**BFCS: DISTRIBUTED STORAGE SERVICE BASED ON SECURE STORAGE**

**Abstract:** Cloud-based storage services have become an emerging trend in big data storage field. Presently there are several problems when designing an efficient storage engine for cloud-based systems with some necessities such as big-file processing, lightweight meta-data, low latency, parallel I/O, de-duplication, high scalability. Key value stores have a vital role and have many advantages when solving those problems. This paper presents about A Scalable Distributed Big File Cloud Storage to handle most of problems in big file cloud storage. It has done by proposing less-complicated, fixed metadata design, which allows fast as well as highly-concurrent, distributed file Input/Output, and simple file and data de-duplication method for static data. The results can be used for building scalable distributed data cloud storage that support big-file with size up to several terabytes.

**Index Terms:** Cloud Storage, Key-Value, NoSQL, Big File, Distributed Storage

**I. INTRODUCTION**

Now-a-days cloud storage system is being used for storing the data in gigabytes and terabytes. Cloud storage is used for the daily use, for backing-up data, sharing file to their colleagues, on the social networking sites. The user of the cloud based system can upload the data on the system and can share it with others and make it available for them and later can download it. The load over the system is very heavy. Hence, to ensure a good quality of service cloud users, the system has to look over various requirement and difficult problems: serving services to the user with high quality without any bottleneck; efficiently storing, retrieving and managing the big data files; resemble and parallel download and upload of data; the deduplication to be taken care of for managing the storage capacity of the system. Traditional file-systems had to face many challenges for service builder when managing a huge number of big-file: How to scale system; How to do distribution of data on a large number of nodes; How to do replication data for load-balancing and fault-tolerance. The solution for these problems is Distributed File Systems and Cloud Storages using commonly is splitting big file to multiple smaller chunks, storing them on disks or distributed nodes and then managing them using a meta-data system. Storing of the chunks and meta-data related to it efficiently and designing a lightweight meta-data related to it are significant problems that cloud storage providers have to face.

**Key value** stores have various advantages for storing data in data-intensive operation. In recent years, key value stores have a very unprecedented growth in every field. They have low latency with less response time and high scalability with small and medium key value pair size. Current key value stores are not designed for directly storing big-values, or big file in our case. We executed several experiments in which we put whole file-data to key value store, the system did not have good performance as usual for many reasons: firstly, the latency of put/get operation for big-values is high, thus it affects other parallel operations of key value store service and multiple concurrent accesses to different value. And, when the value is big, then there is no space to cache objects in memory for fast access. Finally, it is difficult to scale-out system when number of users and data increase. This research is implemented to solve those problems when storing big-values or big-file using key value stores. It has and gets many advantages of key value store in data management to research called cloud-storage system called Big File Cloud Storage (BFCS).

**II. EXISTING AND PROPOSED SYSTEMS**

**2.1. Existing System**

Cloud storage systems are being used for storing the data in gigabytes and terabytes. People use cloud storage for the daily demands. Users upload data from many different types of devices and they can download or share them to others. System load in cloud storage is really heavy. Thus the system has to face many difficult problems and requirements-

i. Storing, retrieving and managing big-files in the system efficiently.

ii. Parallel and resemble uploading and downloading.

iii. Data de-duplication to reduce the waste of storage space caused by storing the same static data from different users.

**2.2. Proposed System**

A common method for solving the problems which is used in many Distributed File Systems and Cloud Storages is splitting big file to multiple smaller chunks, storing them on disks or distributed nodes and then managing them using a meta-data system. Storing chunks and metadata efficiently and designing a lightweight meta-data are significant problems that cloud storage providers have to face.

**2.2.1 Advantages:**

i. Propose a light-weight meta-data design for big file. Very file has nearly the same size of meta-data.

ii. Propose a logical contiguous chunk-id of chunk collection of files. That makes it easier to distribute data and scale-out the storage system.

iii. Bring the advantages of key-value store into big-file data store which is not default supported for big value.

**III. BIG FILE CLOUD STORAGE (BFCS) ARCHITECTURE**

**A. Overview of the Architecture**

BFCS System includes four layers: Application Layer, Logical Layer, File-Chunk Store Layer and Key value store Layer. Each layer of the architecture contains several co-ordinate components. Application Layer consists of application software on desktop computers, mobile devices and web-interface that allows the user to upload, download their files. This layer uses API contained in Logical Layer and uses several algorithms for downloading and uploading process which are described in subsections II-F and II-G. Logical Layer consisted of many services and worker services, ID-Generator services and all logical API for Cloud Storage System. This layer gives the business logic part in BFCSS. The vital components of this layer are uploading and download. Logical Layer stores and retrieves data from File-Chunk Store Layer. File-Chunk Store Layer is the most important layer which has responsibility for storing and caching chunks. This layer manages information of all chunks in the system including user details and file metadata. In this, meta-data describes a file and how it is organized in chunks. File-Chunk Store Layer also contains many distributed back-end services. Two important services of File-Chunk Store Layer are File Information Service and Chunk Storage Service.

****

**Fig.1. Shows the overview of BFCSS Architecture**

File Information Service stores information of files. It is a key value store mapping data from file ID to FileInform structure. Chunk Storage Service stores data chunks which are created by splitting the original files that user uploaded. Splitting and storing a large file as number of chunks in distributed key value store bring a lot of benefits. Firstly, it is easier to store, distribute chunks in key value stores. File chunks can be stored efficiently in a key value store. It is difficult to do this with a large file directly in local file system.

**B. File Description**

File consists of one or more chunks with fixed-size. Each chunk has a unique integer Identity, and all of chunk generated from a file have a contiguous range of chunk-id. This is a different point to many other Cloud Service such as Drop Box which uses SHA-2 of chunk as ID.

**C. Storage of the Chunks**

The basic element in the defined cloud storage system is chunk. A chunk is generated from a file. When the user uploads a file, it will be split into a number of chunks. All chunks which are generated from a file except the last chunk have the same size (the last chunk of a file may have an equal or smaller size). After that, the ID generator will generate id for the first chunk with auto-increment mechanism. Next chunk that follows in the chunks set is to be assigned with an ID and then gradually increase till the final chunk. A FileInform object is created with information such as file-id, size of file, id of first chunk, number of chunks and will be stored to the database and the chunks will be stored in key value store as a record with key as id of chunk and value is data of chunk. Chunk storage is one of the most significance of defines cloud storage. By using chunks to represent a file, we can easily build a distributed file storage system service with replication, load balancing, fault-tolerant and supporting recovery.

**D. Metadata**

Typically, in the cloud storage system such as Drop box, the size of meta-data will respectively increase with the size of original file, it contains a list of elements, and each element contains information such as chunk size, hash value of chunk. Length of the list is equal to the number of chunk from file. So it becomes complicated when the file size is big. BFCS proposed a solution in which the size of meta-data is independent of number of chunks with any size of file, both a very small file and a huge file. The solution just stores the id of first chunk, and the number of chunks which is generated by original file. Because the id of chunk is increasingly assigned from the first chunk, we can easily calculate the ith chunk id by the formula: Chunk\_id[i] = file Inform. start Chunk\_id + i Meta-data is mainly described in FileInform structure consist of following fields:

* File\_Name - the name of file;
* file\_id: - unique identification of file in the whole system;
* sha: - hash value by using SHA algorithm of file data;
* reference\_file:- id of file that have previous existed in System and have the same sha256 - we treat these files as one, reference\_file is valid if it is greater than zero;
* start\_Chunkid: - the identification of the first chunk of file, the next chunk will have id as start\_Chunkid +1 and so on;
* num\_Chunk:- the number of chunks of the file;
* file\_Size :- size of file in bytes;
* file\_status:- the status of file, it has one in four values namely

Uploading File - when chunk are uploading to server;

Completed File - when all chunks are uploaded to server but it is not check as consistent;

Corrupted File - when all chunks are uploaded to server but it is not consistent after checking;

Good Completed - when all chunk are uploaded to server and consistent checking completed with good result. By using this solution, we can create a lightweight meta-data design when building the defined cloud storage.

**E. Uploading and Deduplication Mechanism**

Fig. 2 describes an algorithm for uploading big file to BFCS. Data deduplication can be defined in the cloud storage BFCS. There are many types and methods of data deduplication which can work both on client-side or server-side. We use a simple method with SHA2 hash function to detect duplicate files in the system during the uploading of file. The upload service on BFCS cloud storage system has a little different between mobile client and web interface. The client computes the SHA hash value of data content of this file P. After that, the client creates a metadata of file including file name, file size, SHA value. This information will be sent to server. At server-side, if data deduplication is enabled, SHA value will be used to

****

**Fig.2. Uploading Mechanism**

see associated file\_id, if there is a file\_id in the system with the SHA-value we call it Q, this means that file P and file Q are the same. So we simply refer file P to file Q by assigning the id of file B to reference\_file property of file P - a property that describes that a file is referenced to another file, thus the upload flow complete, there is no more wasteful upload of file. In the case there is no file ID associated with SHA-value of file P or data deduplication is disabled, the system will create some of new properties for the file information including the id of file, the id of first chunk using id\_Generator and number of chunk calculated by file size and chunk size. This process can be done in parallel to maximize speed of operation. Every chunk will be stored in the BFCS storage system as a key value pair.

**F. Downloading Mechanism**

Fig.3 describes an algorithm for uploading big file to BFCS. Firstly, the client selects the id of file that will be downloaded to the server. If FileInform of the file\_id exists, this information will be sent back to the client. The client uses the FileInform information to schedule the download process. Every downloaded chunk will be saving directly to its position in this file. When all chunks are fully downloaded successful, the download process is completed



**Fig.3. Downloading Mechanism**

**IV. COMPARISON WITH OTHER PERSONAL CLOUD STORAGES**

In a paper of Idilio Drago et al [8], many personal cloud storages were benchmarked in a black-box evaluation method. The test cases in [8] used files with size: 10kB, 100kB, 1MB to compare Drop box, Sky Drive, Cloud Drive, Google Drive and Wuala. In this research, we deployed an instance of BFC system in Amazon EC2 to compare with Drop box which uses Amazon EC2 and Amazon S3. Clients of both BFC and Drop box run from Viet Nam. According to paper [9] about some aspects inside Drop box, we compared BFC’s metadata with Drop box. Then, we did experiments for comparing deduplication ability of BFC and other cloud storages such as Google Drive, Drop box, One Drive.

**A. Metadata comparison**

Drop box [9] is a cloud-based storage system that allows users to store documents, photos, videos and other files. Drop box is available on Web interface, and many types of client software’s on desktop and mobile operating systems. The client supports synchronization and sharing between devices with personal storage. Drop box were primarily written in Python. The native client uses third parties libraries such as wxWid gets, Cocoa, libr sync. The basic object in the Drop box system is a chunk of 4MB data. If a file is larger than this configured size, it will be split in several of basic objects. Each basic object is



**Fig.4. Metadata comparison of BFC and Drop Box**

**TABLE I**

**DEDUPLICATION COMPARISON**



an independent element, which is identified by a SHA256 value and stored in Amazon Simple Storage Service (S3). Metadata of each file in Drop box contains a list of SHA256 of its chunks [2], [9]. Therefore, its size is linear to the size of file. For big file, it has a big metadata caused by many of chunks and a long list of SHA256 values from them. In our research BFC has a fixed-size metadata of each file, so it is easier to store and scale storage system for big file. It reduces the amount of data for exchanging metadata between clients and servers. The comparison is shown in Fig 4.

**B. Deduplication**

This comparison was done to study the deduplication ability of BFC and other cloud storages: Drop box, One Drive and Google Drive. We used Wire Shark [7] to capture network flow of cloud storage client application. To estimate the deduplication ability, we did following test cases: (1) A file is multiply uploaded to different folders by a User; (2) A file is multiply uploaded by different users. The result in Table I showed that Drop box supports deduplication per user accounts; it could be done in client applications. BFC support a global deduplication mechanism, it saves the network traffic and internal storage space when many users store the same file content. Google Drive and One Drive do not support deduplication.

**V. CONCLUSION**

BFCS, a simple meta-data to create a high performance Cloud Storage based on MYSQL key value store. Every file in the system has a same size of meta-data regardless of file-size. Every big-file stored in BFCSS is split into multiple fixed-size chunks (May except the last chunk of file). The chunks of a file have a contiguous ID range, thus it is easy to distribute data and scale-out storage system, especially when using MYSQL. This research also brings the advantages of key value store into big-file data store which is not default supported for big-value. The data deduplication method of BFCSS uses SHA-2 hash function and a key value store to fast detect data-duplication on server-side. It is useful to save storage space and network bandwidth when many users upload the same static data.

**VI. REFERENCES**

[1] Thanh Trung Nguyen, Tin Khac Vu, Minh Hieu Nguyen, Le Quy Don Technical University, Ha Noi, Viet Nam,VNG Research, Viet Nam, “BFC: High-Performance Distributed Big-File Cloud Storage Based On Key-Value Store”, IEEE SNPD, June 1-3 2015.

[2] Drop box tech blog. https://tech.dropbox.com/. Accessed October 28, 2014.

[3] Zing me. http://me.zing.vn. Accessed October 28, 2014.

[4] Face book. http://facebook.com, 2014.

[5] D. Borthakur. Hdfs architecture guide. HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf, 2008.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2):4, 2008.

[7] L. Chappell and G. Combs. Wires hark network analysis: the official Wire shark certified network analyst study guide. Protocol Analysis Institute, Chappell University, 2010.

[8] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Benchmarking personal cloud storage. In Proceedings of the 2013 conference on Internet measurement conference, pages 205–212. ACM, 2013.

[9] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras. Inside drop box: understanding personal cloud storage services. In Proceedings of the 2012 ACM conference on Internet measurement conference, pages 481–494. ACM, 2012.

[10] P. FIPS. 197: the official aes standard. Figure2: Working scheme with four LFSRs and their IV generation LFSR1 LFSR, 2, 2001.

[11] S. Ghemawat and J. Dean. Leveldb is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values. https://github.com/google/leveldb. Accessed November 2, 2014.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In ACM SIGOPS Operating Systems Review, volume 37, pages 29–43. ACM, 2003.