**A DISTRIBUTED FILE SYSTEM FOR DEDUPLICATED PRIMARY CLOUD STORAGE**

**1) What is the problem that the thesis deals with?**

With the growing demand for data rates around the world, cloud computing systems are becoming a necessity for computer users. All users upload their data to cloud storage and receive data stored remotely where necessary. However, 70% of the data stored in the cloud storage is not required, which does not allow us to use the storage space properly. Data extraction is the process of removing unwanted items from cloud storage. Data fragmentation can occur in a number of ways: duplicate customer duplication e.g. In server side configuration, after uploading data to cloud storage server, the server will look for unwanted data items, delete them. Data fragmentation is possible in multiple granularities, eg file level, and block level. Storing data in remote storage e.g. cloud storage raises several challenges and security. The most common solution for data encryption is to encrypt data before uploading it to cloud storage.

Cloud storage is an example of a service where data is stored, managed, backup, and made available to users over the Internet. With redundancy prevalent in cloud storage, it is important to use a good storage space. To achieve this, the process of optimization, i.e., writing, is widely used. This method basically separates all incoming files into a set of standard or variable level blocks. Next, the secure hash algorithm 1 is used to find the hash value corresponding to one of these blocks. These hash values, also called fingerprints, are compared to the index finger for duplicates and only one example of duplicate blocks is kept. Each entry in the index finger is like a specific block with its own fingerprint, location, and reference number. Since each file is stored as a set of blocks in duplicate cloud storage (DCS), the entire set of key blocks is required to rebuild the file during reading. To make this easier, the file recipe keeps fingerprints of these existing blocks. These blocks can be unique to a specific file or can be shared to multiple files.

All DCS access checks the metadata, which is an index of fingerprints and a file recipe. During the writing process, all incoming blocks are compared to the input fingerprint of their presence. If the block is already in storage, its reference count alone is increased by 1. Alternatively, it is installed in storage and the index finger is updated with the corresponding installation. During deactivation, the reference count for each block associated with that file is reduced by 1. During the learning process, the file recipe is first touched to find the existing blocks. After that, the locations of these blocks are located on the index finger to cover the entire file.

In the DCS context, blocks are usually shared between multiple files. As a result, the same users can simultaneously enter the fingerprint corresponding to the block. Typically, a read request accesses location information from block login to file reconstruction. Writing or canceling an application to increase or decrease the reference count. Therefore, when parallel requests attempt to update reference simultaneously, the statistical value may be inconsistent. This problem is called "missing update."

The garbage collection process from time to time removes blocks in the storage area where the reference value becomes 0. Due to the missing update problem and inconsistencies in the reference count, it is possible that one or more blocks may be garbage-collected earlier while other files continue to refer to it. This will lead to error of the re-building of the file due to lack of data blocks while attempting to file to go back to work while learning. Therefore, the timing of money management is an important research problem that needs to be addressed in the context of DCS. An earlier proposed the block-level postprocessing deduplication for a live file system cluster where the income deduplicated data only after data is written to disk. In this function, all shared fingerprints are locked by the host at the time of access to their entries. Such a severe closure strategy is not suitable for the current DCS context that performs line duplication, leading to increased response time. Therefore, in this current work, reinforce the structure of the reference fingerprints used in building a good lock and methods necessary to control the money while basic services.

**2) What is the proposed solution?**

There are multiple techniques proposed in this paper:

**Frequent-pattern based prefetching technique**

In cloud storage, while re-utilizing efficient use of storage space, there is also a large availability of storage metadata, namely Fingerprint Index and File Recipe. As this metadata is large, it should be stored on a disk that causes significant learning delays in Deduplicated Cloud Storage (DCS). To improve this learning latency, it can be very helpful to attach the appropriate fingerprints to the repository. Many existing research solutions have used location with similarity or similarity between files to trace appropriate fingers. However, the DCS designed and used in this paper is intended to address a non-backup load that does not reflect a large local area or similarity between files. Therefore, this paper proposes another convenient pre-order approach that enables a customer pattern to gain access to the most accessible files. The proposed pre-planning approach has been implemented and implemented in DCS

**Striped fingerprint index and the concurrency control algorithms**

The traditional fingerprint index contains a set of entries related to each block available in storage. At the same time, when one request reaches the set of block entries, it is not possible to restrict another request to renew this set partially or completely. This leads to incompatible metadata. To avoid this inconsistency issue, it is important to lock all block entries associated with a particular file to submit a specific application. Since a file can have multiple blocked entries, locking and unlocking all block entries involves considerable consideration. Alternatively, the entire fingerprint can be locked in an effort to maintain metadata matching. However, this completely removes any similarities, leading to poor performance. Therefore, as a trade-off between an over-locking area and a complete lack of similarity, it would be better to only lock in a block of compatible block that matches a particular file. Since the current fingerprint design does not help such a good lock, it is important to reset it. In this context, this paper proposes a fingerprint index (SFI) that forms a fingerprint index at two levels. The first-level index maintains the hash value of the file path and the second-level index identifier, which contains inputs corresponding to the linked blocks

**Location-independent index and the virtual server based load balancing algorithm**

The traditional reduction system uses metadata, which is a fingerprint index and a file recipe. Fingerprint details on the fingerprint are used to get duplicates. Similarly, location details related to data blocking in the index finger and the order of the data blocks found in the file reference are required to rebuild any specific file in duplicate storage. However, since the duplicate data node collection built into this paper uses DHT, the file identifier, which is stored in the view table, is responsible for placing and retrieving separate blocks associated with the file. In addition, even if the blocks move from one node to another due to the addition or removal of a node, the view table is automatically updated to show the change from time to time. Therefore, it may be noted that it is no longer necessary to store location information in the fingerprint to keep track of location blocks of data. In light of this, an independent fingerprint index (LIFI) that exploits access to the Chord protocol has been suggested in the current work. All blocks have LIFI access containing fingerprints, its file identifier, and reference number, indicating the number of files shared by this block. Since the blocks are placed in a data set using a file identifier, they need to find the file. While details regarding file identifiers can also be found in LIFI, the computational overhead is quite high due to the large size of LIFI. Therefore, Rich File Recorder (EFR) is designed to store fingerprints and file identifiers that correspond to each file reconstruction. These index properties are stored in a metadata node for duplicate storage

Typically, a collection will be composed of multiple nodes. The load of a particular node in a collection is called the current value of the physical storage in that node. General capacity (λ) and total load (µ) refer to the combination of maximum physical storage and the amount of current storage usage for all nodes in the collection respectively. During use, load imbalances may occur between group nodes.

**Deployment aspects of the proposed distributed file system**

JaguarFS has been suggested to improve DCS performance with appropriate pre-planning process, load balancing process and cost management methods while simultaneously maintaining metadata compliance. This file system removes and duplicates all incoming files according to their type and size. Thanks to this smart retrieval app, metadata processing is greatly reduced compared to programs where a single decision-making process is applied to all incoming files regardless of their field. As JaguarFS learns the learning access pattern in preetch

metadata of regular files, read readings greatly improved. Additionally, it does not require metadata for each block that creates the return file. On the other hand, it is enough to have only the stripes corresponding to the file. These strings can be found in 128 EFR matching that file. Therefore, JaguarFS reduces the pre-configuration of metadata at the file level instead of block. Therefore, drastically reduce the metadata line to disk and cache. Therefore, it is possible to add additional metadata to the FPBPT repository that continues to improve readability because a growing number of read requests can be used by the cache itself. In addition, the installation of moderately loaded nodes for data transfer during load measurement work has had a positive impact on the overall performance of JaguarFS.

**3) What is the future scope of this?**

The proposed JaguarFS DCS focused on providing solutions for optimization, load balancing and metadata compatibility. This function can also be extended to achieve higher availability and error tolerance. In the proposed DFS, despite the use of the appropriate fingerprint from disk to storage, as the predicted number of fingerprints eventually changes over time, the challenge is to design a repository with good size to hold the entire set predicting fingerprints or at least its large set to achieve optimal learning readings. This goal should be pursued in the future. In the DCS context, the duplication of blocks is completely eliminated in order to make better use of the storage space. This leads to difficulty rebuilding files, even if one block is lost. Additionally, if that block is shared with multiple files, the loss will be even greater. Therefore, a better way to simulate a policy-based blocks could be developed. In addition, DCS stores all incoming files as a set of blocks that are distributed in multiple storage locations. In this context, if a node fails, the required blocks in several files will be found. Therefore, an effective way can be developed to provide improved tolerance for mistakes.