CS_344 Assignment 4 Report

Group C21: Akshat Mittal, 200101011
Satvik Tiwari, 200101091
Pranjal Baranwal, 200101083

Part A: File systems and implementation details

We will be comparing data deduplication and large files creation in the ZFS and EXT4 file systems for our experiment.

An introduction to the two filesystems:

ZFS (Zettabyte File System)

- ZFS is a local file system and logical volume manager created by Sun Microsystems Inc. as a part of Solaris OS to direct and control the placement, storage and retrieval of data in enterprise class computing systems.
- It is characterized by data integrity, high scalability and built-in storage features designed to run on a single server, potentially with hundreds if not thousands of attached storage drives.
- ZFS pools the available storage and manages all disks as a single entity. A user can add more storage drives to the pool if the file system needs additional capacity.
- It is highly scalable and supports a large maximum file size.

EXT4 (Fourth Extended File System)

- The EXT4 filesystem primarily improves performance, reliability and capacity. To improve reliability, metadata and journal checksums were added. To meet various mission-critical requirements, the filesystem timestamps were improved with the addition of intervals down to nanoseconds.
- In EXT4, data allocation was changed from fixed blocks to extents. 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.
- EXT4 reduces fragmentation by scattering newly created files across the disk so that they are not bunched up in one location at the beginning of the disk, as many early PC filesystems did.
- It also makes use of delayed allocation which helps improve the performance as well as helps the file system allocate contiguous blocks of memory.

Brief description of the features we have selected for the experiment:

Data Deduplication

- It is a specialized data compression technique for eliminating duplicate copies of repeating data used to improve storage utilization and can also be applied to network data transfers to reduce the number of bytes that must be sent.
- In the deduplication process, unique chunks of data or byte patterns are identified and stored during a process of analysis. As the analysis continues, other chunks are compared to the stored copy and whenever a match occurs, the redundant chunk is replaced with a small reference that points to the stored chunk.

- Given that the same byte pattern may occur dozens, hundreds or even thousands of times, the amount of data that must be stored or transferred can be greatly reduced.
- 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 Files Creation

- The EXT4 file system supports a maximum volume of 1 EiB (ExbiByte)=2^{60} Bytes and a maximum file size of 16 TiB (TebiByte)=2^{44} Bytes with the standard 4KiB blocks with 48 bits 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 does not 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 B: Experiment and observations

We first installed vdbench on Ubuntu virtual machine. **Refer Readme.pdf for complete steps**.

Vdbench has three basic definitions in the configuration file:

1. Filesystem Definition

FSD defines the filesystem storage used for the testing. The FSD name should be unique. FSD parameters include:

a. fsd=name: Unique name for this File System Definition.
b. anchor=/dir: The name of the directory where the directory structure will be created.
c. depth=nn: How many levels of directories to create under the anchor.
d. files=nn: How many files to create in the lowest level of directories.
e. width=nn: How many directories to create in each new directory.
f. sizes=(nn, nn, …): Specifies the size(s) of the files that will be created.

2. Filesystem Workload Definition

FWD defines the filesystem workload used for the testing. The FWD name should be unique. FWD parameters include:

a. fwd=name: Unique name for this Filesystem Workload Definition.
b. fsd=(xx, …): Name(s) of Filesystem Definitions to use.
c. operation=xxx: Specifies a single file system operation that must be done for this workload.

d. fileio=xxx: How file I/O will be done; random or sequential.
e. fileselect=xxx: How to select file names or directory names for processing; random or sequential.
f. threads=nn: How many concurrent threads to run for this workload.
g. xfersize=(nn, …): Specifies the data transfer size(s) to use for read and write operations.

3. Run Definition

RD defines what storage and workload will be run together and for how long. Each run definition name must be unique. RD parameters include:

a. fwd=(xx, yy, …): Name(s) of Filesystem Workload Definitions to use.
b. format=yes/no: Causes the directory structure to be completely created, including initialization of all files to the requested size.
c. elapsed=nn: How long to run in seconds.
d. interval=nn: Statistics collection interval in seconds.
e. fwdrate=nn: Number of operations per second.

Experiment 1: Data Deduplication

We created two partitions of ZFS file system. The names used are *zfs-dedup-on* and *zfs-dedup-off*. Refer Readme.pdf for complete steps.
In this experiment we will be comparing ZFS (with deduplication on) and ZFS (with deduplication off) for the below mentioned workload (*dedup-workload*).

```
dedupunit=1m,dedupratio=2
fsd=fsd1,anchor=$anchor,depth=2,width=5,files=20,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
```

Here, we are creating 500 files (5*5*20) of size 1MB each in a nested directory structure of depth 2 and width 5. dedupunit is set to 1MB and dedupratio to 2.
dedupratio is the ratio of the total number of blocks (of size dedupunit) to 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.

Now, we run this workload by setting correct anchor point.

1. ZFS (with dedup on)

Turn on data deduplication of *zfs-dedup-on* pool using:

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

Inside vdbench directory, run the following command:

```
$sudo ./vdbench -f dedup-workload anchor=/zfs-dedup-on
```

All generated output files can be found inside outputs/zfs-dedup-on folder in the submission.

2. ZFS (with dedup off)

Turn off data deduplication of *zfs-dedup-off* pool using:

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

Inside vdbench directory, run the following command:

```
$sudo ./vdbench -f dedup-workload anchor=/zfs-dedup-off
```

All generated output files can be found inside outputs/zfs-dedup-off folder in the submission.

Memory usage of both pools can be seen using: `$zpool list`

Before Workload:

```
akshat@akshat-VirtualBox:~/vdbench$ zpool list
NAME            SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs-dedup-off  9.50G   130K  9.50G        -         -     0%     0%  1.00x    ONLINE  -
zfs-dedup-on   9.50G   165K  9.50G        -         -     0%     0%  1.00x    ONLINE  -
```

After Workload:

```
akshat@akshat-VirtualBox:~/vdbench$ zpool list
NAME            SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
zfs-dedup-off  9.50G   501M  9.01G        -         -     0%     5%  1.00x    ONLINE  -
zfs-dedup-on   9.50G   252M  9.25G        -         -     0%     2%  1.99x    ONLINE  -
```

In above experiment, we can observe that *zfs-dedup-off* required around 501MB of memory while *zfs-dedup-on* required just 252MB to store the files which is total of 500MB (500*1MB).

We observe a deduplication ratio of 1.99x (which is very close to what we expected, i.e., 2.00x).

A little overhead is observed to store metadata and pointers. Hence, using the data deduplication feature, instead of maintaining whole blocks of data, in case of duplicates, ZFS simply makes a pointer point to old data.

Another interesting observation is the CPU utilization. It can be observed that data deduplication is a more CPU intensive task.

Experiment 2: Large Files Creation

We created two partitions, one of each ZFS and EXT4 file system. The names used are *zfs-pool* and *ext4-pool* respectively. Refer Readme.pdf for complete steps.
In this experiment we will be comparing ZFS and EXT4 file systems for testing large files creation with below mentioned workload (*largefiles-workload*).

```
fsd=fsd1,anchor=$anchor,depth=2,width=2,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
```

Here, we are creating eight files (2^2*2) of size 1GB each in a directory structure of depth 2 and width 2.
We run this workload on both file systems using the below command:

```
$sudo ./vdbench -f largefiles-workload anchor=/zfs-pool
$sudo ./vdbench -f largefiles-workload anchor=/ext4-pool
```

All generated output files can be found inside outputs/zfs-largefiles and outputs/ext4-largefiles folders in the submission respectively.

We found while running the workloads that zfs took **29 seconds** to create files with an average write rate of **279.1 MB/sec** whereas ext4 finishes the task in just **17 seconds** with average write rate of **487.7 MB/sec**.

So, for writing same amount of data, ext4 takes less time than zfs and thus ext4 has better throughput.

We can say ext4 is equipped to create large files and handle them efficiently.

Note: These observations are taken from summary.html file in output folders.

Part C: Conclusions

Disadvantages of data deduplication:

1. Performance: The ZFS system takes more time to set up the filesystem if deduplication is on. If the data does not contain any duplicates, enabling deduplication will add overhead without providing any benefits.
   In the large file creation also, ext4 was giving better performance in terms of speed. The average write speed is faster in case of ext4 file system.
2. CPU Utilization: CPU utilization is more when the deduplication is on at the time of files creation due to deduplication overhead. For *zfs-dedup-on*, utilization is 71.4% while for *zfs-dedup-off*, utilization is only 23.7%.
   In the large file creation also ZFS has more CPU utilization than the ext4 file system.

Disadvantages of large files creation:

1. Greater metadata overhead for small files: If we run first workload *(dedup-workload)* on ext4, we observed that large amount of space was being used for overhead metadata. A lot of additional space is used in maintaining the extent trees compared to actual data for small files.
   But in zfs, we saw only additional 1MB was being used for the same workload.
2. No possible recovery from corruption: Ext4 optimizes large files creation by using delayed and contiguous allocation 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. There is no possible recovery from corruption.