A Study on Various Data De-duplication Systems

Rashmi Vikraman
Department of Information Science and Technology
College of Engineering Guindy
Anna University, Chennai, India

Abirami S
Department of Information Science and Technology
College of Engineering Guindy
Anna University, Chennai, India

ABSTRACT
Data is the heart of any organization; hence it is necessary to protect it. For doing so, it is the needed to implement a good backup and recovery plan. But the redundant nature of the backup data makes the storage a concern; hence it is necessary to avoid the redundant data present in the backup. Data de-duplication is one such solution that discovers and removes the redundancies among the data blocks. This paper focuses on giving a wide study on the technology, process and types of the various data de-duplication system. This paper is helpful to the readers in giving a detailed analysis and study on the various data de-duplication systems that has been proposed by many researchers.

Keywords
Data de-duplication, Flash Storage, Chunking, Backup

1. INTRODUCTION
According to the present scenario, the backup has become the most essential mechanism for any organisation. Backing up files can protect against accidental loss of user data, database corruptions, hardware failures, and even natural disasters. However, the large amount of redundancies which is found in the backups makes the storage of the backups a concern, thus utilizing a large of disk space. Data de-duplication comes as a rescue for the problem of redundancies in the backup. It is a capacity optimization technology that is being used to dramatically improve the storage efficiency. Data de-duplication eliminates the redundant data and stores only unique copy of the data. Here instead of saving the duplicate copy of the data, data de-duplication helps in storing a pointer to the unique copy of the data, thus reducing the storage costs involved in the backups to a large extent. It need not be applied in only backups but also in primary storage, cloud storage or data in flight for replication, such as LAN and WAN transfers. It can help organizations to manage the data growth, increase efficiency of storage and backup, reduce overall cost of storage, reduce network bandwidth and reduce the operational costs and administrative costs. The five basic steps involved in all of the data de-duplication systems are evaluating the data, identify redundancy, create or update reference information, store and/or transmit unique data once and read or reproduce the data.

Data de-duplication technology divides the data into smaller chunks and uses an algorithm to assign a unique hash value to each data chunk called fingerprint. The algorithm takes the chunk data as input and produces a cryptographic hash value as the output. The most frequently used hash algorithms are SHA [22], MD5 and rabin fingerprint algorithm [4]. These fingerprints are then stored in an index called chunk index. The data de-duplication system compares every fingerprint with all the fingerprints already stored in the chunk index. If the fingerprint exists in the system, then the duplicate chunk is replaced with a pointer to that chunk. Else the unique chunk is stored in the disk and the new fingerprint is stored in the chunk index for further process.

Data de-duplication can be performed either in file level, chunk level or byte level. File level involves considering the file as a whole for the duplicate detection by finding the whole file hash and making the comparisons with the already stored whole file hash values. Chunk level involves splitting the file into small pieces called *chunks* and the hash value of the chunk called *fingerprint* of the chunk is used for making comparisons and storage. Byte level involves comparing the file byte by byte. Based on the technology, data de-duplication can be performed as Hash based de-duplication or Content Aware de-duplication. Based on the process or when it is done, it can be inline (*synchronous*) de-duplication, where the de-duplication is done immediately after its arrival and then stored into the disk or post-process (*asynchronous*) de-duplication, where the data is stored into the disk and then it undergoes the process of de-duplication. Based on the type or where it happens, it can be source (client) side de-duplication and target side de-duplication. The major questions involved in performing de-duplication are when and where the process of de-duplication has to be done. Following are the answers for those two major constraints.

1.1 Based on when de-duplication is performed
Please Following are the two ways of performing the data de-duplication on the backup run.

1.1.1 Inline De-duplication
The Data de-duplication is performed before writing the data into the disk. As soon as the data stream comes for storage into the disk, it undergoes the de-duplication algorithm and only unique chunks are stored. It collides with the normal functioning of the computer.

1.1.2 Post process De-duplication
Data de-duplication is performed after the data to be de-duplicated has been initially stored into the disk. The data residing in the disk is later fetched inside the de-duplication system and only unique chunks remain back and all the duplicate chunks are deleted. It doesn't collide with the normal functioning of the computer.

1.2 Based on where de-duplication is performed
Following are the two ways of data de-duplication system that specifies where exactly the de-duplication is happening.

1.2.1 Source side or client side De-duplication
It identifies the duplicate data at the source and transmits only unique segments to a central repository. Two separate

components are present for source and client.

1.2.2 Target side De-duplication

It identifies the duplicate data at the target and stores only unique segments. It is a standalone system. It causes no overhead on the client or server being backed up.

2. STUDY ON CHUNKING ALGORITHMS

Data De-duplication can be performed in two different ways, either Hash based where the fingerprint of the chunk is used in de-duplication of data or Content based, where the de-duplication is done by byte by byte comparison. Following section gives a brief study on such algorithms.

2.1 Hash Based Chunking

Hash Based De-duplication involves using a hashing algorithm to identify the chunks of the data. The hash algorithm takes the chunk as the input and produces a cryptographic hash value for the chunk. The most commonly used hashing algorithms are SHA-1 [22] and MD-5. The hash value is known as the fingerprint of the chunk. The chunks can either be of fixed length or variable length. If the fingerprint already exists in the chunk index, then this chunk is termed as duplicate and it is not stored into the disk, else if the chunk was not found in the chunk index, then this unique chunk is stored into the disk. Following are the two ways of chunking the data file.

2.1.1 Fixed length or Fixed blocks Chunking

Here the evaluation of data includes a fixed reference window used to look at segments of data during de-duplication process. It provides a fixed block boundary e.g. 4KB, or 8KB. Fixed length chunking is used most often when general purpose hardware is involved for carrying de-duplication. Nevertheless the fixed length chunking algorithm achieves significantly very less reduction than a variable length approach. The reason is because the duplicates are usually found between any two transmitting data set or any two consequent backup data sets, the two data sets with a small amount of difference are likely to have very few identical chunks. Advantage is that it requires the minimum CPU overhead, and it is fast and simple. Because of the block size or block boundaries being fixed, it results in boundary shifting problem, where if the data in the file is shifted, then it affects all the data following it, and the duplicates are not detected as a result of this.

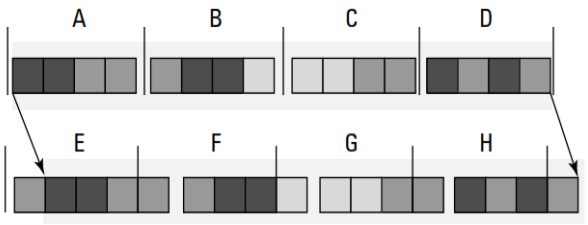

Fig 1: Fixed Length Chunking [33].

Figure 1 illustrates the boundary shifting problem due to fixed size chunking, where chunks A, B, C and D are similar to chunks E, F, G and chunks H respectively. But due to the addition of some text in the beginning before the chunk E affects all the chunks following it and the duplicates are not detected due to the fixed window size.

2.1.2 Variable length or Variable block Chunking

Here the evaluation of data uses a variable length window to find duplicate data in stream or value of data processed. It divides the data stream into variable length data segments using a data dependent methodology that can find the same data block boundaries in different locations and contexts. Here the window size varies based on what algorithm is being used with average window size as 4KB. The most frequently used variable length chunking algorithm is TTTD [1], [2], [3], [4]. Figure 2 illustrates the variable length chunking. Even after adding some data before the chunk E, neither the chunk E nor the chunks following it are affected. This way of creating variable length blocks makes the data to float inside the data file and helps in finding maximum number of duplicates.

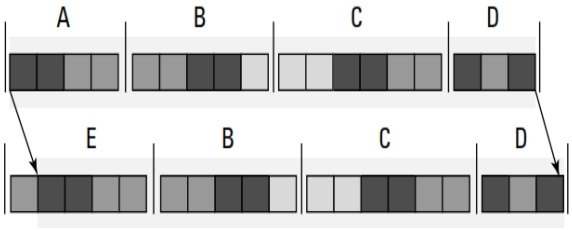

Fig 2: Variable Length Chunking [33]

T. T. Thwel et.al. [2] have used the TTTD algorithm for chunking the data files. This paper has clearly specified about the procedure involved in the variable length chunking. It uses a minimum size and maximum size threshold for setting the maximum and minimum values of every chunk. Two divisor values namely main divisor and second divisor are also used for finding the boundary of the chunk. Main divisor finds the breakpoint and if it unsuccessful in doing so, then the backup breakpoint found using the second divisor acts as the breakpoint. But TTTD has a limitation due to the second divisor which mostly produces breakpoints which are near to the maximum threshold. This results in larger sized chunks where a lot of time is wasted in performing unwanted calculations and comparisons.

T. S. Moh et.al. [1] has proposed TTTD-S algorithm to eliminate the disadvantage of TTTD algorithm. It uses a new parameter called average threshold which is the average of maximum and minimum threshold. When this algorithm reaches this parameter, the original values of main divisor and second divisor is halved. These values are switched back to the original values once the breakpoint is found. This avoids unnecessary comparisons and calculations.

2.2 Content or Application Aware Based Chunking

F. Douglis et.al. [5] used the content aware de-duplication which is performed in a different way. Here the data is considered as an object. It takes the objects and compares it with the other objects for finding the duplicates in an efficient manner. Here the data is divided into large data segments and by using the knowledge of the content of the data, similar segments are determined and only the changed bytes between the objects are saved. This is a byte level comparison.

3. INDEXING TECHNIQUES

Final chunks obtained after performing the chunking algorithm undergoes a cryptographic hash algorithm to produce a unique fingerprint (Hash value) for every chunk. All the unique chunks are placed in the chunk index. As the number of chunks increases, the number of fingerprints increases to be saved in the chunk index. Increase in the size of the chunk index makes the search in the chunk index more

complicated. Hence many researchers have found solution for reducing the search time involved in the comparisons. Brief reviews on such research's have been discussed below.

T. T. Thwel et.al. [2] proposed an efficient indexing mechanism for data de-duplication suing advantage of B+ tree properties. The fingerprints act as the indexing keys in the B+ tree structure. Here the search time gets reduced from O(n) to O(log n) which can avoid the risk of full chunk indexing. This comparatively reduces the searching time and the number of comparisons.

P. Christen [6] has given a survey of indexing techniques for scalable record linkage and de-duplication where the various indexing techniques for the structured data are given in detail.

But when the number of fingerprints in the chunk index increases, the search time also increases. It becomes difficult to hold the entire chunk index inside the main memory; hence the chunk index is placed in the disk. To search for the chunk fingerprint in the chunk index placed in the disk, it takes a lot of IO operations and search time as well. This is called as *chunk lookup disk bottleneck problem*. To avoid this, some part of the chunk index can be placed in the cache based on certain conditions such as based on locality, similarity etc. Another structure called the bloom filter, which acts as a summary index for the entire chunk fingerprints present in the chunk index is placed in the cache to avoid unnecessary searches. The study on the bloom filter is given in below.

3.1 Bloom Filter

Bloom filter [7] is a space efficient probabilistic data structure that is used to test whether an element is a member of the set. The bloom filter can be placed in the cache to test whether a particular fingerprint is a part of the chunk index placed in the disk or not. Using the bloom filter, false positives may be resulted but never a false negative. This technique uses a small hash area but still eliminates the unnecessary accesses. A bloom filter is a bit array of m bits all set to 0 initially. There are k hash functions whose hash values are set to one of the m array positions. To add an element, feed it into k hash functions and based on the hash values, set all the resulting bits in m bit array to 1. To test for an element, feed it into k hash functions and check whether the resulting hash values are set to 1 in the m bit array. During the query, false positives may be resulted which states that any fingerprint is present in the chunk index which is actually not present. But a false negative is never resulted and thus avoiding the unnecessary search. Removing an element from the bloom filter is not possible as false negatives are not allowed in the bloom filter. The most commonly used hash function for producing k hash values is bloom filter is murmur hash function. Figure 3 [12] demonstrates the functioning of the bloom filter. In the figure to test whether w is an element of the {x,y,z}, w is fed into 3 hash functions. As one of the bit array obtained from the hash result is set to 0 it says that w is not a member of the set {x,y,z}.

S. Quinlan et.al.[8] have used the bloom filter to detect the previously stored data. It is intended for the archival data. Fan et.al. [9] has proposed counting filters which is an enhancement to the bloom filter which provides a way to delete the element in the bloom filter. Here the array positions are enhanced from being a single bit to n bit. Insert operation involves incrementing the value of the buckets. Lookup involves testing whether the bucket has a non-zero value set. When an element is deleted from the bloom filter, the value set in the bucket is decremented.

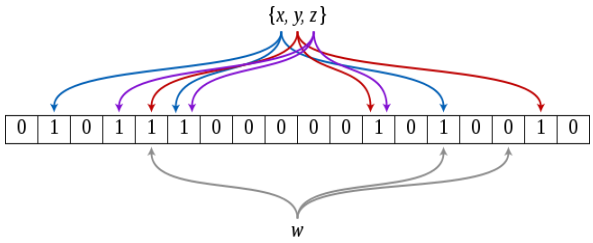

Fig 3: Bloom filter

Chazelle et.al.[10] has proposed bloomier filter which is the generalisation of the bloom filter. It acts like an associative array where every value corresponds to a particular element that has been inserted. The false positives produced from this filter are comparatively lesser than the bloom filter. J. Wei et.al.[11] has proposed dynamic bloom filter for efficiently representing the membership for variable large datasets. It consists of dynamically created groups of bloom filters. Here within the groups, the bloom filters are homogeneous and it allows parallel access to achieve high query performance. It allows element deletion by using lazy update policy. It produces good scalability, query accuracy and space efficiency. But still, one cannot be directly dependent on bloom filter for duplicate detection. Hence to fetch some part of the chunk index into the cache helps in reducing the overall search time involved in duplicate estimation. This has been explained in the next section.

3.2 Cache Based Storage

To avoid the chunk lookup disk bottleneck problem, some part of the chunk index in the disk is kept in the main memory for faster search. The mode of prefetching the chunk fingerprints into the cache depends on certain factors. Some are discussed here.

Zhu et.al.[14] has proposed a combination of three approaches to overcome the chunk lookup disk bottleneck problem. A bloom filter [5] is maintained as a probabilistic summary of all the fingerprints already stored in the disk. Stream informed segment layout, where the incoming chunks are stored inside containers in the arrival order so that every container contains the chunks from one stream. This captures the temporal locality of the data. Locality preserved caching involves capturing the data order when the data is written for the first time and to accelerate the duplicate detection by fetching the chunks into the cache based on it.

D. Bhagwat et.al.[13] has proposed the concept of extreme binning. This method exploits the similarity among the backup runs. The chunk index is split into two tiers, where one of the tiers called the primary index is placed in the RAM and the other in the disk. The primary index contains one fingerprint entry per file. This fingerprint is called the representative_ID of the file, which is the smallest fingerprint out of all the fingerprints of the file. The rest of the fingerprints are placed in disk in small structure known as bin pointed from each representative_ID of the file. Broders theorem is used for exploiting the similarity among the backup runs.. Limitation is that as the amount of information increases the primary index cannot be placed in the RAM. C. Wang et.al.[16] has proposed a fast duplicate chunk identifying method based on Hierarchical indexing structure which is very much similar to the concept of extreme binning. Advantage of FDCI methods is as follows; it reduces the disk

access time, improves the RAM utilization and keeps the de-duplication performance under consideration

M. Lillibridge et.al.[15] has proposed sparse indexing which is a technique that uses sampling and takes the exploits locality information of backup run for faster duplicate detection. Each chunk is de-duplicated against the chosen chunks instead of full chunk index. Very small portion of the chunks are chosen as the samples and their sample hash values are mapped to the already stored chunks. Using very small sampling rate, sparse index can be made placed in the RAM. But it can still result in duplicates if the chosen sample is not proper.

A. Wildani et.al.[17] has proposed HANDS- Heuristically arranged non backup inline de-duplication system. It dynamically prefetches the fingerprints into the cache based on the working sets which are derived from the access patterns of the chunks. The working set table is created using the N-Neighbourhood partitioning. The working set of the element is fetched into the cache when a cache miss occurs.

W. Xia et.al.[19] has proposed a system which exploits both the similarity and locality among the data sets. It utilizes very low RAM overhead and produces high throughput due to the combination of both the approaches. The main idea is as follows. One is to group various strongly correlated small files into a segment which results in much smaller similarity index compared to the chunk index which can fit into the RAM, thus increasing the de-duplication efficiency. Second is segmenting the large files into smaller segments which results in identifying more duplicates in the dataset. By preserving the locality information in RAM, it is able to remove a large number of duplicates in the data streams.

D. Meister et.al.[20] has proposed block locality caching for data de-duplication system which exploits the locality between the data sets. It captures the locality information of the previous backup run and uses this information to predict the future chunks in the next backup run. This approach uses up-to date locality information which is less prone to aging. The chunks are arranged inside the blocks in a sequential order as they arrive in the data stream. As the first data of the second backup run is written to the disk, the block recipes of previous backup run aligned to it is loaded into the cache to decrease the search time and overcome the chunk lookup disk bottleneck problem.

4. FLASH BASED DE-DUPLICATION SYSTEMS

To avoid the chunk index disk bottleneck problem, a recently popular is to make use of flash memory. Flash memory is much faster to access than a disk based search. Flash stands in between DRAM and disk both in terms of access speed as well as the cost. It takes 100 to 1000 time's lower access times than the hard disk. B. Debnath et.al.[21] has proposed a technique called chunkstash: speeding up inline storage de-duplication by making use of flash memory. RAM lookup and bloom filter reduces the disk lookup to a large extent by Zhu et.al.[14] but still some amount of disk lookup involved takes a lot of time. Hence to reduce that overhead, the lookups are made from the flash based index. It is a chunk metadata store on the flash memory. A small fraction of the index is fetched into the RAM to decrease the search time and the number of access.

G. Lu et.al.[23] has proposed a technique called bloomstore, a bloom filter based memory efficient key-value pairs for indexing of data de-duplication on flash. They proposed an efficient KV store on flash with a bloom filter [7] based index structure called bloomstore. The unique features of this proposal include no index store on the RAM and storage of both the index structure and the chunk index in the flash. Their design not only reduces the RAM overhead but also achieves high insertion/lookup throughput by reducing the number of flash page read. The RAM was loaded with a small size bloom filter per bloomstore and keeping a flash page sized data buffer. The index structure acts like a prefilter to avoid many unnecessary flash page lookup.

5. FILE RECONSTRUCTION

The file reconstruction is one of the most important steps of the entire de-duplication system. The file once stored into the disk has to be reconstructed back when a request for the file arrives. File reconstruction is to rebuild the file content from the fingerprints of the chunks stored in the file recipe after it has been de-duplicated and written into the disk File reconstruction can be performed using a structure called *file recipe* [13]. When any particular file is written into the disk, all the information about the file is placed inside the file recipe. File reconstruction estimates the read performance of the file; hence various techniques to control the read performance have been mentioned.

D. Meister et.al. [24] has proposed file recipe compression in data de-duplication systems. Many compression schemes have been applied to shrink the file recipe. The idea behind the file recipe compression is to assign code words to the fingerprints, so the code words can be placed inside of the 20 byte SHA key in the file recipe. One of the techniques is zero chunk suppression. Many of the files contain chunks completely filled with zeroes. The fingerprint of the zero chunks can be found in advance and can be replaced with a code word. Hence the zero chunks need not be checked in the chunk index nor has to be stored inside the disk. Second approach is chunk index page oriented approach where each chunk not longer than necessary are assigned a code word. The chunk index's pages are used for assigning the code words. The last approach in this is statistical approach where the statistics is used to assign code words to fingerprints based on the probability of the usage of the chunks. Limitation of this system is that it cannot be applied to all the data sets.

Due to the high fragmentation of the chunks across many of the disk locations, the quality of the fragmentation and the de-duplication of data are reduced to a large extent. Y. Tan et.al.[26] has proposed a technique for reducing the de-linearization of data placement to improve duplication performance. The de-linearization of the data placement reduces the spatial locality of the data streams which is used for increasing the read performance. This approach reduces such de-linearization by compromising over the compression ratios. They have proposed a method called De-Frag to do the same. The key idea is to choose some duplicate data to be written into the disk instead of removing it. The decision is taken based on a metric called Spatial locality level used to measure the spatial locality for the chunks. If the dynamically calculated spatial locality value is lower than the pre-set value, then the chunk is written into the disk instead of removing it. The calculation of the spatial locality level is based on the broders theorem. From the results D-Frag has improved the de-duplication performance including the de-duplication throughput, efficiency and read performance.

Y. J. Nam et.al.[27] has proposed an approach to improve the read performance of the data de-duplication systems. Here a new indicator called cache aware Chunk Fragmentation Level

(CFL) has been proposed which estimates the degraded read performance by taking both incoming chunk information and read cache effects into consideration. CFL monitor checks whether the read performance of the current data stream is worse than the demanded read performance in terms of a value called CFL value. The CFL monitor includes two parameters, optimal chunk fragmentation and cache aware current chunk fragmentation. Another approach is the selective duplication that improves the read performance by improving the current level of the fragmentation in the chunks.

It has been found from many results that compressing the data after performing de-duplication produces better results than performing de-duplication over compressed data. The most commonly used compression algorithm is ziv Lempel algorithm [28], [14]. Many other compression algorithms such as easy Zlib, Huffman etc can also be made use. It is always better to compress the data before storing the de-duplicated data into the disk.

6. EVALUATION TECHNIQUE

D. Harnik et.al.[29] has performed the estimation of de-duplication ratios in large data sets. Estimation technique depends on two categories which are sampling phase and scanning phase. It provides various formulas used for estimating the performance of the de-duplication system. The main factor for finding the efficiency of the data de-duplication system is to evaluate using the de-duplication ratio. *De-duplication ratio* is defined as the ratio of the original size of the data before performing de-duplication to the de-duped size of the data after applying de-duplication on the data. There are many factors which affects the de-duplication ratio. First is based on the type of the data being used. More user created, unstructured, encrypted and compressed data produces higher de-duplication ratios. Secondly based on the data change rate, less change in the data sets produces higher de-duplication ratio. Third is based on the retention policy, where longer retention policy results in higher de-duplication ratio. Fourth is the full to incremental backup ratio where more full backup's results in higher de-duplication ratio than the incremental backup.

7. CONCLUSION

This paper has covered various research work performed on the data de-duplication. All the steps involved in the de-duplication algorithm have been explained briefly. It provided an overview on all the existing works happening on data de-duplication framework. Comparisons between the methodologies have also been discussed. From these works, it is obvious that still a lot more challenges need to be addressed in the future researches. They are creation of more optimized chunking algorithm. Better methodologies for solving the chunk lookup disk bottleneck problem which are not restricted only to similarity or the locality of the backup runs can be created. And creation of more optimized algorithms to increase the read and write performances of data de-duplication systems. For the future, many other problems related to data de-duplication systems can be discussed including study on performance evaluation factors mentioning the results as well.

8. REFERENCES

[1] Eshghi, K. A. 2005. Framework for Analyzing and Improving Content-Based Chunking Algorithms. Technical Report HPL-2005-30(R.1), Hewlett Packard Laboratories, Palo Alto, CA.

[2] Thein, N.L. and Thwel, T.T. 2012. An efficient Indexing Mechanism for Data De-duplication. In Proceedings of the 2009 International Conference on the current trends in Information Technology (CTIT), 1-5.

[3] Kruss, E., Ungureanu, C. and Dubnicki, C. 2010. Bimodal Content Defined Chunking for Backup Streams. In Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST' 10),

[4] Rabin, M. O. 1981. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University.

[5] Bloom, B. H. 1970. Space/time tradeoffs in hash coding with allowable errors. Communications of the ACM, 13(7), 422-426.

[6] Peter, C. 2012.A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. In Proceedings of the IEEE Transactions on Knowledge and Data Engineering. 24(9), 1537-1555.

[7] http://en.wikipedia.org/wiki/Bloom_filter

[8] Quinlan, S. and Dorward, S. 2002. Venti: A new Approach to Archival Storage. In Proceedings of the USENIX Conference on File and Storage Technologies, 89- 101.

[9] Fan, L., Cao, P., Almeida, J. and Broder, A. Z. 2000. Summary cache: a scalable wide area web cache sharing protocol. In Proceedings of the IEEE Transactions on Networking, 8(3), 281- 293.

[10] Chazelle, B., Kilian, J., Rubinfled, R. and Tal, A. 2004. The Bloomier Filter: an efficient data structure for static support lookup tables. In Proceedings of the 15th annual ACM-SIAM symposium on Discrete Algorithms, 30-39.

[11] Wei, J., Jiang, H., Zhou, K. and Feng, D. 2013. Efficiently Representing Membership for Variable Large Data Sets. In Proceedings of the IEEE Transactions on Parallel and Distributed Systems, Vol.25, 960-970.

[12] https://www.wikipedia.org

[13] Bhagwat D., Eshghi, Long D. D. E., and Lilibridge M., "Extreme binning: Scalable, parallel deduplication for chunk-based file backup", Proceedings of the 7th IEEE International Symposium on Modelling, Analysis and Simulation (MASCOTS), 1-9, 2009.

[14] Zhu, B., Li, k., and Patterson, H. 2008. Avoiding the disk bottleneck in the Data Domain deduplication file system. In Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST), 269-282.

[15] Lillibridge, M., Eshghi, K., Bhagwat, D., Trezise, G. and Camble, P. 2009. Sparse indexing: Large scale, inline deduplication using sampling and locality. In Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST), 111-123.

[16] Can, W., Qin, Z. G., Yang, L. and Juan, W. 2012. A Fast Duplicate Chunk Identifying Method Based on Hierarchical Indexing Structure. In Proceedings on IEEE International Conference on Industrial Control and Electronics Engineering, 624-627.

[17] Wildani, A., Miller, E. L., and Rodeh, O. 2013. HANDS: A Heuristically arranged non-backup inline

deduplication system. In Proceedings of the IEEE 29th International Conference on Data Engineering (ICDE), 446-457.

[18] Sengar, S.S. and Mishra, M. 2012. E-DAID: An Efficient Distributed Architecture for inline data deduplication. In Proceedings of the IEEE International Conference on Communication Systems and Network Technologies (CSNT), 438-442.

[19] Xia, W., Jiang, H., Feng, D. and Hua, Y. 2011. SiLo: a similarity-locality based near exact deduplication scheme with low RAM overhead and high throughput. In Proceedings of the USENIX Annual Technical Conference (ATC), 26-28.

[20] Andre, B., Dirk, M. and Kaiser, J. 2013. Block locality caching for data deduplication system. In Proceedings of the 6th ACM International Systems and Storage Conference, 19-24.

[21] Biplob, D., Jin, L. and Sudipta, S. 2010. Chunkstash: Speeding up Inline Storage Deduplication using Flash Memory. In Proceedings of the 2010 USENIX Annual and Technical Conference, 16-21.

[22] en.wikipedia.org/wiki/SHA-1

[23] Lu, G., Nam, Y. J. and Du, D. H. 2012. BloomStore: Bloom filter based memory efficient key-value store for indexing of data deduplication on flash. In Proceedings of the 28th IEEE Symposium on Mass Storage Systems and Technologies (MSST), 1-11.

[24] Meister, D., Tim, S. and Brinkmann, A. 2013. File recipe compression in data deduplication systems. In Proceedings of 11th USENIX Conference on File and Storage Systems and Technologies (MSST), 175-182.

[25] Kaczmarczyk, M., Barczynski, M., Killian, W. and Dubnicki, C. 2012. Reducing impact of data fragmentation caused by inline deduplication. In Proceedings of the 5th ACM Annual International Systems and Storage Conference (SYSTOR).

[26] Feng, D., Sha, E.H., Ge, X., Tan, Y. and Yan, Z. 2012. Reducing the delinearization of data placement to improve deduplication performance. In Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, 796-800.

[27] Young, J. N., Park, D. and David, H. C. D. 2012. Assuring Demanded Read Performance of Data Deduplication Storage with Backup Datasets. In Proceedings of the IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication systems, 201-208.

[28] Ziv, J. and Lempel, A. 1978. Compression of Individual Sequences via Variable-Rate Coding. In Proceedings of the IEEE Transactions on Information Theory, 530-536.

[29] Harnik, D., Margalit, O., Naor, D., Sotnikov, D. and Vernik, G. 2012. Estimation of deduplication ratios in large data sets. In Proceedings of the 28th IEEE Conference on Mass Storage Systems and Technologies (MSST), 1-11.

[30] Gowsikhaa, D., Manjunath., Abirami, S. 2012. Suspicious Human activity detection from Surveillance videos. In Proceedings of the International Journal on Internet and Distributed Computing Systems, Vol.2,No.2, 141-149.

[31] Gowshikaa, D., Abirami, S., Baskaran, R. 2012. Automated Human Behaviour Analysis from Surveillance videos: a survey. In Proceedings of the Artificial Intelligence Review. DOI 10.1007/s 10462-012-9341-3.

[32] Gowsikhaa, D., Abirami, S. and Baskaran, R. 2012. Construction of Image Ontology using Low level features for Image Retrieval. In Proceedings of the International Conference on Computer Communication and Informatics, (ICCCI 2012, January 10-12), 129-134.

[33] Mark, R. C. and Steve Whitner. Data De-duplication for Dummies. Wiley Publishing, Inc.