Multi-Index Technique for Metadata Management in Private Cloud Storage

Prabavathy B, Subha Devi M, Chitra Babu
Department of Computer Science and Engineering
SSN College of Engineering
Kalavakkam, Tamilnadu, India
Email: prabavathyb@ssn.edu.in, subha1129@cse.ssn.edu.in, chitra@ssn.edu.in

Abstract—Cloud computing provides computing, storage and software resources as services to users on-demand over the Internet. Cloud computing has been increasingly gaining popularity due to the benefits such as its access capability from any location, sharing of data across multiple data centers, and the management of data storage by the service providers. Since security is still an unresolved issue in public cloud storage, business critical data can be preferably maintained in private cloud storage. It is normally built with the utilization of unused commodity machines. As storage space is limited in private cloud storage, it has to be efficiently utilized. Hence, a specialized technique known as deduplication is used to reduce the data footprint in the storage. This technique splits the file into several chunks. Chunks of different files are compared against each other to ensure only unique chunks are stored in the storage. Chunk index maintains the chunkID and the location of the actual chunk for reconstructing the file. Whenever a file is stored in the storage system, corresponding metadata is also placed along with it to facilitate the retrieval of that file. Metadata generally includes the properties of the file and the beginning location of physical block of the file. In deduplication enabled cloud storage system, since the file is stored in the form of a collection of chunks, chunk index also need to be maintained in addition to metadata. This improves the search time of a particular file. Thus, the efficiency of the chunk indexing mechanism will directly influence the file retrieval time. In order to achieve faster retrieval time, this paper proposes multi-index structure for metadata management.

Index Terms—metadata management, indexing, deduplication, optimization.

I. Introduction

Cloud computing improves the performance of any organization by utilizing minimum resources and providing pay-per use service. The other key characteristics of cloud computing include device and location independence which means that the users can access from any location and from any device (eg., laptop, PC or mobile). Due to these characteristics, organizations choose to use software, platform and infrastructure as services over the cloud. One of the infrastructure services provided by cloud computing is cloud storage.

Cloud storage is a service model in which data is maintained, managed and backed up remotely and made available to users over the Internet. There are four types of cloud storage deployment models such as public, private, hybrid and community cloud storage. Public cloud storage is provided by the third-party service provider who hosts and manages the infrastructure. Private cloud storage is created, managed and monitored completely within an organization's firewall settings. Hybrid cloud storage is the combination of the public and private cloud. An organization may use the private cloud storage for the structured and active data, whereas public cloud storage for unstructured and archival data. Community cloud storage is an infrastructure for multiple customers, which is shared among several organizations.

Although there are many advantages in public cloud storage such as low cost, ease of management, it also has certain drawbacks. Multi-tenancy nature of the cloud may lead to the possibility of leakage of business critical data. Whenever the data needs to be moved to a new service provider, there is no guarantee that the complete data will be removed from the storage of old service provider. Hence, it is safe to keep the business critical data in private cloud storage environment as it offers a higher level of security and control over the data.

Private cloud storage is set up with the storage resources of unused commodity machines within the organization. This storage is provided as a service to the users within an organization. Since storage space is limited within an organization, it has to be utilized in an efficient manner. When different users share the storage, there may be lot of duplicate data across the files that belong to those users. Deduplication [3] is a technique which helps to eliminate the redundant data and utilize the storage space in an optimized manner. It splits the file into several chunks and computes the hash value for those chunks. The hash value of a chunk is termed as its chunkID. A chunk index is maintained with the chunkID and the location of the corresponding actual chunk. Whenever a chunk of a file enters the storage, its chunkID is compared against the chunk index to detect whether it is a duplicate chunk. Since only chunks of the file are maintained in the storage, it is necessary to maintain the chunkIDs that constitute the file. This is termed as File Recipe.

The objective of this paper is to decide upon the structure for the maintenance of the chunk indices to improve the performance of duplicate detection and retrieval of the file. Rest of the paper is organized as follows: Section 2 describes the background of metadata management, Section 3 discusses the works related to this paper, Section 4 describes the pro-posed system architecture of optimized private cloud storage, Section 5 discusses the experimental setup and results, Section 6 concludes the work.

ISBN:978-1-4799-1024-3/13/$31.00 ©2013 IEEE

II. Background

Whenever a file is stored in the cloud storage, the corresponding metadata should be maintained in order to retrieve the data. Metadata of a file describes the attributes of the file which includes pointer to the disk blocks which constitutes the file. Often, 80% of the time is spent in searching the metadata rather than searching the data. Metadata footprint increases with the increase in data that is stored in the storage nodes. If such metadata is not effectively maintained, it leads to under utilization of cloud storage nodes. Hence, it is very important to manage the metadata in an efficient manner to improve the utilization of storage space and file retrieval time.

A. Metadata management in Traditional File System

In traditional file system, a file is distributed across disk blocks in a machine. Metadata of that file includes the pointer to the disk blocks that constitutes the file. Whenever a request for that file is received by the storage, metadata is consulted to retrieve the file.

B. Metadata management in Distributed File System

In distributed file system, files are distributed across several storage nodes. Metadata of those files are maintained in a metadata server. When the user requests for a file, metadata server is first contacted and the corresponding metadata of the file is found. Metadata contains the information about the machine where the file is stored. It is then retrieved from the corresponding storage node.

C. Metadata management in Deduplication Enabled Storage

In general, chunks of a file are distributed across several storage nodes of a cluster in a deduplication enabled storage. Hence, metadata alone is not sufficient to retrieve the requested file. In addition to metadata, it is necessary to maintain both the file recipe as well as the chunk index. This is illustrated in Figure 1. Whenever the user requests for a file, the metadata server is contacted. It contains the information regarding the chunk index to retrieve the relevant chunks that constitute the requested file.

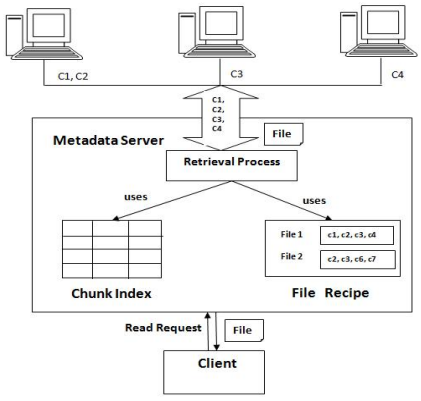

Figure 1. File Read Operation in Deduplication Enabled Storage

III. Related Work

Deduplication enabled storage system has the challenge of maintaining the file recipe and chunk index for the quick retrieval of a file. There are two types of workload for the deduplication enabled cloud storage. They are backup and non backup workload. Backup workload consists of files that have locality of reference. Non backup workload does not possess the locality between the files that are streamed to the storage.

Extreme Binning enables faster and efficient identification of duplicate chunks as it compares every incoming chunk only with the chunkIDs of similar files. According to Broder's theorem [4], two files are more or less similar if they have the same representative chunkID. Representative chunkID is the minimum hash value of the chunks that belongs to a file. It maintains the hierarchical index to hold the chunkIDs of a file. Primary index holds the representative chunkIDs. Secondary index holds the unique chunks of all similar files. This hierarchical index reduces the search time for detecting the duplicates. SHHC [2] maintains a distributed hash table to store the chunk entries in Solid State Drives (SSDs). As it places the hash table in SSD and prefetched data in RAM, it aids in fast retrieval of data.

Efficient B+ tree indexing [5] organizes the chunkIDs in a B+ tree. It helps to maintain a huge number of chunkIDs for the quick retrieval during file read operation and also to speed up the searching to identify the redundant chunks. Due to this advantage, there is a significant reduction in search time and comparison space. Application aware deduplication [1] enables the storage to compare the chunks of a same application. In this work, applications are categorized as static and dynamic. Based on the type of application, different duplicate detection methods and hashing algorithms are used. Duplication detection methods such as whole file, fixed and content based chunking [6] are applied. It suggests to use different index for different types of files.

The objective of this paper is to build an Optimized Private Cloud Storage (OPCS) using commodity machines by employing deduplication technique. The workload of this cloud storage is non-backup workload consisting of different types of applications. Extreme Binning quickly identifies the duplicates and facilitates fast retrieval of the file. However, it is performed for storage with backup workload. SHHC uses sophisticated device SSD to maintain the chunk index. As OPCS is built using commodity machines, it is not possible to maintain index in such devices. Efficient B+ tree considers a centralized B+ tree to organize the chunkIDs without consideration of types of applications. Application-aware deduplication categorizes the types of applications and also proposes the maintenance of different indexing schemes. However, specific structure for the index is not mentioned.

IV. Design of Optimized Private Cloud Storage

A private cloud storage is built by utilizing the unused commodity machines. In this cloud storage, one machine is chosen to be a Centralized Coordinator (CC) which manages all

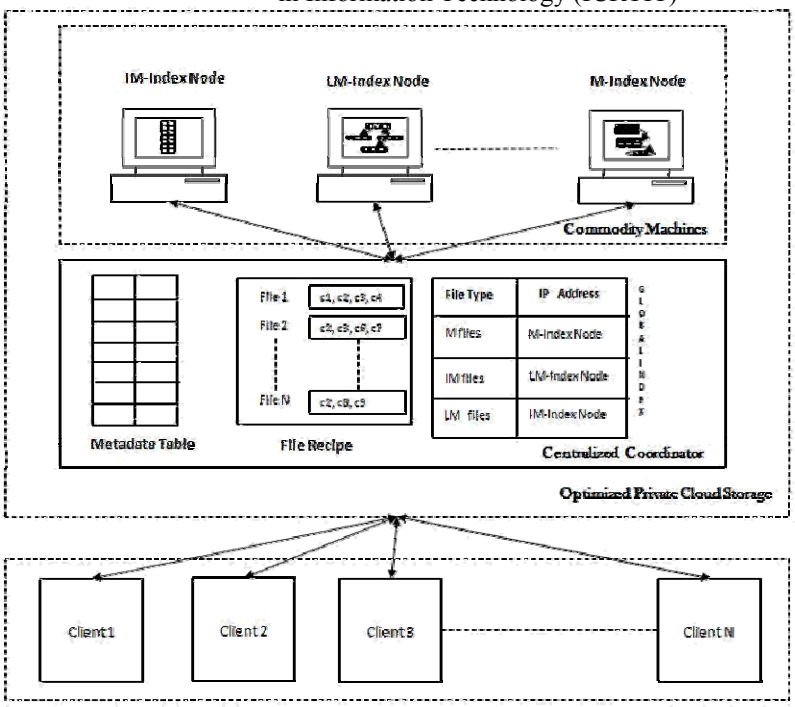

Figure 2. Architecture of Optimized Private Cloud Storage

the other storage nodes. It also acts like an interface between clients and storage nodes.

As the storage space is limited within the private cloud storage, it has to be efficiently utilized. Hence, deduplication technique is carried out for the optimal utilization of cloud storage. This cloud storage facilitates the users of the University or Organization to share the files across their laboratories instead of carrying flash drives. The categories of files considered for cloud storage are .txt, .odt, .pdf, .odp, .exe, .rar, .avi, .mp3 and .mp4. These files can be broadly classified based on their nature. They are either mutable, less mutable or immutable files.

Mutable files: Files of types .txt, .odt, .odp and .pdf are categorized as mutable file. They involve more modification and are most frequently accessed by the user. As the content of these files may get modified, content based chunking is used to divide the file to identify the duplicates effectively.

Less mutable files: Files of types .exe and .rar are less mutable files as they contain only less modification. Duplicates among these types of files are efficiently identified by performing fixed size chunking method.

Immutable files: Files of types .avi, .mp3 and .mp4 are categorized as immutable files. As the content of these types of files is not modified, such file can be compared in its entirety to identify the duplicates.

Deduplication enabled file system places unique chunks of the files across the storage nodes. File metadata describes the attributes of the file. However, a file is constructed with the consultation of the file recipe. File recipe has chunkIDs that constitute the file. Further, the location of the chunks in different storage nodes needs to be known. These details are maintained in a chunk index. OPCS is built using 'K' commodity machines where one of the commodity machines acts as a CC. In 'K-1' Commodity machines, 'm' machine serves as storage nodes and 'n' machines serves as index/storage nodes. Figure 2 shows the architecture of OPCS. In OPCS, it is necessary to maintain the metadata of the files, file recipe and the global index(to retrieve the index node in CC). Chunk index is distributed across the index nodes based on the types of the files. Index node that holds the chunk index for mutable files is M-Index node. Similarly, there are LM-Index node and IM-Index node hold the chunk index of less mutable and immutable files respectively.

Index for mutable files: OPCS has the workload that consists of files of several users. Hence, there is a huge probability of multiple versions of the files with minor variations existing across the files of different users or within the files of the same user. These versions of files are similar files. According to Broder's theorem, files f1 and f2 are similar files, if their minimum chunkIDs are the same. Hence, it will be beneficial to maintain a hierarchical index. The primary index holds the representative chunkID and the secondary index accommodates the chunkIDs of similar files. Once the usage of hierarchical index is decided, it is necessary to choose suitable data structures for the primary and secondary index. B+ tree is a balanced tree from root to leaf level. A node in B+ tree constitutes a key and a pointer. Order of B+ tree determines the number of keys

in a node. For example, consider a node with 100 keys and 101 pointers. Each key holds the chunkID of the file.

Level 0 = 100 chunkIDs.
Level 1 = 100 * 101 = 10100 chunkIDs.
Level 2 = 10100 *101 = 1020100 chunkIDs.
Level 3 = 1020100 *101 = 103030100 chunkIDs.

Hence, a B+ tree with height 2 is capable of holding 103,030,100(one hundred three million thirty thousand three hundred). In addition, as it is possible to accommodate large volume of chunkIDs within limited height, retrieval time is much less.

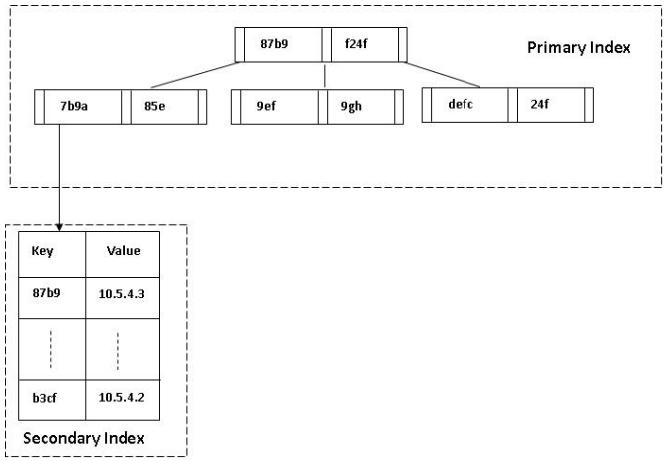

Figure 3. Hierarchical index

Hash table is a data structure consisting of a bucket that maps a key to value. It computes the hash function for the key and allocates the unique location for the entry in the bucket. Hence, a 'key' read in the hash table finds the value in O(1) time. Due to these advantages of B+ tree and hash table, representative ids of the files are placed in B+ tree. In the similar way, chunkIDs of the files are kept in hash table. This is illustrated in Figure 3. This largely reduces the timing overhead that incurs during the file read and write operation.

Index for less mutable files: The content of this kind of file is prone to minor modification only. ChunkIDs of these types of files are organized in a B+ tree to identify the duplicates and improve the retrieval time of the file.

Index for immutable files: Contents of these types of files will be the same across the users. Hence, it is enough to keep only the hashes of the files. Hash table is maintained to store the chunkIDs of files. Figure 4 shows the index structure of the hash table in which chunkID is mapped to the corresponding location of the chunk.

V. Implementation and Results

A private cloud storage has been set by using the open source framework Eucalyptus in commodity machines. Walrus controller handles the storage request processing read, write, delete and list operations. Files are transferred across the commodity machines using File Transfer Protocol (FTP). Deduplication technique is incorporated through the modification of source code of the Walrus Manager.

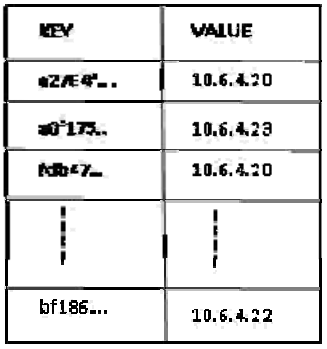

Figure 4. Hash table

A. Analysis of data structure for indices

As the workload for OPCS consists of various categories of files, chunkIDs of these files need to be placed in efficient data structures for faster lookup during file read and write.

1) Analysis of index structure for mutable files: B+ tree is a dynamically scalable tree with the capability to hold numerous data. The non-leaf nodes of B+ tree holds only keys, whereas the leaf nodes holds both the keys and the values. The key is the chunkID and the value is the actual location of the chunk. Searching of the key starts at the root node and moves downwards to the leaf node. B+ tree is implemented with each node holding 100 keys and 101 pointers. Various number of chunkIDs are placed in the B+ tree. They are loaded in the main memory and a value is searched in the B+ tree. The number of chunkIDs is increased from 100 to 25,00,000 and the loading and searching times are noted and tabulated in Table I.

TABLE I
PERFORMANCE ANALYSIS OF B+ TREE IN MEMORY AND ON-DISK

S.No	No of entries	B+tree in-memory (in sec)		B+tree on-disk (in sec)	
		Load	Search	Load	Search
1	100	0.030	0.002	0.014	0
2	1000	0.050	0.002	0.014	0
3	10000	3	0.002	0.015	0
4	100000	-	-	0.016	0.001
5	1000000	-	-	0.020	0.001
6	1500000	-	-	0.020	0.001
7	2000000	-	-	0.020	0.002
8	2500000	-	-	0.025	0.002

During its dynamic expansion, B+ tree cannot be loaded in memory. It is inferred from the table that B+ tree whose entries beyond 10,000 are not able to be loaded in memory. Hence, it is shown in '-'. B+ tree on-disk implementation retains only

the root node in the memory. Based on the search key, nodes in the search path alone are loaded in the memory. Time taken to load the contents of the tree and searching a value for B+ tree in-memory and on-disk implementation are tabulated in Table I.

Similarly, in-memory and on-disk implementation for hash table has been implemented. Time taken to load the chunkIDs and for searching a value in the hash table are tabulated in Table II. Hence, after analyzing the performance of various implementations of B+ tree and hash table, it was found that on-disk implementations of both B+ tree and hash table are more suitable to maintain the chunkIDs for mutable files.

TABLE II
PERFORMANCE ANALYSIS OF HASH TABLE IN MEMORY AND ON-DISK

S.No	No of entries	Hashtable in-memory (in sec)		Hashtable on-disk (in sec)	
		Load	Search	Load	Search
1	100	0.006	0	0.014	0
2	1000	0.050	0.001	0.014	0
3	10000	3	0.001	0.015	0
4	100000	60	0.001	0.016	0
5	1000000	240	0.001	0.020	0
6	1500000	540	0.001	0.020	0
7	2000000	600	0.001	0.020	0
8	2500000	660	0.001	0.025	0

2) Analysis of index structure for less mutable files: Fixed sized chunking method is applied for less mutable files as they are prone to only less modification. Hence, it is assumed that chunks created for these types of files will be less compared to that of mutable files. Due to the advantages of on-disk B+ tree inferred from Table I, it is chosen to hold the chunkIDs of these types of files.

3) Analysis of index structure for Immutable files: Types of files like .avi, .mp3 and .mp4 involve no modification. Hence, hashes for these types of files are enough to be maintained in index. Sequential index and hash table are implemented to hold the chunkIDs of immutable files. In worst case, sequential index is performed in O(n) time to retrieve a file. Whereas, hash table performs efficiently by retrieving the file in O(1) time. It is inferred from Table III.

TABLE III
PERFORMANCE ANALYSIS OF INDEX STRUCTURE FOR IMMUTABLE FILES

S.No	Chunk Entry	Sequential search (in sec)	Hash table (in sec)
1	100	0.001	0
2	1000	0.172	0
3	10000	0.296	0
4	100000	0.431	0
5	2000000	0.740	0.001
6	2500000	0.973	0

B. Identification of duplicate chunks using Indices

Whenever a write request arrives to CC of OPCS, it splits the file into various chunks and their hash values (ChunkIDs) are computed. Based on the type of file, the machine address which contains the index is retrieved. ChunkIDs of the file are sent to that machine for detecting duplicates. They are checked across the entries of the index and only the chunkIDs that are not present in the index are sent back to the CC for placing the data chunks corresponding to those chunkIDs.

TABLE IV
PERFORMANCE ANALYSIS TO DETECT DUPLICATES

S.No	No of chunks	Time taken to detect duplicates (in sec)
1	1500	0.850
2	2000	1
3	2500	1
4	3000	1

Table IV shows the number of chunks checked across the index to detect the duplicates and the detection time is noted.

C. Analysis of Retrieval time of mutable files

When a request arrives to retrieve a file, the corresponding file recipe is obtained. It is forwarded to the storage node where the index for mutable files is maintained. The locations of the actual chunks are identified from the chunk index and the corresponding storage nodes are contacted. All the chunks that are retrieved from different storage nodes are combined to constitute the file. It is then forwarded to the user through CC.

The hierarchical index with on-disk B+ tree with hash table and sequential index is implemented to hold reasonably large number of chunkID entries. The minimum and maximum size of the file that will be stored in OPCS are assumed to be 10 KB and 1 MB respectively. Table V shows the time taken to retrieve a file of type .odt with various sizes.

TABLE V
PERFORMANCE ANALYSIS OF RETRIEVAL TIME OF .ODT FILES

S.No	File Size	No of chunks	Retrieval time (in sec)	
			B+ tree with hashtable	Sequential
1	10 KB	5	0.124	1
2	50 KB	10	0.313	4
3	100 KB	26	0.501	10
4	500 KB	112	11	14
5	1 MB	257	42	52

D. Analysis of Retrieval time of less mutable files

Types of files .exe, and .rar are divided into fixed size of8KB and the chunkIDs are computed for those chunks. These

chunkIDs are organized as B+ tree and in sequential for the retrieval of the files. The minimum and maximum size of file that will be stored in OPCS are assumed to be 10 KB and 1 MB respectively. Table VI shows the time taken to retrieve a file of type .rar with various file sizes.

TABLE VI
PERFORMANCE ANALYSIS OF RETRIEVAL TIME OF .RAR FILES

S.No	File Size	No of chunks	Retrieval time (in sec)	
			B+ tree	Sequential
1	10 KB	2	0.158	2
2	30 KB	3	0.199	6
3	100 KB	12	0.604	9
4	500 KB	68	7	16
5	1 MB	80	8	30

E. Analysis of Retrieval time of Immutable files

Whole file chunking is performed for this file type and the chunkIDs of these types of files are maintained in hash table on-disk and sequential structure. The minimum size of these types of files are assumed to be 10 MB and the maximum size to be 1 GB. The time taken to retrieve these types of files with various sizes is tabulated in Table VII.

TABLE VII
PERFORMANCE ANALYSIS OF RETRIEVAL TIME OF IMMUTABLE FILES

S.No	File Size	Retrieval time (in sec)	
		Hash table	Sequential
1	10 MB	57	65
2	100 MB	792	871
3	250 MB	1447	1865
4	500 MB	1935	2069
5	1 GB	3245	3725

VI. CONCLUSION

A private cloud storage was built within the organization by utilizing the computing and storage capacities of a few commodity machines. In order to efficiently utilize the storage space within the organization, an optimization technique, namely deduplication was performed. Consequently, the storage which was enabled with deduplication contains only the unique chunks of files in storage. In order to facilitate the identification of duplicate chunks while storing and also to retrieve and reconstruct the file from the chunks, chunk index and file recipe are maintained. Chunk index entries of the entire storage cluster are split into several indices based on the types of files. These indices are distributed across different storage nodes. Experiments were conducted to choose a suitable index structure to organize the chunk entries for their quick retrieval. Workload consisting of different types of files with various sizes are deduplicated and stored in the storage cluster. The retrieval time for different types of files are computed with the distributed multi-index and is compared against the time taken with the sequential index. Results show that the distributed multi-index performs better compared to the sequential index.

REFERENCES

[1] Y. Fu, H. Jiang, N. Xiao and L. Tian. "Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment." In IEEE International Conference on Cluster Computing, pp. 112-120, 2011.

[2] L. Xu, J. Hu, S. Mkandawire and H. Jiang. "SHHC: A Scalable Hybrid Hash Cluster for Cloud Backup Services in Data Centers." In IEEE 31st International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 61-65, 2011.

[3] W. Zeng, Y. Zhao, K. Ou and W. Song. "Research on cloud storage architecture and key technologies." In Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, pp. 1044-1048. ACM, 2009.

[4] D. Bhagwat, K. Eshghi, D. D. E. Long and M. Lillibridge. "Extreme Binning: Scalable, parallel deduplication for chunk-based file backup". In IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 1 - 9, 2009.

[5] T. T. Thwel and N. L. Thein. "An Efficient Indexing Mechanism for Data Deduplication." In IEEE International Conference on the Current Trends in Information Technology (CTIT), pp. 1-5, 2009.

[6] C. Policroniades and I. Pratt. "Alternatives for detecting redundancy in storage systems data." In Proceedings of the USENIX Annual Technical Conference, pp. 73-86, 2004.