International Journal of Engineering Research and Development
***e**-ISSN: 2278-067X, **p**-ISSN: 2278-800X, www.ijerd.com*
Volume 4, Issue 4 (October 2012), PP. 01-06

Two Stage Max Gain Content Defined Chunking for De-duplication

Arul Selvan R[1], Dr K Porkumaran[2]
[1]Software Engineering, Novell, Bangalore, India
[2]Principal, NGP Institute of Technology, Coimbatore, India

Abstract:—Data de-duplication is a very simple concept with very smart technology associated in it. The data blocks are stored only once, de-duplication systems decrease storage consumption by identifying distinct chunks of data with identical content. They then store a single copy of the chunk along with metadata about how to reconstruct the original files from the chunks, this takes up the less storage space than storing the whole thing again. Most of the de-duplication programs are based on fingerprinting. Data de-duplication segments the data into chunks and writes those blocks to disk. And signature is created for each data segment much similar to the fingerprints. And an index is associated for the fingerprint. The index, which can be recreated from the stored data segments. When the repeated new data is encountered in the data stream the de-duplication algorithm inserts a pointer in a metadata which related to the data block which is already present in the data stream. If the same block appears more than once multiple pointers are created for a single block on the disk. This eliminates the redundant blocks in the data stream and improves the storage space efficiency. Content defined chunking means the duplicate blocks are identified based on the internal properties on the data stream. There are many algorithms are in use to identify the repeated content on the data stream. In this paper, a new two stage content based chunking algorithm, the first stage can be any content based algorithm based on the cut points and in the second stage the newly developed max gain algorithm, identifies the new cut points on the repeated data.

Keywords:—chunk, fold factor, content defined.

I. INTRODUCTION

File systems often contain redundant copies of information: identical files or sub-file area, possibly stored on a single host, on a shared across many through storage cluster. De-duplicating storage systems take advantage of this redundancy to reduce the underlying space needed to contain the file systems and the backup space. De-duplication can work at whole file level and sun file level, the scope of this research is limited to sub file level. Data de-duplication technology works best when the data stream has lots of repeated data segments. For most people that is a perfect description of a backup, whether you backup everything every day (and lots of us do this) or once a week with incremental backups in between, backup jobs by their nature send the same pieces of data to storage system over and over again.

Though data compression helped to an extent, until the data de-duplication evolution, there wasn't a good alternative to storing all the duplicates. Consider the following example, let us consider the text "**Himalaya has glaciers"** is stored on the hard drive, now the owner opens the document and modifies it to "**Himalaya has glaciers and the glaciers are awesome**". In this example, the text "**glaciers"** is repeated twice. So we do not need to store the two copies. That is data de-duplication, and more importantly this is a content defined way of addressing the storage space reduction and the same is explored here. Though application knowledge is unknown at this time, there are several reverse engineering methods are employed to identify the repeated blocks in the data stream. Typical duplication eliminations process includes chunking, hash value generation and redundancy detection.

II. STATIC CHUNKING

There are numerous algorithms to determine the cut points in a data stream of the file. The simplest one would be the static chunking or fixed chunking. Divide the data stream into chunks of fixed size. The file is partitioned into blocks with a fixed size[3], 4K bytes for example, this means that there is no exceptional intelligence required to calculate the chunk boundaries, the chunk size is predetermined. Most of the time the chunk size is determined based on the file type and size. And apply the SHA-1 hashing to calculate the hash value of all the chunks as their identifiers. During the process of hash comparison, once a same hash is found, consider the chunk as redundancy, otherwise, store the chunk and the associated hash for reference. As one would expect, the effectiveness of this approach on duplicate elimination is highly sensitive to the insertion and modifications. For example an insertion of a single byte at the beginning of the file will change the content of all blocks in the file resulting in different chunking and the duplicate elimination ratio or the fold factor will vary widely.

Original Data:
24063 12345 67891 23456 78953 00101 23456 78992 77800 12345 67890 73012 34567 89012 01912 34567 8910
Modified Data:
24063 **N**1234 56789 12345 67895 30010 12345 67899 27780 01234 56789 07301 23456 78901 20191 23456 78910

In the above example, the chunks size is fixed on five alphabetic characters. The chunk content is not analyzed for the cut point definition. And when the alphabetic character 'N' is inserted in the beginning of the file, the identified cut points are moved by one alphabetic character length that leads to the new hash generation and hence new duplicate elimination or fold factor. This type of cut point identification is highly vulnerable to file modifications and insertions, the new cut points are generated after the file modification or insertion.

III. DYNAMIC CHUNKING

To retain the identified cut points, dynamic or variable size chunking is used. It does not break files at fixed offsets rather break the file based on the file content. Instead of setting boundaries at multiples of the predefined chunk size, the content defined chunking (CDC) approach defines break points where a certain condition becomes true. This is usually done with a fixed-size overlapping sliding window. Chunk sizes are variable in nature and based on the content of the data stream. The chunk is defined between two cut points.

Some statistical methods are used to find the cut points in the data stream. Many algorithms are based on the sliding window technique, the window of fixed size moves ahead by byte. Cut point selection is based on values of a function evaluated on local data window. And every position of the window the contents are fingerprinted. Rabin[1]fingerprinting is widely used for this purpose. Rabin is the mother of most of the content based de-duplication algorithms. The canonical algorithm for dynamic sized content-defined blocks is based on Rabin Fingerprints. At every offset of the file, the contents of the sliding window are analyzed and a fingerprint, **f**, is calculated. If the fingerprint satisfies the break-condition, a new breakpoint has been found and a new chunk will be created. Typically, the break-condition is defined such that the fingerprint can be evenly divided by a divisor D, i.e. f mod D = r with 0 <r < D, where typically r = 0.

By deciding chunk boundaries based on the content, files that contain identical content that is shifted due to insertions and deletions, will still result in identical chunks those were identified earlier. Rabin-based algorithms are typically configured with a minimum and maximum chunk size, as well as an expected chunk size. And the algorithm operates on a specified random value and a rolling hash until the specified condition is encountered.

Rabin’s algorithm uses randomly chosen irreducible polynomials to fingerprint the bit-strings. This method is applied to produce a very simple real-time string matching algorithm and a procedure for securing files against unauthorized changes. The method is provably efficient and highly reliable for every input since the cut points are determined based on the content. The input byte-string is first partitioned into non-overlapping sub-strings called chunks. Rabin’s algorithm plays a critical role in this partitioning process. Let us explore this in more detail, shall see below how it is done. A few policy decisions need to be made upfront as follows,

A) Average number of bytes per chunk desired, power of 2 (typical value being 4096Bytes). Logarithm to the base 2 of this number tells us how many bits are of interest. In our case 4096 Bytes as the average chunk size, the number of such bits of interest is twelve.

B) Break condition or the specific value for these interested bits and the identified bits. Typically the bits are chosen from LSB, and the value zero.

C) Number of bytes for the break function, typically 32 bytes.

Here is the algorithm flow

- Compute the Rabin fingerprint, starting at each byte in the input and over the chosen number of bytes to its right
- If the bits of interest in that fingerprint have the value specified (for example, the least significant 12 bits have value 0), then mark that starting byte as a “break point”
- The sub-string between two consecutive break points is a chunk
- It is clear that if the Rabin fingerprint is computed over 32 bytes at each byte of the input then if the input byte-string is of 1 million bytes then we need to compute 1 million minus 31 such fingerprints to find all the chunks in that string. That can get very expensive if one has to compute each of the fingerprints individually, as the amount of input data grows. A beautiful property of the Rabin algorithm is that only the first fingerprint needs to be computed in full and all subsequent fingerprints can be “incrementally” derived from the previous one quite inexpensively.
- Compute the Rabin fingerprint at the first byte or the bytes required for the break function (i.e., over the first 32 bytes)
- Compute the fingerprint incrementally from the first break block
- Now that the fingerprint at the 2nd block is available, use it to incrementally compute the fingerprint at the 3rd block, and continue this process till the end of the data stream.

The break condition, of ‘D’ match could be a simple ‘.’itself or a predefined value specific to nature of the stream data and may vary content to content. Though for a different purpose, the Rabin’s paper uses the same process and computes incremental fingerprints. Each change only impacts its direct neighbors unlike in fixed chunking it impacts the whole data stream. Chunk size is typically in between 2KBytes to 64Kbytes.The marker may be a sequence of bytes such that some mathematical function of the data results in a certain bit pattern or as simple as a full stop. In fact, the original motivation for the Rabin algorithm was the desire to come up with a hash function that can be incrementally computed in a constant time. $H = c_1a^{k-1} + c_2a^{k-2} + c_3a^{k-3} + \ldots + c_ka^0$ where a is a constant and $c_1, \ldots, c_k$ are the input characters. Rabin’s string search algorithm is used with a very simple rolling hash function that only uses multiplications and additions, the new hash value is rapidly calculated given only the old hash value, the old value removed from the window, and the new value added to the window.

IV. EFFICIENCY OF RABIN’S ALGORITHM

Sub string match and not knowing the sub string itself in advance is the defined problem. A brute-force substring search algorithm checks all possible position and operates on O(m*n).A variation on this idea is to compute a hash value of the sub string and then compare that hash value to the hash value of each of the substrings of the original string. The difference is that we compare the hash values, not the original bytes themselves. For a perfect hash function, the hash matches the string also matches. Moreover computing the hash of a substring takes time proportional to the size of that substring then it is all the same as string comparison and we have not accomplished anything great. The algorithms still operates on O(m*n).

Few other sub string algorithms like, Knuth-Morris-Pratt and Boyer-Moore were optimized for performance. The Boyer-Moore algorithm as presented in the original paper has worst-case running time of O(n+m) only if the pattern does. More interestingly, there may be a possibility that for identified duplicate chunk's cut points are not necessarily the beginning and end of the real duplication area. And these algorithms are not application aware and operates on some predefined break functions.

Typically the A simpler approach is to be content-agnostic and simply select the content size and marker. Since cut points are simply chosen by position, this does not incur additional computational cost. The goal is to identify the real or actual origin and end of redundant blocks. This kind of scenario will arise when the f(x) does not fall on the not appear in the text. When the pattern does occur in the text, running time of the original algorithm is O(nm) in the worst case. Whereas in case of Knuth-Morris-Pratt has complexity of O(m+n). This is where Rabin's search begins for a hash function that can be computed incrementally, at a constant cost, after the first one. Armed with such a hash function, Rabin computes 2 full hashes first, of B and of the m-byte substring starting at the very first byte of A and then incrementally computes all the other n – m hash values, for a total cost of O(n).

V. MOTIVATION

It is possible that the identified cut points are not necessarily the chunk boundaries. If the identified chunk repeats then the hash, usually SHA-1, will get added in the de-duplication table. It may be possible that de-dup chunks are chained up. We do not need a hash for every chunk instead we can have a hash of the whole chunk which repeatedchunk boundary and the cut points might shrink inwards. There could be two cases the non-duplicate block may be equal or less than the max chunk size. Or it can be more than the max chunk size.

VI. MAX GAIN ALGORITHMS

Object decomposition will result in sequence of chunks, one may find the sparse chunk repetition and sub sequence repetition. In the first stage, we have table consists of finger prints and the associated pointer to the chunks in the data stream and the identified duplicate chunks are removed from the data stream. Since we are doing the content defined chunk identification, this leads to higher duplication elimination. Since the cut points are based on the predefined sequence and the sequence itself may not necessarily the origin and end of the sequence. Now we will be attempting to redefine the cut points for the identified duplicate chunks in the data stream, this is achieved in the second stage. There are two methods proposed to redefine the chunk boundaries. First approach is a byte by byte method and the second is rabin's forward and backward (FB) sliding method with overlapping chunks.

VII. BYTEBYBYTE METHOD

Let us denote the identified cut points as follows Cp1, Cp11, Cp2, Cp22…….. Cpn, Cpnn.
Cpi – Cut point that begins the duplicate chunk
Cpii- Cut point that ends the duplicate chunk
Every chunk resides in between the two consecutive cut points and the chunk size not necessarily evenly distributed, called variable chunk de-duplication. Let us denote the chunks that reside in between the two cut points as C1, C2, C3,C4 …..Cn. And the occurrence frequency is denoted as c1Oi, c2Oi, c3Oi……..CnOi. The value of 'i' denotes the number of appearance in the given data stream. The occurrence frequency is calculated based on the fingerprint match. New outbound cut points are identified for the redundant chunk as follows. Let us identify the chunk frequencies through the fingerprint comparison and the identified frequency is follows, for a redundant chunk the frequency is denoted as follows

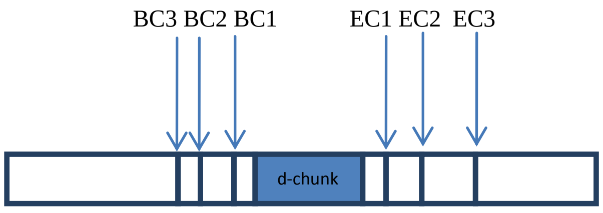
```
CiOj j<=1, →Oj denotes duplication
Ci has two cut points(cPb, cPe)→ Pb entry cut point, Pe exit cut point
All CiOj – Do for chunks has j >1
n= 0,1….m
n-k~~not equal to~~ zero – Max crawling till the beginning of the file data stream
cPb-k, cPe – FingerPrint(cPb-k, cPe)
//Now we have the finger prints for a chunk duplicate
Calculate the CiOk  if k~~ not equal to~~j --- or k<=1→ break retain the original cut points
else continue for the set CkOk duplicate chunks
Now we got a new entry cut points(cPb) for the chunks.

//Attempt to find the new exit cut points(cPe)
All CiOj – Do for chunks has j >1
n= 0,1….m
n+k  ~~not equal to~~ endoffile – Max crawling till the end of the file data stream
cPb, cPe+k – FingerPrint(cPb-k, cPe)
Calculate the CkOk  for all chunks if k~~ not equal to~~j or k<=1→ break retain the original cut points
else continue for the new set of CkOk duplicate chunks
Now we got a new exit cut points(cPe) for the chunks
```

Stage 1: (content defined chunking)
24063 12345 67891 23456 78953 00101 23456 78992 77800 12345 67890 73012 34567 89012 01912 34567 8910

Stage 2: (content defined re-chunking)
24063 **123456789** **123456789** 53 0010 **123456789** 92 77800 **123456789** 0 730 **123456789** 012 019 **123456789** 10
In the above example, 12345, 23456 and 34567 are the identified duplicate chunks in the first stage. The improved duplicate elimination, the outbound bytes are compared with the identified duplicate blocks. As chunk "34567" has "12" as a prefix and "89" as suffix matches with the other duplicate chunks contributed for the space reduction. Hence the 6 byte space saved. And it is important to note that not all the time all the duplicate blocks will contribute to further space reduction.

Rabin's Bidirectional Sliding
Additional cut points are identified bi-directionally for the duplicate chunks outwards from the cut points using the Rabin's algorithm. In the first stage, when the cut point is identified, the sliding windows move ahead on a fixed length based on the policy decision. Irrespective of the chunk property, the window moves ahead. But the skipped chunk or the chunk until the next cut point identification may or may not be a duplicate chunk. If the skipped chunk is not falling under duplicate category and the subsequent chunk falls under duplicate category then Rabin's bidirectional sliding algorithm is applied on the preceding and succeeding non duplicate chunks of the identified duplicate chunks. In the following example the bidirectional



sliding is applied on the duplicate d-chunk and BC1, BC2 and BC3 are the proceeding cut points based on the Rabin's algorithm on the non-duplicate block. And EC1, EC2 and EC3 are the succeeding cut points on the succeeding non duplicate block. And the d-chunk can be a duplicate block or duplicate blocks chained. The bidirectional sliding works outwards from the duplicate chunk or chunk chain. The newly identified cut points (BCiand ECi) distance is measured from the chunk boundary, if the cut point distance matches then hash is calculated and compared and hash matches the algorithm generates newer cut points till it reaches the adjacent duplicate chunk boundary. The important attribute of this approach is the sliding windows works on both the directions by compromising the policy decision. The open source version of Rabin's algorithm in java implementation was taken and the max gain algorithms are implemented. Also to get the file list volume search was implemented using the depth-first search algorithm and stack was used to identify the resume point on every level of the file system tree. And the resultant application was named as TSdup application which operates on bytebybyte and bidirectional sliding methods at a time.

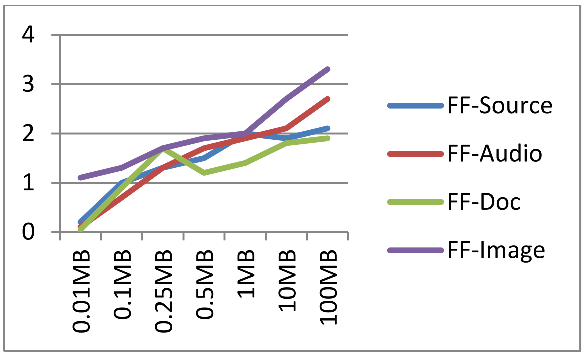
VIII. EXPERIMENTAL EVALUATION

We wanted to experiment the following as part of the evaluation process.

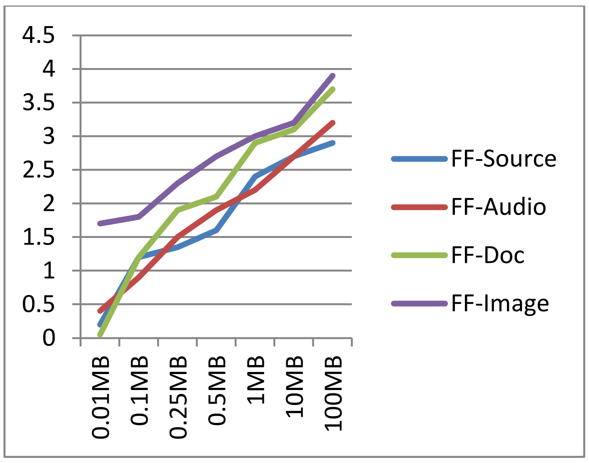
- Storage space reduction or the fold factor using the second stage content defined algorithms for the real world datasets
- Average chunk size recorded for various data types

A good candidate data volume for de-duplication would include a file server which hosts the user documents, virtualization files, software source files containing data that is modified infrequently and read frequently. Test bed was created on a windows 2008R2 server on Dell PowerEdge R910 with 8GB RAM with that was attached to a storage center S30 SAN with two volumes, both created for a production server volume mainly consists of office documents, photos, music and videos. The volume space and per file usage was measured ahead of the test run. And more importantly SSD wasn't used in the storage stack. Now the TSDup application was executed locally on the given volume. The scanned volume files are read one by one by the TSDup application and the fold factor is calculated based on the file type and size and the data is compared with the first stage Rabin' algorithm with a predefined break function. The meta-data additions and the associated complexities are not evaluated here. Before the test execution, the server was restarted to flush the cache away from the memory. In a data set collection process we have segregated the data set of various sizes such as 10KB 100KB 1MB 10MB 100MB, the evaluated files types are Source files, MS doc files, Image files and Audio files. The average chunk size policy decision was done based on the file size.

In the following experiments we measured the fold factor for the two stage chunking especially for the bytebybyte implementation. Folder factor is measured based on the percentage of the data reduction. If 50% of the original data is reduced, then the fold factor is two, for 75%reduction the fold factor is four. The effective of finding the repeated chunks in the data stream depends on the file type, used three file types, outlook express .ost files, mpeg layer three mp3 files and the files size scales to few mega bytes. And for the first stage chunking we used fixed size and Rabin finger print algorithm. And the improvements measured.

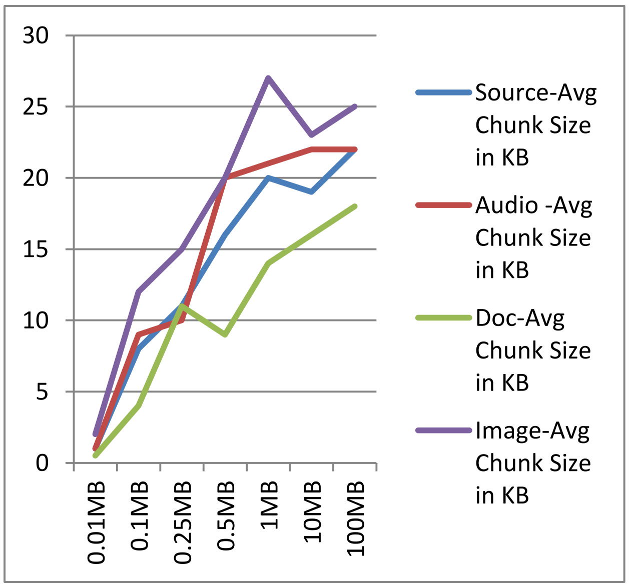


Rabin's Fold Factor



Bytebybyte tsdup fold factor

The average chunk size is chosen for every file size and the average chunk size increases exponential with the size of the file data. It is a not good idea to increase the chunk size forever, If we do so the fold factor might not increase and after a point it will come down.



Average Chunk Size vs File Size

Data de-duplication has predominantly applied with the secondary storage systems. The reason is data de-duplication involves overhead to identify and remove the duplicate data. In the online storage this will affect the performance of the file server. The performance of a storage device, particularly hard disks, can vary based on whether the accesses that are being serviced are reads or writes. The improved folder factor has come with a cost. We have only calculated the extra IOPS raised during the second stage execution. The second stage algorithm is immediately after the first stage for every file, the chances of file data available in the cache is more for smaller files hence the file read requests are served from the cache and when he file size increases the.

IX. CONCLUSION

This paper presents an enhanced chunking approach to achieve high performance de-duplication. The proposed de-duplication system adds significant value on top the existing content defined de-duplication algorithms. We have found that identifying the predefined boundary conditions is very challenging and most of the cases the boundary conditions will keep varying across the data stream. Significantly, these algorithms require no additional meta-data to be stored. And we see that the average chunk size is increased on all possible occasions and constitutes the increased fold factor and space reduction. Chunking parameter2nd stage de-duplication fragments the data and that will result in slower data reads, therefore deliberate selection and tuning the underlying file system is crucial for the better performance. And I would recommend to execute the TSDup evaluation under a wide variety of chunking parameters such as break condition for cut point identification and chunk size etc.

REFERENCES

[1]. M. Rabin. Fingerprinting by Random Polynomials. Harvard University Center for Research In Computing Technology Technical Report TR-CSE-03-01, 1981. Boston, MA.

[2]. Research on chunking algorithms of data de-duplication, cai bo, zhang feng li, wang can,American Journal of Engineering and Technology Research Vol. 11, No.9, 2011

[3]. BOBBARJUNGDeepak R;JAGANNATHAN Suresh; DUBNICKI Cezary. Improving Duplicate Elimination in Storage Systems[J]. ACM Transactions on Storage (TOS), 2006.

[4]. A Study of Practical DeduplicationDutch T. Meyer*† and William J. Bolosky* *Microsoft Research and †The University of British Columbia {dmeyer@cs.ubc.edu, bolosky@microsoft.com}

[5]. Efficient randomized pattern-matching algorithms, by richard m. karp michael 0. Rabin

[6]. Bimodal Content Defined Chunking for Backup StreamsErik Kruus, NEC Laboratories Americakruus@nec-labs.com. Cristian Ungureanu NEC Laboratories America cristian@nec-labs.com. Cezary Dubnicki 9LivesData, LLC.

[7]. Content-dependent chunking for differential compression, the local maximum approach- nikolaj bjørner, andreas blass, and yuri gurevich

[8]. Frequency Based Chunking for Data De-Duplication Guanlin Lu, Yu Jin, and David H.C. Du Department of Computer Science and Engineering University of Minnesota, Twin-Cities, Minneapolis, Minnesota, USA.

[9]. Segregating and Extracting Overlapping Data Points in Two-dimensional Plots ! William Browuer, Sujatha Das and C. Lee Giles, Pennsylvania State University

[10]. Online De-duplication in a Log-Structured, File System for Primary Storage, Technical Report UCSC-SSRC-11-03 May 2011. Storage Systems Research Center, Baskin School of Engineering, Baskin School of Engineering, University of California, Santa Cruz

[11]. content-dependent chunking for differential compression, the local maximum approach nikolaj bjørner, andreas blass, and yuri gurevich Microsoft Research, One Microsoft Way, Redmond,

[12]. Efficient Data Deduplication System Considering FileModication Pattern- Ho Min Jung_, Sang Yong Park, Jeong Gun Lee, Young Woong Ko. Department of Computer Engineering, Hallym University, Chuncheon, Korea.

[13]. Theoretical Aspects of Storage SystemsAutumn 2009-Chapter 3:Data DeduplicationAndré Brinkmann.

[14]. ChunkStash: Speeding up Inline Storage Deduplication using Flash Memory- Biplob Debnath¤ Sudipta Senguptaz Jin Liz zMicrosoft Research, Redmond, WA, USA¤University of Minnesota, Twin Cities, USA.

[15]. The Effectiveness of Deduplication on Virtual Machine Disk Images- Keren JinStorage Systems Research Center University of California, Santa Cruz.

[16]. Minimizing remote storage usage andsynchronization time using deduplication and multichunking- Syncany as an examplebyPhilipp C. Heckel, 2012.

[17]. Finding Similar Files in Large Document Repositories George Forman, Kave Eshghi, Stephane Chiocchetti Intelligent Enterprise Technologies Laboratory, HP Laboratories Palo AltoHPL-2005-42(R.1)- June 15, 2005