RepoZip: A Technique for Lossless Compression of Document Collections

D.N. Sumanaweera, F.Fahima Doole, D.P. Pathiraja, G.G.K. Deshapriya, Prof. Gihan Dias
Department of Computer Science and Engineering,
University of Moratuwa, Moratuwa, Sri Lanka
{dinithi.10, fahima.10, daham.10, kelum.10}@cse.mrt.ac.lk, gihan@uom.lk

***Abstract*—Many computer systems; especially in corporations, contain large amount of documents such as letters, reports and presentations. Many such documents are present in several versions. Such data needs to be synchronized with branch offices and mobile devices, often over slow and expensive connections. However, as many documents are stored in an already compressed format, it is difficult to compress them further by exploiting the hidden redundancies. We present a novel approach named RepoZip which improves the compression of an existing compression algorithm over a document collection, by exploiting the inter-document meta-data and content-level redundancies. It concentrates on compressing OOXML documents that have been constructed through the archival of a hierarchy of meta-data files and PDF documents which include deflated content streams. Therefore, the RepoZip approach achieves larger compression gains over OOXML document collections or PDF document collections by exploiting usually undetected meta-data level similarities.**

Keywords—lossless compression; meta-data similarity; OOXML; PDF; clusters; generalized suffix tree

I. Introduction

In a typical computer system, huge collections of similar documents can be available due to a user keeping many versions of the same document but with minor variations. Also he might even keep on editing a document for saving more and more similar versions. Being able to store those documents through optimal storage utilization is one of the key concerns of a user who maintains such a large document collection. Hence, for a user to achieve convenient and efficient document storage, it is important that the document file collection is de-duplicated and compressed, without affecting its authenticity. This helps to reduce the data storage space or the bandwidth occupancy when storing or transmitting such an entirely large document collection. In order to fulfill this purpose, a greater amount of compression methods, techniques and algorithms have been introduced from time to time; conducting various research and experimentation to come up with higher compression ratios. However, most of them focus only on the problem of individual file compression [1].

A compression scheme performs compression based on the redundancies it detects within the content that is to be compressed. Thus, its level of compression relies upon how well it can detect those redundancies and therefore, it is very much essential to figure out all the possible redundancies for obtaining a very good compression. Especially when it comes to a collection of documents, it is effective to identify intra-document as well as inter-document similarities before the conduction of actual compression. The redundancies can be found among the documents at two levels; the content-level which accounts for actual text or image content saved by a user and the meta-data level which accounts for the format, style and structural data. Content-level redundancies can be clearly observable whereas the meta-data redundancies might include a portion which is hidden even from the best compression algorithms such as LZMA. Due to this limitation of redundancy detection, two widely used document types; Office Open XML (OOXML) and Portable Document Format (PDF) are not getting compressed in an optimal manner.

An OOXML document itself is an archive file in which there is a collection of XML documents. Also if a PDF document is considered, its content varies from the raw text content to other meta-data content such as fonts, images and page structure data which are present inside streams that have already been compressed at the document creation. Thus, it is difficult to further compress a collection containing a large number of such documents in which, the similarities that are hidden beneath the internal compression spread across the entire collection. Hence, in order to attain a high compression ratio for a collection of OOXML documents or PDF documents, it is important to exploit those redundancies among XML components of OOXML documents or streams of PDF documents prior to the actual compression.

This paper presents a novel approach named RepoZip, which addresses the earlier explained problem scenario and improves the compression of such a collection of documents by preprocessing their content in such a way that an existing compression algorithm could detect and exploit all of the possible redundancies prior to the actual compression. It mainly involves in taking the entire document content within the collection into account, decompressing the already compressed content, reorganizing the collection by clustering the content of all documents by their type and compressing them using LZMA; currently one of the best compression algorithms due to its very large sliding window up to 4GB when compared to other existing algorithms [3]. The application is also capable of decompressing the RepoZipped archive, extracting and retrieving each of the individual document content from the available clusters, compressing the content which is meant to be compressed at the OOXML and PDF document creation and reconstructing the original documents in the collection. The RepoZip preprocessing gives quite a considerable amount of improvement upon LZMA compression; proving the fact that both content and meta-data level redundancy detection is essential for a better compression.

II. Related Work

The concept behind existing file system compression addresses the redundancies that spread across large number of files in a repository under the techniques such as data de-duplication [5], migratory compression [3] and delta compression [1]. Their ultimate goal is to achieve better compression than from individual file compression.

The key idea in data de-duplication is to identify and eliminate duplicate data in file level as well as block level [5]. Under the block level technique, a file is divided into content-defined chunks. Then, the unique chunks are identified by fingerprinting each chunk with a unique, strong hash [4] using a hash algorithm and are stored in an index during the process of analysis. The index also contains meta-data about each chunk [5]. This index is queried whenever the system needs to find whether there is a duplicate chunk to the chunk representing a new update to the file. Whenever a match occurs, the redundant chunk is replaced with a small reference that points to the stored chunk. Each fingerprint in the index has to be queried in order to determine if the system already stores a copy of the chunk.

Delta Compression [1] is widely used in version controlling systems where there is the current file and a set of source files. Most probably the current file and a source file are related versions with very similar content. Delta refers to the difference between the two files. The measurement of compressed delta is the size of the delta divided by the current version of the target file. Such deltas are stored instead of the versions of the system. Deltas are sometimes organized into chains in which the target version of one delta is the source version of the next delta in the chain. If a deltas' target version is newer than its source version in the history, then it is referred to as a forward delta. A reverse (backward) delta has the opposite side such that, the target version is older than its source version in the history [1]. The forward delta chain approach stores the first version of a particular file as its source version and then, stores every subsequent version in a forward delta chain.

Data de-duplication and delta compression act in low level where a file system deals with disk storage. Even though they do file chunking; whether fixed sized or content based, they have their own drawbacks. In redundancy detection between chunks, fixed sized chunking might result in shadowing existing duplications (no content sharing with other chunks). Although variable sized chunking gives a better opportunity to detect similarities than fixed sized chunking, the hashing may still produce two completely different hashes for highly similar chunks with a very small variation.

At application level compression, Migratory compression (mzip) [3] acts as an improvement to the usage of existing individual based compression tools that use a limited sized sliding window in redundancy detection. It can improve the compression of standard compressors by reorganizing the file chunks to exploit their intra similarity. In similarity detection, similarity features are generated for each chunk. Two chunks are said to be similar if they share many such features or a group of features called a super feature. Upon identifying similar chunks, mzip preprocessor creates two recipes; migratory recipe and restore recipe. Migrate recipe contains the chunk order of the reorganized file where the similar chunks are migrated and located together. A chunk is represented by its offset within the original file. Restore recipe contains the chunk offsets in order; reflecting their original order in the file but by its offsets within the reorganized file.

III. Background Study

RepoZip is a novel approach that takes meta-data level as well as document content-level redundancies which spread across a large collection of Office Open XML (OOXML) documents and PDF documents into consideration before compressing them using an existing compression algorithm such as LZMA. Basically the idea is to let the entire set of redundancies including visible and hidden redundancies to get discovered and reduced before moving on with a usual compression algorithm. Thus, RepoZip can be considered as an improvement tool for existing compression techniques.

Before moving into the main preprocessing steps, it is essential to understand how the document content-level and meta-data level redundancies do occur in different file formats. Here, OOXML and PDF have their own ways of structuring the raw content including the actual document content and meta-data.

A. Office Open XML (OOXML)

The OOXML format is based on XML and ZIP (basically zlib [2] archive). The OOXML documentation defines formats for word processing, spreadsheets and presentation documents in the form of specific markup languages known as word processing ML, spread sheet ML and presentation ML. Therefore, .docx, .xlsx, .pptx files are archive files that could be decompressed. [8][9]

When examining the structure of each such document file, a root structure can be observed as the basic folder structure. It includes sub folder structures and different XML files containing meta-data information, depending on the document type and their content.

Therefore each document consists of a great number of redundant content inside those structured files. Document content-level and meta-data level redundancy can be exploited in these structured files where each file contains a great number of redundant characters, set of strings and tags.

TABLE I. SOME EXAMPLE XML FILES AND FOLDERS WITHIN A OOXML ROOT FOLDER STRUCTURE

[Content_Types] .xml	to identify every unique type of part found within the document
core.xml	holds the typical document properties that are filled out to identify documents
custom.xml	specifies the custom document properties
styles.xml	lists the available accents
_rels folder	includes files that indicates the connections between files
docProps folder	includes document properties files
word folder	Includes content-specific parts

Some basic set of characters, strings and tags of meta-data level files under the structured folder content are, <document> and <body elements>,<p> for paragraphs, <r> for run which is a region of text with a set of common properties and many more.

B. PDF Documents

Portable Document Format (PDF) [6][7] is a well-known international standard for representing electronic documents through its reliance on the same imaging model as the Postscript Page Description Language. The format itself is a page description language that has the capability to present textual and graphical content in a device and resolution independent manner. The model presents a PDF as a single, self-contained sequence of bytes, representing a collection of objects along with associated structural information according to the respective PDF specification. It describes the appearance of graphics and typography within the document.

There are several levels of abstraction in a PDF document: object level, file level, document level and content streams [6][7]. A content stream is usually a compressed sequence of instructions consisting of operands and operators that specify how to paint graphical elements on a page, found within a stream object. For instance, 'BT' and 'ET' define a region for text to be painted. Other operators such as 'BMC' or 'BDC' and 'EMC' that marks content stream portions to be processed as single units do occur very often and repetitively within streams. Also there are different types of graphics such as text, text state, color, shading patterns being specified within a stream by using relevant operators. Out of the four types of fonts defined in PDF, type 3 font glyphs are defined by streams of PDF graphics operators. Also for type 0 (CID Fonts) whose glyphs are obtained from an object called CID Font, its character encoding is in a character map called CMap which can appear inside a separate content stream. Apart from font streams, there could be cross reference streams as well.

If any PDF document is considered, those content streams might have completely identical streams; especially font streams as well as streams that share a greater portion of their content. Thus, a great redundancy spreads across large number of PDF documents. Not only the content streams but also other non-stream content that contains keywords such as 'obj', 'stream', 'endstream' ,'startxref', 'xref', 'trailer' also appear within a document for multiple times, contributing for intra-document level redundancy as well as the inter-document level redundancy. However, PDF uses FlateDecode; deflate algorithm provided by zlib [2] as the compression filter for encoding its content streams at the construction level and thus, the meta-data redundancies are hidden beneath those deflated streams.

IV. METHODOLOGY AND IMPLEMENTATION

A. RepoZip Compression

RepoZip methodology was designed as to have two main preprocessing steps at compression; decompressing already compressed document content and clustering raw content based on their meta-data type. Fig. 1 shows the RepoZip compression work flow and Fig. 2 shows the RepoZip Decompression workflow.

By considering the nature of raw content in both OOXML and PDF files, the very first step on this compression process goes into the meta-data level of each document type by decompressing the originally compressed content at their construction time. If it is an OOXML document, it is extracted at first and all internal XML files within its internal folder hierarchy are taken. If it is a PDF document, all of the content streams which have been compressed by the deflate algorithm are inflated and the entire raw content is taken. Likewise, for each document in the collection, this decompression is performed.

The second step is to cluster the XML files by their type (i.e. style.xml, settings.xml etc.) or cluster the PDF raw content streams by their type. For any basic PDF document, six main cluster types can be identified including a separate one to which the rest of the non-stream raw content apart from the content streams can go into. Others are font/image streams, text streams, CIDFont streams, cross reference streams and page structure related streams. Clustering is performed on each and every document in the collection until all similar type of XMLs or content streams of all documents are put into their respective cluster. This can be considered as

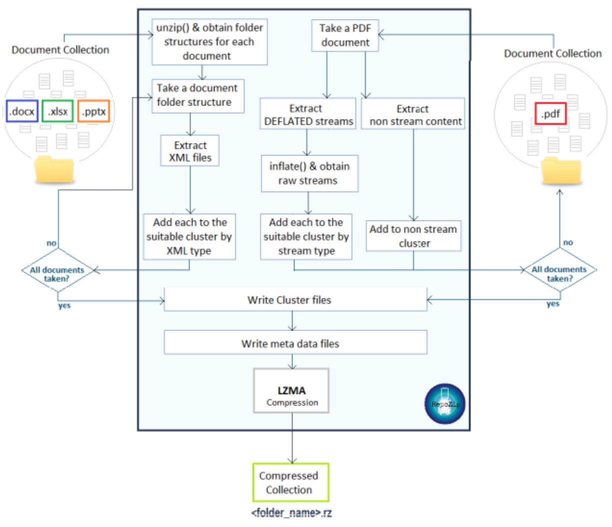

Fig. 1. RepoZip Compression Workflow

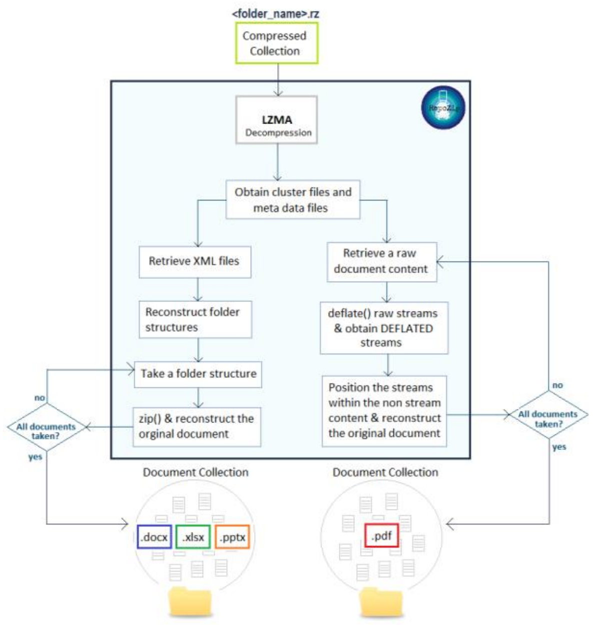
<folder_name>.rz
Compressed Collection
LZMA Decompression
Obtain cluster files and meta data files
Retrieve XML files
Retrieve a raw document content
Reconstruct folder structures
deflate() raw streams & obtain DEFLATED streams
Take a folder structure
zip() & reconstruct the orginal document
Position the streams within the non stream content & reconstruct the original document
All documents taken?
no
yes
All documents taken?
no
yes
Document Collection
.docx
.xlsx
.pptx
Document Collection
.pdf

Fig. 2. RepoZip Compression Workflow

the step which takes the concept of file reorganization in [3] for migrating similar content chunks into adjacent places.

After going through the above mentioned preprocessing steps, the final step is to write each cluster into a file and keep meta-data required for the reconstruction of original files including their original names in a separate file. This set of files including the cluster files and meta-data files will be the input for an existing compression algorithm like LZMA.

B. *RepoZip Decompression*

At decompression, the RepoZips' main objective is to reconstruct the original document files from their compressed state. In order to achieve that, all of the content in the cluster files are read back after decompressing the RepoZip archive file; retrieving back the cluster content from each cluster and then reorganizing the XMLs/PDF content streams to their original documents. A file name index was written with all original document names as meta-data at RepoZip compression; positioned at their programmatically assigned IDs in order to be used at this stage. After reorganizing the documents, they are reconstructed.

V. Results

This section presents the compression ratios obtained by applying RepoZip technique prior to LZMA compression. As previously mentioned, LZMA is used for the purpose since it is the currently existing compression algorithm which uses a larger window size [3]. The results were taken on .docx and .pdf file collections of different sizes for different cases; collections having similar documents and collections having different documents. A similar test document collection had several random documents having a number of duplicates for each. A different test document collection had random documents; each having different document content saved by a user.

Equation (1) is the formula which is used to calculate the compression ratio. In the context of RepoZip, the size of the uncompressed content accounts for the original size of the document collection whereas the size of the compressed content refers to the size of the compressed collection of RepoZip cluster and meta-data files.

$$\text{Compression Ratio} = \frac{\text{Uncompressed content}}{\text{Compressed content}} \qquad (1)$$

A. *Results for OOXML Document Collections*

For similar .docx document collection as in Fig. 3, the RepoZip compression ratio is almost 3.4 times than that of the direct LZMA compression. Direct LZMA compression ratio always stays under the value of 2, with only slight variations whereas RepoZip shows an increase of the rate up to 3.6 times of a compression ratio.

On the other hand, Fig. 4 Shows that for Different .docx document collections, RepoZip has only been able to provide a slight improvement for the direct LZMA compression. However, as the size increases, the rate of increase has become much higher compared to direct LZMA; reaching a ratio of around 3.

TABLE II. Compression Ratios of Similar DOCX Document Collections

No of Files	LZMA Ratio	RepoZip Ratio
25	1.544747	5.057325
50	1.629328	5.298013
75	1.632089	5.339367
100	1.663265	5.948905

TABLE III. Compression Ratios of Different DOCX Document Collections

No of Files	LZMA Ratio	RepoZip Ratio
25	1.089552	1.223464
50	1.868914	2.169565
75	2.472973	2.983696
100	2.542373	3.208556

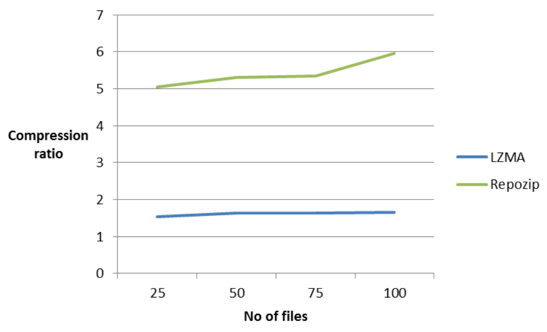

Fig. 3. Direct LZMA compression ratios vs. RepoZip compression ratios for similar .docx document collection

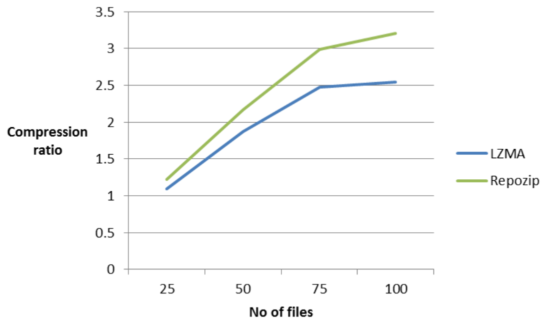

Fig. 4. Direct LZMA compression ratios vs. RepoZip compression ratios for different .docx document collection

B. *Results for PDF Document Collections*

With similar PDF documents, the improvement is even higher. According to Fig. 5, the start itself reflects how well the RepoZip can improve the redundancy detection. At 10 documents, it is almost the double of LZMA ratio. The increasing rate is again much higher while LZMA continues to be leveled with slight variations below the compression value of around 8. However, there is a tendency for it to get increased at the number of documents being 100. Still, RepoZip achieves almost 3 times greater compression than LZMA by that point.

According to the observations regarding different PDF document collections, the rate of compression ratio increase in RepoZip is much higher than that of the rate of compression ratio increase in direct LZMA. Even though both start the trend with nearly similar ratios, it can be well seen that the LZMA compression ratio always stays under the value 2, whereas for RepoZip it dramatically increases up to a value of around 12 when the number of documents is increasing. It is almost a 6 times of compression than that of the direct LZMA. With more and more number of documents, it can be expected the future trend of RepoZip to be continuously increasing.

TABLE IV. COMPRESSION RATIOS OF SIMILAR PDF DOCUMENT COLLECTIONS

No of Files	LZMA Ratio	RepoZip Ratio
20	5.453934	12.29697
40	6.385484	18.72163
60	5.333639	19.69974
100	9.112742	33.80558

TABLE V. COMPRESSION RATIOS OF DIFFERENT PDF DOCUMENT COLLECTIONS

No of Files	LZMA Ratio	RepoZip Ratio
20	1.812225291	6.944494
40	1.776723635	10.04498
60	1.689890608	10.65335
100	1.891340342	12.38411

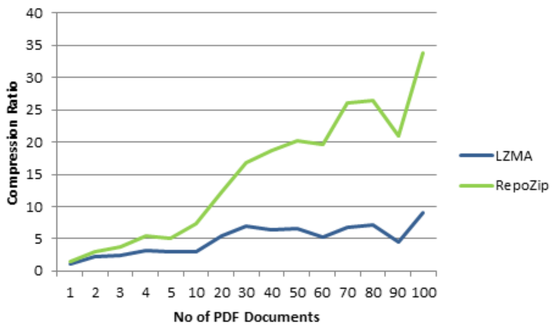

Fig. 5. Direct LZMA compression ratios vs. RepoZip compression ratios for similar PDF documents

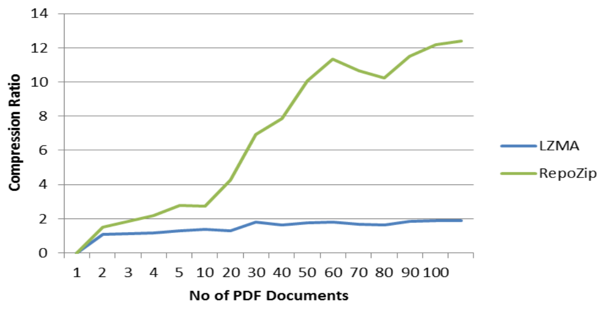

Fig. 6. Direct LZMA compression ratios vs. RepoZip compression ratios for different PDF documents

C. *Overall Discussion*

From overall results of .docx and PDF documents, it can be seen that when a repository with similar documents is to be compressed, RepoZip can largely improve the compression of LZMA. For that kind of a document collection, both meta-data level and content-level redundancies spread across the collection can be exploited. When it is a collection of completely different documents, RepoZip improves the compression, exploiting the meta-data level similarity spread across the collection. However, the attainable ratio depends heavily on the percentage of similarities which spread across the collection. Therefore, further results are required to be obtained over a standard benchmarking document collection for each case. Also the above results were taken for each document type upon compressing samples containing varying number of documents; accounting for the sample size with number of samples as 10 for each collection size. This might be the reason for observing dips and dramatic drop downs of the compression ratios. Therefore it is also required to increase the sample size and test the technique further on more documents collections. However, the clear improvement of LZMA through RepoZip is very well depicted from the current results.

VI. PARALLEL AND FUTURE WORK

In parallel to current RepoZip implementation, another method was looked upon in order to further improve the compression. It was to use a Generalized Suffix Tree data structure [10] for storing the meta-data clusters. The research was focusing on capturing each and every redundancy by storing the document strings in a single generalized suffix tree. The plan was to write each cluster file as a single generalized suffix tree into a binary file. The collection of such binary files will be the input to a compression algorithm.

This approach has shown very positive results for the documents which are about 15KB in size. However it takes a long processing time for larger documents than existing techniques. Thus we hope to concentrate on that issue at a lower level, focusing on concurrent programming approaches. It would be a feasible and an effective solution as the implemented system can be executed by running tasks in parallel. Constructing a separate suffix tree for each cluster is one of the major steps which can be parallelized. The space that the generalized suffix tree consumes in the construction phase is also another main issue that should be taken into consideration. The proposed solution for this issue is, keeping data related to a single cluster at a given time by writing the rest back to the auxiliary memory.

Another improvement is to perform content based chunking that divides documents into chunks and generates hash values for each chunk. This approach can also be used to enhance the compression of obtained clusters since the major redundancy detection has already been performed just as in migratory compression [3].

VII. Conclusion

RepoZip is a technique which improves the compression of an existing compression technique over a collection of OOXML documents or a collection of PDF documents. Research related to this novel approach for compressing such document collections basically focused on exploiting the meta-data and content-level redundancies that spread across such large collections of documents and proving the hypothesis that a higher compression ratio can be obtained by decompressing already compressed OOXML documents and PDF content streams in order to take advantage of their hidden inter-document redundancies as well. To facilitate the redundancy detection done by usual sliding window approaches like LZMA, the documents content is extracted and clustered based on their meta-data type (i.e. XML type/content stream type). The clusters are then written into files, which in turn will be the input to an existing compression scheme.

Compared to direct LZMA compression, RepoZip gives a very high compression ratio in compressing not only a highly similar document collection but also very different document collections. Through this research, we have been able to prove the level of compression improvement which we can achieve by going into the meta-data level of a large collection of documents. Also we have been able to adapt the concept of migratory compression [3] at an application level for improving the meta-data level redundancy detection by using a clustering approach based on the meta-data content type. Hence, RepoZip can be considered as an initiative for better improvement of existing compression techniques.

References

[1] T. Suel and N. Memon, “Algorithms for Delta Compression and Remote File Synchronization”, In Khalid Sayood, editor, Lossless Compression Handbook, 2002.

[2] Jean-loup Gailly and Mark Adler, *zlib* 1.2.8 Manual. [Online]. Available: http://www.zlib.net/manual.html

[3] X. Lin, G. Lu, F. Douglis, P. Shilane, G. Wallace, “Migratory Compression: Coarse-grained Data Reordering to Improve Compressibility”, In FAST, pp. 257-271. 2014.

[4] P. Shilane, G. Wallace, M. Huang, and W. Hsu, ”Delta Compressed and Deduplicated Storage Using Stream-Informed Locality”, in Proceedings of the 4th USENIX conference on Hot Topics in Storage and File Systems, USENIX Association, 2012.

[5] D. Bhagwat, “Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup”, in Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009.

[6] “Document management -Portable document format -Part 1: PDF, 1.7”. [Online]. Available: http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf

[7] “PDF Reference, sixth edition, Adobe® Portable Document Format", Adobe Systems Incorporated [Online]. Available: http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

[8] “Microsoft Developer Network Office 2007 Documentation”. [Online]. Available: https://msdn.microsoft.com/enus/library/aa982683(v=office.12).aspx.

[9] Structure of a WordprocessingML document (Open XML SDK) [Online]. Available: http://msdn.microsoft.com/enus/library/office/gg278308%28v=office.15%29.aspx.

[10] E. Ukkonen, "On-line construction of suffix trees", Algorithmica 14.3 (1995)