Deploying De-Duplication on Ext4 File System

Usha A. Joglekar[1], Bhushan M. Jagtap[2], Koninika B. Patil[3,]
1. Asst. Prof., 2, 3 Students

Department of Computer Engineering
Smt. Kashibai Navale College of Engineering
Pune, India

Abstract

Memory space has become one of the most important and costly resources today in both personal as well as corporate areas. Amount of data to be stored is increasing rapidly as number of people using computing devices has increased. For effectively improving memory utilization, Data Deduplication has come under the spotlight. In deduplication, a single unique copy of data is stored in memory and all other copies refer to the address of this original data. The urgent need now is to make data deduplication more mainstream and integrate it with file systems of popular operating systems using inline deduplication. Our focus is on EXT4 File System for Linux, as Linux is one of the most widely used open source operating system. Also, since bit level deduplication lays severe overhead on the processor, chunk level deduplication is the most suitable for our task.

1. Introduction

Data storage techniques have gained attention lately due to exponential growth of digital data. Not just data, but backup and distributed databases with separate local memories give rise to several copies of the same data. Large databases not only consume a lot of memory, but they can lead to unnecessarily extended periods of downtime. It becomes an enormous task to handle the database; and backup and restoration also bogs down the CPU.

Since we cannot compromise on backup space, we must make sure that there are no duplicate files in the data itself. Prevention of further transmission of multiple copies of data is necessary for which we need data deduplication.

For example, a typical email server containing hundred instances of the same file of size about 1MB. Every time we take back up of the email platform, all hundred instances have to be saved again, requiring 100MB more of storage space.

Data Deduplication is a specific form of compression where redundant data is eliminated, typically to improve storage utilization[4] so, all the duplicate data is deleted, leaving a single copy of data which is referenced by all the files containing it.

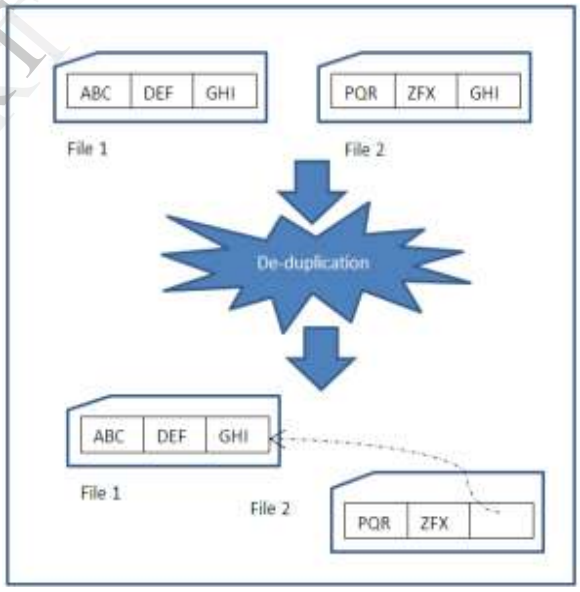

Figure 1.Deduplication concept

Figure 1,shows two files, File 1 and File 2, which contain one repeating block of data. A block containing GHI is present in both files. So, the deduplication process replaces the GHI block in File 2 with a pointer to the same block in File 1.

Data deduplication is not the same as data compression. In data compression, a function is applied for replacing a larger, frequently repeating pattern with a smaller pattern. Even though storage space is lessened, it will still contain redundant patterns. Data

deduplication on the other hand, searches for redundant patterns and saves only one instance of it.

2. Classification of Deduplication

Deduplication is a vast topic and can be classified in various ways as discussed below.

2.1. Level of Deduplication

Deduplication can take place at various levels depending on the granularity of the operation. We can have very high level deduplication in the form of File Level deduplication, or extremely fine Deduplication at very low level in the form of Byte Level deduplication. An intermediate between the two is block level deduplication.

In File level deduplication, duplicate files are eliminated and replaced by a pointer to the original file. It is also known as Single Instance storage.
It can be achieved by comparing each file to all the other files present in memory.

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. It is also known as sub file level deduplication.

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.

2.2. Post Process and Inline Deduplication

Post process deduplication carries out deduplication after the data has already been stored in memory. Here the need for hash calculations is removed, but when the memory is almost full, it may lead to problems as duplicate data may be present until deduplication is done.

Before the data is written to disk, we check whether it is already present or not. Thus we ensure that duplicate data is never present in disk and we are not wasting precious memory space.

2.3. Source and Target Deduplication

Source Deduplication refers to when deduplication is carried out at source itself. Before sending the data to the destination we ensure that no duplicate or repeating data is present in the data. On the other hand, in Target deduplication, we carry out deduplication in the same place where we finally store the data, that is, in the destination.

3. System Architecture

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

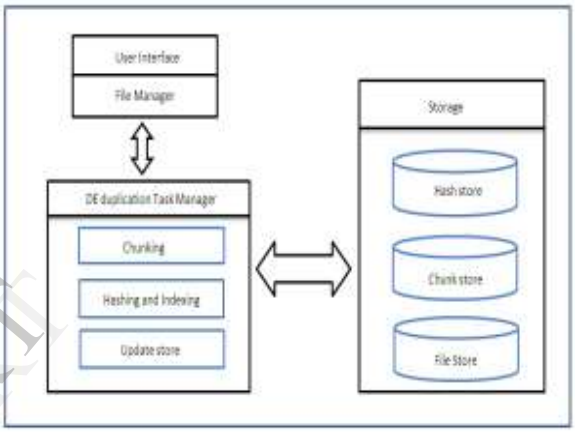

Figure 2.System Architecture

3.1. User Interface

This is the operating system interface with the help of which the user reads, writes, modifies or deletes files.

3.2. File Manager

The File Manager manages namespace, so that file name in each namespace is unique. It also manages metadata.

3.3. 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 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.

3.4. 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.

4. General Working

Whenever a user wants to create, modify or delete any file, we need to invoke inline deduplication process.

Since we are using target deduplication, we carry out deduplication exactly where the data is being store, in memory.

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.

5. Chunking

We can carry out deduplication at three levels as we have seen, that is, File Level, Block Level and Bit Level. Block level deduplication is more employable in the practical sense. The more important discussion is about the size of each block and whether it should be of fixed or variable size. Generally block size is taken to be 512 bits or a multiple of 512 as conventional hash algorithms work to this tune. As the data stream enters, we can mark the boundaries at the end of every 512 bits so that from the next bit, the next block of 512 bits will start. We can thus segregate the data stream into distinct well defined blocks.

There are two ways in which we can chunk the data, by Fixed Size Chunking (FSC) or Content Defined Chunking (CDC). Fixed Size Chunking is simpler; the data stream is divided into equal sized chunks of data. But any changes made in these chunks will cause all of the following chunks to shift by that number of bits, due to the 'avalanche' effect. So even if a previous match in the blocks was found, even a single bit change will prevent matching of a 99% similar block. On the other hand, if we use CDC, which sets the boundaries of the chunk according to how much of the data coincides, dynamically, we will spend a lot of time and processing power as the complexity of the operation increases greatly.

As shown in Figure 3(b), a data stream ABCDEF is divided into two blocks of equal sizes, ABC and DEF. When we add Z after C, in FSC, the hash value of the entire sequence will change. Even if DEF is present in the following sequence, we will not be able to identify it as a duplicate block. If we use VSF as shown in Figure 3(a), we can allocate a separate block to 'Z' and DEF will remain a separate block so that hash value will remain same and deduplication of data will be identified.

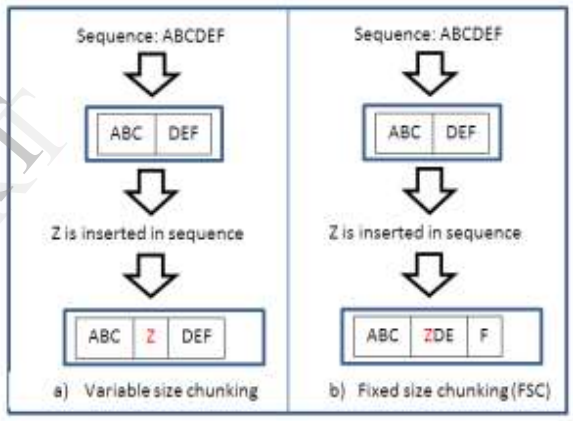

Figure 3 .Problems in fixed size chunking (FSC)

Using a sliding window of 512 bits, we will have to shift it bit by bit for the entire data stream.

6. Hashing

Rather than handling the entire original sequence, we compress each chunk using a hash function giving us a much smaller, handy fingerprint. It helps make comparison and storage simpler. We can use any conventional hash function or algorithm such as MD5 or SHA for creating the fingerprint.

The MD5 algorithm is capable of reducing a string of 512 bits to 128 bits. Since the compressed fingerprint is much smaller than the original, chances are that two different strings may give the exact same fingerprint. This is known as collision and in MD5, the probability of collision is 2^{-64}.SHA1 is more secure in this regard than MD5. It reduces the original string of 512 bits to 160 bits, therefore reducing the probability

of collision to 2^{-80}, which is much less than that of MD5. However, MD5 gives better throughput in terms of system performance and is faster and costs less than SHA, just about a third.

MD5 and SHA algorithms consist of many rounds and steps of encryption making it complicated so as to reduce the possibility of a collision occurring. Different functions of logical, arithmetic and shift operations are performed on the block of data.

Both MD5 as well as SHA1 suffer from certain drawbacks as mentioned above and the way to go forward would be to design a deduplication specific Hash Algorithm which will combine the benefits of both SHA1 and MD5 and will be more suitable for our work.

7. Indexing

Once we have created fingerprints for all of the chunks, we can check if our current chunk is already present in memory or not.
First of all let us understand what storage structures or 'stores' are being used.

7.1 Chunk store

The actual unique data blocks are present in the chunk store. These chunks may be a part of many different files and only one copy is maintained which is referenced by all the files containing it.

7.2 Hash Store

Hash store contains the compressed fingerprints of the corresponding chunks stored in the Hash Store. It is these fingerprints which are compared when searching for a duplicate data block.

7.3 File Store

The File Store is a logical table consisting of all the files. Each file is broken into chunks. A reference to each chunk is stored in the file store. In the Linux file system, it is known as the inode table.

Indexing deals with the searching of hash store for a matching fingerprint.

8. Update stores

As said above, the stores need to be updated whenever a modification is made to any file. If a match is found in the Hash store, then only the File Store needs to be updated.

If the matching Hash is not found then this new fingerprint is added in the Hash Store, the new block is written to the memory and its address is stored in the appropriate place in the File Store.

9. Future Scope

One of the major drawbacks of Data Deduplication is that a significant amount of time is lost in chunking and fingerprinting, as well as indexing. This degrades system performance.

For this purpose, we can use pipelining and parallelism, especially in multicore systems[1][2]. We can use parallel processing using the multiple available cores simultaneously. Since the chunking and hashing are independent of each other. Even the chunking and hashing of adjacent blocks is not interdependent. Hence all of these operations can be conducted concurrently. The results of all these can be combined at the Indexing stage, where we require the results of all the stages serially so that we do not miss out on the blocks which we even carry out parallely. If efficient use of parallelism is made then it will be easier to integrate deduplication on the computer as it will reduce runtime overheads and prevent the processor from diverting all of its resources to deduplication process. All popular Operating Systems should employ deduplication process before the actual writing of data can take place so as to make best use of memory.

Also, the hashing function should be specific to deduplication. We can combine advantages of existing hashing algorithms to make a new one which gives high performance as well as minimizes the risk of collision.

Thus deduplication has a lot of scope in future and should be converted into a basic functionality available and inbuilt in all new age systems.

10. Conclusion

Data deduplication is more of a necessity than a mere tool to make maximum usage of memory. It is simple to implement and it's only drawback, of consuming too much of processing power, can be overcome with the help of parallelism and pipelining. All operating systems should have inbuilt facility for deduplication, in all file systems.

11. Mathematical model

We now provide a model of the system in terms of the Set Theory

1. Let 'S' be the de-duplication system
S = {. . .
2. Identify the inputs F,M where F is File and M is Metadata
S = {F, M. . .
F = {i | 'i' is collection of row data}
M = {t |'t' is a collection of information, i.e. inodes of the file}
3. The output is identified as O
S = {F, M, O. . .
O = {'O'|O is metadata i.e. inode structure of file}
4. The processor is identified as P
S = {F, M, O . . .
P = {S1, S2, S3, S4}
5. S1 is the set of chunking module activity and associated data
S1 = {$S1_{in}$, $S1_p$, $S1_{op}$}
$S1_{in}$= {f |'f' is valid stream of bytes}
$S1_p$= {f |'f' is chunking function to convert $S1_{in}$ to $S1_{op}$}
$S1_p$ ($S1_{in}$) = $S1_{op}$
$S1_{op}$ = {t |'t' is output generated by chunking algorithm i.e. chunks of data}
6. S2 is the set for fingerprinting activities and associated data
S2 = {$S2_{in}$, $S2_p$, $S2_{op}$}
$S2_{in}$= {t |'t' is valid chunk of data}
$S2_p$= {t |'t' is the set of functions for fingerprinting}
$S2_p$ ($S2_{in}$) = $S2_{op}$
$S2_{op}$ = {t |'t' is finger print of the $S2_{in1}$}
7. S3 is the set of data modules for searching the hash store and related activities
S3 = {$S3_{in1}$, $S3_{in2}$, $S3_p$, $S3_{op}$}
$S3_{in1}$ = {t |'t' is valid fingerprint of chunk}
$S3_{in2}$= {t |'t' is hash store}
$S3_p$ = {t |'t' is set of functions for searching fingerprints in hash store}
$S3_p$ ($S3_{in}$) = $S3_{op}$
$S3_{op}$ = {t |'t' is valid fingerprint indicating presence or absence of chunk}
8. S4 is the set of modules and activities to update the store
S4 = {$S4_{in}$, $S4_p$, $S4_{op}$}
$S4_{in}$ = {(t1, t2, t3)} | 't1' is the valid chunk value, t2 is the fingerprint of t1, t3 isthe search flag associated with t1}
$S4_p$ = {t |'t' is the set of functions for updating chunk, hash and file metadataStore}
$S4_p$ ($S4_{in}$) = $S4_{op}$
$S4o_p$= {t|'t' is the metadata, i.e. inode of updated cluster}
9. Identify the failure cases as F'
S = {F, M, O, F'. . .
Failure occurs when
F' = { Φ }
F' = {p | 'p' is collision of fingerprints}
F' = {p | 'p' does not have fingerprint in hash store}
10. Identify success cases (terminating case) as 'e'
S = {F, M, O, F', E. . .
Success is defined as
E = {p |'p', i.e. successfullyupdated hash, chunk, file store}
11. Identified initial condition as S0
S= {F, M, O, F', E, S0}
Initial condition for deduplication is there should be redundant data present inChunk store and respective fingerprint of chunks must be present in hash store, i.e., $S2_{in2} \neq \{\Phi\}$ if chunk store is not null.

12. References

[1] Keren Jin, Ethan L. Miller,"The Effectiveness of De-duplication on Virtual Machine Disk Images",*SYSTOR 2009*, Haifa, Israel, May 2009.
[2] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Zhongtao Wang,"P-Dedupe: ExploitingParallelism in DataDe-duplication System",*IEEE Seventh International Conference on Networking, Architecture, and Storage*. 2012
[3] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: large scale, inline deduplication usingsampling and locality," in *Proccedings of the 7th conference on Fileand storage technologies*. USENIX Association, 2009, pp. 111–123.
[4] http://en.wikipedia.org/wiki/Data_de-duplication [Online; accessed 28- September -2013]
[5] http://www.herongyang.com/Cryptography/index.html[Online;accessed 5- October -2013]
[6] http://moinakg.wordpress.com/tag/pcompress[Online; accessed 5- October -2013]
[7] http://blog.mcpc.com/?Tag=data%20deduplication [Online; accessed 5- October -2013]