International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Big File Cloud Storage (BFCS): Distributed Storage Service with De-Duplication and Secure Storage

Sukruti Gajare[1], R. A. Khan[2]

[1, 2]Department of Computer Engineering, MES College of Engineering, Pune, University of Pune, Maharashtra, India

Abstract: ***Nowadays, cloud-based storage are growing and has become an emerging trend in big data storage field. Many problems arise while designing an efficient and low complicated storage engine for cloud-based systems with some issues like big files processing, metadata, latency, parallel Input/Output, de-duplication, distributed nature, high scalability. Key value stores has a vital role and showed many advantages when solving those problems. This paper presents about Big File Cloud Storage(BFCS) with its modules and architecture to handle most of problems in a big file cloud storage which is based on key value store. Here we are 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. This method can be used to build a distributed storage system that can accommodate data whose size is upto terabytes***.

Keywords: Cloud Storage System, Key value, Big File, Distributed Storage System

1. Introduction

Now-a-days cloud storage system are 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; resumable 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 unpre-cedented 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).

2. 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-ordinated 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 upload 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 FileInformationService and Chunk Storage Service.

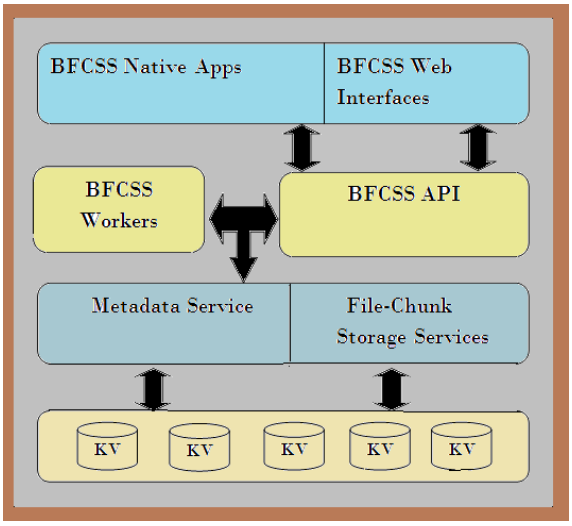

Figure 1: Shows the overview of BFCSS Architecture

File Information Service stores information of files. It is a key value store mapping data from fileID 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 DropBox[12] which uses SHA-2[16] 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 Dropbox [12], the size of meta-data will respectively increase with the size of original file, it contains a list of elements, 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 or 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] = fileInform.startChunk_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

UploadingF ile - when chunk are uploading to server;
CompletedFile - when all chunk are uploaded to server but it is not check as consistent;
CorruptedFile - when all chunk are uploaded to server but it is not consistent after checking;
GoodCompleted - 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

Figure 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 [3] 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

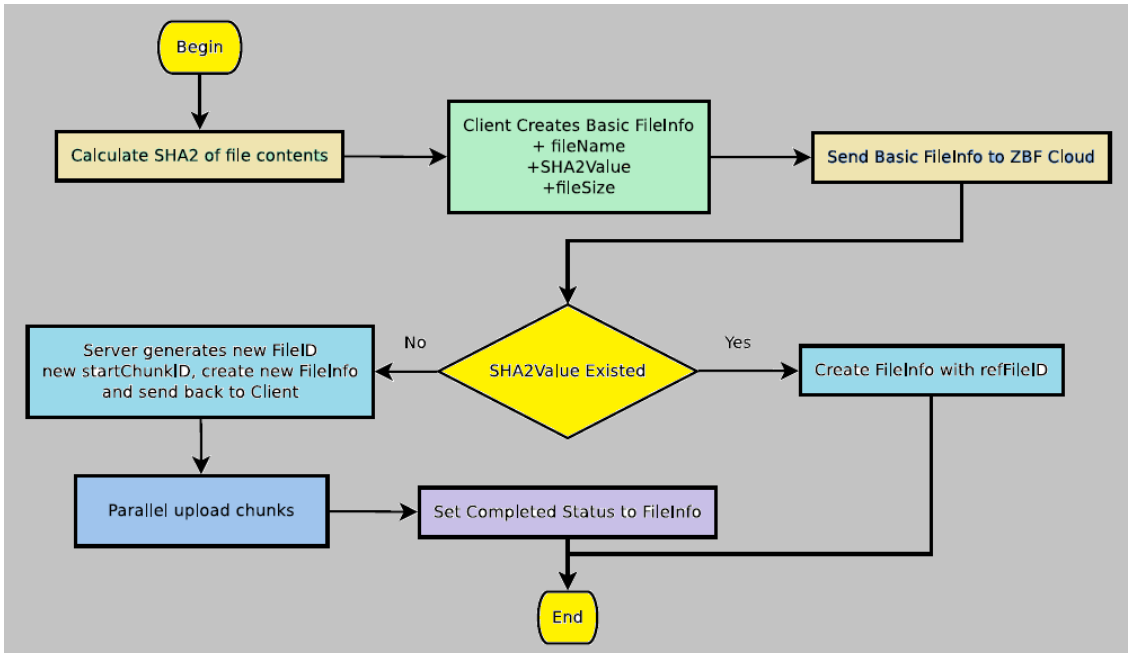

Figure 2: Uploading Mechanism

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 fileID 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

Figure 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 save directly to its position in this file. When all chunks are fully downloaded successful, the download process is completed

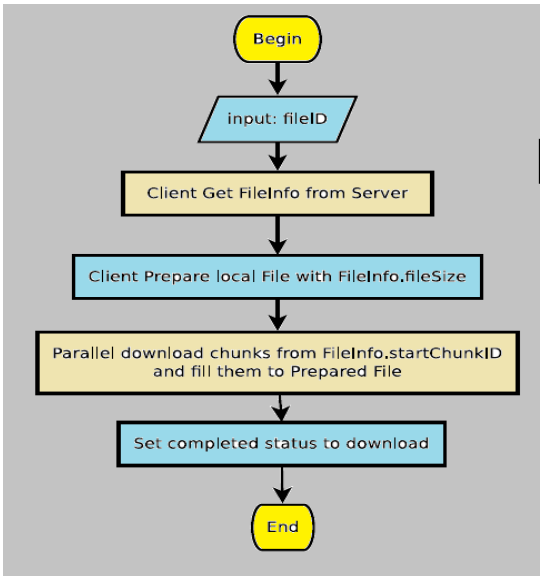

Figure 3: Downloading Mechanism

3. 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.

References

[1] Thanh Trung Nguyen, Tin Khac Vu, Minh Hieu Nguyen, Ha Noi, Viet Nam, "**BFCSS: High-Performance Distributed Big-File Cloud Storage Based On Key value Store**", June 1-3 2015, IEEE SNPD 2015, 978-1-4799-8676-7/15(Base Paper).

[2] T.T.Nguyen and M.H.Nguyen , "Design Sequential Chunk identity with Light weight Metadata for Big File Cloud Storage", IJCSNS International Journal of Computer Science and Network Security, VOL.15 No.9, September 2015.

[3] Jin Li, Xiaofeng Chen, Xinyi Huang, Shaohua Tang and Yang Xiang, Mohammad Mehedi Hassan, Abdulhameed Alelaiwi, "**Secure Distributed Deduplication Systems with Improved Reliability**", 2015, 10.1109/TC.2015.2401017, IEEE Transactions on Computers.

[4] Thanh Trung Nguyen · Minh Hieu Nguyen, "**Zing Database: High-Performance Key value Store For Large-Scale Storage Service**", 17 August 2014, Springer - Vietnam J Comput Sci (2015), DOI 10.1007/s40595-014-0027-4.

[5] Joshi Vinay Kumar, V Ravi Shankar, "**De-duplication and Encryption in Cloud Storage",** May 2015, International Journal of Innovative Research in Science, Engineering and Technology, Vol. 4, Special Issue 6.

[6] Leeba Varghese , Suranya G, "**Test Pattern Generation Using LFSR With Reseeding Scheme For BIST Designs",** December 2014, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 3, Special Issue 5.

[7] Christian Forfang, **"Evaluation of High Performance Key value Stores",** June 2014, Norwegian University of Science and Technology.

[8] Nesrine Kaaniche, Maryline Laurent, **"A Secure Client Side Deduplication Scheme in Cloud Storage Environments"**, 2013, Institut Mines-Telecom, Telecom SudParis, UMR CNRS 5157.

[9] Idilio Drago, Enrico Bocchi, Marco Mellia, Herman Slatman, Aiko Pras, **"Benchmarking Personal Cloud Storage",** October 23–25, 2013, ACM, 978-1-4503-1953-9/13/10.

[10] Mihir Bellare, Sriram Keelveedhi, Thomas Ristenpart, **"DupLESS: Server-Aided Encryption for Deduplicated Storage",** 2013, USENIX Security Symposium**.**

[11] Iuon-Chang Lin and Po-ChingChien, "**Data Deduplication Scheme for Cloud Storage",** 2012**,** International Journal of Computer, Consumer and Control (IJ3C), Vol. 1, No.2.

[12] Idilio Drago, Marco Mellia, Maurizio M. Munafò, Anna Sperotto, Ramin Sadre, Aiko Pras, "**Inside Dropbox: Understanding Personal Cloud Storage Services",** November 14–16, 2012, ACM, 978-1-4503-XXXX-X/12/11.

[13] Russell Sears, Raghu Ramakrishnan, "**bLSM: A General Purpose Log 0Structured Merge Tree",** May 20–24, 2012, ACM, 978-1-4503-1247-9/12/05.

[14] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, "**Bigtable: A Distributed Storage System for Structured Data",** Google, Inc.

[15] David Karger, Eric Lehma, Tom Leighton, Matthew Levine, Daniel Lewin, Rina Panigrahy **"Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web".**

[16] **"Secure Hash Standard",** Computer Systems Laboratory National Institute of Standards and Technology Gaithersburg, MD 2089, Issued April 17, 1995.

[17] Martin Placek, Rajkumar Buyya, **"A Taxonomy of Distributed Storage Systems".**

[18] Dhruba Borthakur, **"HDFS Architecture Guide",** Copyright © 2008 The Apache Software Foundation.

[19] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "**The Google File System",** October 19–22, 2003, ACM, 1-58113-757-5/03/0010.