3rd National Conference On Emerging Trends In Computer Science & Engineering (NCETCSE-2017)
INTERNATIONAL JOURNAL OF SCIENTIFIC PROGRESS AND RESEARCH (IJSPR) ISSN: 2349-4689

Distributed Big-File Cloud Storage Based On Key-Value Store

[1]NALINA H S, [2] DR. K THIPPESWAMY,
[1] M. Tech Student,[2]Guide and Head of the Department (CSE),
VTU PG Center, Mysuru

Abstract - Currently, cloud-based storage facilities are fast forming and transforming into a rising example in data storage field. There are various problems while planning a capable storage engine for cloud-based systems with a couple of requirements, for example, huge file processing, lightweight meta-data, low latency, parallel I/O, deduplication, high flexibility. Key-value stores expected a key part and demonstrated various advantages when dealing with those issues. This paper presents about Big File Cloud (BFC) with its estimations and development demonstrating to handle the immense majority of problems in a noteworthy record disseminated capacity system considering key-value store. It is done by proposing lowbewildered, changed size meta-data diagram, which sponsorships brisk and especially concurrent, scattered record I/O, a couple of estimations for resumable exchange, download and essential data deduplication for static data. This examination joined the advantages of ZDB - an in-house key value store which was progressed with auto-build entire number keys for handling large report storing issues successfully. The results can be used for building versatile flowed data conveyed capacitythat supports tremendous report with size up to a couple of terabytes.

Keywords: Cloud Storage, Key-Value, NoSQL, Big File, Distributed Storage.

I. INTRODUCTION.

Cloud storage services generally facilitate millions of people with storage capacity of gigabytes to some terabytes for each user to save their valuable data according to the importance on daily basis, for example data backup, sharing files to their friends via Facebook, Twitter etc… Client uploads their data into cloud using different devices like tablets, mobiles and PCs and then they can download it to any other different devices according to need. Usually system load will be heavy in cloud storage, thus to assure best quality of facilities for clients, organization has to overcome many problems and requirements. Providing powerful data facilities for a huge number of users without tailback. Saving, using and maintaining large files into organization effectively. Simultaneous and resumable upload and download, data deduplication to minimize the usage of storage to save the same copy of files from various users.

Key-Value stores have many benefits for saving in data-intensity services. They frequently beat traditionalrelational databases in capacity of huge load and large-scale system. Since past years key-value store has immense growth in both industries and academic fields. They have less-delay response time and best scalability with small and medium key-value pair size. Present key-value stores are not well built for saving big-values or big files. We tried with many experiments where we put complete file-data to key-value store, the result did not had a good outcome because the latency was high for I/O operation and parallel access to different value is limited. When the worth is high there will be no space to cache other objects for faster access from main memory. Eventually it is tough to build system when users and data increase. This research is executed to solve these problems while saving large values using key-value store.

It provides several benefits of key-value store in data management to build cloud-storage system called BFC(Big File Cloud Storage).

II. SYSTEM MODEL

A. Architecture

BFC system contains four layers: Application Layer, Storage Logic Layer, Object store Layer, Persistent Layer.

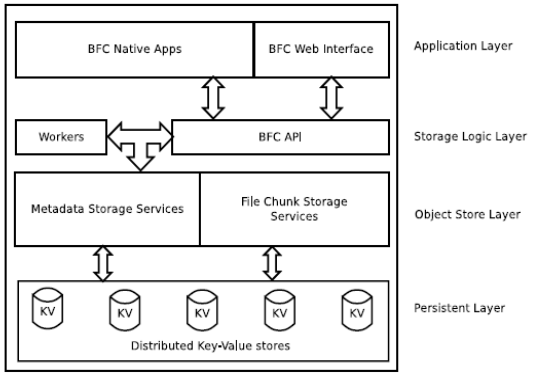

Fig.1 BFC Architecture

Application Layer Contains the application codes and navigation interface on desktop, mobile device and tablets which allows users upload into and download data from cloud. This uses the API from Logical layer and many algorithms for uploading and downloading process.

Storage Logical layer contains queuing service and worker services, ID generator services and all logical API for cloud storage system. It implements business logic part in BFC. The important component of this layer is upload and download service and CloudAppsService which resolves all client request. When number of clients reaches limit we can implement CloudAppsService into different servers to overcome. Clients do not request CloudAppServices directly but through the dispatcher where dispatcher provides public APIs for clients. The dispatcher checks for the concurrent connections from clients where it can block from the same user. This layer fetches data from Object Store Layer which is responsible for saving and caching objects. It maintains information of all objects including client data, file information data specially meta-data. Meta-data describes how the data is arranged as a list of small blocks/chunks. Object Store layer includes backend services like FileInfoServices includes information about files and key-value store maps data from fileID to FileInfo structure and ChunkInfoServices stores data blocks which are formed by dividing the original files that user has uploaded. Splitting and saving a large file as a list of chunks in distributed key-value store provides more avantages. Small Chunks can be saved effectively in key-value store, it is tough to do this with large file directly into local storage system, in addition provides parallel upload and download and resumable. All data in this layer are persevered to Persistent Layer which stores key-value based on ZDB.

B. Storing of Chunk

The main content in BFCSS is chunk. A chunk is a data block created by splitting up large files into smaller data blocks when user uploads a file, if the file size is larger than the provided size it will be split into group of chunks. All chunks which are generated will be of same size except the last chunk where it might be of same size or lesser. Then the generator will create id for first file and the first chunk with auto increment mechanism until the final chunk. A FileInfo object is created with information like file-id, id of the first chunk and number of chunks stored to ZDB. Chunk will be stored in key-value store as a record with *key* is chunk id and *value* is chunk data.

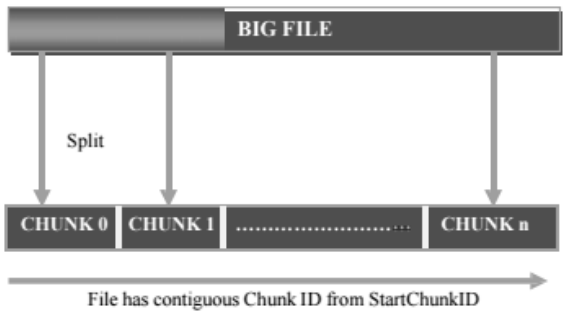

Fig 2 Splitting big files into smaller chunks

C. Metadata

Naturally, in the cloud storage such as DropBox, CEPH, the size of meta-data will gradually increases with the size of original file, contains list of elements where each holds information like chunk size, hash value of chunk. Length of list is up to the quantity of chunks ina file, it will be tangled when the file size is big. BFCS proposed an answer within which the

number of meta-data is individual of range of chunks with any size of file, both small file and large file. The answer simply stores the id of first chunk, and also the range of chunks that is generated by original file. As a result of the id of chunk is more and more allotted from the primary chunk, we are able to simply calculate the i^{th} chunk id by the formula:

Chunked[i] = fileInfo.startChunkID+i (1)

Metadata is especially narrated in FileInfo structure of below fields:

fileName – the name of file.*fileID* – 8 bytes, distinct identificationof file in the complete sysem.

Sha256 – 32 bytes, hash value using sha256 algorithm of file.

refFileID – 8 bytes – id of file that have previous existed in system and have identical sha256 , we treat these as one, *refFileID* is justifiable if it is greater than zero.

startChunkID – 8 bytes , the recognition of first chunk of file, the further chunk will have id as startChunkID+1, startChunkID+2…..

numChunk – 8 bytes, number of chunks in file.

fileSize – 8 bytes, file size in bytes.

status – enum 1 bytes, status of file, the status of file has one in four values.

EuploadingFile – chunk are uploading to server.

ECompletedFile – chunk are uploaded to server but not check as stable.

EcoruptFile – chunks are uploaded to server but not stable after checking.

EGoodCompleted – chunks are uploaded to server and stable checking completed with best result.

III. PREVIOUS WORK

Cloud storage is used by people for daily demands,for instance backing up data, sending file to friends through social networks. Users will also possibly upload data from different types of devices and they can download or share with others. In Cloud storage system load is usually very heavy. Thereby, in order to assure excellent quality of service for users, the system will have to deal with certain problems and requirements.

Disadvantages

- Efficiently storing, retrieving and managing big files in the system.
- Data duplication in order to reduce wastage of storage space which is due to storing same static data from different users.
- Parallel and resumable upload and download

IV. DATA ADMINISTRATION AND CLONING

BFC is designed based on ZDB (Zing Data Base) a distributed key-value storage system. It is clear that meta-data which is stored can be cloned for fault-tolerance and load-balancing. FileInfoService and ChunkStoreService administer data with the help of consistent-hashing, which is used to clone key-value data. Every store services have its own distributed ZDB illustration. Each has a range [$h_{lowerbound,}$ $h_{upperbound}$] used to utilize the scope of key to store. On the off chance that hash(key) is in the range, it is put away in that case. In BFC, document id and lump id are auto increase whole number keys. We can utilize straightforward hash work.

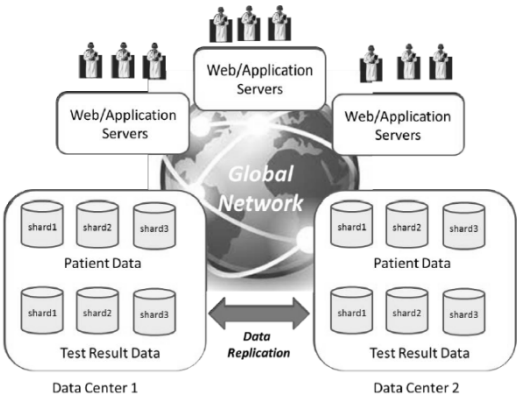

Fig 3 Data Administration and Cloning(replication)

V. UPLOADING AND DEDUPLICATION

Information deduplication is upheld in BFC. There are many sorts what's more, strategies for information deduplication which can work both on customer side or server-side. In BFC, we actualized it on server-side. We utilize a straightforward strategy with key-esteem store and SHA2 hash capacity to identify

copy records in the entirety framework in the stream of transferring. An examination between BFC furthermore, other distributed storage frameworks in deduplication is appeared in Table I. The transfer stream on BFC distributed storage framework has a little diverse between versatile customer and web interface. On portable customer, after a record to transfer is chosen, we call it A, the customer figures the SHA hash estimation of substance of this record.

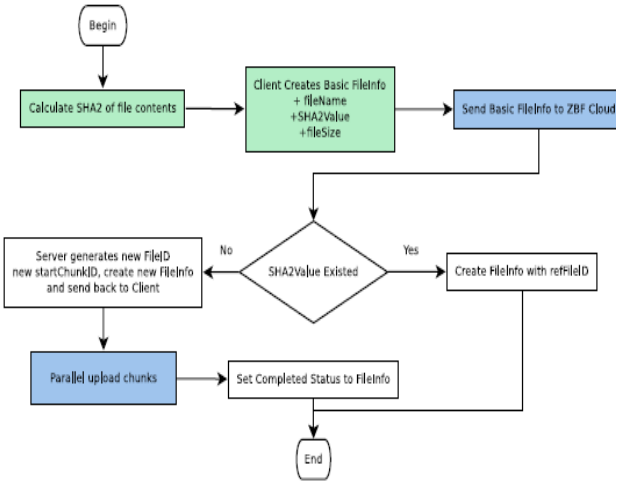

Fig 4 Uploading algorithm

After that, the customer makes a fundamental data of document including record name, document measure, SHA esteem. This essential data will be sent to server. At server-side, if information deduplication is empowered, SHA esteem will be utilized to query related fileID, if there is a fileID in the framework with the SHA-esteem we call it B, this implies record An and document B are precisely the same. So we just elude document A to record B by appointing the id of document B to refFileID property of record A - a property to portray that a record is referenced to another document. The essential data will be sent back to customer, and the transfer stream finish, there is not any more inefficient transfer. For the situation there is no fileID related with SHA-estimation of document An or information deduplication is handicapped, the framework will make some of new properties for the document data including the id of record, the id of first piece utilizing IDGenerator and number of piece figured by document measure what's more, piece size. The customer will utilize this data to transfer record substance to the server. At that point, all lumps will be transferred to the server. This procedure can be executed in parallel to amplify speed. Each lump will be put away in the capacity framework as a key-esteem match, with the key is the id of lump, and the esteem is information substance of the lump. At the point when all lump are transferred to the framework, there is a methodology to confirm transferred information such as checking the condition of SHA-esteem figured by customer also, SHA-estimation of document made by transferred lump in server. In the case of everything is great, the status of field of FileInfo is set toEGoodCompleted.

Deduplication	Dropbox	OneDrive	Google Drive	BFC
Single user	yes	no	no	yes
Multi-user	no	no	no	yes

Table 1 Deduplication comparison

VI. DOWNLOADING ALGORITHM

Versatile customers of BFC have download calculations portrayed in Fig 8. Right off the bat, the customer sends the id of document that is destined to be downloaded to the server. The dispatcher server will check the session and number of association from the customer. On the off chance that they are substantial, the dispatcher sends download demand to the CloudAppsService server, then it will query the document data in the FileInfoService which stores meta-information data with document ID as a key. On the off chance that FileInfo is existed with the asked document ID, this data will be sent back to the customer. The most essential data of the record from FileInfo structure incorporates: first id of piece (chunkIdStart), number of lump (chunkNumber), size of piece (chunkSize) and size of Record (fileSize). The customer utilizes these data to plan the download handle.From that point forward, the versatile customer downloads pieces of records from ChunkStoreService by means of CloudAppDispatcher and CloudAppService, lumps with range ID from chunkIdStart to chunkIdStart+numberChunk−1 are simultaneously down stacked in a few strings, each lump has a size of chunkSize, but last lump. Local application will pre-apportion document in neighborhood filesystem with document estimate determined in fileSize field of FileInfo. Each downloaded lump will be spare specifically to its position in this document. At the point when all pieces are completely

downloadedfruitful, the download procedure is finished.

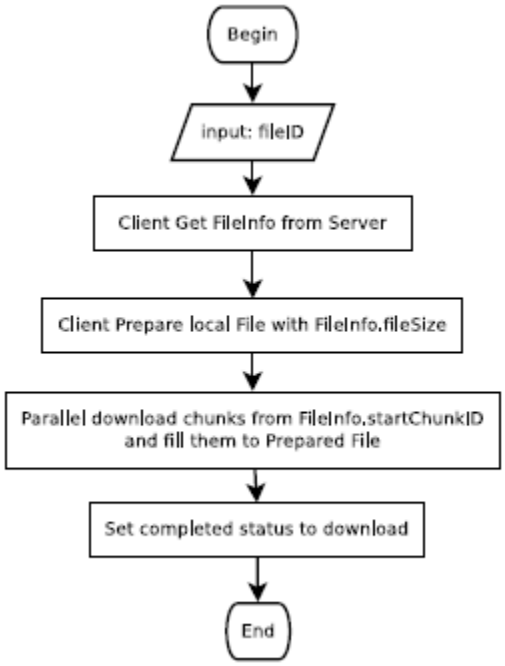

Fig 5 Downloading Algorithm

VII. PROPOSED SYSTEM

A common method which is used for solving these problems is by dividing big file to multiple smaller chunks, storing them on disks and then managing them by using a meta data system.

Cloud storage providers have to face significant problems like, storing chunks and meta-data effectively and designing a light weight meta data. After a long time, current cloud storage services have a complex meta data system; somewhat the size of metadata will be linear to the file size for every file. Thereby, the space complexity of these Meta data system is not scalable for big file. In this research, we implement a big file cloud storage architecture and also a superior solution to reduce the space complexity of meta-data. Propose a light-weight meta-data design for big file. Every file has nearly the same size of meta-data. BFC has space complexity of meta-data of a file, while size of meta-data of a file in Dropbox, HDFS has space complexity of where n is size of original file.Propose a logical contiguous chunk-id of chunk collection of files. That make it easier to distribute data and scale-out the storage system.ring the advantages of key-value store into big-file data store which is not default supported for big-value

Advantages

- A lightweight metadata design for big file. Every file has approximately the same size of metadata.
- A logical contiguous chunk-id of chunk collection of files that makes it manageable in order to distribute data and scale out the storage system.
- File compression which overcomes the problem of increased data storage and information transfer.

VIII. CONCLUSION

The proposed cloud storage system provides a simple meta-data to create a highly scalable distributed Cloud Storage based on key-value store. Every file in the system has a same size of meta-data regardless of filesize. Every big-file stored in the cloud 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. This research work also brings the advantages of key-value store into big-file data store which is not default supported for big-value. The data de-duplication method 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, Information Technology Faculty,“BFC: High-Performance Distributed Big-File Cloud Storage Based On Key-Value Store”, Le Quy Don Technical University, Ha Noi, VietNam, VNG Research, VietNam.2015.

[2] Sukruti Gajare, R. A. Khan, Department of Computer Engineering, “Big File Cloud

Storage (BFCS): Distributed Storage Service with De-Duplication and Secure Storage″MES College of Engineering, Pune, University of Pune, Maharashtra, India.2014.

[3] Ms. Priyadarshini, Mrs. Shruthi K R"A Highly Scalable Distributed Big File Cloud Storage with Data De-Duplication", Department of CSE Rajeev Institute of Technology Hassan, Karnataka, India.2016

[4] M. Vasavi, J. Narasimha Rao,BFC: save storage space in big-file cloud through key-value store and sha-2 function" ,Department of CSE, CMR Technical Campus, Kandlakoya(v), edchal(m),Ranga ReddyDistrict, Telangana State, India. 2016

[5] Dropbox tech blog. https://tech.dropbox.com/. Accessed October 28, 2014.

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

[7] 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.*