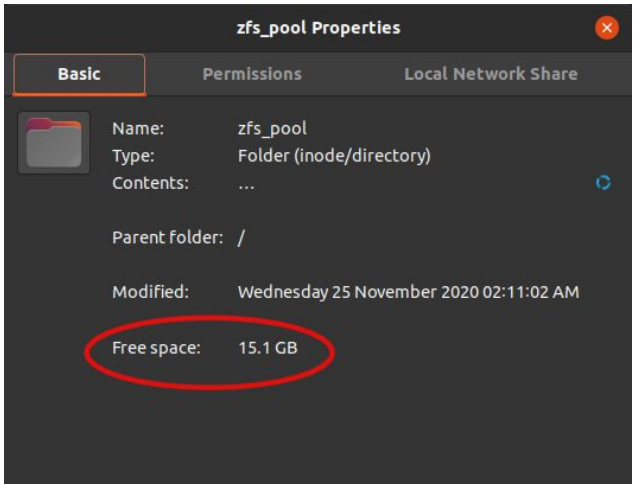
```
$sudo ./vdbench -f ./compression_off -o output_comp_off
```

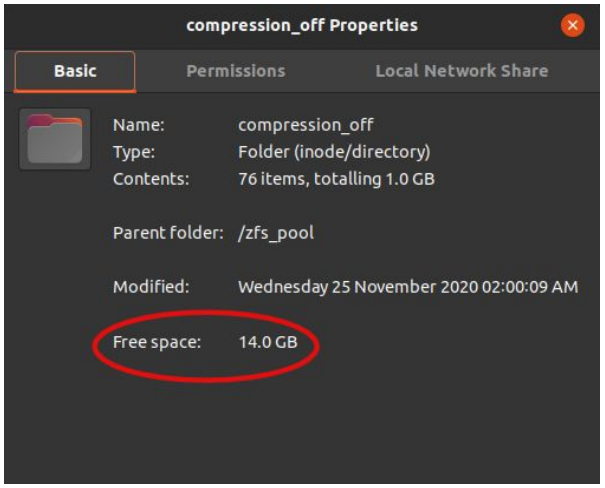
./compression_off

```
compratio=5

fsd=fsd1,anchor=/zfs_pool/compression_off/,depth=1,width=25,files=2,size=20m

fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2

rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1
```

RESULT OF EXPERIMENT 2.1

As expected, the available memory reduces by 1 GB (approx.).





EXPERIMENT 2.2: *compression=on*

Step 1: In order to use the lz4 compression algorithm in ZFS, run the following command:

```
$sudo zfs set compression=lz4 zfs_pool
```

Step 2: Now we create another file of 1GB, but since the compression flag is set to on, we observe that the file compresses by a factor of 5 and occupies only 0.2 GB (one-fifth of 1 GB). Run the following command to create the files:

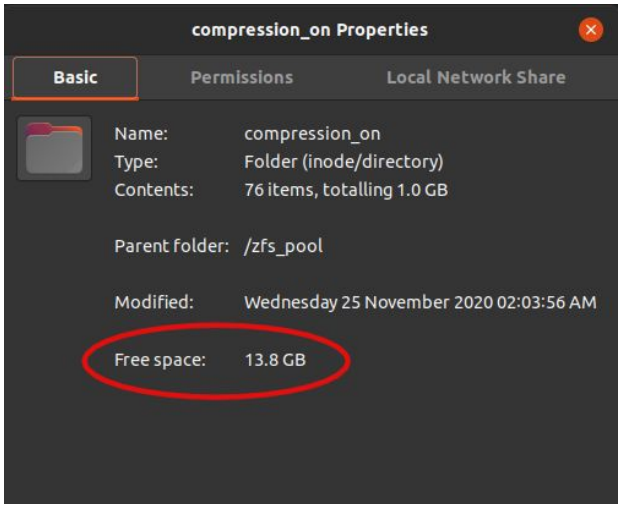
```
$sudo ./vdbench -f ./compression_on -o output_comp_on
```

./compression_on

```
compratio=5

fsd=fsd1,anchor=/zfs_pool/compression_on/,depth=1,width=25,files=2,size=20m

fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2

rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1
```

RESULT OF EXPERIMENT 2.2:

Final available memory is 13.8 GB instead of the expected 13 GB (14 GB - 1 GB = 13 GB). That is with compression set to lz4, the files occupy less space.



3. LARGE FILE CREATION

We provide comparison on an additional feature, ext4 optimises large file creation while zfs does not. We measure the time taken to create a large file (3.2 GB) in both zfs and ext4.

Ext4 uses extents (as opposed to the traditional block mapping scheme used by zfs), which improves performance when using large files and reduces metadata overhead for large files. In addition, ext4 also labels unallocated block groups and inode table sections accordingly, which allows them to be skipped during a file system check. This makes for quicker file system checks, which becomes more beneficial as the file system grows in size. This difference is visible in the file creation time in Ext4 and ZFS as shown.

EXPERIMENT 3.1: (ZFS)

Run the command to create the file:

```
$sudo ./vdbench -f ./large_file -o output_large_file
```

./large_file

```
fsd=fsd1,anchor=/zfs_pool/large_file/,depth=1,width=1,files=1,size=3g

fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2

rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1
```

Note: The flash drive was loaded with the ZFS file system. The depth, width, and files parameters are set to 1 because we want to create just one file.
The size of the large file is chosen as 3 GB.

```
11/25/2020-01:33:27-IST         Starting slaves
11/25/2020-01:33:27-IST         Slave localhost-0 (pid  90784) connected to master 90747
11/25/2020-01:33:27-IST         Slaves connected
11/25/2020-01:33:27-IST         Query host configuration started
11/25/2020-01:33:28-IST         Query host configuration completed
11/25/2020-01:33:29-IST         Starting        rd=format_for_rd1       For loops: None
11/25/2020-01:33:30-IST         Warmup done     rd=format_for_rd1       For loops: None
11/25/2020-01:44:32-IST         Workload done   rd=format_for_rd1       For loops: None
11/25/2020-01:44:32-IST         Slaves done     rd=format_for_rd1       For loops: None
11/25/2020-01:44:33-IST         Starting        rd=rd1  For loops: None
11/25/2020-01:44:34-IST         Warmup done     rd=rd1  For loops: None
11/25/2020-01:44:43-IST         Workload done   rd=rd1  For loops: None
11/25/2020-01:44:43-IST         Slaves done     rd=rd1  For loops: None
11/25/2020-01:44:43-IST         Shutting down slaves
11/25/2020-01:44:43-IST         Vdbench complete
```

RESULT OF EXPERIMENT 3.1:

Start time: 01:33:27
End time: 01:44:43
Time taken: 616 seconds

EXPERIMENT 3.2: (ext4)

Run the command to create the file:

```
$sudo ./vdbench -f ./large_file_ext4 -o output_large_file_ext4
```

./large_file_ext4

```
fsd=fsd1,anchor=/media/usb/large_file_ext4,depth=1,width=1,files=1,size=3g

fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=random,threads=2

rd=rd1,fwd=fwd1,fwdrate=100,format=yes,elapsed=10,interval=1
```

Note: The flash drive was loaded with the ext4 file system. The depth, width, and file parameters are set to 1 because we want to create just one file. The size of the large file is chosen as 3 GB.

RESULT OF EXPERIMENT 3.2:

Start time: 03:53:19
End time : 04:00:12
Time Taken: 413 seconds

```
11/25/2020-03:53:19-IST         Starting slaves
11/25/2020-03:53:20-IST         Slave localhost-0 (pid 120625) connected to master 120586
11/25/2020-03:53:20-IST         Slaves connected
11/25/2020-03:53:20-IST         Query host configuration started
11/25/2020-03:53:20-IST         Query host configuration completed
11/25/2020-03:53:22-IST         Starting        rd=format_for_rd1       For loops: None
11/25/2020-03:53:23-IST         Warmup done     rd=format_for_rd1       For loops: None
11/25/2020-04:00:01-IST         Workload done   rd=format_for_rd1       For loops: None
11/25/2020-04:00:01-IST         Slaves done     rd=format_for_rd1       For loops: None
11/25/2020-04:00:02-IST         Starting        rd=rd1  For loops: None
11/25/2020-04:00:03-IST         Warmup done     rd=rd1  For loops: None
11/25/2020-04:00:12-IST         Workload done   rd=rd1  For loops: None
11/25/2020-04:00:12-IST         Slaves done     rd=rd1  For loops: None
11/25/2020-04:00:12-IST         Shutting down slaves
11/25/2020-04:00:12-IST         Vdbench complete
```

We observe that the time taken by ext4 to create a large file is much less than the time taken by ZFS to create a file of the same size. Hence, the ext4 file system optimizes the creation and handling of large files as compared to the ZFS file system.

Also, on comparing the two output folders, we observe that the throughput of ext4 is greater than the throughput of ZFS.

Label (double click on row)	IOPS	Resp	Resp max	Resp stddev	MB/sec
output_large_file	32.115	31.093	69,070.761	1,316.214	4
output_large_file_ext4	42.585	23.442	3,428.042	58.590	5

Ext4 is 25% faster than ZFS when creating a large file, this is because of the use of extents instead of a block mapping scheme which provides a smaller file overhead and a significant speedup when the size of file increases

FILE SYSTEM	***FILE CREATION TIME (sec)***	***THROUGHPUT (MB/sec)***
ZFS	616	4
EXT4	**413**	**5**
