OPERATING SYSTEM LAB ASSIGNMENT-4
GROUP-35

SAURABH BARANWAL, Roll no: 180101072
MOHAN KUMAR, Roll no: 180101042
RAHUL KRISHNA, Roll no: 180123035
PANKAJ KUMAR, Roll no: 180123031

This report consists of two parts: **Experimental Part + Theoretical analysis of another File System (ext4)**

First, we'll try to look into the experimental part of the Assignment. For the experimental part, we'll be analysing both features (deduplication and compression) within ZFS only with respective configurations being turned ON or OFF.

In the second part, we would aim to learn how to install ext4, how respective features are implemented in it and how it compares with ZFS. We would only layer it out theoretically.

So for this purpose, we first need to learn how to install ZFS on our system.

INSTALLATION OF ZFS

(Complete installation guide can be found at this link: https://ubuntu.com/tutorials/setup-zfs-storage-pool#3-creating-a-zfs-pool)

The main components of ZFS are maintained as a standard Ubuntu package, so to install simply run: `sudo apt install zfsutils-linux`
After this, type following command to check if zfs was correctly installed: `whereis zfs`

Check installed drives by running: `sudo fdisk -l`
Note the device names of drives you want to pool. For this purpose in our Virtual Box, we added a virtual hard disk which was stored by the name **/dev/sdb**

Now, a pool needs to be created. A stripped pool, with the name **new-pool** is created by following command:

```
sudo zpool create new-pool /dev/sdb
```

Finally, to check the pool status you can type command: `sudo zpool status`

FEATURE 1: DEDUPLICATION

For experimentation purposes, we're trying to analyse observations corresponding to two different configurations of ZFS: ***dedup on*** and ***dedup off***

IMPLEMENTATION DETAILS OF DEDUPLICATION IN ZFS

Data deduplication ensures that duplicate copies of data are eliminated. It occurs when writing to the disk is done. Dedup, in ZFS, is implemented using checksums. Data written with deduplication enabled is entered into the data structure deduplication table (DDT) indexed by the data checksum. Deduplication forces the use of the cryptographically strong SHA-256 checksum. Subsequent writes will identify duplicate data (by comparing checksums of respective blocks) and if they are the same, it is considered as a duplicate and only a pointer to the original block is written to disk without actually writing that block. It happens only between blocks of the same size and data written with the same record size.

WORKLOAD AND OBSERVATIONS

We write the following workload with file name **dedup_zfs** for vdbench:

```
dedupratio=3
dedupunit=128k
fsd=fsd1,anchor=/new-pool,depth=2,width=2,files=10,size=128k
fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=ran
dom,threads=2
rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=10,interval=1
```

De-dup ratio to be 3 means that the rough estimate of reduction of disk-space, if dedup is on, should be one third. Dedupunit=128k means that deduplications compares 128kb of data at once. The third line in the above snippet, is the code that creates some files in the anchor directory. The anchor directory should be specified to be the one corresponding to ZFS (for us, it is /new-pool as we had created a ZSF pool with name new-pool). The rest of the parameters give a measure of total files that will be created. In total, 40 files will be created with size of each file be 128Kb. The fourth line describes the workload, where the the operation is defined to be read and two threads are used for this purpose. File selection is done randomly and then the chosen file (size: 128 kb) is read sequentially in 4k blocks (xfersize= 4k)
The RD parameter 'format=yes' causes the directory structure to be completely created (after deleting any existing structure), including initialization of all files to the requested size of 128k. After the format completes it runs for 10 seconds at a rate of 100 reads per second.

Dedup configuration is made on by typing the following command on Terminal:

```
sudo zfs set dedup=on new-pool
```

After this, we run the workload using the following command inside vdbench directory:

```
sudo ./vdbench -f dedup_zfs
```

The above command runs the workload, and populates the anchor directory with respective files. After this, the following command is run to check the space allocated in zpool new-pool;
zpool list

The output came out to be as follows:

```
saurabh@saurabh-VirtualBox:~/Desktop/Assignment-4/vdbench$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH
  ALTROOT
new-pool  6.50G  1.91M  6.50G        -         -     0%     0%  3.07x    ONLINE
  -
```

The experimental dedup ratio came out to be 3.07 (which is pretty close to the theoretical value set to be 3). Further, the ALLOC column value 1.91 M indicates the total file allocation when dedup is ON,

Similarly, we make the dedup configuration off by following command:

```
sudo zfs set dedup=off new-pool
```

Again, first the same workload is run. And then we run command: **zfpool list** to see the following result when dedup configuration is off.

```
saurabh@saurabh-VirtualBox:~/Desktop/Assignment-4/vdbench$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH
  ALTROOT
new-pool  6.50G  5.46M  6.49G        -         -     0%     0%  1.00x    ONLINE
  -
```

Experimental dedup ratio comes out to be 1.00x (as dedup was off). Also, the total space allocated is 5.46 M

KEY OBSERVATION AND COMPARISON:

1. Space allocation when dedup was off is 5.46M while when dedup is on, is 1.91 M. This is a significant observation that disk space usage decreases when de-duplication occurs. Hence, ZFS's deduplication performance in terms of disk usage becomes more efficient when deduplication is on
2. When dedup is ON, the dedup ratio is 3.07 which is significantly equal tothe theoretical value 3 set. This proves the experimental success of our result.

DISADVANTAGE OF DEDUPLICATION

De-duplication leads to more CPU consumption. This is because of two factors. One, it involves extra business for checksum calculation. This increases latency on storage-write operations. Second, once the checksum has been calculated; extra CPU clock-cycles are required for lookup in DDT(De-duplication table).

It can be quantified by looking at **summary.html** files of each case in the corresponding output

folders. **Output_dedup_on** corresponds to output folder of case dedup on whereas **ouput_dedup_off** corresponds to output folder of case dedup off. The average CPU usage in both cases is noted from summary.html files in both folders and come out as follows:

Total average CPU Usage when dedup is on: 68.7%

Total average CPU Usage when dedup is off: 53.8%

Clearly, when deduplication is enabled; average CPU usage is more.

FEATURE 2: COMPRESSION

For experimentation purposes, we're trying to analyse observations corresponding to two different configurations of ZFS: ***compression off*** and ***compression on (by using lz4 algorithm)***

IMPLEMENTATION OF COMPRESSION IN ZFS

When data is stored using less disk space, it is called compression of data. Compression of data, in ZFS, can be achieved through various algorithms (which are specified as a parameter). These algorithms are:

1. **gzip:** Standard UNIX compression
2. **gzip-N:** selects a specific gzip level. `gzip-1` provides the fastest gzip compression. `gzip-9` provides the best data compression. `gzip-6` is the default.
3. **lz4-** provides better compression with lower CPU overhead
4. **lzjb:** optimized for performance while providing decent compression
5. **Zle:** zero length encoding is useful for datasets with large blocks of zeros

For our experimentational purpose, we have used lz4 algorithm. Let us describe it in some more detail. The LZ4 algorithm represents the data as a series of sequences. Each sequence begins with a one byte token that is broken into two 4 bit fields. The first field represents the number of literal bytes that are to be copied to the output. The second field represents the number of bytes to copy from the already decoded output buffer (with 0 representing the minimum match length of 4 bytes). A value of 15 in either of the bitfields indicates that the length is larger and there is an extra byte of data that is to be added to the length. A value of 255 in these extra bytes indicates that yet another byte to be added. Hence arbitrary lengths are represented by a series of extra bytes containing the value 255. After the string of literals comes the token and any extra bytes needed to indicate string length. This is followed by an offset that indicates how far back in the output buffer to begin copying. The extra bytes (if any) of the match-length come at the end of the sequence.

WORKLOAD AND OBSERVATIONS

We write the following workload with file name **compression_zfs** for vdbench

```
compratio=2
fsd=fsd1,anchor=/new-pool,depth=2,width=2,files=10,size=128k
fwd=fwd1,fsd=fsd1,operation=read,xfersize=4k,fileio=sequential,fileselect=ran
```

```
dom,threads=2
rd=rd1,fwd=fwd1,fwdrate=max,format=yes,elapsed=10,interval=1
```

Compratio refers to the compression ratio that we want (we have set it to 2 in our case).The rest of the code for file creation, reading and formatting remains the same as previous case. So we avoid explaining the same here, again.

NOTE: In file **compression_zfs,** you will have to change the anchor folder corresponding to ZFS in your local system. For my system, ZFS pool that I created was mounted at this anchor folder location: **/new-pool** hence I have specified the same there.You will have to change the same accordingly to your local file system.

COMPRESSION OFF

Compression is made off for ZFS first by typing the following command on terminal:

```
sudo zfs set compression=off new-pool
```

After this the workload is run using vdbench. The following command is typed for this:

```
sudo ./vdbench -f compression_zfs
```

After the workload running has completed, you need to run 2 commands to make two key observations.
First: **zpool list**
The output is as below:

When there is no compression, the ALLOC column corresponds to memory 5.45M. This means space allocation without any compression (as it was off).

Second, **zfs get-compressratio new-pool**
(Note: new-pool is the name of my ZFS pool. You might have created with some other name, enter that)
We get the following output:

This effectively signifies that for now, compression ratio is 1.00X (which means, compression is off).

COMPRESSION ON:

Compression is now made on, by typing the following command (**Note that we specify lz4 as the value of parameter compression, to employ lz4 algorithm as previously described)**

```
sudo zfs set compression=lz4 new-pool
```

After the compression has set to be on using the lz4 algorithm, now we run the workload using the same command previously. After that we run again the two commands which we did earlier for observations. The results corresponding to case of compression ON are as follows:

zpool list

```
saurabh@saurabh-VirtualBox:~/Desktop/Assignment-4/vdbench$ zpool list
NAME       SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH
  ALTROOT
new-pool  6.50G  3.00M  6.50G        -         -     0%     0%  1.00x    ONLINE
  -
```

ALLOC column field value is 3.00M which is the space allocation value when compression was ON and was done using the lz4 algorithm.

zfs get compressratio new-pool

```
saurabh@saurabh-VirtualBox:~/Desktop/Assignment-4/vdbench$ zfs get compressratio new-pool
NAME      PROPERTY       VALUE  SOURCE
new-pool  compressratio  1.99x  -
```

The experimental compression ratio comes out to be 1.99X which is very close to the theoretical value of 2X that we had set in the workload.

KEY OBSERVATION AND COMPARISON

1. The space allocation, when compression was ON using lz4 algorithm, was lesser than the case when compression was OFF. This is quite obvious because space usage is lower when compression is turned on.
2. When compression is ON, we found that the experimental value of compression ratio (1.99X) was very close to the theoretical value of compression ratio (2X) we had set in the workload. Further, when compression was OFF, compression ratio is found to be 1.00X which is as expected (as no compression takes place).

DISADVANTAGE OF COMPRESSION

CPU cost is higher when compression is on because extra CPU utilisation is required for implementing the respective compression algorithm and reducing the original data size. The observations can be quantified by the respective vdbench workload results.

In folders **output_comp_lz4** and **output_comp_off,** we have file **summary.html** that contains the summary of statistics for cases when lz4 compression is used and when compression is off respectively. The results for CPU utilisation is as follows:

Total average CPU Usage when compression is on: 72.7%

Total average CPU Usage when compression is off: 68.4%

Another disadvantage, of compression, is that even though the space allocation is reduced; the quality of data may be reduced.

PART-2 : THEORETICAL ANALYSIS OF FILE SYSTEM EXT4 (AND ITS COMPARISON WITH ZFS ON CERTAIN FEATURES)

Ext4 Installation

(Read for more info: https://www.tutorialspoint.com/how-to-create-a-new-ext4-file-system-in-linux)

Linux created its original extended file system (ext) as early as 1992. It was the first to use a virtual file system (VFS) switch, which allowed Linux to support many file systems at the same time on the same system. ext4 comes embedded on Linux.

Add a new ext4 file system in linux

Step1: add a new partition

```
sudo fdisk /dev/sda
      Command (m for help): l
```

Step2: we choose the n option to create a new partition.

```
      Command (m for help): n
```

Step3: Now we issue the below command to make the sda5 partition as an ext4 partition.

```
sudo mkfs.ext4 /dev/sda5
```

Let us now analyse the two features we took for experimentation, with respect to ZFS.

DATA DEDUPLICATION

Ext4 doesn't support De-Duplication.

We can however suggest a mechanism to implement de-duplication within ext4 (**Source Research paper:** https://www.ijert.org/research/deploying-de-duplication-on-ext4-file-system-IJERTV3IS11159.pdf)

Data Deduplication is a specific form of compression where redundant data is eliminated, typically to improve storage utilization, so all the duplicate data is deleted, leaving a single copy of data which is referenced by all the files containing it.

In File level deduplication, duplicate files are eliminated and replaced by a pointer to the original file. **Block Level Deduplication** makes file level deduplication finer by dividing each file into clocks or chunks. If any of these blocks are repeated in memory, they will be removed and replaced by a pointer. **Byte level deduplication** is even more refined than block level but it requires far more processing. The data stream is analysed byte by byte

and any recurring bytes are simply given references to the original byte instead of storing it again.

System Architecture:

Data Deduplication consists of several sub processes such as chunking, fingerprinting, indexing and storing. For managing these processes, there are logical components.

1. User Interface This is the operating system interface with the help of which the user reads, writes, modifies or deletes files.
2. File Manager The File Manager manages namespace, so that file name in each namespace is unique. It also manages metadata.
3. Deduplication Task Manager Before deduplication can be done, we need to divide the file into chunks and then ID each chunk using a hash function. Next, these ID‟s or fingerprints are stored in a table. Each time we call the deduplication task manager, we check in a table, known as a Hash Store, if our current fingerprint matches any of the previously stored fingerprints. The Hash store consists of all of the fingerprints which are stored in it right from the first block which is stored in memory.
4. Store Manager Deduplication exploits three different storage components, File Store, Hash Store and Chunk Store. These stores are managed and updated by the Store Manager.

General Working:
Once we have obtained the data stream being modified, Deduplication Task Manager intervenes and takes over control of the data stream, instead of directly writing it to memory.
This data stream is separated into chunks or blocks of equal size and using a hash function, we find the hash value or fingerprint. Next, check if this fingerprint is present in the hash store. If we find a match then already a block having the same contents is present in memory and we do not need to write the same block again. We must simply store a reference to this block in the File Store. However, if a match is not found in the Hash Store, then we need to write this block to memory, update this fingerprint in the Hash Store and give a reference of this block in the File Store in the appropriate position.

COMPRESSION

Ext4 doesn’t support data Compression.

Ext4 and family reserve an attribute for compression but don't implement it. This feature was originally put off because there were more urgent things to do, and then it disappeared as the size of storage media increased a lot faster than the size of data that isn't already compressed. Most large files today (videos, music, even word processor documents) are already compressed.
Compression can still make sense for medium-sized files. Performance-wise, it's a trade-off: it costs more CPU time but less I/O time.

How to run previous workloads for ext4?
Simply change the location of the anchor directory (corresponding to ext4 partition) in the code. Same workloads can be run then.

The ext4 file system supports persistent pre-allocation—useful for applications like streaming media where sequential access performance is paramount. It also supports delayed allocation. The journals are checksummed for improved reliability, and there is a multiblock allocator. File system checking is significantly faster.

Key Comparison between ZFS and ext4:

	ZFS	ext4
Data Deduplication	Support Data Deduplication feature	Doesn't support
Compression	if compression is on in ZFS it may work faster than ext4	however ext4 doesn't support compression feature
Allowable characters of directory entries	Unicode except NULL	byte except NULL
Maximum volume size and applications	256.000.000.000.000.000 ZiB It is more useful for large enterprises with large data	1 EiB More faster file system than ZFS for low max-vol size
CheckSum (Data Integrity)	Support by default	Not supported by default
Encryption	Yes	Yes, experimental
Persistent Cache	Yes	Not supported
Sequential I/O and Random Seek	generally lower than ext4	generally greater than ZFS
