OPERATING SYSTEMS LAB

Understanding Filesystems

Sumith (140050081)
Shubham Goel (140050086)

Theoretical background

We pick a feature, **data deduplication** in our case, and analyze it for several benefits, in terms of lower disk usage, better performance, and so on. In this lab, we will consider two filesystems that differ in this feature, and understand the pros and cons of having that feature.

Feature of interest

In computing, **data deduplication** is a specialized data compression technique for eliminating duplicate copies of repeating data. Related and somewhat synonymous terms are intelligent (data) compression and single-instance (data) storage. This technique is 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 match frequency is dependent on the chunk size), the amount of data that must be stored or transferred can be greatly reduced [1].

Description of implementation

Data can be deduplicated at the level of files, blocks, or bytes.

File-level assigns a hash signature to an entire file. File-level dedup has the lowest overhead when the natural granularity of data duplication is whole files, but it also has significant limitations: any change to any block in the file requires recomputing the

checksum of the whole file, which means that if even one block changes, any space savings is lost because the two versions of the file are no longer identical.

Block-level dedup has somewhat higher overhead than file-level dedup when whole files are duplicated, but unlike file-level dedup, it handles block-level data such as virtual machine images extremely well. With block-level dedup, only the blocks that are unique to each file consume additional storage space. All other blocks are shared.

Byte-level dedup is in principle the most general, but it is also the most costly because the dedup code must compute 'anchor points' to determine where the regions of duplicated vs. unique data begin and end.

ZFS provides block-level deduplication because this is the finest granularity that makes sense for a general-purpose storage system[2].

Experimental evaluation

The experiments were done on an Ubuntu 14.04 system. ZFS on FUSE was installed via the Ubuntu Software Center that gives access to *zfs* and *zpool* commands. We use an external disk (16 GB pendrive) to provide the unallocated space for the ZFS filesystem.

Setting up

The following commands were used to setup:

```
sumith1896@yoda:~$ sudo zpool create zfs1 /dev/sdb
sumith1896@yoda:~$ sudo zfs set dedup=on zfs1
sumith1896@yoda:~$ sudo zpool list
NAME   SIZE  ALLOC   FREECAP  DEDUP  HEALTH  ALTROOT
zfs1        15G  1.65M  15.0G             0%  1.00x  ONLINE  -
sumith1896@yoda:~$ df /zfs1/
Filesystem           1K-blocks  Used Available Use% Mounted on
zfs1        15482829           21  15482808   1% /zfs1
```

Note the memory usage of the disk.

Workload chosen

We use 500 2MB files (total size 1GB) created for the previous labs. Each file contains the same repeated character to make up the 2MB. This directory is copied to the ZFS drive and appropriate metrics are measured.

Metrics noted

We observe the memory usage before and after copying these files with two settings of dedup off and on respectively. We also note the CPU usage during the copying process using the *iostat* command. The results of this have been submitted as `on.txt` and `off.txt` for dedup on and off respectively.

Experiments run

With dedup on and after the files were copied:

```
sumith1896@yoda:~$ sudo zpool list
NAME   SIZE  ALLOC   FREE       CAP  DEDUP  HEALTH  ALTROOT
zfs1                            15G  1.53M  15.0G     0%  2639.00x  ONLINE  -
sumith1896@yoda:~$ df /zfs1/
Filesystem                      1K-blocks    Used Available Use% Mounted on
zfs1                            16490559 1025091  15465468   7% /zfs1
```

After these files were deleted:

```
sumith1896@yoda:~$ sudo zpool list
NAME   SIZE  ALLOC   FREE       CAP  DEDUP  HEALTH  ALTROOT
zfs1                            15G  1.65M  15.0G     0%  1.00x  ONLINE  -
sumith1896@yoda:~$ df /zfs1/
Filesystem                      1K-blocks  Used Available Use% Mounted on
zfs1                            15482817    21  15482796   1% /zfs1
```

Setting dedup off for the next experiment:

```
sumith1896@yoda:~$ sudo zfs set dedup=off zfs1
```

After the files were copied:

```
sumith1896@yoda:~$ sudo zpool list
NAME   SIZE  ALLOC   FREE       CAP  DEDUP  HEALTH  ALTROOT
zfs1                             15G   987M  14.0G     6%  1.00x  ONLINE  -
sumith1896@yoda:~$ df /zfs1/
Filesystem                       1K-blocks    Used Available Use% Mounted on
zfs1                             15482802 1010774  14472028   7% /zfs1
```

Observation

Memory: With dedup on we observe that the time for copying is less and we also observe that the actual space taken is much less than the logical space needed. In contrast with dedup off where is the copying time is much more as actually disk space needs to be provided even for the duplicated data and we can observe that the actual space taken is comparable to the logical space needed.

CPU: If we have a look at the CPU usage (comparing the `on.txt` and `off.txt` files), we can observe that the copying with dedup on is much more CPU intensive and dedup off is relatively much lesser. This also hints the fact that in scenarios where there are no duplicate data we may better off using dedup off than dedup on. We present more views in the upcoming section.

Conclusion

Is it all good?

It all depends on your data.

If your data doesn't contain any duplicates, enabling dedup will add overhead without providing any benefit. If your data does contain duplicates, enabling dedup will both save space and increase performance. The space savings are obvious; the performance improvement is due to the elimination of disk writes when storing duplicate data, plus the reduced memory footprint due to many applications sharing the same pages of memory.

Most storage environments contain a mix of data that is mostly unique and data that is mostly replicated. ZFS deduplication is per-dataset, which means you can selectively enable dedup only where it is likely to help. For example, suppose you have a storage pool containing different dedup settings[2].

References

[1] https://en.wikipedia.org/wiki/Data_deduplication
[2] https://blogs.oracle.com/bonwick/entry/zfs_dedup