5 Classification criteria for data deduplication methods

Sulabh Bansal, Prakash Chandra Sharma

SCHOOL OF COMPUTING AND INFORMATION AND TECHNOLOGY, MANIPAL UNIVERSITY JAIPUR, JAIPUR, INDIA

5.1 Introduction

The automatic elimination of duplicate data is commonly known as deduplication. This reduction can be of storage data, network data, or virtual data. Storage data is the most general form of data known. It refers to data stored on various permanent and temporary storage mediums such as disks, tapes, and so on. Network data refers to data commuting through digital mediums between different machines connected over LAN, WAN, or Internet. Virtual data refers to data contained in virtual memory (VM).

Deduplication perceives the data as having two different views. First, the logical view that contains identifiable duplicates. Second, the physical view showing how data is stored physically from which duplicates have been removed. The deduplication establishes mapping from logical view to physical view for I/O operations in applications.

The logical view of data can be used to know the relevance of duplicate data elements and to decide which elements out of them need to be kept and which need to be removed. The data may or may not be partitioned into discrete chunks. The data elements are compared and data redundancy is eliminated. The granularity of chunks is defined to be chunk boundaries and their sizes. The spatial or temporal locality of duplicate chunks appeals for specific design decisions that can exploit locality to improve the efficiency and effectiveness of deduplication (Lillibridge et al., 2009). The physical view, on the other hand, reveals the technique used to represent duplicate eliminated data on the physical medium. It must be possible to reconstruct the logical view. The distribution scope of the technique is an important design decision in this respect.

The deduplication process must work with a storage management system which includes subsystems such as a file system. The timing of the main operations of deduplication, like when to find duplicates with respect to the critical path of I/O operations, impacts the design of deduplication. Identification of duplicates requires resources intensively and thus efficiency of the data structure used for indexing which supports matching of duplicate chunks is highly important. The choice of indexing method puts a trade-off between exactness and speed. Thus, this choice impacts heavily the efficiency and effectiveness of the deduplication process.

Data Deduplication Approaches. DOI: https://doi.org/10.1016/B978-0-12-823395-5.00011-2

The data elements are compared using indexes generated through one-way hash functions so that they are unique for every data element with different content. Examples of one-way hash functions are secure hash algorithms (SHA-1, SHA-2) and message digest (MD5) (Daehee, Sejun, & Baek-Young, 2017). The indexes supposed to be saved on permanently on disks are prefetched in memory for comparison before accessing data from disks. The additional techniques may also be used for reducing the time taken by comparison like Bloom filter (Daehee et al., 2017).

The objective of deduplication changes with storage environments they are designed for. In the case of archived data, removal or restoration is rare. Quantities of data that must be archived are large and the archival process is expected to be completed within a defined duration (Quinlan and Dorward, 2002). Thus, the deduplication for archived data target more on throughput than latency. Whereas, since backup data is restored and deleted more frequently, its deduplication require more efficient reference management and better garbage collection mechanisms (Guo and Efstathopoulos, 2011; Zhu, Li, & Patterson, 2008). Since active data is not immutable, the scenario of deduplication in primary storage is entirely different. The latency in deduplication in this scenario is more significant as it affects the application executing on primary storage very badly. Requests to access live data are more frequent than backup and archival storage data (Hong & Long, 2004; Srinivasan, Bisson, Goodson, & Voruganti, 2012). The modern operating systems take care for redundancy in random access memory (RAM) by sharing the duplicate pages. Sharing of duplicate pages is not done in case of VM. Therefore, deduplication is required to eliminate redundancy across VMs (Waldspurger, 2002). Deduplication of data on RAM has stringent requirement for I/O performance and space-efficiency. Deduplication in solid-state drives (SSDs) target to improve utility of space and lifespan (Chen, Luo, & Zhang, 2011).

Characteristics of duplicated dataset can also affect the technique adopted for deduplication. Let us consider two scenarios, first when there are many duplicate files that need to be handled and second when there are many similar files rather than identical files. In the first case, for deduplication files can be compared without looking at their content whereas in the second scenario the duplicate portions of the file cannot be identified without looking into it. Similarly, system capacity can affect the deduplication strategy too. Heavyweight deduplication strategy requiring more overhead can be done only on high-capacity servers rather than the low-capacity clients. Data packets at routers need to be processed in less time so that they can overwork to reduce redundancies at nodes within a network.

There are various deduplication techniques being used and available, but there is not a single best solution that can handle all types of redundancies. Different deduplication technique has been developed depending on the scenarios where they are supposed to be used. The different scenarios may be defined by various parameters like characteristics of data sets which are duplicated and need to be eliminated, system capacity available for bearing the overhead of deduplication, time of deduplication, etc. The algorithms in deduplication differ due to different decisions made to resolve issues such as granularity, technique, indexing, locality, scope, and timing (Daehee et al., 2017).

There are many aspects of deduplication process that effects its applicability, effectiveness, and efficiency. The definition, design, perspective, or solution of each aspect of this

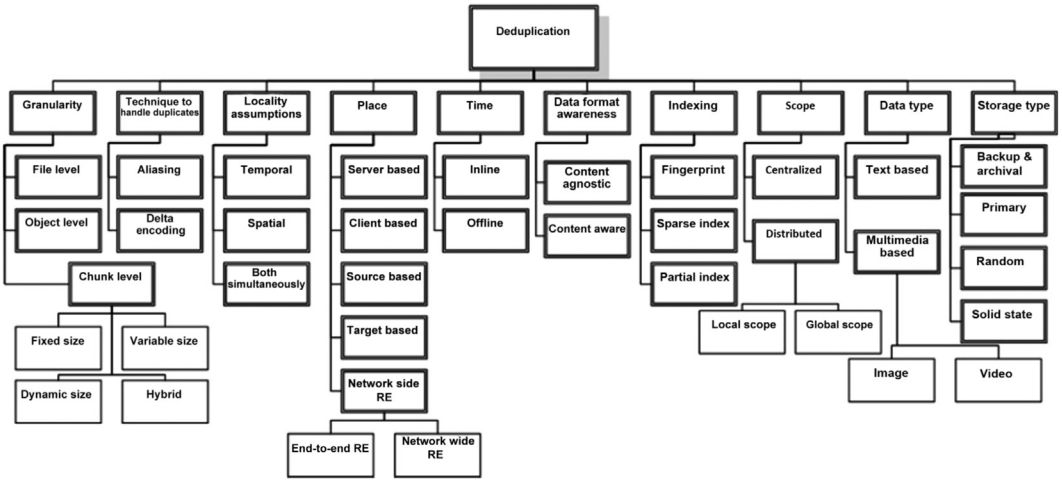

FIGURE 5–1 Classification criteria for deduplication. *RE*, redundancy elimination.

process depends on many parameters, scenarios, etc. To make the entire process as effective and efficient as possible, the nitty-gritties of deduplication process have been worked out in different ways by different people depending on the things like the environment where deduplication is to be used, applications it is designed for, resources available, etc. The variations experimented and implemented are so many that these efforts put for deduplication are categorized based one several different classification criteria.

In this chapter, we present the various classification criteria found in the literature for categorizing different deduplication processes (Kaur, Chana, & Bhattacharya, 2018; Kulkarni, Douglis, Lavoie, & Tracey, 2004; Paulo & Pereira, 2014). Fig. 5−1 shows pictorial view of this classification in a compact form.

A brief layout of these classifications is shown below. Following the brief layout each of the classification criteria is detailed in subsequent sections in the same order.

Classification criteria:

- **a.** Granularity
  - **a.1.** File level
  - **a.2.** Chunk level
    - **a.2.1.** Fixed size
    - **a.2.2.** Variable size
    - **a.2.3.** Dynamic size
    - **a.2.4.** Hybrid
  - **a.3.** Object level
- **b.** Technique to handle duplicates
  - **b.1.** Aliasing for exact duplicate
  - **b.2.** Delta encoding

- **c.** Locality assumptions for efficiency
  - **c.1.** Temporal locality
  - **c.2.** Spatial locality
  - **c.3.** Both spatial and temporal locality
- **d.** Place
  - **d.1.** Server based
  - **d.2.** Client side
  - **d.3.** Source based
  - **d.4.** Target based
  - **d.5.** Network side
    - **d.5.1.** End-to-end redundancy elimination
    - **d.5.2.** Network-wide redundancy elimination
- **e.** Time
  - **e.1.** Inline
  - **e.2.** Offline
- **f.** Data format awareness
  - **f.1.** Content agnostic
  - **f.2.** Content aware
- **g.** Indexing and techniques to find duplicates
  - **g.1.** Fingerprint or hash index
  - **g.2.** Sparse index
  - **g.3.** Partial index
- **h.** Scope
  - **h.1.** Centralized
  - **h.2.** Distributed
    - **h.2.1.** Local scope
    - **h.2.2.** Global scope
- **i.** Data type
  - **i.1.** Text based
  - **i.2.** Multimedia based
    - **i.2.1.** Image based
    - **i.2.2.** Video based
- **j.** Storage type
  - **j.1.** Backup and archival storage based
  - **j.2.** Primary storage based
  - **j.3.** Random access memory based
  - **j.4.** Solid-state drives based

5.2 Granularity

The data deduplication process consists of comparison step where duplicates are identified before they can be removed to eliminate redundancy. An important aspect of overall design of

a data deduplication system is to whether the data is to be partitioned for deduplication or not. If it needs to be partitioned, then by which method data is partitioned. These partitions become the basic unit for comparison and then elimination and define the granularity of the process. This aspect significantly affects the efficiency and effectiveness of the whole process of deduplication. Granularity may be considered as the only decision parameter which primarily defines the deduplication algorithm (Mandagere, Zhou, Smith, & Uttamchandani, 2008). However, other parameters also play significant role in deduplication algorithm.

5.2.1 File level or single-instance storage

The simplest approach for granularity is to consider the whole file itself as the unit for deduplication which is the most coarse-grained granularity. Here, boundaries set by a file system for files are taken as that of chunks (Bolosky, Corbin, Goebel, & Douceur, 2000).

In file-level deduplication, hash values generated using algorithms like SHA-1 (N. I. o. S. a. T. NIST, 2015) are compared to find duplicates. If a duplicate entry for computed hash value for the file exists in the index table, the duplicate file is not saved but can be referred through the duplicate entry otherwise new file and the index are saved. File-level deduplication is popular to eliminate redundancy for duplicate data in storage, email, and cloud-based systems. File-level chunking is appropriate for many backup systems as they are file oriented and the partitioning overhead is removed resulting in reduced number of chunks (Policroniades & Pratt, 2004).

Like file-level granularity, single-instance storage (SIS) is method of data deduplication that works at the most coarse-grained granularity, that is, object level which may be file or email message. Various Microsoft application use SIS for eliminating redundancy due to duplicates. It is popular for deduplication in file systems, email servers, data backup systems and other storage-related computer systems. In this a common storage is maintained where files are stored. Whenever duplicate file is detected a SIS link is created which refers to file contents saved to the common store. SIS (Bolosky et al., 2000) is used in Microsoft Exchange 2003 (Microsoft, a) and 2007 (Microsoft, b) email system for deduplication. An email sent to many results in its multiple copies but only one in the primary recipient's mailbox is saved through SIS, in other recipients' mailboxes only the pointers of the email are saved. SIS can eliminate when entire files are duplicate but not large duplicate portions within similar files. Hybrid Email Deduplication System (HEDS) resolves this issue (Kim et al., 2012).

Other examples which uses file-level deduplication are like EMC Corporation's (EMC's) Centera (EMC, a) for reducing redundancies in storage, storage services used for cloud data such as (JustCloud), and (Mozy) for reducing latency in a client.

5.2.2 Chunk level

File-level deduplication is good to find redundancies when entire files are duplicate but not for redundancies due to duplicate data within files or when files are similar but not exactly duplicate. For such requirements, the file data is partitioned into segments or portions popularly called chunks in deduplication. These chunks are the basic unit for comparing and

eliminating duplicates. The approaches falling in this category can be further classified based on methods of partitioning as follows.

5.2.2.1 Fixed size

As a simple approach the data is partitioned commonly into chunks of fixed size also called blocks. There are many storage systems that works on fixed-size partition and thus, this deduplication approach naturally suits such storage systems (Hong & Long, 2004; Quinlan & Dorward, 2002). This deduplication after partitioning a file into the blocks of same size, generate hash index of these blocks and compares them to find redundant blocks. The fixed-size partitions of data also suit for other scenarios like cloud and computer networks that deals with data in predefined units. For example, in data networks, the data when transmitted is considered in fixed-size units called packets. Thus, deduplication for network data may fix the size of chunks to be that of packets.

When changes in data are processed and handled in smaller sizes this approach can offer higher processing rates and reduced CPU overhead (Constantinescu, Glider, & Chambliss, 2011; Policroniades & Pratt, 2004). Reducing the size of chunks may improve deduplication gain but increases processing overhead due to the increased size of metadata and fragmentation (Kaczmarczyk, Barczynski, Kilian, & Dubnicki, 2012; Policroniades & Pratt, 2004).

However, here an issue, referred as the offset-shifting or boundary-shifting problem (Eshghi & Tang, 2005), arises in finding duplicate contents. Suppose, the content of file is changed at the beginning and saved as a different file. In these similar files, whatever be the granularity whether whole file as single or fixed-size partitions or chunks, in the worst case, no chunks in these similar files will match.

Venti (Quinlan & Dorward, 2002) uses this approach for elimination of redundancy for archival storage. The hash keys generated using SHA-1 for fixed-size blocks are compared. The concerned file is then reconstructed with the help of the index represented as a hash tree.

Dropbox (Dropbox), a system popular for cloud storage also uses the same approach with a very large size of blocks, that is, 4 MB to reduce redundancy in network traffic and server storage. Hash values are generated using SHA256 (N. I. o. S. a. T.NIST, 2015). It uses two type of servers. Control servers keeps the metadata like block hash values mapped to the file data and storage servers saves the deduplicated blocks on Amazon S3 (Amazon).

5.2.2.2 Variable size

When deduplicating at chunk level, data may be partitioned into variable-sized chunks too. By keeping size of blocks a variable, solves the offset-shifting problem occurring due to their fixed sizes. It also helps to find more redundancy in the data. Size of blocks varies based on contents rather than defined by a fixed offset. Using an n digit predefined number called content-based divisor, the boundaries are marked for chunking of a file. For that, fingerprint (Rabin, 1981) is computed for each window of fixed-size sliding byte by byte through the file. The predefined divisor is compared with the last n digits of each fingerprint. The fingerprint that matches is considered as a boundary for chunking. The contents between any two subsequent boundaries thus identified defines a chunk. This approach generates variable-sized chunks.

Content-defined chunking (CDC) algorithm set the boundaries by content itself (Muthitacharoen, Chen, & Maziéres, 2001). A fixed content pattern is predefined to be the boundary of any chunk. Then by sliding a window over the data until the fixed content pattern used as divisor to define the chunk boundary is found. To prevent too small or too large sizes of chunks, the restriction for a minimum and a maximum size is imposed. Now suppose a byte is inserted in the beginning of one file and saved as another copy, then only the first chunk in two files will differ whereas the other chunks will match and get eliminated. In this scheme also, the boundary-shifting problem may still exist in large chunks formed due to boundaries identified due to maximum size threshold instead of content matching with divisor. A modified version of CDC called the two thresholds−two divisors (TTTD) algorithm imposes restriction on maximum and minimum chunk size as in CDC (Eshghi & Tang, 2005). However, to define chunk boundaries now it uses two types of fixed patterns as divisors instead of one while sliding the window of CDC. The first divisor is chosen as in CDC. Another pattern having more chances of occurrence is chosen as second divisor. Now, while sliding window as in CDC until first divisor is found, whenever the second divisor is found the last occurrence is recorded as a possible breakpoint. If no occurrence is found before the maximum size limit, the position of recorded last occurrence of the second divisor is considered as the chunk boundary. This way in TTTD, the significant reduction in boundary-shifting problem is possible.

The variable-sized chunking has been used for deduplication in backup systems (Dong et al., 2011; Dubnicki et al., 2009; Guo & Efstathopoulos, 2011; Lillibridge et al., 2009; Xia, Jiang, Feng, & Hua, 2011; Zhu et al., 2008) and file systems (Bonwick, 2009; Silverberg). As it provides fine-granularity, better utilization of storage space is achieved. The number of disk access are reduced to speed up the processing time through efficient caching schemes and locality-based disk layout. Data domain deduplication file system (DDFS) (Zhu et al., 2008) employs techniques like compacting summary vector kept in memory, stream-informed segment layout on the disk and sparse indexing to extradite search, better exploit spatial locality both for data and indexes and to reduce the requirement for RAM respectively.

5.2.2.3 Dynamic size

Instead of limiting the size by predefined thresholds as in variable-sized chunks, dynamic partitioning may also be used for increasing the variability of chunk size without impacting the deduplication gain. Fingerdiff, partitioning dynamically, creates larger chunks for unmodified portions and smaller chunks for modified regions (Bobbarjung, Jagannathan, & Dubnicki, 2006). Having subchunks of size small enough to capture small changes reduces chunk storage space, whereas the large size for unmodified data reducethe indexing costs. Two other algorithms which increases chunk size variability, breaking apart algorithm and the building-up algorithm, are presented in bimodal CDC (Kruus, Ungureanu, & Dubnicki, 2010). First, initially dividing data into larger chunks divides them further into smaller ones when required. Second algorithmstarting withsmaller chunks of the data stream combines them till the deduplication gain does not degrades. Another variant combines it with a statistical chunk frequency estimation algorithm (Lu, Jin, & Du, 2010).

5.2.2.4 Hybrid

The hybrid approaches have also been used in this context. Any of variable size chunking or file-level chunking may be selected adaptively based on either a static or dynamic policy (Kim, Song, & Choi, 2013; Min, Yoon, & Won, 2011). Min et al. (2011) choses out of file-level chunking and variable size chunking based on content. File-level chunking is chosen for compressed or encrypted files while variable size chunking is chosen for text files. HEDS (Kim et al., 2012) uses variable size chunking for larger objects and a file-level chunking for smaller ones.

5.2.3 Object level

Although, the chunking, whether fixed or variable size, can be used for any type of files, it is expensive for files having specific formats. Object-level deduplication can be used in such cases. A file is partitioned based on the format of a file in this approach. A few of such approaches, also called structure-aware data deduplication techniques, have been proposed that partitions data on the basis of structured formats or metadata of file (Kim et al., 2013; Li, He, Sengupta, & Aiyer, 2009; Liu et al., 2008; Yan & Tan, 2011).

According to a study (Meyer & Bolosky, 2011), cost of simple file-level deduplication is low, but it saves less space than block-level deduplication. Moreover, the study shows that if semantic knowledge of file structures is used, a greater number of redundancies can be eliminated at very little increment in overhead (Meyer & Bolosky, 2011).

It is obvious that use of fixed-size chunks requires comparatively less computational overhead than variable size chunks due to extra processing required to find boundaries. With the decreasing size of chunks, the number of indexes increases. The memory requirement increases but better deduplication gain is obtained. Variable-sized chunking is much better than others in terms of deduplication gain and worst in terms of processing time. In terms of index overhead, file-level deduplication is best while it varies in chunk level deduplication with their size. Thus, for files that are updated variable-sized chunking should be preferred. Variable size chunking is better to be used for server-based deduplication owing to their high-capacity to handle overheads. In contrast, file-level deduplication should be preferred to deal with duplication due to replicated files and when deduplication is client based.

5.3 Technique to handle duplicates

Technique to represent stored deduplicated data may differ in terms of degree of exactness in duplicates. Aliasing is the techniques used for elimination of exact duplicates whereas delta encoding eliminates nonexact but similar duplicates (Aronovich et al., 2009; Nath et al., 2006; Policroniades & Pratt, 2004; Quinlan & Dorward, 2002). Every method in different categories based on granularity discussed in Section 5.2. can be combined with either aliasing or delta encoding. However, in these deduplication techniques, the optimal size of chunks will differ. The chunk size that is optimal and with which the deduplication gain is not reduced is preferably larger in the delta encoding as compared to aliasing. For both types of

deduplication, metadata structures are required to abstract the sharing. The data is visible as shared in physical view but from the logical view this sharing is hidden. Many storage systems that work at file-level granularity use tree structures for mapping files to their chunk addresses which are referred for restore operations (Quinlan & Dorward, 2002). On the other hand, systems that deal with data at block-level granularity require metadata to map logical blocks to storage addresses (Chen et al., 2011; Hong & Long, 2004). Additional structures may also be required to process read requests for content of data that does not have an associated signature so that it can be searched directly in indexing metadata. The chunks that are not being referenced after their deletion or modification must be passed over for garbage collection (Guo & Efstathopoulos, 2011). Efficient I/O translation methods and reference management are required for reducing the I/O latency and recover free space, respectively.

5.3.1 Aliasing for exact duplicate

Aliasing is used to eliminate exact duplicate chunks in chunk-based deduplication. Each deduplicated chunk is made to refer a single chunk physically in corresponding metadata structure so that I/O requests for aliased chunks can be processed correctly. Aliasing is used in most deduplication systems like Microsoft SIS (Bolosky et al., 2000) and Venti (Quinlan & Dorward, 2002).

5.3.2 Delta encoding

Delta encoding is used when the chunks to be eliminated are similar but not identical. When many different similar chunks are found, only one of them referred as the base chunk is fully stored. Only the part of other similar chunks which is distinct from base is stored. This different portion of the similar chunk is called a delta or diff. The corresponding delta or diff is applied on base to restore any other similar chunks. Metadata which maps files and blocks to chunks must have mechanisms to locate the base chunks and concerned deltas. To encode delta or diff a pair of similar chunks is considered. The deduplication factor will be higher when similarity is considered pair of chunks is more. Therefore, the mechanisms which identify similar chunks and pairs within them for elimination and delta encoding are key for deduplication. Its performance also depends on the algorithms used for delta encoding (Hunt, Vo, & Tichy, 1998). Delta deduplication has been applied on large collections of files to study their efficiency (Douglis & Iyengar, 2003; Ouyang, Memon, Suel, & Trendafilov, 2002). The deduplication system that uses only delta encoding was introduced by IBM Protect Tier (Aronovich et al., 2009).

5.3.3 Hybrid

A few systems combine both techniques. Aliasing is used to eliminate exact duplicates and delta deduplication is used for eliminating redundancy in chunks which does not match exactly with any other chunk but match partially with other chunks (Shilane, Wallace, Huang, & Hsu, 2012; You, Pollack, & Long, 2005). A combination of these two techniques can also be combined with chunk compression to improve their performance (Daehee et al., 2017; El-Shimi et al., 2012; Gupta, Pisolkar, Urgaonkar, & Sivasubramaniam, 2011).

Aliasing is faster than delta deduplication. Processing overhead and time required to restore is less in aliasing as processing for deltas is neither required during elimination nor during restoration of chunks (Burns & Long, 1997). The size of chunk which needs to be deduplicated can be increased without reducing the deduplication gain when delta encoding is used (Aronovich et al., 2009; You & Karamanolis, 2004).

5.4 Locality assumptions for efficiency

Locality refers to the assumption that a certain element of data if referred once is expected to be referred again in the near future and a set of data elements which when referred are generally referred together within short duration. Caching strategies and on-disk layout in storage systems are improved by exploiting locality assumptions in storage systems. Similarly, locality assumptions can be exploited to improve deduplication gain (Clements, Ahmad, Vilayannur, & Li, 2009; Dubnicki et al., 2009; Yang, Feng, Niu, & Wan, 2010a).

5.4.1 Temporal locality

Temporal locality refers to chunks that are close to each other with respect to time. The assumption here is when a chunk is referred once, it is expected to be referred again in near future. Least-recently used (LRU) policies are usually used for eviction of data items to exploit temporal locality (Quinlan & Dorward, 2002).

5.4.2 Spatial locality

Spatial locality refers to the assumption that a set of chunks which occur once are expected to appear again in the same order in the near future. That means a set of chunks is expected to be occurring together during a sequence of accesses to data. To exploit spatial locality, a group of chunks expected to be occurring together is stored in their original order physically. So that, when any one of the signatures out of the chunks in a spatially local group is referred, the signatures of all chunks of that group are brought to the cache to avoid expected more disk accesses for the chunks in same group (Zhu et al., 2008; Rhea, Cox, & Pesterev, 2008).

5.4.3 Both simultaneously

As temporal and spatial locality refers to different aspects of locality, they can both be exploited simultaneously also (Srinivasan et al., 2012).

5.5 Place

The data deduplication methods may also be classified according to the place they are performed. For example, when data is supposed to be communicated over a network then different places for deduplication may be the server side, client side, or the network, that is, routers, etc. From another perspective, deduplication may be done at place where data is

created or where it is stored. The earlier called the source of data while later called the target. All these classes are described as follows.

5.5.1 Server based

For a long time, tape storage has been popularly used in high-performance and dedicated backup systems for quickly backing up large amounts of data. However, to substitute disks in place of tapes in backup systems, the server-based deduplication has emerged (EMC, b; NEC; Symantec, a). In server-based deduplication, clients send data to servers that need to be backed up and data is deduplicated at server. The system at client side is lightweight which only sends data to servers. The server bears most of the overhead due to computation and memory in deduplication. In the process, first a file is sent to a server, then segmented into mainly variable size chunks at server. Further, the duplicates are identified by computing indexes and comparing them with saved indexes.

Server-based deduplication is good to find redundancies significantly. However, it poses heavy load on network by incurring excessive redundant traffic being delivered to servers to be deduplicated. Moreover, as servers bear large CPU and memory overhead, they require efficient layouts for in-memory and on-disk chunks. DDFS (Zhu et al., 2008) provides a solution for backing up within a limited backup window.

5.5.2 Client side

It refers to those deduplication systems, where uniqueness of data is checked at client side itself and only chunks which are not duplicate are delivered to servers. The indexes of data to be deduplicated are either compared with local indexes saved on client side or backup agent is used to compare them with remote indexes saved on server. In this deduplication, the redundant traffic is removed from network but overhead for the client increases (EMC, c; Symantec, b). In pure client-based deduplication where redundancy is eliminated at client side without any collaboration with any server the redundant data transferred from different clients increases the network traffic. Thus, collaboration with a server is preferred. The client here partitions data file into chunks, computes, and sends chunk indexes to a server. Then, server replies with indexes of unique chunks not present on server so that only those can be transferred back by client. Similarly, low-bandwidth network file system (LBFS) (Muthitacharoen et al., 2001) adds the communication protocol which sends indexes before the actual data to a server to reduce the space overhead at the cost of increased latency. Overall, this deduplication strategy suffers from the problems due to clients having limited capacity to bear extensive overhead of deduplication process.

5.5.3 Source based

This class refers to deduplication systems, where the elimination of redundant data is performed at the data source before sending it to target. A file system can be considered as a source in this perspective, where the generation of data starts. New files in such systems are periodically scanned, and hashes are created and compared with existing hashes. Pointer of

duplicate file, if found, is made to point to the old file but is considered as separate entities. Redundant copies of such duplicate files are thus not written on to the storage. Copy-on-write mechanism is used to handle modifications in such duplicated files. This process is hidden from users and backup systems. Thus, backup systems will include the copies of such duplicate files in the content which results in larger sizes of backups than the source data (Microsoft, c). To reduce comparison overhead in this type of deduplication, it can be attached with copying operation by default.

File systems define specific ways of linking such duplicate files. Linux names them as the ref link MacOS as clone file (Becker, 2009). The method of delta encoding is preferably used if such duplicate files are modified copy-on-write (Linux, 2017).

5.5.4 Target based

Target-based deduplication is the process of removing duplicates at the location for which data was generated. For example, consider a server connected to a storage system such as storage area network (SAN) and network-attached storage (NAS). In this case, these network-based storage systems would be a target location for which data was generated at the server. The target-based deduplication will be performed for such network storage systems. SAN and NAS store data at the block level and file level, respectively. Thus, deduplication in SAN would be chunk based whereas in NAS it would be file based. In this case, the server is the point of data generation but not aware of any deduplication. Another example of target-based deduplication would be performed in a backup store such as a data repository or a virtual tape library (Daehee et al., 2017).

5.5.5 Network side

This refers to approach when deduplication takes place on the entire network rather than a server or client and generally called as redundancy elimination (RE). It can further be classified as follows (Daehee et al., 2017).

5.5.5.1 End-to-end redundancy elimination

Here, the redundancy is removed at two end-points of network paths in WAN optimizers (Cisco; Citrix; Riverbed). Two devices are installed for this purpose, one before router at sending end and another after the router at receiving end. The file is partitioned into chunks. Their indexes are saved on to the cache before the transfer. These indexes and chunks are later saved on the disk. Suppose two clients try to send the same files. The file received first is sent to the server after compressing and saving its indexes and chunks on to the cache. When the same file is sent another time, indexes of the duplicate chunks are identified by their comparison with those saved previously at sending device, these are then replaced by the indexes so that size of such encoded packet is reduced. At receiving device, the file is reassembled using the chunks corresponding to the indexes in the encoded packet and this reconstructed file is routed to the specific server.

A LBFS (Muthitacharoen et al., 2001) does not transfer the data that already exists in the file system on server or the cache at client side to exploit the cross-file similarities. This way latency and network bandwidth is reduced. Both client and server in LBFS maintain a chunk index database. When a client writes a file on server, LBFS works as follows. As soon as the file is closed by user, client creates a temporary file on a server through remote process call (RPC). The file is partitioned into chunks and their hash keys are computed by the client. Then it tries to write on server file through RPCs with hash keys. Server replies to client again through an RPC with hash keys of nonexisting chunks so that only those can be sent back. The server reassembles the entire file with all chunks including the ones saved previously and those received newly. Since it is client that partitions the file and saves the indexes of those chunks, the LBFS is considered a client-based deduplication. It is also considered as an end-to-end RE since the duplicate chunks and indexes are held only at end-points (i.e., the client and the server) whereas through the network only unique chunks are transferred.

5.5.5.2 Network-wide redundancy elimination

It is fixed-size chunk deduplication performed at routers in place of hosts. First, the considered packet is captured in a router on the fly or at network end-points. Indexes (Rabin, 1981) for the incoming packet payload is then computed. Further, these indexes are compared with those of saved packets and duplicate ones are eliminated by encoding into small shims. These are later decoded before coming out from the network.

The network-wide RE removes redundancies of packet payloads considering byte strings in a payload as the unit. Deduplication is done through RE devices. These are some special routers (or switches). Through a sliding small window on the received payload computing the fingerprints of each window, duplicate fingerprint(s) are found by comparing to those in the local cache. The maximal duplicate region is identified by expanding duplicate fingerprints in the left and the right sides which matches with a packet in the local cache. This maximal duplicate byte region is replaced by a small shim header consisting a fingerprint and byte offsets. Such an encoded payload is decoded for reconstruction by a RE device on a path and delivered to a server.

By eliminating duplicate network traffic, bandwidth in links between an encoder and a decoder in network-wide RE is saved (Anand, Gupta, Akella, Seshan, & Shenker, 2008; Anand, Sekar, & Akella, 2009; Spring & Wetherall, 2000). However, computational overhead increases due to sliding fingerprinting and memory overhead increases due to cache for saving the packets. More important is that the server is supposed to run expensive process of deduplication again after performing entirely at a decoder before reaching the server. Hence, deduplication is done redundantly twice, first in the network and then on the server.

5.6 Time

The deduplication may also be classified based on the time which refers to the stage (before or after the storage) when duplicate data is detected and removed. If performed before during the process of storing, it is called inline and called offline or postprocess

otherwise. Whether deduplication should be done inline or offline is often heavily debated (Daehee et al., 2017; Paulo & Pereira, 2014).

5.6.1 Inline

Inline deduplication refers to removal of redundancies before storing data on disk (Debnath, Sengupta, & Li, 2010; Dubnicki et al., 2009). Workloads are of two types, primary like user directories, email, databases, and secondary like backups and archives. It has been used for both the primary (El-Shimi et al., 2012; Srinivasan et al., 2012) and secondary (Debnath et al., 2010; Lillibridge et al., 2009; Zhu et al., 2008; NEC) workloads. It is also called in-band deduplication as it can only be performed by intercepting the I/O or write requests in their critical path.

For primary workloads, deduplication executes direct on the I/O path. After intercepting the requests, the chunk boundaries and signatures, if necessary, are identified. The index is investigated to find duplicate chunk. If the chunk is found, then it is either shared or delta encoded. Only the data and indexes which are not duplicate are saved to storage and cache. The latency of the operations is critical for the applications using primary workloads. Thus, in such systems with the use of in-memory cache, the disk I/O requests are reduced in deduplication.

This deduplication is used in several backup and archival storage (Quinlan & Dorward, 2002; Rhea et al., 2008) as well in file systems (Ungureanu et al., 2010; Zhu et al., 2008). This deduplication is used in content-addressable storage (CAS) systems (Quinlan & Dorward, 2002; EMC, a). Some file systems use this deduplication for primary storage (Bonwick, 2009; Silverberg).

Through this deduplication the overhead of extra space required for deduplication is eliminated. However, it adds up the overhead of latency in a write path. In fact, this is one of its main drawbacks as most of the processing is done in the write path. The latency of on-disk index operations is major bottlenecks in the performance and full index can be loaded to RAM to solve this problem for a limited size of the datasets.

In scenarios like primary storage systems this overhead may not be acceptable due to strict I/O latency requirements. An example is iDedup, where both temporal and spatial locality is exploited for improving processing time (Srinivasan et al., 2012). To exploit spatial locality, it reduces the additional seek time for files read sequentially by performing selective deduplication. In order to exploit temporal locality, dedup metadata is maintained using LRU eviction policy in cache.

ChunkStash (Debnath et al., 2010) saves chunk metadata to flash memory rather than disk to increase inline deduplication throughput as flash memory is faster. The metadata consists of the key as chunk index and location as the value. ChunkStash uses in-memory hash tables too. These are created with the help of cuckoo hashing (Pagh & Rodler, 2004). The size of metadata signature is reduced to reduce the RAM requirement.

HYDRAstor (Dubnicki et al., 2009) stores blocks distributed throughout a node grid using a distributed hash table (DHT). HYDRAstor uses inline deduplication and achieves efficient utilization, scalability, system performance, and fault tolerance by deduplication, DHT, load balancing, locality and prefetching, and by data resiliency respectively.

5.6.2 Offline

Offline deduplication is one where data is scanned in the background to eliminate duplicates after writing it as it is to the storage (Alvarez, 2011; EMC, 2009; IBM, 2002). It is also called as postprocess or off-band deduplication. As it does not intercept to delay the process during write critical path, the I/O latency is improved. Modifications required in the I/O layer are reduced at the expense of more resources required to scan the storage to search updated chunks for elimination. Several modifications have been proposed to avoid scanning of storage at the cost of negligible overhead in I/O operations (Clements et al., 2009; Hong & Long, 2004).

Offline deduplication has several other drawbacks as compared to inline deduplication apart from requiring more space on disk to hold entire data before the process starts. Its execution depends on availability of idle time on the system. Also, disk bandwidth is unnecessarily consumed as duplicate data on disk must again be loaded to memory for deduplication.

The offline implementations may offer policy-based operation which provides benefits like deferring optimization on files still being used, or processing files based on type and location. As the advantage of inline deduplication, less storage and network traffic are required because duplicate data is never stored or transferred. However, the drawback is computationally expensive hash calculations which may reduce the storage throughput.

5.7 Data format awareness

5.7.1 Content agnostic data deduplication

It refers to the deduplication which does not require awareness of application-specific data formats. Deduplication strategies like fixed block chunking or file-level chunking do not require to know the content or their data formats for elimination. The alpha encoding does not require the knowledge of data formats used in the duplicated files as the entire file or object needs to be eliminated.

5.7.2 Content-aware data deduplication

It refers to the data deduplication method that leverages on the knowledge of application-specific data formats. For example, variable-length chunking makes use of knowledge of content and data formats to decide variable-length boundaries dynamically. The object-level chunking may also be grouped in this category requires to know the data formats to identify objects. A file is partitioned based on the format of a file in this approach. In the delta encoding, similar chunks or files are deduplicated such that only common section of two duplicate items is saved entirely whereas remaining uncommon part is saved separately as delta encodes. So, this type of method will be benefited from the knowledge of data formats used in the duplicate elements to be eliminated.

5.8 Indexing and techniques to find duplicates

Efficiency of process to find duplicates depends on the choice of data structure that is used for indexing. Mostly the indexes are built based on summarized content (Bolosky et al., 2000;

Quinlan & Dorward, 2002). However, some systems build indexes for the entire chunk content (Arcangeli, Eidus, & Wright, 2009). The reduction in the size of indexes not only reduces space costs but also speeds up the comparison process in deduplication. Different deduplication techniques may be also be classified based on different indexing techniques used as follows.

5.8.1 Fingerprint or hash index

As a popular method for summarizing the content is to generate hash values to represent it. The identity signatures thus created can be successfully used to find exact duplicates within the chunks and even before aliasing them which prevents hash collisions (Rhea et al., 2008). However, computation required for hashing increases the computational overhead and raises problems in some systems (Chen et al., 2011). Inline deduplication minimizes the chance of hash collisions. However, the latency of I/O operations and the entire process is increased due to byte-by-byte comparison of chunks (Quinlan & Dorward, 2002).

In the process a set of Rabin fingerprints are computed for chunks using a sliding window. Then, the similarity between chunks is determined by comparing a defined number of common fingerprints (Manber, 1994). Calculation of Rabin fingerprints is linear. Moreover, since their addition is distributive, a sliding window can be efficiently used to scale up the calculation of the fingerprints for variable-sized chunks (Broder, 1993; Rabin, 1981). A set of heuristics is further proposed to coalesce a group of similar fingerprints into super fingerprints to reduce the number of fingerprints. When these super fingerprints match for two chunks, it indicates high resemblance between them (Broder, 1997). The indexing data structure is built using these signatures or fingerprints.

When all signatures are indexed, potentially all duplicate data elements can be found (Bolosky et al., 2000; Quinlan & Dorward, 2002). However, in this way the size of the index may become larger than the space allocated in RAM. Further, storing the index in disks will deteriorate the throughput of deduplication very badly (Quinlan & Dorward, 2002).

5.8.2 Sparse index

A sparse index is used against the full index to reduce the overhead on space. Here, single entry of index refers to a group of stored chunks may be called a segment. To build a sparse index, first the chunks are grouped together and in place of identity signatures their similarity signatures are calculated to refer these groups together (Lillibridge et al., 2009). This leads to reduction in the size of primary index thus formed which can be kept in RAM. Further, next level of indexing can be done for each segment using identity signatures for grouped chunks. This index can be stored on disk. For deduplication of a new segment, identity signatures of only most similar segment are brought to RAM. This helps in achieving acceptable deduplication gain with less utilization of RAM. Using sparse indexes, deduplication can be scaled up for larger data sets but with limited deduplication gain (Bhagwat, Eshghi, Long, & Lillibridge, 2009). Chunk sizes can be reduced with reducing footprint of RAM to scale for larger datasets.

5.8.3 Partial index

An index is partial when all the stored unique chunks are not indexed. However, each index entry maps to only one unique chunk. It is another alternative to keep the RAM utilization within a specified range at the cost of incomplete deduplication (Guo and Efstathopoulos, 2011; Chen et al., 2011; Gupta et al., 2011; Kim & Choi 2012). A policy is predefined for eviction like LRU policy.

5.9 Scope

To improve throughput or gain or both, the deduplication can be done over a set of nodes. Such distributed deduplication can also be scaled for larger datasets and more clients. The overhead in designing such deduplication increases because routing mechanisms need to be defined across different nodes. So, it presents another possibility of categorizing the deduplication technique based on whether they make use of multiple machines present in the cluster or not.

5.9.1 Centralized

This category will include all the deduplication techniques that are performed on a single node. Whether the supported data is from a single or multiple clients does not matter (Quinlan & Dorward, 2002; Rhea et al., 2008). Thus, even if the system works on a cluster of independent machines having independent resources, this approach does not take any advantage in processing by eliminating duplicate chunks across remote nodes.

5.9.2 Distributed

The distributed systems assume that deduplication is working on a cluster of independent machines with individual CPU and RAM. The throughput can be increased through parallelism in distributed deduplication. Parallelism can also help in handling node failures and increasing availability (Bhagwat et al., 2009; Cox, Murray, & Noble, 2002; Douceur et al.). When different nodes in distributed systems can access a shared storage device abstraction, the metadata information can be shared between nodes through the shared storage device otherwise it can be passed across the network to different machines for sharing (Clements et al., 2009; Kaiser, Meister, Brinkmann, & Effert, 2012). Moreover, in a distributed system, distinct tasks may be assigned to distinct nodes. A few nodes are assigned the task of partitioning the data and computing their signatures while others may process query and update indexes (Yang et al., 2010a,b).

In this perspective the process of deduplication can further be distinguished based on the scope in which duplicates are matched and represented after being removed.

5.9.2.1 Local scope

It refers to distributed deduplication when each node performs deduplication only locally. No elimination is done for duplicate chunks present across distinct nodes globally. Some systems in this category have separate index saved on each node and deduplication is

performed independent of other nodes in the system (You et al., 2005). Some systems increase the cluster deduplication gain through routing intelligently only those files or set of chunks which are similar to a specific node (Bhagwat et al., 2009; Dong et al., 2011).

5.9.2.2 Global scope

It refers to the contrasting option of eliminating duplicate chunks across the whole cluster globally. For global identification of duplicates and their elimination, an index mechanism which can be accessed by all nodes in the cluster is required. The global index can be either centralized which suffers from scalability and fault tolerance issues (Hong & Long, 2004), or decentralized global indexes may be used which poses increased overhead of lookup and update operations (Clements et al., 2009; Douceur et al.; Dubnicki et al., 2009; Hong & Long, 2004). Globally distributed deduplication saves more space than local approaches at the cost of additional overhead required to access index. This may increase I/O latency to unacceptable level in primary storage systems (Ungureanu et al., 2010).

5.10 Data type

Data types are defined by the storage formats and implicit characteristics of data. Here it refers to data represented in the form of text or images and videos. The matching process in deduplication techniques differs significantly for these. Finding duplicate in heterogeneous data on Internet and cloud storage systems is major challenge where the requirement of deduplication is increasing with growing social networking platforms. Thus, deduplication techniques applied on data primarily existing on cloud can also be categorized based on data type namely text, image, or video.

5.10.1 Text-based deduplication

In this technique duplicates in the data represented as text are identified through byte-by-byte comparison. We have already seen many efforts made in this direction, which can broadly be further categorized based on various design issues such as granularity, locality, indexing, and security.

5.10.2 Multimedia-based deduplication

Multimedia deduplication is meant for images or videos. Thus, we have two different classes of deduplication: image-based deduplication and video-based deduplication. Deduplication in video and image started primarily in 2011 (Katiyar & Weissman, 2011; Ramaiah & Mohan, 2011). To detect similarity in multimedia-based deduplication hash values or image fingerprints based on features extracted from images and videos are calculated and compared using some threshold on the difference. Several methods to extract the information (features) from images or videos are there in developed images and videos retrieval systems. The overall performance including scalability and high accuracy in multimedia deduplication depends heavily on the feature extraction algorithm.

5.10.2.1 Images deduplication

The image deduplication may be used to eliminate redundancy in exact duplicate images or near-exact duplicate (similar) images. Near-exact duplicate image can be obtained by modifying or copying an image. The images can be modified due to operations such as adding noise, cropping, scaling, rotation, and compression. There are different techniques to find duplicate images that differ in their accuracies.

It is more complex to store the duplicate copies of images when they are not exactly the same. First the centroid is selected for all similar images. Then, centroid image and the near-exact image transformation are stored separately. A transformation matrix is used to store image transformations. Although, the technique for extraction of feature in images remains essentially same whether it is exact and near-exact image deduplication but the storage for near-exact images differs as image transformation is stored separately in way like delta encoding.

Deduplication techniques for images differ inextraction methods used for image features, hashing algorithm used for indexing the images and methods to measure distance between images or videos for measuring similarity.

5.10.2.2 Video deduplication

For video deduplication a video is fragmented into a set of frames. These frames are deduplicated like images (Nie, Hua, Feng, Li, & Sun, 2014). Hash values for each key frame of video iscalculated using visual features or descriptors. Visual features help in detecting duplicate sequences between the frames of concerned video and library (Nie et al., 2014). Video deduplication requirement slightly differs from image as subsequent frames of a video are expected to be very similar or duplicate and thus duplication ratio in videos is expected to be more than images. Video deduplication also refers to exact and near-exact duplicates like image deduplication.

5.10.2.3 Examples of multimedia deduplication

Chen et al. constructed B + tree index based on gray block features of images and extract the edge information of the images using Haar wavelet algorithm for optimizing the accuracy (Chen, Wang, & Tian, 2013).

Ramaiah et al. extract features using histogram refinement a content-based image retrieval (CBIR) strategy (Ramaiah & Mohan, 2011). It is used to eradicate ration cards identified duplicates based on their images. To speed up the process district level and K-means algorithm for clustering is used.

Zargar et al. again use CBIR to eliminate duplicate electricity bills based on their images (Zargar, Singh, Rathee, & Singh, 2015). These images are partitioned into blocks using block truncation coding (BTC). Images are clustered based on size for better space utilization.

Hua et al. propose SmartEye, an in-network coarse-grained deduplication (Hua, He, Liu, & Feng, 2015). The features are extracted from images using a variant of principal component analysis technique. Feature correlation is used to identify similarity in images. Deshmukh et al. use MapReduce and Pearson correlation techniques for fast identification of a duplicate image

resulting in more efficient and reliable system (Deshmukh & Lambhate, 2012). Rashid et al. compress the image and encrypt it partially to prevent sit from cloud service provider (CSP) without extra computational overhead (Rashid, Miri, & Woungang, 2016). Wavelet coefficients is used to generate unique hashes so to have secured image deduplication. Specialized local sensitive hashing named local-difference-pattern (LDP) is used.

Shen et al. proposed a fast and robust duplicate video detection system (Shen, Zhou, Huang, Shao, & Zhou, 2007). It detects duplicates using K-nearest neighbor algorithm.

Naturel et al. proposes video deduplication for television broadcast (Naturel & Gros, 2005). The discrete cosine transform (DCT) of the video is fragmented into shots or frames.

Katiyar et al. propose an application-aware video deduplication system (Katiyar & Weissman, 2011).

Li et al. uses block-level deduplication (Li, J., Li, & Jin, 2016). The disintegrated frames are further partitioned into fixed-size blocks. After encryption they are uploaded on cloud. Redundant blocks are eliminated at server side.

Velmurugan et al. speeded up feature extraction and used K-dimensional (Kd) tree to index the features and find similarity between them (Velmurugan & Baboo, 2011). Dong et al. proposed a scale in variant transform to extract features (Dong, Wang, Charikar, & Li, 2012). Nonexact duplicates are detected using query expansion method.

5.11 Storage type

Existing deduplication systems may be re-grouped by storage type also. As compared to earlier presented classification, the classification based on storage type does not specify any criteria which define some algorithmic aspect of the deduplication. However, the different storage system poses different challenges which need to be handled differently in deduplication. The requirements and restrictions are different for different storage environments. Thus, the deduplication design specifically what combinations of them are suitable depends considerably on the storage system being targeted.

5.11.1 Storage for backup and archives

Storage systems used for archives and backup pose some similar requirements whereas they differ on some other issues. For example, they both have similar assumptions regarding data immutability, and both prefer throughput over latency (Yang et al., 2010b). On the other hand, backups are expected to be restored and deleted more frequently than archives. Rather in some archive systems data deletion is not even possible (Quinlan & Dorward, 2002). Archival and backup production storage systems differ considerably in terms of duplication ratios too (Meister & Brinkmann, 2009; Meyer & Bolosky, 2011; Quinlan & Dorward, 2002; You & Karamanolis, 2004; You et al., 2005).

Microsoft SIS (Bolosky et al., 2000) and Venti (Quinlan & Dorward, 2002) introduced deduplication in backup and archival systems. SIS backs up Windows images using offline deduplication system. However, Venti introduced an inline deduplication CAS for nonerasable and

immutable archival data. It does not exploit locality of data due to which frequent access to disk results into index lookup problem. Different techniques have been used to address this problem like different designs are used for indexing (Eshghi, Lillibridge, Wilcock, Belrose, & Hawkes, 2007), spatial locality is exploited (Guo & Efstathopoulos, 2011; Lillibridge et al., 2009; Shilane et al., 2012; Zhu et al., 2008), and the index is stored on SSDs (Debnath et al., 2010; Meister & Brinkmann, 2010).

Hash-based directed acyclic graphs (HDAGs) optimizes the representation of directory trees and concerned files by placing them together. The comparison of directories required to find duplicates can be done efficiently using the HDAG structures. Compacting the index which keeps the signatures of chunks allows it to be kept in RAM boosting lookups significantly (Eshghi et al., 2007).

Another solution for the issues of bottle neck due to index lookup and high RAM consumption is the partial or sampled indexes. Only a subset of signatures can be kept in cache which is evicted upon reaching a certain RAM utilization threshold (Guo & Efstathopoulos, 2011).

A peer-to-peer deduplication was introduced as distributed deduplication in Pastiche (Cox et al., 2002). Different nodes cooperate among themselves to back up their data. The nodes choose other remote nodes based on their proximity in network and similarity of data to back up. This deduplication is performed inline. Each node having its own independent CAS sends only the new chunks to the nodes having the most similar content. It reduces overhead on both network bandwidth and storage space. Deduplication is performed only on a definite set of nodes with no guarantee of eradication of duplicate from the entire cluster.

This problem partial deduplication is handled in Farsite, an inline global deduplication system (Douceur et al.). It is a system which focuses on deduplication across the cluster. The virtually centralized index is represented using a distributed data structure which is accessible to all nodes in cluster. It maps unique chunk signatures to particular nodes. All similar chunks are sent to the same nodes for local elimination. The exact cluster deduplication is achieved.

Parallel local deduplication is used to avoid global indexes and their issues to increase deduplication throughput at the cost of gain (Bhagwat et al., 2009; Dong et al., 2011; Fu, Jiang, & Xiao, 2012; Xia et al., 2011; You et al., 2005). A large-scale archival storage functions in such systems by routing the stored files to specific cluster nodes according to their signatures. Each node with its own processor, disk and memory, partitions the files into variable-sized chunks. Duplicates are eliminated locally using both aliasing and delta deduplication. Routing of files may be done according to their similarity to specific nodes as follows. First a backup node, possibly have less load, is chosen to start the backup process for a file. The similarity of files is found by calculating their identity signatures at backup node. Then, based on these, the correct backup node having similar data is identified to which the chunk is routed. Indexes are maintained at two levels. Index of file similarity and identity signature is stored in RAM. Index of chunk identity signatures is stored on disk. The primary index is compared to search for a similar file before routing. Similarity in the identity signature is then to determine if the entire file is duplicated. If whole file is not duplicate, the most similar file is identified. The deduplication then is performed at chunk level by bringing into the memory the chunk identity signatures of similar file and comparing with those of the new

file. A Sparse index implementation is used for this. The RAM footprint is reduced at the cost of deduplication gain. In this system several backup files can be eliminated in parallel.

The CAS systems used for backup and archival can also be used for file system deduplication but at the expense of reduced I/O performance (Liguori & Hensbergen, 2008; Ungureanu et al., 2010). A lot of research has happened in the area of archival and backup deduplication and is continuing. This research focuses mainly to optimize deduplication in terms of both the deduplication gain and I/O throughputs. A common assumption prevails in such research that data is immutable and I/O throughput is preferred over latency. The primary storage systems do not have such an assumption. Thus, deduplicating file systems using deduplication systems meant for backup and archival does not perform well in terms of random I/O operations.

5.11.2 Primary storage

With the popularity of cloud computing, interest in live volume deduplication is growing. Some of the assumptions of deduplication systems used for backup and archival data does not hold good in primary storage deduplication. Requirements for I/O latency are strict here for performance. The expectation from a primary storage deduplication is that I/O performance is not degraded after it is implemented. Data is expected to be modified frequently against write-once assumption of archival and backup systems. The copy-on-write technique may be required to prevent updates on aliased data. Moreover, references need to be managed efficiently to track frequent changes in chunk references (Hong & Long, 2004). The proportion of duplicate data is generally lower in primary storage as compared to backup storage (Meyer & Bolosky, 2011). However, that can be higher in general purpose primary VM volumes in large clusters (Clements et al., 2009; Jin & Miller, 2009).

5.11.3 Random access memory based

The assumption in RAM deduplication systems is different than both above discussed storage systems. The possibility of finding duplicates within applications or VMs is only in the same server. Thus, RAM deduplication is performed locally and centralized. Its performance depends on how efficiently the copy-on-write and references are managed as the memory pages change very frequently. RAM deduplication is used to eliminate the redundancy which is not eliminated even by sharing the content through forked processes or shared libraries. The memory thus saved can be used for launching more applications or VMs. Through RAM deduplication memory consumption may be reduced by 33% (Waldspurger, 2002).

The Disco virtual machine monitor (VMM) is one such deduplication system used for sharing memory pages of different VMs (Bugnion, Devine, & Rosenblum, 1997). VMware ESX is based on content-aware approach that performs nonintrusive memory scans to find duplicates (Waldspurger, 2002).

5.11.4 Solid-state drives based

SSDs are improved the performance of random I/O operations drastically. The page is considered the unit for these operations whose size is usually fixed at 4KB. However, deletion is

done in the units called erasure block having 64−128 pages. An erasure block having all unused pages can only when be erased. Since updates cannot be done in place, old version of a block must be deleted when modified. However, there is a limit on number of erase operations that can be performed on a block which limits the lifespan of drive posing a major bottleneck of this technology (Chen et al., 2011).

Deduplication has also been used within SSDs which is still emerging. It raises interesting new challenges. New metadata structures and hashing algorithms must be designed to overcome limitations of DRAM space and computational power. SSD partial indexes can detect few duplicates and depend heavily on locality assumptions. However, on better note as compared to other storage environments, SSD partial indexes decrease RAM requirements and data fragmentation does not exist due to efficient random read requests in SSDs (Kim & Choi, 2012).

5.12 Conclusion

The list of different classification criteria discussed here may not be comprehensive and there may exist few other perceptions for classification. However, most relevant and classifications perceived broadly in literature for the different deduplication techniques have been included in this chapter. Apart from simply specifying different classes the chapter provides considerable details and variations that exist for a class using the examples of specific implementations existing. This provides a fair understanding of algorithmic and processing aspects of deduplication as a whole.

References

Alvarez, C. (2011). NetApp deduplication for FAS and v-series deployment and implementation guide, TR-3505, NetApp.

Amazon, Amazon simple storage service. <http://aws.amazon.com/s3/>.

Anand, A., Gupta, A., Akella, A., Seshan, S., & Shenker, S. (2008). Packet caches on routers: the implications of universal redundant traffic elimination. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*.

Anand, A., Sekar, V., & Akella, A. (2009). SmartRE: an architecture for coordinated network-wide redundancy elimination. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, Barcelona.

Arcangeli, A., Eidus, I., & Wright, C. (2009). Increasing memory density by using KSM. In: *Proceedings of the Linux Symposium*.

Aronovich, L., Asher, R., Bachmat, E., Bitner, H., Hirsch, M., & Klein, S.T. (2009). The design of a similarity based deduplication system. In *Proceedings of International Systems and Storage Conference (SYSTOR)*, New York, NY.

Becker, J. (2009). *The reflink(2) system call v5*. <https://lwn.net/Articles/335380/>.

Bhagwat, D., Eshghi, K., Long, D.D.E., & Lillibridge, M. (2009). Extreme binning: scalable, parallel deduplication for chunk-based file backup. In *Proceedings of International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Washington, DC.

Bobbarjung, D. R., Jagannathan, S., & Dubnicki, C. (2006). Improving duplicate elimination in storage systems. *ACM Transactions on Storage*, *2*(4), 424−448.

Bolosky, W.J., Corbin, S., Goebel, D., & Douceur, J.R. (2000). Single instance storage in Windows 2000. In *Proceedings of the USENIX Windows System Symposium (WSS)*, Berkeley, CA.

Bonwick, J. (2009). *ZFS deduplication*. <https://blogs.oracle.com/bonwick/entry/zfs-dedup>.

Broder, A. (1997). On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, Washington, DC.

Broder, A.Z. (1993). Some applications of Rabin's fingerprinting method. In *Sequences II: Methods in Communications, Security, and Computer Science*, New York, NY.

Bugnion, E., Devine, S., & Rosenblum, M. (1997). Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, *15*(4), 143–156.

Burns, R.C., & Long, D.E. (1997). Efficient distributed backup with delta compression. In *Proceedings of the Workshop on I/O in Parallel and Distributed Systems (IOPADS)*, New York, NY.

Chen, F., Luo, T., & Zhang, X. (2011). CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Chen, M., Wang, S., & Tian, L. (2013). A high-precision duplicate image deduplication approach. *Journal of Computers*, *8*(11), 2768–2775.

Cisco. *Wide area application services*. <http://www.cisco.com/c/en/us/products/routers/widearea-application-services/index.html>.

Citrix. *Cloudbridge*. <http://www.citrix.com/products/cloudbridge/overview.html>.

Clements, A.T., Ahmad, I., Vilayannur, M., & Li, J. (2009). Decentralized deduplication in SAN cluster file systems. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

Constantinescu, C., Glider, J., & Chambliss, D. (2011). Mixing deduplication and compression on active data sets. In *Proceedings of the Data Compression Conference (DCC)*, Washington, DC.

Cox, L.P., Murray, C.D., & Noble, B.D. (2002). Making backup cheap and easy. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Berkeley, CA.

Daehee, K., Sejun, S., & Baek-Young, C. (2017). *Existing deduplication techniques. Data Deduplication for Data Optimization for Storage and Network Systems* (pp. 23–76). Springer International Publishing.

Debnath, B., Sengupta, S., & Li, J. (2010). ChunkStash: Speeding up inline storage deduplication using flash memory. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

Deshmukh, A. S., & Lambhate, P. D. (2012). A methodological survey on mapreduce for identification of duplicate images. *International Journal of Science and Research*, 206–210.

Dong, W., Douglis, F., Li, K., Patterson, H., Reddy, S., & Shilane, P. (2011). Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Dong, W., Wang, Z., Charikar, M., & Li, K. (2012). High-confidence near-duplicate image detection. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*.

Douceur, J.R., Adya, A., Bolosky, W.J., Simon, D., & Theimer, M. (2002). Reclaiming space from duplicate files in a serverless distributed file system. In *22nd International Conference on Distributed Computing Systems*, Vienna, Austria.

Douglis, F., & Iyengar, A. (2003). Application-specific delta-encoding via resemblance detection. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

Dropbox. <http://www.dropbox.com>.

Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., et al. (2009). HYDRAstor: A scalable secondary storage. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

El-Shimi, A., Kalach, R., Kumar, A., Oltean, A., Li, J., & Sengupta, S. (2012). Primary data deduplication large scale study and system design. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

EMC. (2009). *Achieving storage efficiency through EMC celerra data deduplication*. <http://china.emc.com/collateral/hardware/white-papers/h6265-achieving-storage-efficiency-celerra-wp.pdf>.

EMC. (2008a). *Centera: content addresses storage system*, Data Sheet. <http://www.emc.com/collateral/hardware/data-sheet/c931-emc-centera-cas-ds.pdf>.

EMC. (2008b). *Networker*. <http://www.emc.com/domains/legato/index.htm>.

EMC. (2008c). *Avamar*. <http://www.emc.com/backup-and-recovery/avamar/avamar.htm>.

Eshghi, K., & Tang, H. K. (2005). A framework for analyzing and improving content-based chunking algorithms. *Technical Report HPL-2005-30*, 1−10.

Eshghi, K., Lillibridge, M., Wilcock, L., Belrose, G., & Hawkes, R. (2007). Jumbo store: Providing efficient incremental upload and versioning for a utility rendering service. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Fu, Y., Jiang, H., & Xiao, N. (2012). A scalable inline cluster deduplication framework for big data protection. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, New York, NY.

Guo, F., & Efstathopoulos, P. (2011). Building a high-performance deduplication system, in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

Gupta, A., Pisolkar, R., Urgaonkar, B., & Sivasubramaniam, A. (2011). Leveraging value locality in optimizing NAND flash-based SSDs. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Hong, B., & Long, D.D.E. (2004). Duplicate data elimination in a SAN file system. In *Proceedings of the Conference on Mass Storage Systems (MSST)*, Washington, DC.

Hua, Y., He, W., Liu, X., & Feng, D. (2015). SmartEye: Real-time and efficient cloud image sharing for disaster environments. In *IEEE Conference on Computer Communications (INFOCOM)*, Kowloon.

Hunt, J. J., Vo, K.-P., & Tichy, W. F. (1998). Delta algorithms: An empirical analysis. *ACM Transactions on Software Engineering and Methodology, 7*(2), 192−214.

IBM. (2002). *IBM white paper: IBM storage tank − a distributed storage system*. <https://www.usenix.org/legacy/events/fast02/wips/pease.pdf>.

Jin, K., & Miller, E.L. (2009). The effectiveness of deduplication on virtual machine disk images. In *Proceedings of the International Systems and Storage Conference (SYSTOR)*, New York, NY.

JustCloud. <http://www.justcloud.com/>.

Kaczmarczyk, M., Barczynski, M., Kilian, W., & Dubnicki, C. (2012). Reducing impact of data fragmentation caused by in-line deduplication. In *Proceedings of the International Systems and Storage Conference (SYSTOR)*, New York, NY.

Kaiser, J., Meister, D., Brinkmann, A., & Effert, S. (2012). Design of an exact data deduplication cluster. In *Proceedings of the Conference on Mass Storage Systems (MSST)*, Washington, DC.

Katiyar, A., & Weissman, J.B. (2011). ViDeDup: an application-aware framework for video de-duplication. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems (Hot Storage)*, Portland, OR.

Kaur, R., Chana, I., & Bhattacharya, J. (2018). Data deduplication techniques for efficient cloud storage management: a systematic review. *The Journal of Supercomputing, 74*, 2035−2085.

Kim, D., & Choi, B. (2012). HEDS: hybrid deduplication approach for email servers. In *2012 Fourth International Conference on Ubiquitous and Future Networks (ICUFN)*, Phuket.

Kim, D., Song, S., & Choi, B. (2013). SAFE: structure-aware file and email deduplication for cloud based storage systems. In *Proceedings of the 2nd IEEE International Conference on Cloud Networking*, San Francisco, CA.

Kim, J., Lee, C., Lee, S., Son, I., Choi, J., Yoon, S., et al. (2012). Deduplication in SSDs: Model and quantitative analysis. In *Proceedings of the Conference on Mass Storage Systems (MSST)*, Washington, DC.

Kruus, E., Ungureanu, C., & Dubnicki, C. (2010). Bimodal content defined chunking for backup streams. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Kulkarni, P., Douglis, F., Lavoie, J., & Tracey, J.M. (2004). Redundancy elimination within large collections of files. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

Li, J., He, L., Sengupta, S., & Aiyer, A. (2009). Multimodal object de-duplication. US Patent 12028840, 138.

Li, X., Jie, L., Li, J., & Jin, B. (2016). A video deduplication scheme with privacy preservation in IoT. In *International Symposium on Computational Intelligence and Intelligent Systems*, Singapore.

Liguori, A., & Hensbergen, E.V. (2008). Experiences with content addressable storage and virtual disks. In *Proceedings of the USENIX Workshop on I/O Virtualization (WIOV)*, Berkeley, CA.

Lillibridge, M., Eshghi, K., Bhagwat, D., Deolalikar, V., Trezise, G., & Camble, P. (2009). Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Linux (2017). *Linux programmer's manual IOCTL_FICLONERANGE*. <https://man7.org/linux/man-pages/man2/ioctl_ficlonerange.2.html>.

Liu, C., Lu, Y., Shi, C., Lu, G., Du, D., & Wang, D. (2008). ADMAD: application-driven metadata aware de-duplication archival storage system. In *Fifth IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, Baltimore, MD.

Lu, G., Jin, Y., & Du, D.H.C. (2010). Frequency based chunking for data de-duplication. In *Proceedings of the International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Washington, DC.

Manber, U. (1994). Finding similar files in a large file system. In *Proceedings of the USENIXWinter Technical Conference*, Berkeley, CA.

Mandagere, N., Zhou, P., Smith, M.A., & Uttamchandani, S. (2008). Demystifying data deduplication. In: *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, New York, NY.

Meister, D., & Brinkmann, A. (2009). Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of the International Systems and Storage Conference (SYSTOR)*, New York, NY.

Meister, D., & Brinkmann, A. (2010). dedupv1: Improving deduplication throughput using solid state drives (SSD). In *Proceedings of the Conference on Mass Storage Systems (MSST)*, Washington, DC.

Meyer, D.T., & Bolosky, W.J. (2011). A study of practical deduplication. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Microsoft. (a). *Exchange Server 2003*. <http://technet.microsoft.com/en-us/library/bb123872%28EXCHG.65%29.aspx>.

Microsoft. (b). *Exchange Server 2007*. <http://www.microsoft.com/exchange/en-us/exchange-2007-overview.aspx>.

Microsoft. (c). *Windows Storage Server 2008*. <https://web.archive.org/web/20091004073508/http:/www.microsoft.com/windowsserver2008/en/us/WSS08/SIS.aspx>.

Min, J., Yoon, D., & Won, Y. (2011). Efficient deduplication techniques for modern backup operation. *IEEE Transactions on Computers, 60*(6), 824–840.

Mozy. <http://mozy.com/>.

Muthitacharoen, A., Chen, B., & Maziéres, D. (2001). A low-bandwidth network file system. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, New York, NY.

N. I. o. S. a. T.(NIST). (2015). *Secure Hash Standard*. <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

I. o. S. a. T.(NIST). (2015). *Secure Hash Standard*. <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

Nath, P., Kozuch, M. A., O'Hallaron, D. R., Harkes, J., Satyanarayanan, M., Tolia, N., & Toups, M. (2006). Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

Naturel, X., & Gros, P. (2005). A fast shot matching strategy for detecting duplicate sequences in a television stream. In *ACM Proceedings of the 2nd International Workshop on Computer Vision Meets Databases*, New York, NY.

NEC, *Hydrastor*. <https://www.necam.com/hydrastor/>.

Nie, Z., Hua, Y., Feng, D., Li, Q., & Sun, Y. (2014). Efficient storage support for real-time near-duplicate video retrieval. In *Algorithms and Architectures for Parallel Processing ICA3PP*, Dalian.

Ouyang, Z., Memon, N. D., Suel, T., & Trendafilov, D. (2002). Cluster-based delta compression of a collection of files. In *Proceedings of the International Conference on Web Information Systems Engineering* (WISE), Washington, DC.

Pagh, R., & Rodler, F. (2004). Cuckoo hashing. *Journal of Algorithms*, 122–144.

Paulo, J., & Pereira, J. (2014). A survey and classification of storage deduplication systems. *ACM Computing Surveys*, *47*(1), 11.

Policroniades, C., & Pratt, I. (2004). Alternatives for detecting redundancy in storage systems data. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, Berkeley, CA.

Quinlan, S., & Dorward, S. (2002). Venti: A new approach to archival storage. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Rabin, M. O. (1981). Fingerprinting by random polynomials. *Technical Report TR-15–81*, 1–12.

Ramaiah, N. P., & Mohan, C. K. (2011). De-duplication of photograph images using histogram refinement. In *IEEE Recent Advances in Intelligent Computational Systems*, Trivandrum.

Rashid, F., Miri, A., & Woungang, I. (2016). Secure image data deduplication through compressive sensing. In *14th Annual Conference on Privacy, Security and Trust (PST)*, Auckland.

Rhea, S., Cox, R., & Pesterev, A. (2008). Fast, inexpensive content-addressed storage in foundation. In *Proceedings of the USENIX Annual Technical Conference (ATC*), Berkeley, CA.

Riverbed. *Steelhead for wan optimization*. <http://www.riverbed.com/products/wanoptimization/>.

Shen, H.T., Zhou, X., Huang, Z., Shao, J., & Zhou, X. (2007). UQLIPS: A real-time near-duplicate video clip detection system. In *Proceedings of the 33rd International Conference on Very Large Data Bases VLDB Endowment*, Vienna.

Shilane, P., Wallace, G., Huang, M., & Hsu, W. (2012). Delta compressed and deduplicated storage using stream-informed locality. In *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, Berkeley, CA.

Silverberg, S. *SDFS*. <http://opendedup.org>.

Spring, N., & Wetherall, D. (2000). A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of the ACM SIGCOMM 2000 Conference on Data Communication*, Stockholm.

Srinivasan, K., Bisson, T., Goodson, G., & Voruganti, K. (2012). iDedup: Latency-aware, inline data deduplication for primary storage. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Symantec a. *Netbackup*. <http://www.symantec.com/netbackup>.

Symantec b. *Puredisk*. <http://www.symantec.com/netbackup-puredisk>.

Ungureanu, C., Atkin, B., Aranya, A., Gokhale, S., Rago, S., Dubnicki, G.C.C., & Bohra, A. (2010). HydraFS: A high-throughput file system for the HYDRAstor content-addressable storage system. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.

Velmurugan, K., & Baboo, L. D. (2011). Content-based image retrieval using SURF and colour moments. *Global Journal of Computer Science and Technology*, *11*(10).

Waldspurger, C. A. (2002). Memory resource management in VMware ESX server. *SIGOPS Operating Systems Review*, *36*(SI), 181−194.

Xia, W., Jiang, H., Feng, D., & 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)*, Berkeley, CA.

Yan, F., & Tan, Y. (2011). A method of object-based de-duplication. *J. Networks*, *6*(12), 1705−1712.

Yang, T., Feng, D., Niu, Z., & Wan, Y. P. (2010a). Scalable high performance deduplication backup via hash join. *Journal of Zhejiang University—Science C*, *11*, 315−327.

Yang, T., Jiang, H., Feng, D., Niu, Z., Zhou, K., & Wan, Y. (2010b). DEBAR: A scalable high-performance de-duplication storage system for backup and archiving. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, Washington, DC.

You, L., & Karamanolis, C. (2004). Evaluation of efficient archival storage techniques. In *Proceedings of the Conference on Mass Storage Systems (MSST)*, Washington, DC.

You, L.L., Pollack, K.T., & Long, D.D.E. (2005). Deep store: An archival storage system architecture. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Washington, DC.

Zargar, A.J., Singh, N., Rathee, G., & Singh, A.K. (2015). Image data-deduplication using the block truncation coding technique. In *Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE) International Conference on IEEE*, Noida.

Zhu, B., Li, K., & Patterson, H. (2008). Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA.