INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X

Impact Factor: 6.078

(Volume 8, Issue 3 - V8I3-1379)

Available online at: https://www.ijariit.com

Efficient cloud data storage and file management using AWS S3 bucket

B. Akhil Kumar
akhilkumar332@gmail.com
Gandhi Institute of Technology and Management, Visakhapatnam, Andhra Pradesh

Dr. T. Uma Devi
utatavar@gitam.edu
Gandhi Institute of Technology and Management, Visakhapatnam, Andhra Pradesh

ABSTRACT

End-user data is regarded as the foundation of everything in the IT world. Every day, a massive amount of data is added to the system, and an even greater amount of data is modified every second. Everyone expects data storage to be mobile and for their data to be safe and secure. While keeping data secure, data acquisition should be simple and quick. To that end, one of the most dominant and dependable technologies of the twentieth century has been provided by one of Amazon Web Services many products, Simple Storage Service (S3), which can store and retrieve massive amounts of data from any type of application, whether web-based or mobile, at incredible speed. The data in S3 is stored in a flat-file system that is simple to understand and implement. Amazon S3 (Simple Storage Service) offers object storage designed for storing and retrieving any amount of information or data from anywhere on the internet. It offers this storage via a web services interface. It can also store various file formats up to 5 terabytes in size. The goal of this project is to make it simple to upload and retrieve files from AWS S3 storage using various compression techniques, as well as to perform file operations on them. This way, one can determine which compression technique works best in terms of compressed data size, and If the original data or file is corrupted, compromised, or destroyed, a cloud backup can be used to restore it. The proposed methodology also addresses some of the limitations of AWS S3\'s flat file system and provides a solution in the form of a directory structure, which when implemented can result in increased security and can address inconsistency even for large databases

Keywords: *AWS S3 Bucket, Huffman Compression Algorithm, LZW Compression Algorithm, File Management*

1. INTRODUCTION

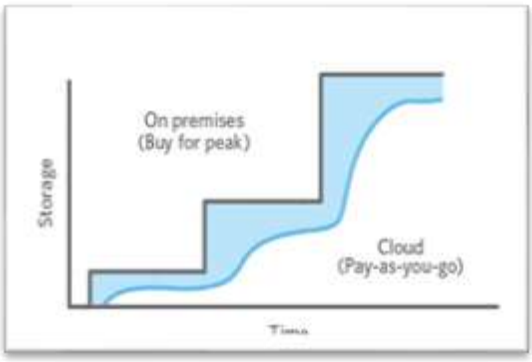
Cloud computing is a type of cloud computing in which data storage is provided as a service that is managed and administered through the Internet by a cloud computing provider. This is referred to as cloud storage. A hosting business often owns and manages the physical environment, which is typically dispersed over numerous servers (occasionally in various countries). These cloud storage providers are in charge of ensuring the safety, security, and operationality of the data as well as the physical environment's security, protection, and functionality. Users and corporations purchase or rent storage capacity from service providers for the storage of personal, organizational, and application data. Storage in the cloud is built on highly virtualized infrastructure and multi-tenancy, metered resources, and instant elasticity and scalability are all examples of this, which is comparable to larger cloud computing. On-premises and off-premises cloud storage services are both available. A third-party cloud provider owns and manages data storage space that it sells on a pay-as-you-go basis over the Internet. On-premises and off-premises cloud storage services are both available. A third-party cloud provider owns and operates data storage space, which it sells on a pay-as-you-go basis over the Internet. These cloud storage solutions monitor capacity, security, and durability to ensure that your data is accessible to your apps from any location on the planet. Applications use traditional storage protocols or an API to access cloud storage. Many companies offer additional services to help with data collection, management, security, and analysis.

1.1 Benefits of Cloud Storage

IT departments can transform three areas by storing data in the cloud:

Total Ownership Cost: There is no need to purchase hardware or provision storage with cloud storage. You can easily add or remove capacity, modify performance and retention settings, and only pay for storage space used, resulting in cost savings. Infrastructure should never hold back development teams when they are ready to execute. Cloud storage allows IT to provide exactly the amount of storage needed when it is needed. This frees IT resources so that they can focus on complex application problems rather than storage system administration.

Information Management: New use cases gain tremendous scalability by storing data in the cloud. Using policies for cloud storage lifecycle management, you can support compliance requirements by performing powerful data management tasks such as automated tiering or data locking.



1.2 Applications of Cloud

Backup and Recovery: Backup and recovery are essential components of data protection and accessibility, but keeping up with increased capacity demands can be a constant challenge. Cloud storage makes backup and recovery solutions more affordable, durable, and scalable. Based on frequency or timing settings, embedded data management policies can migrate data to lower-cost tiers automatically, and archival vaults can be built to help meet legal or regulatory requirements. These benefits enable massive scale possibilities in industries such as finance, healthcare, and media that generate large amounts of data with long-term retention requirements.

Software Test and Development: Environments for software testing and development frequently necessitate the construction, management, and decommissioning of separate, independent, and duplicate storage environments. Aside from the time required, as well as the initial capital costs, can be substantial. Some of the globe's largest and most successful corporations have developed applications in record time by leveraging cloud storage's flexibility, performance, and low cost. Even some of the most basic static webpages can be greatly enhanced for a surprisingly low cost. Pay-as-you-go storage options are being used by developers all over the world to address management and scalability issues.

Big Data and Data Lakes: Traditional on-premises storage technologies can be inconsistent in terms of cost, performance, and scalability, especially over time. Data lakes, which are large-scale, low-cost, highly available, and secure storage pools, are required for big data projects. Data lakes built on object storage keep information in its original form and include rich metadata that enables selective extraction and analysis. Web data lakes can be used to power all types of data warehousing, processing, big data, and analytical engines, allowing projects to be completed in less time and with greater relevance.

2. EXISTING SYSTEM

The cloud's storage constraints, when it comes to finding cloud storage service providers, one of the most common concerns is often the cost. Unfortunately, calculating the exact costs of cloud computing services can be difficult due to complicated pricing models and the overall challenges associated with predicting how many resources your company will require for complete cloud data management. The pricing is directly proportional to the amount of storage consumed; which is, in turn, a major drawback when dealing with huge volumes of data. Businesses that manage such huge amounts of data, quickly exhaust their limit; and find themselves requiring more physical storage from the hosting company.

3. PROPOSED SYSTEM

The project entitled "Efficient Cloud Data Storage and File Management Using AWS S3 Bucket" is a web-based application which offers a cloud platform for a user to store files. However, contrary to conventional storage systems, this storage does so by compressing the files into a compact manner using a few algorithms. This saves not only space on the cloud but also allows more files to be saved. Compression is a highly efficient way of file management in that; it reduces the size of the file by 90% without any data loss. The algorithms used in the compression of the files are discussed in the below sections.

4. PURPOSE OF THE SYSTEM

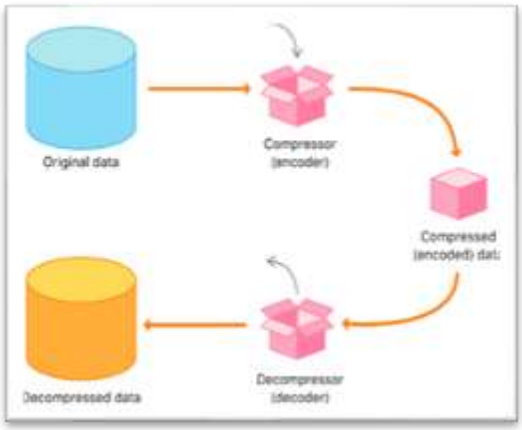
The data stored in S3 uses a flat-file system that is easy to understand and implement. Amazon S3 (Simple Storage Service) offers object storage designed for storing and retrieving any amount of information or data from anywhere on the internet. It offers this storage via a web services interface. It can also store various file formats with a maximum size of 5 terabytes. The goal of this project is to make it simple to upload and retrieve files from AWS S3 storage using various compression techniques, as well as to perform file operations on them. This way, one will be able to check which compression technique works efficiently in terms of compressed data size and also if the actual data or file is compromised, affected, or destructed, a copy on the cloud is available for recovery.

5. DATA COMPRESSION

The process of reducing the number of bits required to represent data is known as data compression. Compression of data can help you save storage space, speed up file transfers, and save money on storage hardware and network bandwidth.

5.1 How Compression Works

A programme performs compression by using a method or algorithm for determining how to reduce data size. For instance, a method may constitute a sequence of numbers — by transforming it with a dictionary of zeros and ones. Alternatively, the technique could include a reference to a previously noticed string of zeros and ones. To compress text, simply remove all unnecessary characters, consider replacing a commonly encountered bit string with a relatively tiny bit string, and use it to imply a string of repeated characters, use the repeat character. A file can be compressed to half the original size or a much greater percentage. Compression can be applied to individual data items or to the entire transmission unit, including header data for transmission. Larger files may be sent or received over the internet in ZIP, GZIP, or another compressed format, either alone or as part of an archive file.



5.2 Why Is Data Compression Important

Lossless compression can significantly reduce the amount of space required for a file. A 40-megabyte (MB) file, for example, takes up 20 MB of space in a 2:1 compression ratio. Administrators spend less money and time on storage as a result of compression. Compression improves backup storage performance and has recently been used to help reduce data in storage media. As data continues to grow at an exponential rate, compression will become an important method of data reduction. Almost any file type can be compressed, but best practices should be followed when planning what ones to compress. A few files, for example, may already be compressed, making compressing them ineffective.

5.3 Data Compression Methods: Lossless and Lossy Compression

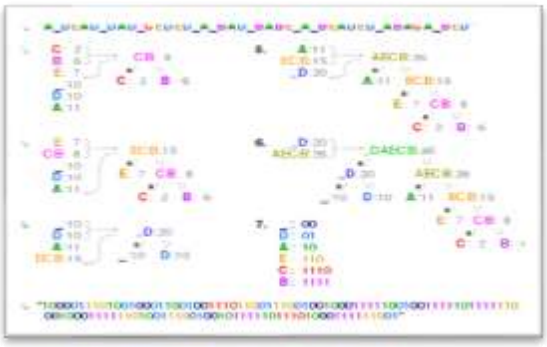
Data compression can be either lossless or lossy. When a file is uncompressed, it can be restored to its original state using lossless compression without losing a single piece of information for application programs, as well as text and word - processing files, where the failure of words or numbers would change the information, lossless compression is the standard approach. Algorithm for lossy compression removes redundant, insignificant, or imperceptible data bits permanently. Compression algorithm is useful in situations where removing some data bits, such as graphics, have hardly any noticeable impact on the content's representation, audio, video, and images. Lossy or lossless technique can benefit from text/media compression. Because Visual image file formats are substantially larger because they are intended to compress data. Digital image appears to support lossy compression as an image file format. Formats such as Picture use lossless compression.

6. ALGORITHMS

In this project, the Huffman Algorithm and the LZW Algorithm are used for data compression.

6.1 Huffman Algorithm

Huffman Coding (also known as Huffman Encoding) is a data compression algorithm which is the foundation of file compression. Huffman encoding is an algorithm that uses the frequency (or probability) characteristics of symbols and the structure of binary trees. Huffman encoding is a data compression algorithm used for data reduction, generating optimum, varying-length codes. Huffman coding is a common type of optimized encoding prefix used to compress data without sacrificing quality. Huffman code is usually useful to compress data where frequent characters are present. It is recommended that the Huffman tree discard the characters not used in the text in order to generate a more optimal coding length. Huffman encodings approximate each character's probability in order to avoid the challenges of encoding characters with a power of half their actual probabilities using an unintegral number of bits. Huffman's greedy algorithm constructs the best way to represent each character is represented as a binary string by using a table that gives the probability of appearance of each character (e.g., its frequency). The time complexity is not terribly important because the number of symbols in an alphabet is very minimal (in comparison to the length of a message that needs to be encoded), when choosing a greedy algorithm for Huffman. If the estimated frequencies for letters are in line with the actual frequencies found in an encoded message, the length of this message is usually shorter than it would have been had a fixed-length code been used. Because all the lengths of binary codes are different, decoding software has difficulty detecting if encoded data is damaged. In other words, as soon as we get to the last bit of a code in the decoding process, we know what letter that is a code for. When decoding a character using a Huffman code tree, we take a path through the tree that is determined by bits in the code line. Any prefix-free binary code can be visualized as a binary tree, with the coded characters stored on the leaves.



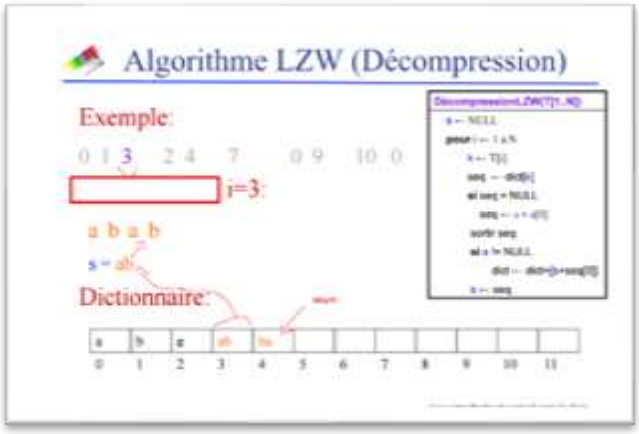
In general, the act of translating the prefix-code stream is referred to as decompression, into the individual byte values, typically by walking the frequency-ordered binary each bit from the input stream is read, and the tree is built up node by node (reaching the leaf node necessarily ends the search for that specific byte value). Prefix codes are variable-length codes assigned to input characters, which assumes that the code assigned to a single character is not a prefix to the code attributed to any other character. The input characters are assigned variable-length codes, and their sizes are ascertained by the frequencies of the corresponding characters. Building a Huffman Coding TreeAP Huffman codes assign codes to characters, as a result, the length of the codes is determined by the relative frequencies, or weights, of the corresponding characters. Huffman encoding uses variable-length coding schemes for assigning binary codes to characters according to their frequency of appearance in the given text. The output of David A. Huffman can be thought of as a variable-length coding table used to encode source symbols (such as characters). David A. Huffman is optimal for encoding symbols one by one with known probability distributions of inputs, that is, for separately encoding irrelevant symbols within such a data flow. Because Huffman encoding algorithms employ probabilities, symbols that are more frequent--symbols that are highly probable--are typically represented using fewer bits than symbols that are less frequent. This research also discusses fixed-length vs. variable-length encoding, unique decodable codes, prefix rules, and the construction of the Huffman tree because inserting an effective prioritized queue data structure takes O(log(n)) time, there are 2n-1 nodes in a full binary tree with n leaves and the Huffman Code. The tree is completed in O(n.log(n)) time, where n represents the total characters.

6.2 LZW Algorithm

The LZW algorithm is a compression algorithm from a commonly used Unix - based operating utility which can be used for GIF file types. LZW is the leading technology in the field of generic data compression because of its simplicity and universality. LZW compression works by reading a sequence of symbols, bundling the symbols together in strings, and turning the strings into codes. LZW recognizes repeated sequences in the data and appends them to the code table as the encoding progresses. Each code from a compressed file is decoded by running it through a code table to determine which symbol or characters it represents. At each compression step, input bytes are constructed until the next character yields a series with no codes in the dictionary. Both algorithms have an encoder and a decoder, which are used to compress.txt files and decompress

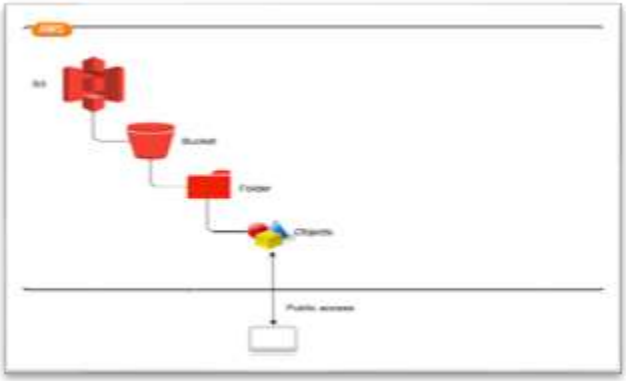
them back to the original size. Some code streams are compressed into printable characters using binary-to-text encoding, which increases encoding length while decreasing compression speed. Various Methodologies to compression techniques designed for text files also use Dictionary to demonstrate all words in a file, such that the file is encoded as a list of pointers to dictionaries, or to represent common words and word endings, such that the file is composed of pointers to the dictionary and encodings of less common words. Most formats use LZW, which incorporates this information into the format specification or includes explicit fields in a data compression header for it to clear and stop codes.

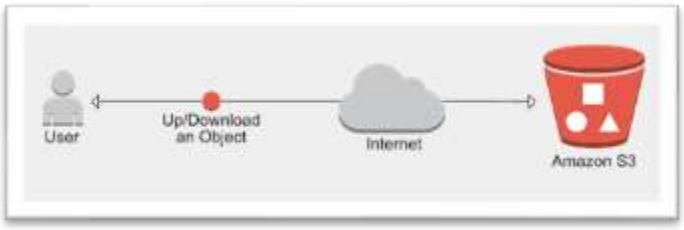
Fixed-length 12-bit codes represent one-character sequences made up of corresponding 8-bit characters, For sequences encountered, codes 256 through 4095 were created in the dictionary, while encoding 8-bit data. Huffman will detect frequency in the bytes (let us say that the above text is ASCII or UTF-8 (which would make ABC all one-byte code points), then A = 3, B = 3, C = 3, and no more, then Huffman could represent all characters using 1.5 bits (well, 1 bit combined with two bits). In the main, LZW is concerned with the frequency of repetitions, and Huffman is concerned with the frequency of single-byte occurrences. LZW combined with Huffman encoding is a lossless compression technique which takes advantage of both repeated words but also things occurring most often as opposed to things occurring least often (the point of that phrase is likely to take some time to sink in).



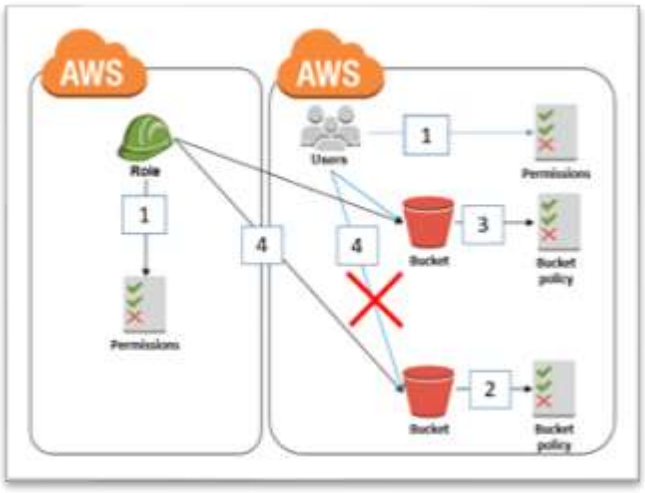
With some abstracting, it is possible to use Huffman encoding to enhance LZW compression results (indeed, this is what most people do). LZW compression outperforms LZW-based Huffman encoding. We could use the LZW Encoding Technique to reduce file size. At the beginning, I wanted to compress the original data with Huffman encoding and then with LZW encoding. However, this does not provide a better compression ratio than LZW single compression.

7. SYSTEM ARCHITECTURE

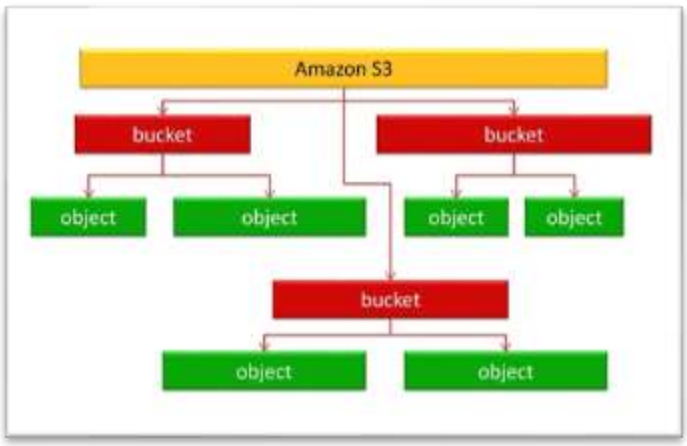




8. CLASS DIAGRAM



9. SEQUENCE DIAGRAM



10. RESULTS & DISCUSSION

Following the initial file upload, the user will be presented with two compression algorithm techniques from which to choose in order to download the encoded version of the uploaded file.

As the file is uploaded, go to the "status" page and compare which compression algorithm provides the best compression ratio.

To successfully upload any file to an AWS S3 bucket (storage service), the user must go to "File Manager" and click "Upload file," where the file manager supports all file formats.

After successfully uploading the file to the file manager, the us er can perform operations such as "Download," "Share," "Delet e," or "Upload."

11. CONCLUSION AND FUTURE SCOPE

This project is an attempt at overcoming the shortcomings of the Amazon S3 Bucket storage architecture, by providing the clients with the feasibility of constricted storage by implementing dedicated compression algorithms. Contrary to conventional cloud storage systems, this project allows users to manage their space by choosing the compression algorithm of their choice. The use of Huffman's Algorithm and LZW Algorithm helps in compressing a wide variety of file types. The choice of algorithms used is based on the requirement of either lossless or lossy compression as and where applicable for the concerned file types. Additionally, the performance metrics of the application show its capability to compress huge files in a short amount of time; thereby promising faster performances and efficient time management in a practical scenario.

Real-time implementations of the project show vast prospects in the field of enterprise storage systems, where it is imperative to handle huge amounts of data on a regular basis. Backups and archives can be made quickly and the cloud storage handled efficiently; all thanks to the implementation of the compression techniques as discussed.

Future iterations of the application can be rolled out with updates that feature user-based application functionality, by restricting and segregating accessibility and functionality for different users. Also, access to the cloud storage can be configured to be based on an individual account model; thereby allowing users to maintain the privacy of files and data stored on the cloud. Additional levels of security can be configured to prevent unauthorized access and other cyber-attacks.

This project is an initial proposal to the idea of running cloud storage based on Amazon S3 by implementing data compression algorithms on various files. Any and all modules of the application including front end, back end, algorithm implementation, etc. have been practically implemented and this document was made by assessing all the aspects of the project.

12. REFERENCES

[1] Amazon simple storage service api reference, api version. *Amazon Web Services LLC.*

[2] Lakshman A, Malik P. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*; 2010.

[3] Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D. The eucalyptus open-source cloudcomputing system. *IEEE International Symposium on Cluster Computing and the Grid*; 2009, p. 124–131.

[4] Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, and Zaharia M. Above the clouds: A berkeley view of cloud computing. *Technical report*; 2009.

[5] Extension of Huffman Code & Pattern Recognition through fuzzy logic.

[6] Enhanced LZW Algorithm with Less Compression Ratio.

[7] Compression Using Fractional Fourier Transform A Thesis Submitted in the partial fulfillment of requirement for the award of the degree of Master of Engineering In Electronics and Communication, By Parvinder Kaur.

[8] RL-Huffman Encoding for Test Compression and Power Reduction in Scan Applications-mehrdad nourani and Mohammadh tehranipour, The University of Texas at Dallas.

[9] Hemasundara Rao, "A Novel VLSI Architecture of Hybrid Image Compression Model based on Reversible Blockade Transform C", Student Member, IEEE, M. Madhavi Latha, Member, IEEE.

[10] D.A.Huffman, "A Method for the construction of Minimum-redundancy Codes", Proc. IRE, vol.40, no.10, pp.1098-1101,1952.

[11] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, May 1977.

13. BIBLIOGRAPHY

B. Akhil Kumar
Gandhi Institute of Technology and Management, Visakhapatnam, Andhra Pradesh, India

Dr. T. Uma Devi
Gandhi Institute of Technology and Management, Visakhapatnam, Andhra Pradesh, India

