An Efficient and Responsive Architecture for Data De-duplication Using MD5 Hash

Seemant Kumar Sharma, Saurabh Sharma

Gyan Ganga Collage of Technology, Jabalpur, India(M.P)

Abstract - In this digital world of internet, data storage and server storage use is often and every home user, enterprises, several organizations are using email and online storage as a storing node. Online backup storage is an easy option for everyone to store digital data, files and other multimedia files. This makes the storage servers loaded as well as more disk storage is required to save a large amount of same data. Due to the same reason the search operation takes more time to found a specific file and time taken to give positive result and acknowledgement is increased. This problem was overcome with a mechanism known as Data de-duplication. This process is used for removing duplicate data and to reduce redundancy at server node. In this paper we have studied previous and recent work on de-duplication and proposed a solution which is a Parallel architecture for inline data de-duplication which uses the secure hash algorithm 256 for performing data de-duplication task in order to overcome the issues of time and to reduce hash collision. In this architecture write and delete operations are performed for efficiency and time evaluation. This architecture is useful for storage servers where a huge amount is stored every day and software industries always looks for new developments so that they can keep their storage systems up to date and free for efficient utilization of the server nodes.

Keywords - Data De-duplication, Data Architecture MD5 Hash.

I. INTRODUCTION

Nowadays, online backup storage, content delivery networks, blog sharing, news broadcasting and social networks as an ascendant part of Internet services are data centric. Hundreds of millions of users of these services generate petabytes of new data every day. For instance, as of April 2011, an online file-sharing and backup services called drop boxes, has more than 25 million 2GB drop boxes (total of 50 petabytes). A large portion of internet service data is redundant for the following reasons. One is, now a days people tend to save data at multiple times for data safety reasons and avoids purchasing storage for high cost; this leads to more redundant data. One another reason is, while incremental (or differential) data backups or disk image files for virtual desktop tend not to have duplicated whole-file copies, but still there is large ratio of duplicated data portion from the modifications and revisions of the files.

Rapidly increasing data arises many challenges to the exist-ing storage systems. The large amount of data requires more storage medium to be used [1]. As the data increases, more data is for backup cite. Due to increment in storage data it is found that it brings some difficulties in backup systems. The cost of the storage media has decreased, but the main problem is to manage number of disks in the back-up systems. In fact, in storage archives a large amount of data is redundant and slight changed to another chunk of data. The identification of these duplicate chunks is fundamental to improve the quality of information retrieval [10].

In de-duplication the redundant data is deleted by using the cryptographic hash concept .In backup servers hash is used for finding the duplicate data. Hash is a fixed length representation of any arbitrary length message. The complexity of comparisons can be reduced by using hash as the original length of data is much more than the hash size. In de-duplication process whenever any record comes for server, it calculates the hash signature for the record using secure hash algorithm (SHA). Once hash signature is generated server checks this signature in hash index, which is already maintained in the system. While searching for the signature in hash index if the server finds its entry in the hash index (record already exists) then rather storing it again server creates a reference for this. This reference will point to the location of block on the disk. In second case if server does not find the entry of record in hash index table it will store the record on the disk and adds an entry for its hash signature in hash index.

A. Classification of data de-duplication

Generally de-duplication methods consists of two main approaches for data de-duplication storage systems: finger-printing based and delta-based data de-duplication. Nowa-days, fingerprinting-based de-duplication is prevalent in prac-tice and research and this thesis deals exclusively with this type. The baseline algorithm for fingerprinting-based data de-duplication, shown in Figure below is the sequence of chunking, duplicate detection and storage: A Broad view of de-duplication classification is presented in Figure 1.

B. De-duplication based on processing position

De-duplication can occur where data is created, which is referred as source de-duplication". De-duplication performed at the place where data is saved or stored is

called"target de-duplication" [1].
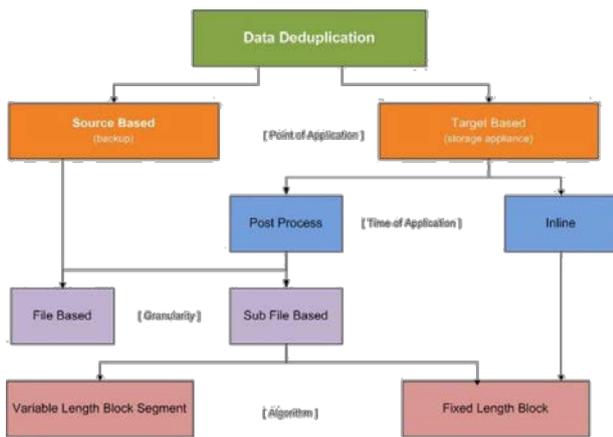
Fig. 1: Classification of Data De-Duplication Strategies

1) source and target de-duplication: When describing de-duplication for backup architectures, it is common to hear two terms: source de-duplication and target de-duplication.Source de-duplication ensures that data on the data source is de-duplicated. This generally takes place directly within a file-system. The file system will periodically scan new files creating hashes and compare them to hashes of existing files. When files with same hashes are found then the file copy is removed and the new file points to the old file. Unlike hard links however, duplicated files are considered to be separate entities and if one of the duplicated files is later modified, then using a system called Copy-on-write a copy of that file or changed block is created. The de-duplication process is transparent to the users and backup applications. Backing up a de-duplicated file system will often cause duplication to occur resulting in the backups being bigger than the source data.

Target de-duplication(De-duplication Appliance-based) is the process of removing duplicates of data in the secondary storage. Generally this will be a backup store such as a data repository or a virtual tape library. In the target de-duplication, when the data is coming to store, we apply post process or in-line data de-duplication depends on our needs at the target side [2] [3].

C. De-duplication based on time of processing

Data de-duplication can be broadly classified into two types based on time of operation processing, De-duplication can occur "In-line", when data is in flowing condition or "Post-process" ,in which it has been written.

1) Offline de-duplication(post-process deduplication or asynchronous) : With post-process de-duplication, de-duplication analysis and calculations are made after the data is stored in storage device. Once the data is stored then only the process will be applicable. A benefit of using post process is no one need to wait for hash based

calculations. The lookup is completed before storing the data also ensuring about performance degradation not achieved. On the negative side of this process,one may unnecessarily save redundant data for a small time which could be an important issue if the system is near to full capacity.

2) In-line de-duplication(synchronous): This is the process where the de-duplication hash calculations are created on the target device. When the data enters the in the device,if the device spots a that block that is already stored on the system. It does not store the new data block and just references to the existing block. The benefit of in-line de-duplication is that it requires less amount of storage. On the other side, because of hash calculations as well as lookups takes long time, the data ingestion may be slower. Due to which throughput of the device is reduced.

D. De-duplication based on storage position

There are major two types of data de-duplication based on storage are known as primary data de-duplication and secondary data de-duplication.

1. Data se-duplication method in which the data retain in primary storage is used for de-duplication mechanism is called "primary de-duplication". Primary storage is the storage which contains currently used data, which is easily accessible by the central processing unit on computer system. This is generally a data with small capacity. Due to so much frequently change in active or primary data this method is not so much applied.

2. Data de-duplication method in which secondary storage is used for de-duplication process is called "secondary de-duplication". Secondary data is the data which is not so much used and stored from a long time on a system. This kind of data is usually not directly attached to CPU and not directly accessible too. Due to a permanent behaviour of secondary storage it is used more than primary de-duplication method.

E. Hash Based De-duplication

In hash based data de-duplication process we use cryptographic hash to detect redundant copy of any record. In the general process storage server maintains a hash table, which contain two fields. One is hash signature and other is its real address. It calculates the hash signature for each record requesting for backup by using secure hash algorithm. Now it search for this hash signature in hash table. If signature not found, that means record is unique, and do an entry for this in hash table.

F. Levels of de-duplication

Data de-duplication technology is used to identify duplicate data, eliminate redundancy and reduce the need to transfer or store the data in the overall capacity. The redundancy may occur within a file, in a specific block of data and in a specific byte of data. At present mainly three levels of data de-duplication are classified as

1) File level de-duplication

2) Block-level de-duplication

3) Byte-level de-duplication

1) Whole File Hashing (or File-level De-duplication) : In the file level de-duplication, entire file is assumed as a record. When a file comes to back up, we compare the hash signature of incoming file to already stored files hash signatures. If the file is already stored, we store a reference to it otherwise store entire file and make an entry for hash signature of this file in the hash table.

2) Block-level De-duplication (or Sub File Hashing): In the block-level data de-duplication method, incoming data stream is divided into various data blocks and compared with the hash of data block. Then it determine whether it is same as with the previously stored data block (use hash algorithm for each data block to form a digital signature or unique identifier). If the hash of the data block is unique, then store this block to disk, and store its identifier in the hash index; otherwise, only store the reference to the same data block's original location. It stores a reference of a comparatively small size in place of the data block, rather than storing duplicate data blocks again, hence a significant saving of disk storage space. Hash algorithm used to judge duplicate data, may lead to conflict between the hash signatures, so we use SHA-256 algorithm for generating hash signatures because it generates 256 bit of hash signature and can create different hash signature for 2^{160} blocks of the data. There are two types of Block Level de-duplication-

Fixed size block level de-duplication Fixed Block de-duplication involves determining a block size and segmenting files/data into those block sizes. Then, those blocks are what are stored in the storage subsystem. For example suppose we take a fixed size 1 byte to divide an incoming file.

Variable size block level de-duplication Variable Block

de-duplication involves using algorithms to determine a variable block size. The data is split based on the algorithm's determination. Then, those blocks are stored in the subsystem.

3) Byte-level De-duplication: In byte stream level data is in another way . In this, the incoming data stream is divide into the number of bytes and then the hash signature of each incoming bytes are compared with the stored bytes on the disk and take appropriate action (Refer fig). Byte level de-duplication gives highest accuracy as compared to file level de-duplication and block level de-duplication. But byte level de-duplication lead to many problems, which are as follows

1) Size of the hash table will become very large.

2) It may lead to large file fragmentation.

3) Finally, byte level de-duplication will lead to performance degradation.

G. Advantages of De-duplication

Data de-duplication provides higher data reduction ratio from 10 to 1 to 50 to 1. Storage amount can be reduced because less space is required as there is no redundant data is present, it leads to fewer disk and less frequent purchases of disk. The less amount of data will help to utilize small backups and it will increase the overall recovery time[6]. The main advantage of data de-duplication algorithm is that the service can be provided in data centers to more number of users with the same amount of available resources. With the help of data reduction resulted by de-duplication, disk management is much easier as well as it decreases the overall cost of managing any buying storage cost. An efficient de-duplication algorithm is time saving which means service is provided to many users with lesser amount of time at same storage value. Apart from these benefits de-duplication also saves network bandwidth as only fewer data is in use. The main features of de-duplication are

Provides variable and fixed Block De-duplication Efficient Storage Utilization)

Scalability

High Availability

II. BACKGROUND AND RELATED WORK

The research of data de-duplication presently focuses on different aspects. Effectiveness of data reduction is one of them, that is, in order to reduce the storage capacity requirement, re-move the duplicate data as much as possible. Data reduction is definitely an important parameter for better data de-duplication architectures. Another aspect is the efficiency of data de-duplication, i.e.to achieve the effectiveness of algorithm what amount of resources are required. Many researchers worked in the field of data de-duplication previously and resulted with different methods for better efficiency.

While surveying the recent methods and advancements we can see that most available backup systems uses file-level de-duplication [4] traditionally. But the data de-duplication technology can exploit inter-file and intra-file information redundancy to eliminate duplicate or similarity data at the granularity of file, block or byte. Some of the available architecture follows the source de-duplication approach and provide the de-duplication technology in the available users file system [5]. Because of this file system de-duplication, user faces delay in sending data to backup store, and the rest of the available architectures which support target de-duplication strategy provide single system de-duplication which means at the target side only single system or Server handles all the user requests to store data and maintains the hash index for the number of disks attached to it [2].

Name of some previously proposed architectures are VENTI, LBFS(lower bandwidth file system) [5],MAD2, SIS (single instance store), CDC(Contenet defind chunking) [6], INS(Index Name Server) and PASTICHE. VENTI and Single Instance S adopt fixed-size file dividing method to partition the file into blocks [7] [8].

LBFS and PASTICHE divide each file into variable sized blocks [5] [9]. Fixed-size file dividing method is simple and easy,but the salient disadvantage is that all the blocks after the change point will be affected, and then misjudged as non-duplicate blocks.

Zhu ET use the Summary Vector, an in-memory, conser-vative summary of the segment index, to reduce the number of times that the system goes to disk to look for a duplicate segment only to find that none exists. Then they use Stream-Informed Segment Layout (SISL) to create spatial locality and to enable Locality Preserved Caching (LPC) to prefetch hash codes of adjacent segments into cache. LPC method avoids disk operation and accelerates the process of identifying duplicate segments [10] [2]. some researchers worked in the field of cloud storage and worked with using both fixed size block and variable size blocks. As there are a lot of de-duplication techniques depending on the algorithms chunking of the data blocks. In paper, they chosen Fixed Block [1]and Rabins Fingerprint [11] which are the most well known algorithms as the representatives. Fixed Block algorithm uses fixed size block as a unit of the de-duplication while Rabins Fingerprint uses variable block size.

Tin-Yu Wu, Wei-Tsong Lee, Chia Fan Lin2 proposes a new data management structure named Index Name Server (INS), which integrates data de-duplication with nodes optimization mechanisms for cloud storage performance enhancement. INS manages and optimize the nodes

according to the client-side transmission conditions. Sengar and Mishra [2] proposed a very scalable and efficient in-line data de-duplication using SHA-160. This algorithm supports bloom filter to reduce the disk access time for segments which are not present in the Disk.It support load balancing in storage nodes.

A problem with the available architectures is that the hash algorithm may lead to hash collision, that is, different blocks produce the same hash codes, which will result in discarding unique block mistakenly. However, LBFS [5], fingerdiff [12] used hash algorithm (SHA-1 or MD5), and most of them considered that the probability of hash collision is extremely lower than the probability of hardware errors. In our archi-tecture we selected SHA-256 hash algorithm because of its strong collision resistant and encryption function. "A Parallel Architecture for In-Line Data De-Duplication Using SHA-256 Hash" is our goal. The proposed architecture uses the hash index for redundancy identification between files so it should fulfill some other features-

Use of upgraded hash algorithm leads to lesser probability of hash collision as SHA-256 Provide hash signature upto 2^{128} bytes.

Parallel implementation helps reducing time consumption and shows better performance for larger file sizes.

Space reclaiming with use of reference count mechanism. To decrease the communication overhead it should sup-port better interaction between storage node and server.

III. PROPOSED DE-DUPLICATION ARCHITECTURE

Our proposed architecture for inline data de-duplication is given in (Refer fig 2) Our proposed parallel architecture with SHA-256 algorithm,is using following concepts given below-

A. Client

The node that contains or need back up for data is client. When client require to store any data , it sends that data to server node.

B. Server

After a client request a file to backup, server first receives that file at backup store and after accepting the file, server di-vides it in fixed size blocks(example 1024 KB) and group these divided blocks into super block and these super blocks are distributed among nodes of available storage using strategies of data distribution. Now storage nodes and server create hash signature of

distributed parts and a sequential search of hash signature is performed . This search is performed in parallel way at the maintained hash table.

C. Meta Server

All the database tables are collected at meta data server. The meta data tables contain the file name, number of parts in each file,parts path information,number of references to each file part,hash signature of each files part and storage utilization of each storage node information.
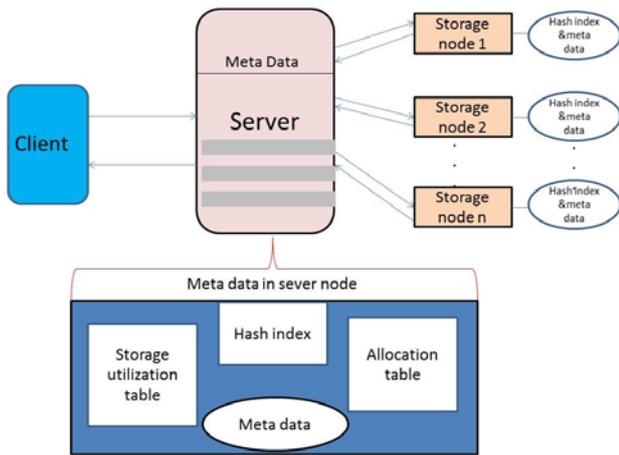
Fig. 2: Proposed Parallel architecture

D. Storage Server

Every storage server has its hash table and information related to it. Storage node first receives data for backup and performs hash calculation of data block and compares this with its hash table which is maintained by server itself. The data is stored in disk storage if hash signature found unique. If its signature is found similar then it only store the reference.

IV. PROPOSED PROCESS

Our proposed architecture is executed with some module and nodes are taken as components. Important components of this architecture are client node and server nodes. By using these nodes such parallel architecture is implemented and performance study is obtained using the operations as write, delete and read The main components of this architecture are client, server, and storage nodes. The operations which we have implemented in this architecture are read, write and delete. Fourth operation is nothing which leads to exiting from system. An schema for these operations are discussed in following sections.

A. Descriptions of operations

Write operation algorithm At the server

1) Client sends the file or record to backup.

2) Server receives this requested file and numbers of blocks are calculated by server itself.

3) These number of blocks are divided depends on the block size allotted. For an example if a file of 7 KB is requested for backup and block size is decided as 1 KB , then the number of blocks for the file will be 7.

4) In allocation table these blocks are entered.

5) Adds an entry in allocation table.

At the storage client

1) Storage client receives the super block from server and proceed to step 2.

2) Calculate hash signature for all available blocks in the super block.

3) Determine if any of these or all of those blocks are already stored or not. In case if already stored , then go for referencing it; otherwise entire block should be stored in disk.

Delete operation algorithm

1) Client sends the file or record it wants to delete.

2) Server takes this request and send the name of the file requested to module of metadata and to all storage nodes. This both sending process are done simultaneously.

3) Meta data module on server now decrease the number of references and every storage node also decreases the number of references , corresponding to that file.

4) Data block is deleted if the number of reference value is zero for that data block.

5) Once deletion is performed, server acknowledged to client with a message of success.

Read operation algorithm

1) Client requests to server to read file or record with sending file name.

2) Server then sends that file name to metadata module. In meta data allocation table each record is main-tained.

3) Server gets the information about super block

i.e. which storage node contains which super block.

4) Send the read request to these storage nodes.

5) Storage clients accepts this request from server and send the requested super block to server.

6) Finally server replies to client with received parts from server.

7) B. Technologies used

Java programming language is used for the implementation of this architecture. Java sockets are used for connection of network. for implementing the above mentioned architecture. We are using java sockets for implementing connection of network. This implementation od connection network is be-tween client and server. Java programming database is used for establishment of connection between mysql server and java. Different java classes are also used for different modules. Mysql server is used for maintaining the metadata and their tables. These tables contains the data about hash index, number of references, allocation, file part meta data.

V. EXPERIMENTS

The main aim of our proposed architecture is to provide a system or platform by which can remove duplicate data from the data centers in parallel and allow load sharing. Proper distribution of all incoming load is done using storage balancing technique. This architecture also uses distribution of requested data and server using different storage nodes.
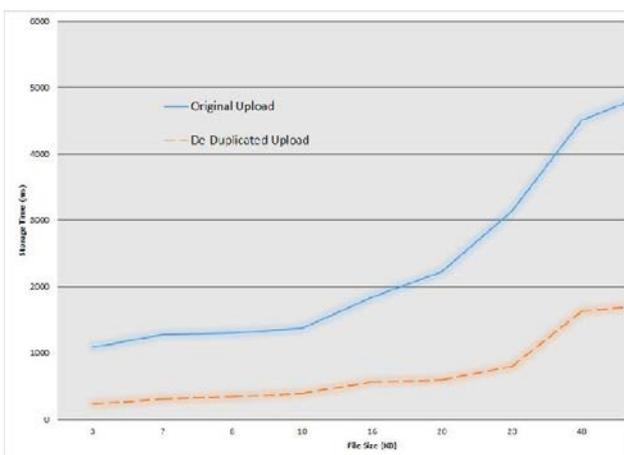
Fig. 3: Write process with 1 KB block size

Our implemented architecture provides inline data de-duplication mechanism. Such method od de-duplication takes place before storing data. It means the system checks for redundancy when data is not stored. For experimental purpose,we performed our experiments with different

Block Size i.e. 1 KB ,2 KB , 3 KB and 4 KB and studied their effect on different file size. For experiment we used two core 2 Duo work stations with 4 GB RAM with memory specification as 160 GB Hard drives and network connection of 1GBPS LAN.

A. Performance on using 1KB block size for write and delete process

For experimental purpose,we performed our experiments with different Block Size i.e. 1 KB ,2 KB , 3 KB and 4 KB and studied their effect on different file size. All the graphs are representing two parameters as file size and storage time taken for different files during write process and delete process.

While using block size of 1 KB the performance graph for write process different file sizes is given below- (Refer fig 3)

While using block size of 1 KB the performance graph for deleting different file sizes is given below- (Refer fig 4)
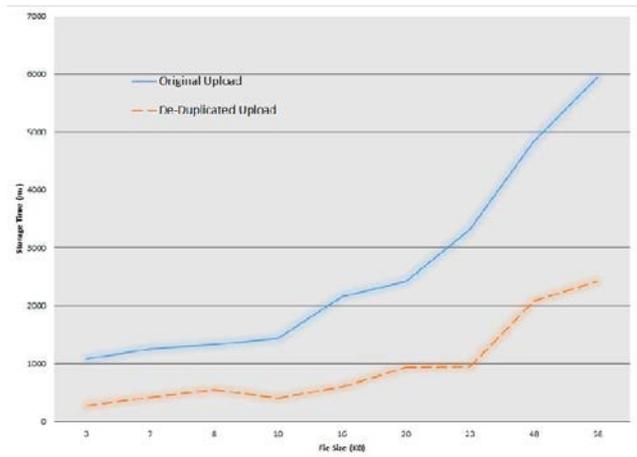
Fig. 4: Delete process with 1 KB block size
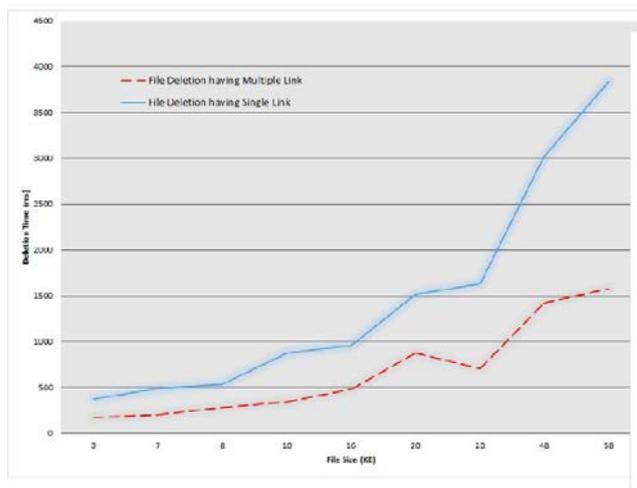
Fig. 5: Write process with 2 KB block size

B. Performance on using 2KB block size for write and delete process

While using block size of 2 KB the performance graph for write process different file sizes is given below- (Refer fig 5)

While using block size of 2 KB the performance graph for deleting different file sizes is given below- (Refer fig 6).
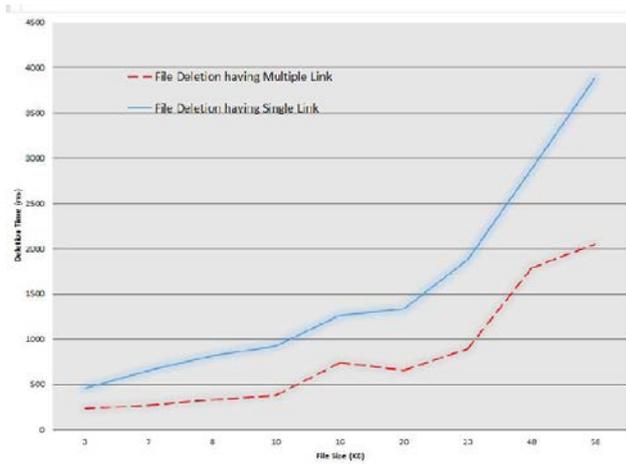
Fig. 6: Delete process with 2 KB block size

C. Performance on using 3KB block size for write and delete process

While using block size of 3 KB the performance graph for write process different file sizes is given below- (Refer fig 7)
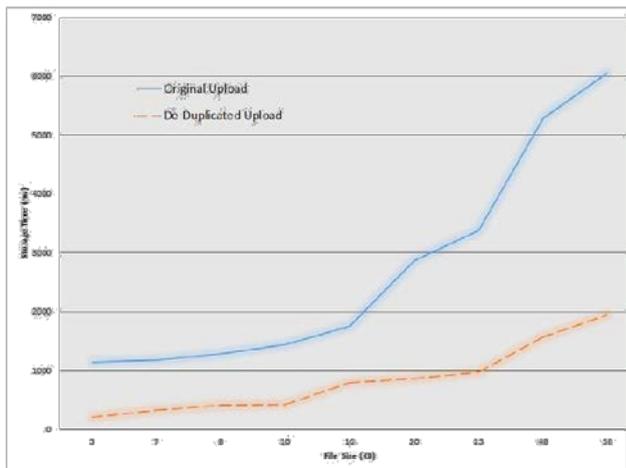
Fig. 7: Write process with 3 KB block size

While using block size of 3 KB the performance graph for deleting different file sizes is given below- (Refer fig 8).
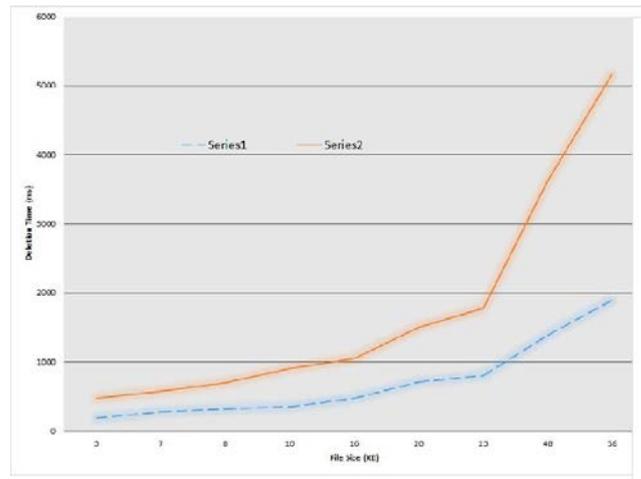
Fig. 8: Delete process with 3 KB block size
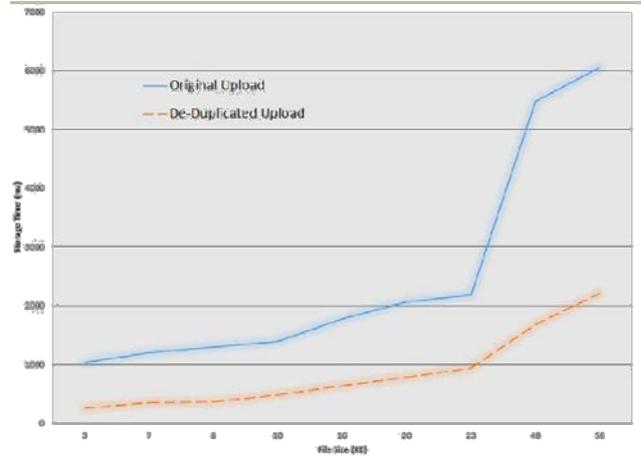
Fig. 9: Write process with 4 KB block size
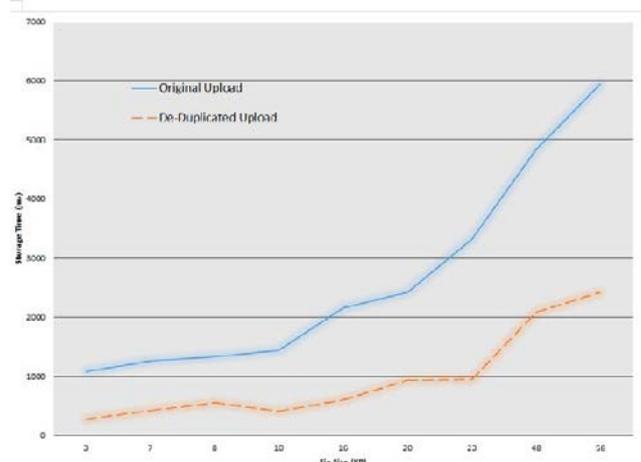
Fig. 10: Delete process with 4 KB block size

D. Performance on using 4KB block size for write and delete process

While using block size of 4 KB the performance graph for write process different file sizes is given below- (Refer fig 9

While using block size of 4 KB the performance graph for deleting different file sizes is given below- (Refer fig 10).

When we applied write process on different files sizes using inline parallel architecture , we found that; if we are storing a file having multiple references then the time for write process will be less than the time taken by the system ,when there was no references.

When we applied delete process on different file sizes using inline parallel architecture , we found that;if we are deleting a fie having multiple references at first time deletion will take very less time, because it only delete the data base entries, and if we are finally deleting the data having 1 reference count the deletion time will increase. Our implementation is showing a same trend with varying Block size. So it can be used with varied block sized file system and it will give same results. The overall performance on parameter time is given in the graph given below- (Refer fig 11) The results we get with this implementation and experiment are-

Hash collision probability is much lesser due to use of SHA-256 hash signature. It provides a message digest of size 2^{128} 1 , which means the probability is much lesser then SHA-1 algorithm.
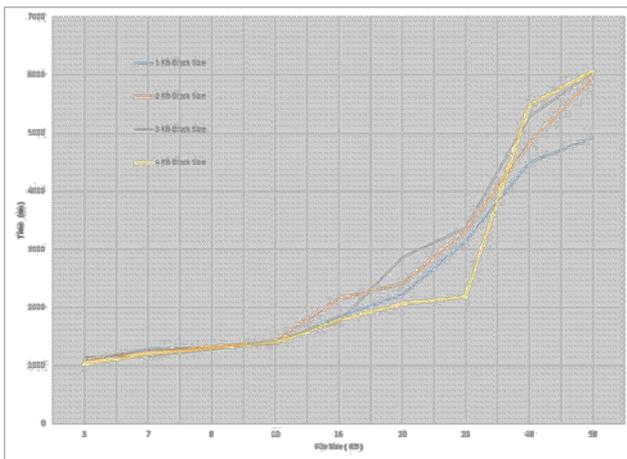
Fig. 11: File Storage time Variation with Varying Block Size

For different block sizes the parallel architecture is running successfully and the time taken(in milliseconds) by the system to perform write operation for redundant data is less then the time taken to store that data first time. One important fact we get during implementation is that this architecture is more efficient for files with larger size. The overall time taken to store redundant large files is lesser then the time taken for small file size.

For performing the delete operation for block size of 1 KB, 2 KB, 3 KB and 4 KB,we get a fact that for deletion of a single link the time taken by system is lesser; where for deletion of actual data takes more time . This depends on the reference count information. If the reference count is greater than one then only the entries in database are

remove and leads to lesser time but if the reference count is one, in that case it has to go to the actual data storage and will perform delete operation; that leads to more time taking process.

Parallel architecture supports for better time efficiency and load sharing. By using various storage nodes in parallel with server node, the incoming data is properly shared with all storage nodes and hash is created for each node individually. For reading, writing or deleting these file it is also easier for server to assemble all parts together and sending the data to client.

It creates unique names of file if duplicates are present. If a data is already stored then it fetches that particular part of data and never write a duplicate name.

VI. COMPARISON WITH PREVIOUS HASH ALGORITHMS

Hash signature or message digest is a very important part of data de-duplication process. While doing experiments with this architecture using SHA-256 algorithm, we got better results. In previously used methods MD5 or SHA-1 Algorithm was used. We can see a comparison with these hash algorithms and we can conclude that the SHA-256 algorithm is more efficient than other algorithms.

Most important parameter for any cryptographic hash algorithm is the hash size. SHA-256 gives the hash output size with 256 bit, whereas MD5 and SHA-1 gives 128 bit and 160 bit hash respectively. This means SHA-256 is toughest to crack.

The probability of collision is very less in the case of SHA-256 algorithm with 2^{128}. In the case of MD5 and SHA-1 the collision chances are 2^{64} and 2^{80} respectively, which have more probability of having collision. The maximum message size for SHA-256 is 2^{128} 1.

It is higher than SHA-1 and MD5, those have maximum message size as 2^{64} 1.

These properties of SHA-256 made it robust to use and gives better results at higher speed.

VII. CONCLUSION AND FUTURE WORK

In this paper a parallel architecture for inline data de-duplication is presented, in this we used SHA-256 algorithm for less collision probability. The main advantage of using SHA- 256 algorithm is that, it provides 128 bit hash by which the collision probability is much more lesser than previous de duplication methods. We have implemented an inline parallel architecture for data de

duplication by experimenting write and delete operations with different size of data blocks. Finally we got the positive results for storing operations as well as for delete operations. In our architecture we have implemented in-line parallel architecture using server and client storage based de-duplication of data. Further modification can be done using different hash algorithms with variable block size concept. Implementation of such architecture can be done possibly on Linux kernel. Study is going on to check our algorithm with varying Hashing methods and it will be very interesting to check its performance compared to other algorithms.

REFERENCES

[1] Q. He, Z. Li, and X. Zhang, "Data deduplication techniques," in Future Information Technology and Management Engineering (FITME), 2010 International Conference on, vol. 1. IEEE, 2010, pp. 430–433.

[2] S. S. Sengar and M. Mishra, "A parallel architecture for in-line data de-duplication," in Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on. IEEE, 2012, pp. 399–403.

[3] Wikipedia, "Deduplication — wikipedia, the free encyclopedia," 2011, [Online; accessed 20-September-2014]. [Online]. Available: http: //en.wikipedia.org/w/index.php?title=Deduplication&oldid= 434419167

[4] G. Wang, Y. Zhao, X. Xie, and L. Liu, "Research on a clustering data de-duplication mechanism based on bloom filter," in Multimedia Technology (ICMT), 2010 International Conference on. IEEE, 2010, pp. 1–5.

[5] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in ACM SIGOPS Operating Systems Review, vol. 35, no. 5. ACM, 2001, pp. 174–187.

[6] X. Ge, N. Deng, and J. Yin, "Application for data de-duplication algorithm based on mobile devices," Journal of Networks, vol. 8, no. 11, pp. 2498–2505, 2013.

[7] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage." in FAST, vol. 2, 2002, pp. 89–101.

[8] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, "Single instance storage in windows 2000," in Proceedings of the 4th USENIX Windows Systems Symposium. Seattle, WA, 2000, pp. 13–24.

[9] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pp. 285–298, 2002.

[10] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in Fast, vol. 8, 2008, pp. 1–14.

[11] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving du-plicate elimination in storage systems," ACM Transactions on Storage (TOS), vol. 2, no. 4, pp. 424–448, 2006.

[12] M. Jiantinga, "A deduplication-based data archiving system." Interna-tional Proceedings of Computer Science & Information Technology, vol. 50, 2012.