Optimized Cloud Storage with High Throughput Deduplication Approach

Y. V. Lokeshwari
PG Student Department of Computer Science and Engineering SSN College of Engineering, Anna University Chennai – 603110, Tamil Nadu, India

B. Prabavathy Assistant Professor Department of Computer Science and Engineering SSN College of Engineering, Anna University Chennai – 603110, Tamil Nadu, India

Chitra Babu Professor and Head Department of Computer Science and Engineering SSN College of Engineering, Anna University Chennai – 603110, Tamil Nadu, India

ABSTRACT
Cloud computing has revolutionised e-commerce by facilitating the consolidation of computing and storage resources. Many organizations have set up private clouds as it results in better utilization of resources. Private cloud storage can be built from the unused resources to store the data that belongs to the organization. Since private cloud storage has a limited amount of hardware resources, they need to be optimally utilized to accommodate maximum data. Deduplication is an effective technique to optimize the utilization of storage space. Two methods adopted for deduplication, namely, chunk level and file level, are studied here. This paper discusses the implementation of both these methods in cloud storage through a case study. The present work also proposes a variation in file level deduplication to further increase the throughput.

Keywords
Private cloud storage, duplicate detection methods, deduplication, Eucalyptus, Gluster file system,

1. INTRODUCTION
Cloud computing enables the consumers to obtain computing and storage resources as services over the internet. It provides advantages such as consolidation of resources, elasticity, scalability and pay per use business model. Cloud computing has facilitated better utilization of available resources. It enables organizations to set up a private cloud with unused commodity machines.

Private cloud storage can be built by consolidating the storage resources of an organization. This can be used for several purposes. Backup workloads, both traditional and non-traditional, can occupy the cloud storage. Traditional backup workload consists of streams of data with large stretches of repetition. In other words, there will be more locality of reference among different streams of data. For example, the arrangement of files in *My Documents* directory appear approximately in the same order in different backups of the *My Documents* directory on different days. Non-traditional backup workload consists of individual files belonging to different users with no locality of reference.

Private cloud storage needs to be better utilized as there is a physical limitation on the storage space. Deduplication is an effective technique for optimization of instances of data stored in cloud storage [7]. Deduplication can be classified into chunk level and file level deduplication. Chunk level deduplication method enforces the storage of unique chunks by comparing every incoming chunk for duplicate identification. This method achieves better deduplication efficiency because it does exact deduplication [1]. However, the throughput is low as it checks every incoming chunk for duplication. File level deduplication method enforces the chunks of similar files to be compared against duplicates. Files with incremental changes are referred to as similar files. This method achieves better throughput as it compares every incoming chunk only with chunks of similar files. However, the deduplication efficiency is comparatively low as some duplicate chunks may be found across different groups. Hence, this technique performs only approximate deduplication.

This paper focuses on a specific use case for private cloud storage, details of deduplication techniques with their performance and suggests a variation in the index of file level deduplication, which further improves the throughput. The paper is organized as follows. Section 2 discusses the architecture of various deduplication approaches. Section 3 explains the experimental setup for optimized cloud storage in detail. Section 4 analyses the optimized cloud storage for different kinds of workload. Section 5 proposes the architecture of the variation in file level deduplication and discusses the results. Section 6 concludes and provides possible future directions.

2. APPROACHES FOR DEDUPLICATION
In order to better utilize the storage space, duplicates among the files need to be identified. Every incoming file is either divided into fixed or variable sized chunks [6]. As variable sized chunking leads to better identification of duplicates, each incoming file is divided into variable sized chunks. Based on how the incoming chunk is checked against duplicates, deduplication can be categorized into two types, namely, chunk and file level.

2.1 Chunk level Deduplication
Whenever a data stream has to be written, every chunk in the stream is checked for duplicates before writing. This is termed as chunk level deduplication.

2.1.1 Data Domain Deduplication file System (DDDFS)

DDDFS [4] is a file system which performs chunk level deduplication. Figure 1 shows the architecture of DDDFS. DDDFS supports multiple access protocols. Whenever a file to be stored enters into the system, it is handled through one of the standard interfaces such as Network File System (NFS), Common Internet File System (CIFS) or Virtual Tape Library (VTL) to a generic file service layer. File service layer manages the metadata and forwards the file to the content store. Content store divides the file into variable sized chunks. Secure Hash Algorithm (SHA-1) [5] finds the hash value of each chunk. This is termed as Chunk_ID. Content store also maintains the file recipe to construct the file when it is read. File recipe contains the sequence of Chunk_ID which constitutes that file. Each chunk is checked for duplicates against a set of chunk indices maintained at the chunk store. Chunk index is the metadata that includes Chunk_ID and the location of actual chunks in storage. Unique chunks will be compressed and stored in the container. Container is the unit of storage. Container manager is responsible for allocating, de-allocating, reading, writing and reliably storing containers.

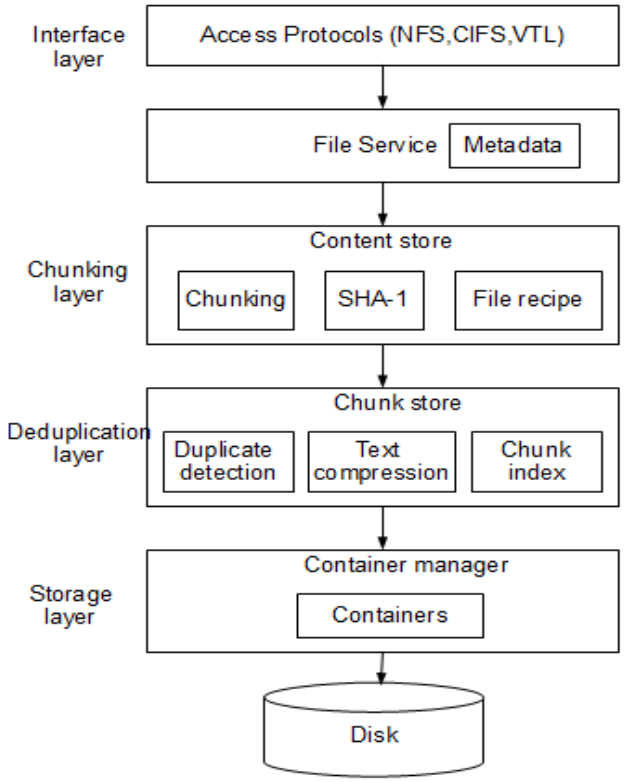

Fig. 1. Data Domain Deduplication File System Architecture

Since every incoming chunk is checked for possible duplication, only unique chunks occupy the cloud storage. Therefore, chunk level deduplication has better deduplication efficiency. However, as each incoming chunk is checked against a large list of chunk indices, the number of disk I/O operations is large. This has a significant impact on deduplication throughput. Storage of traditional backup workload demands good deduplication efficiency as it involves large data redundancy among different workloads. Hence, this deduplication approach is best suited for such workloads.

2.2 File Level Deduplication

Whenever a data stream has to be written, every chunk in the stream is checked against the chunks of similar files. This is termed as file level deduplication.

2.2.1 Extreme Binning: Scalable, parallel deduplication approach

This approach provides a scalable solution with the division of chunk index into two tiers namely Primary and Secondary index [3]. In this approach, all the Chunk_IDs that constitute a file and the minimum Chunk_ID among them are found. This minimum Chunk_ID is termed as representative Chunk_ID. According to Broder's Theorem, two files are said to be nearly similar, when the representative Chunk_IDs of both the files are same. Figure 2 depicts the structure of a two tier chunk index. Primary chunk index consists of representative Chunk_ID, whole file hash and the address of the secondary index or bin. Bin is made up of three fields namely, Chunk_ID, chunk size and the storage address of the chunk.

Whenever a file has to be backed up, it is chunked and both the representative Chunk_ID and the hash value for the entire file are found. Representative Chunk_ID is searched in the primary index and if it is not present, the incoming file is considered as new. Hence, a new bin is created and all Chunk_IDs, their corresponding size and a pointer to the actual chunks are added to the disk. Representative Chunk_ID, hash value of a new file and the pointer to the newly created bin are added to the primary index.

On the other hand, if representative Chunk_ID of the incoming file is already present in the primary index but the hash value of the whole file does not match, the incoming file can be considered to be nearly similar to the one that is already on the disk. Most of the chunks of this file will probably be available in the disk. The corresponding bin is loaded into the RAM, from the disk and it is searched for matching chunks of the incoming file. If the Chunk_ID is not found in the bin, then the metadata of the chunk is added to the bin and the corresponding chunk is written to the disk. The whole file hash value is not modified in the primary index and the updated bin is written back to the disk. If the whole file hash value of the incoming file is found in the primary index, then the incoming file is considered as a duplicate. Hence, the bin need not be updated.

Since every incoming chunk is checked only against the indices of similar files, this approach achieves better throughput compared to the chunk level deduplication. Since non-traditional backup workload demands better deduplication throughput, file level deduplication approach is more suited in this case.

3. EXPERIMENTAL SETUP FOR OPTIMIZED CLOUD STORAGE

Cloud storage is built with open source software such as Eucalyptus Framework and Gluster file system. Both the chunk and file level deduplication techniques are implemented. Cloud storage has been tested for deduplication efficiency and throughput using both the techniques. Testing has been

performed for the storage of files of different sizes. A case study involving a university scenario has been studied and the corresponding results are discussed.

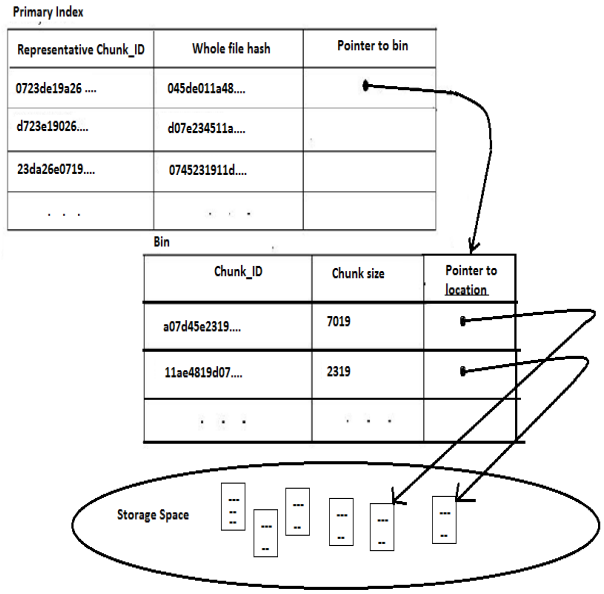

Fig. 2. Structure of the primary index and the bins

3.1 Case Study

Students in a university have access to multiple computers in different workplaces. For example, students working in laboratory, dormitory and class room uses different computers. Due to this, there may be several inconsistent copies of the same data. After some time, inconsistencies among the files in different workplaces lead to confusion. Hence, a new version of the file cannot be created from the existing one. This demands common online storage for the students of university. Private cloud storage can be built from unused commodity machines and can be utilized as a common storage to satisfy this demand. In this scenario, every student can be provided a user-id to access the common storage.

3.2 Cloud Test Bed

Figure 3 depicts the test bed of cloud storage which can be used for online common storage. This storage needs to be better utilized as there is a physical limitation on the storage size. Hence, an optimization technique has been incorporated in this to make this storage as an optimized one. An Optimized Cloud Storage (OCS) is built with the following steps.

3.2.1. Setting up Private Cloud using Eucalyptus

EUCALYPTUS is an open source software framework for cloud computing that implements Infrastructure as a service (IaaS) [8]. This provides the users an ability to run and control the entire set of virtual machine instances [9] that are deployed across variety of physical resources.

Eucalyptus consists of five entities namely Cloud Controller (CLC), Cluster controller (CC), Storage controller (SC), Walrus and Node controller (NC). The Cloud controller is in charge of the entire cloud that manages the resource allocation and user accounts. The Cluster controller performs per-cluster scheduling

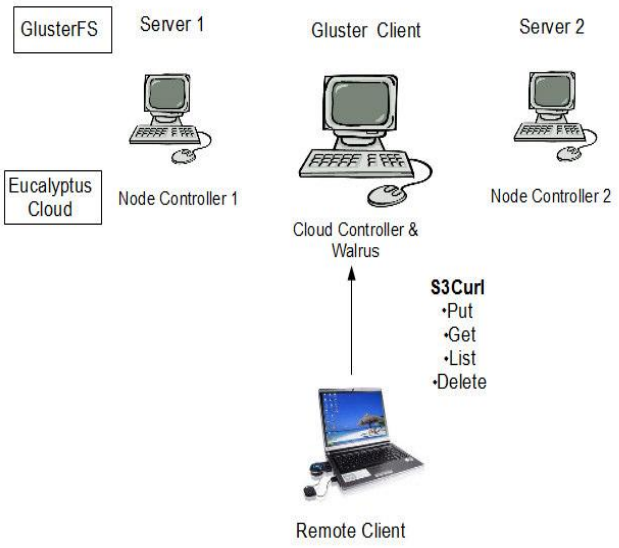

Fig.3. Cloud Test Bed

and networking. The Storage controller provides block storage services similar to Amazon's Elastic Block Service (EBS) [12]. The Walrus allows the users to interact with cloud storage via S3 interface or REST based tools similar to Amazon's Simple Storage Service (S3) [11]. The Walrus can store the data only on the machine in which it is installed [2]. The Node controller controls the hypervisor on each compute node and sets up the virtual machines.

3.2.2. Consolidation of storage resources

As Eucalyptus can not consolidate the storage resources of the commodity machines, a distributed file system has to be used for consolidation. Hadoop Distributed File System (HDFS) is the most common file system used in cloud applications. However, this is designed to support parallel and data intensive applications. Though this file system consolidates the storage resources, it cannot be mounted on a single point. Further, it does not provide any control to the user on the storage and retrieval of the files.

Gluster File System is used to set up a high-availability storage with many storage servers that use GlusterFS [14]. The client system can access the storage in the same way as it accesses a local file system. Gluster File System is used to combine the storage resources of different machines. It allows the user to mount the consolidated storage at a single mount point [15]. Further, it allows the user to control the storage and retrieval of the files. Hence, Gluster file system has been chosen to consolidate the storage resources.

One of the four machines is configured as CLC, CC, SC and Walrus. Rest of the machines are configured as Node controllers. Gluster File System is installed on the machines which are part of the cloud to consolidate their storage resources. This consolidated storage can be accessed from a single mount point.

Eucalyptus is essentially a set of web services. When the user makes a request for storage to the front-end web service, the

cloud controller forwards the request to Walrus. Walrus is compatible with Amazon's S3 which allows users to perform PUT, GET, DELETE, and LIST operations on the data.

3.2.3. Interacting with Walrus and Modifying Eucalyptus

Walrus allows the users to store persistent data organised as buckets and objects [10]. Users may use third party tools to interact with walrus. Third party tools for interacting with Walrus are:

1. S3curl is a command line tool that is a wrapper around curl [13].
2. S3cmd that allows easy command line access to storage that supports the S3 API.
3. S3fs that allows users to access S3 buckets as local directories.

Eucalyptus source code *WalrusManager.java* deals with the bucket creation, deletion, listing, putObject and getObject methods. Deduplication optimization technique can be incorporated into the *putObject and getObject* methods to build an optimized cloud storage.

4. PERFORMANCE STUDY

The cloud test bed has been used for testing the two deduplication techniques for file sizes between 5 KB and 50 KB with increments of 5 KB. The space saving ratio is termed as deduplication efficiency and time saving ratio is termed as deduplication throughput. There can be two kinds of data stream for the common online storage. They are:

i. Data stream of students belonging to the same department (Multiple users of same group)

ii. Data stream of students belonging to different departments (Multiple users of different groups)

Figure 4 shows the variation of deduplication efficiency with file size. The following inferences can be drawn from this figure:

1. Data stream of multiple users belonging to the same group takes less space compared to the one from different groups when they get stored in the OCS with chunk level technique. This is due to the fact that, there may be possibilities of more duplicates among the data stream of same group.
2. Storage of data stream of multiple users of different group takes same space irrespective of the deduplication technique used in the OCS.
3. Data stream of multiple users belonging to the same group takes more space in OCS with file level compared to OCS with chunk level. The reason being, there may be possibilities of duplicates existing among the bins in the case of file level deduplication, since it performs only approximate deduplication.

Figure 5 shows the variation of deduplication throughput with file size. The following observations can be made based on this:

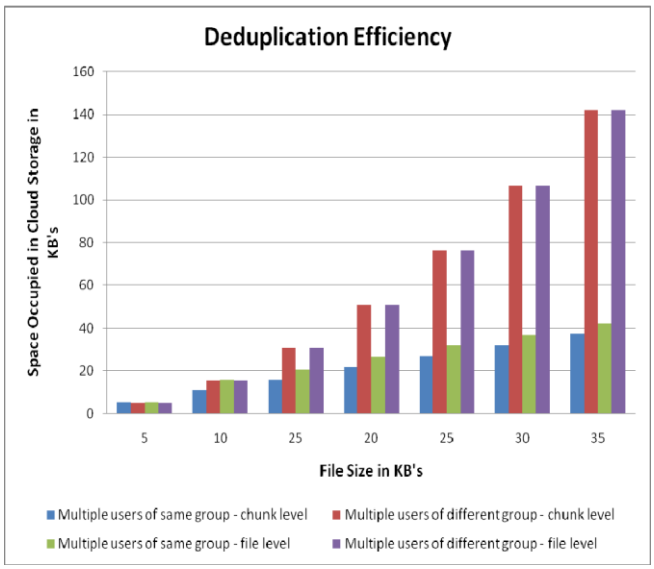

Fig. 4. Deduplication Efficiency

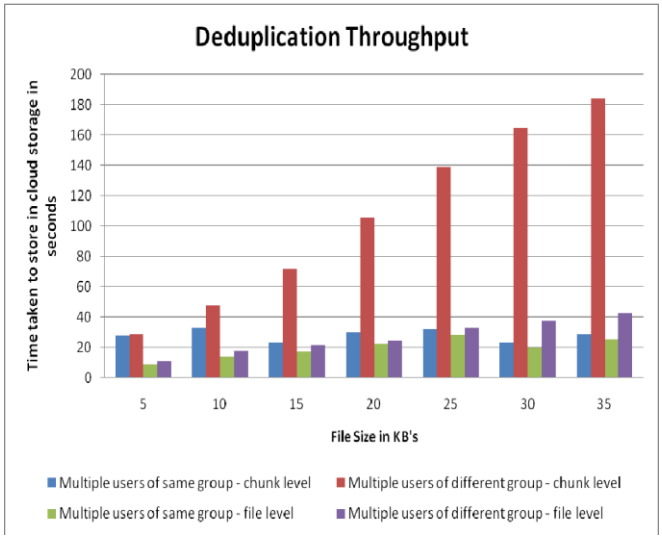

Fig.5. Deduplication Throughput

1. Data stream of multiple users belonging to the same group takes less time compared to the one from different group when they are stored in the OCS with chunk level technique. This is due to the fact that, there may be possibilities of more duplicates among the data stream of the same group. Hence, they need not be saved in the storage.
2. Storage of data stream of multiple users belonging to different group takes more time in OCS with chunk level deduplication compared to its file level counterpart.
3. Data stream of multiple users belonging to the same group takes less time in OCS with file level deduplication compared to the chunk level counterpart. The reason for this is, that chunk index

entries compared against duplicates will be less in file level compared to the chunk level deduplication.

It can be seen from the above case study that there are two groups of data stream that can enter into the cloud storage. When the data stream is identified based on the groups, throughput can be further increased. The data presented in Figure 5 shows that, throughput achieved through file level deduplication is better compared to that of chunk level deduplication. However, file level deduplication fails to group the incoming data stream. Hence, the primary index of the file level deduplication technique can be varied to achieve further increased throughput.

5. VARIATION IN FILE LEVEL DEDUPLICATION

Users who want to access the common storage are identified by a unique user-id. Every file belonging to an individual user is associated with the user-id. With this kind of arrangement, it is possible to group the data stream. In file level deduplication, there will be only one primary index for all incoming files. Hence, files that belong to different groups spend more time in checking the primary index unnecessarily. Thus, the structure of primary index can be slightly varied to accommodate groups of data streams. Figure 6 shows the varied structure of primary index in the file level deduplication.

Figure 7 shows the graph that depicts deduplication throughput for multiple users of same as well as different group for file level and varied file level deduplication. From the graph, it is clear that data stream of different group takes less time in OCS with varied file level deduplication compared to OCS with file level deduplication.

6. CONCLUSION AND FUTURE WORK

With the evolution of cloud computing, compute and storage resources of commodity machines can be efficiently utilized. This allows every organization to build its own private cloud for a variety of purposes. In order to better utilize the limited storage available in a private cloud, a suitable approach for optimization has to be used. The detailed architecture of chunk and file level deduplication techniques are discussed in the present study. In addition, it presented a case study and analyzed the performance of optimized cloud storage. Further, this paper proposed a novel variation in the file level deduplication technique and showed that this achieves better throughput. Currently, optimized cloud storage has been tested only for text files. In future, it can be further extended to support files of other types.

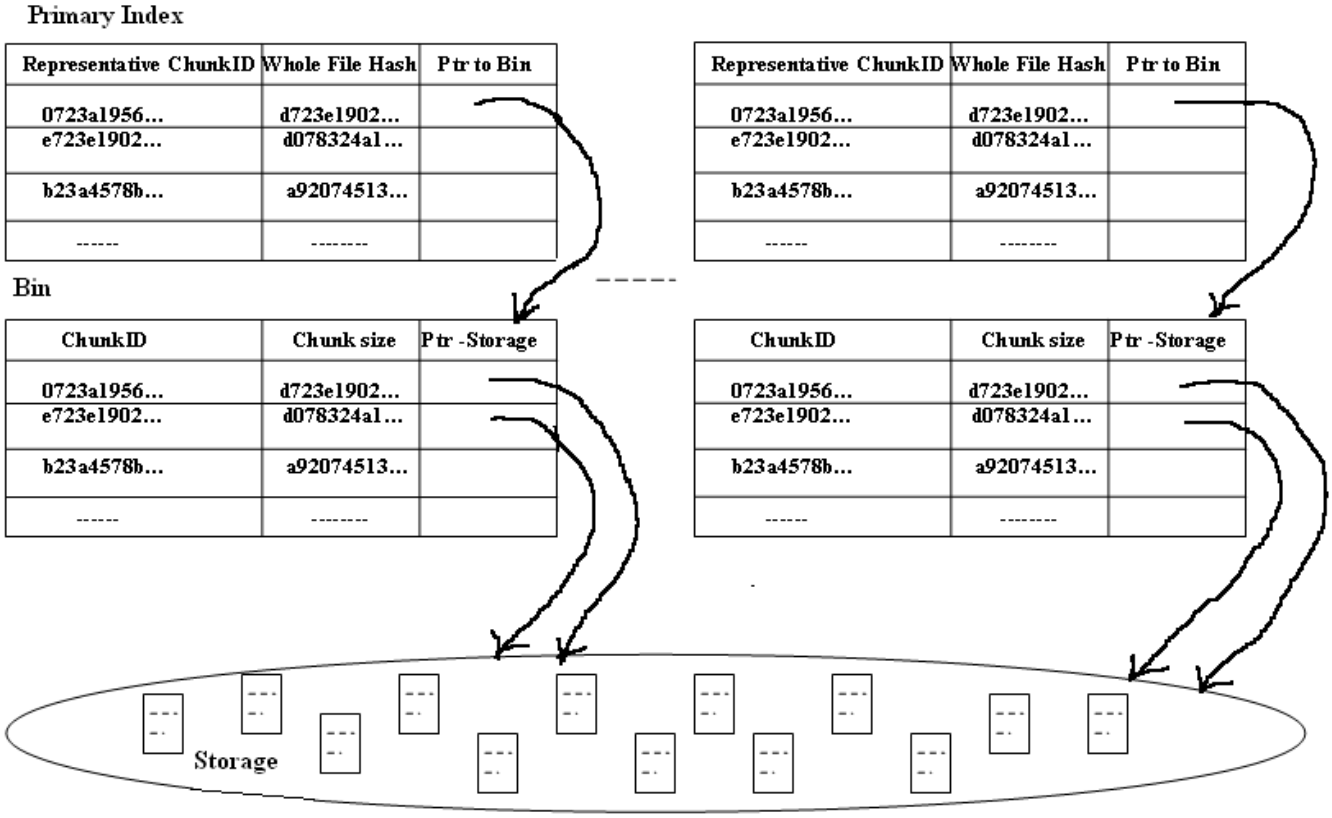

Fig. 6. Varied structure of primary index

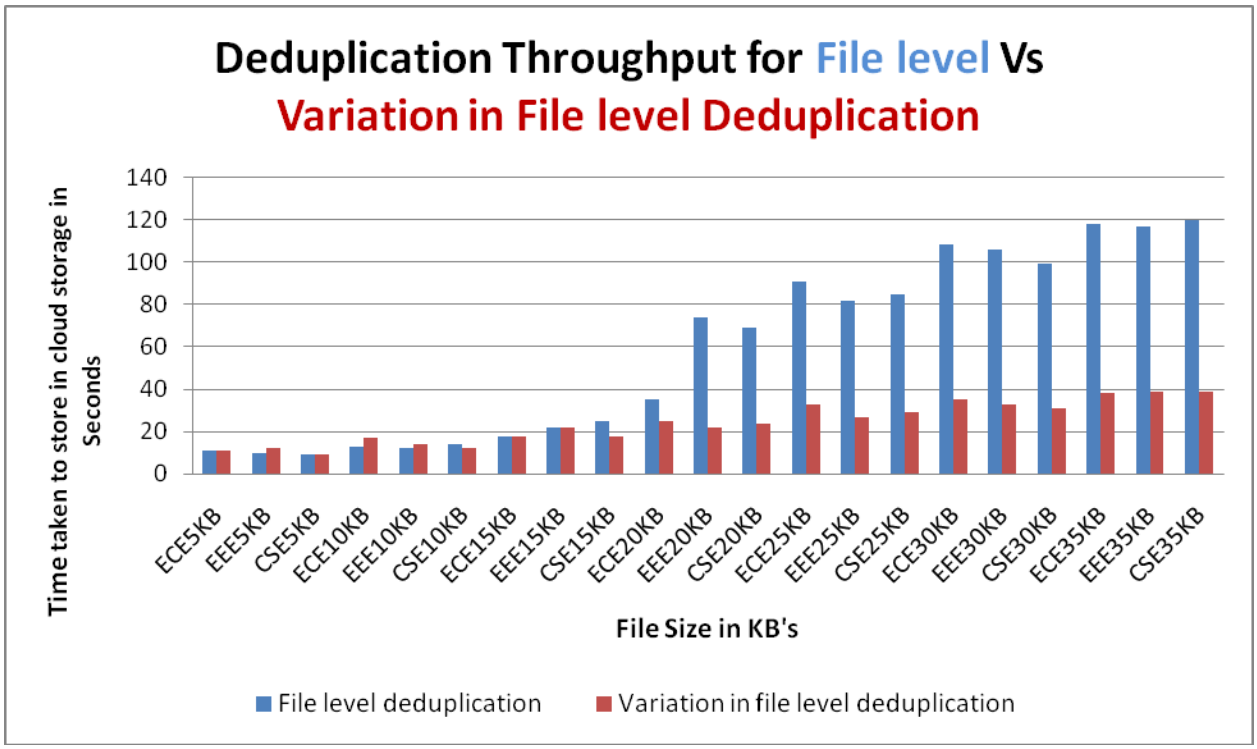

Fig.7 Deduplication Throughput for File level Vs Variation in File level Deduplication

7. REFERENCES

[1] Wei, J. Jiang, H. Zhou, K. and Feng, D. 2010. Mad2: A scalable High-throughput exact deduplication approach for network backup services, Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on, 0:1–14.

[2] Abe, Y. and Gibson, G. 2010 pwalrus: Towards better integration of parallel file systems into cloud storage. In Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), IEEE International Conference, pp. 1 –7.

[3] Bhagwat, D., Eshghi, K., Long, D. D. E., and Lillibridge, M. 2009. Extreme binning: Scalable, parallel deduplication for chunk-based file backup.

[4] Zhu, B., Li, K., and Patterson, H. 2008. Avoiding the disk bottleneck in the data domain deduplication file system. In Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST'08, Berkeley, CA, USA. USENIX Association, pp. 18:1–18:14,

[5] Stallings, W. 2002 Cryptography and Network Security, Principles and Practice, Pearson Education, 3rd edition.

[6] Policroniades, C. and Pratt, I. 2004. Alternatives for detecting redundancy in storage systems data, ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference, pp. 1-15.

[7] Zeng W, Zhao Y, Ou K and Song W, 2009, Research on cloud storage architecture and key technologies, ICIS '09: Proceedings of the second International Conference on Interaction Sciences, pp.1044-1048.

[8] Open Source Eucalyptus manual. Eucalyptus Installation [Online]Available: http://open.eucalyptus.com/wiki/EucalyptusInstall_v2.0.

[9] Amazon's Elastic Compute Cloud. Elastic Compute Cloud, [Online] Available : http://aws.amazon.com/ec2/.

[10] Open Source Eucalyptus manual. Interacting with Walrus, [Online] Available :http://open.eucalyptus.com/wiki/EucalyptusWalrusInteracting_v2.0.

[11] Amazon's Simple Storage Service. Simple Storage Service, [Online] Available : http://aws.amazon.com/s3/.

[12] Amazon's Elastic Block Storage. Elastic Block Storage, [Online] Available : http://aws.amazon.com/ebs/.

[13] Open Source Eucalyptus manual. Third Party tool to interact with walrus, [Online] Available :http://open.eucalyptus.com/wiki/EucalyptusWalrusS3Curl_v2.0.

[14] Gluster file system. http://www.gluster.org.

[15] Gluster file system documentation. http://gluster.com/community/documentation/index.php/MainPa ge