Application and Implementation of Batch File Transfer in Redis Storage

Hu Meng, Yongsheng Pan, Lang Sun

Application and Implementation of Batch File Transfer in Redis Storage

Hu Meng[1*], Yongsheng Pan[1] and Lang Sun[1]

[1] HEFEI City Cloud Data Center Co., Ltd. Hefei, China 230601
menghu@citycloud.org.cn

Abstract. In the face of massive information, batch processing of files is an important way of information transmission and storage, and the application is quite common. With the increasing demand for batch files processing reliability and speed, and the problem of low storage efficiency for current batch file processing, the paper proposes a storage method that combine distributed storage system HDFS file storage advantage and Redis cache technology to form a rapid batch merge files. The files that meet the conditions are merged into the Sequence File and stored in the HDFS. The multiple linear regression analysis method is used to determine the load factor, so that the load balancing is adjusted and the Redis cache hash data is used to ensure the efficiency. Through experiments on the corresponding file platform for file upload, query, delete and memory usage, we analysis batch processing method and non-batch method comparatively. It can be concluded that compared with the non-batch direct upload file to HDFS way, improved batch file processing method can process files more faster and ensure the stability and reliability of the file at the same time.

Keywords: Redis, HDFS, Batch Processing, Distributed File System.

1 Introduce

File system is an important way to transmit and store information. The use of the scale continues to expand, such as the office system, mail, message system through which information can be shared and distributed quickly. Users in such applications, not only requires high-speed processing speed, but also requires the reliability of storage. Therefore, massive files in cloud storage research has important practical value.

Massive file storage is generally based on HDFS. HDFS is a distributed file system, through the cheap multi-machine support large-scale data sets of large file storage, with strong scalability, and solve the storage problem of space constraints. Meantime, HDFS can provide high-throughput data access. It is ideal solution for large-scale data set applications, and even in the case of error can guarantee the reliability of data storage. It assumes that the calculation elements and storage would fail, so it maintains multiple copies of the work data to ensure that they can be redistributed against the failed nodes. It works in parallel to ensure efficient processing. But the storage efficiency of small files in HDFS is not high. It uses NameNode to maintain the mapping of file path to the data block and the mapping of the data block to

DataNode, and also monitor DataNode heartbeat and maintain the number of data block copies. When a large number of small files stored in HDFS then the NameNode will run out of most of the memory, resulting in low storage efficiency, limiting the file access speed.

Taking into account the above-mentioned problems, we use a separate server with large memory to cache the data to be merged. It would improve the performance of the management node, and avoid the main server bottlenecks. The cache server uses Redis to store data. Redis is a memory-based high-performance Key / Value database. It writes updated data to disk or writes modified operation to additional log files periodically to ensure data persistence. And the Master-Slave synchronization provides a high availability and reliable platform to users. The first upload files cache in Redis, writing to disk operation only need to execute one time after merging the files which would reduce the times of disk I / O. And file uploading processes in memory which can provide a significant reduction of response time of file uploading.

2 Algorithm Summary of batch processing

The file storage scheme designed in this paper is to build an intermediate platform between users and HDFS system to handle the upload, query and delete operations of received files. As large files can be stored directly and efficiently in HDFS, the platform only process small files. Processing of files that larger than 32M would return a processing-received tag directly. Consolidated storage scheme as shown in Figure 1. Users interact with the platform through socket. Redis is used to cache user files. Caching files merge and store in the Sequence File of HDFS through the HDFS interface. The metadata records cache in Redis.

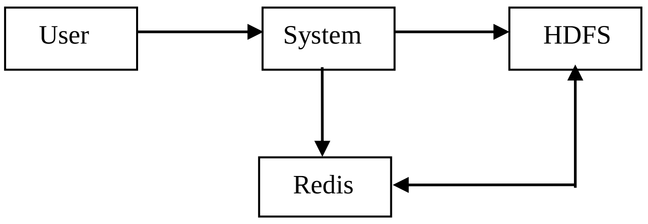

Fig. 1. Files consolidated storage solution

2.1 Storage structure

Redis as a file cache database, save the cache file content and metadata records. The cache structure design is as follows:

Table 1. Redis cache data storage structure

Name	Type	Description
RCF	Hash	Cache the data of the file, including the contents of

		the file. Key is the file name, Value is the contents of the file
RCFL	Long	The length of the file in the cache, that is, the total length of the file data stored in the RCF
MH:DID	Hash	File information that update to metadata record after serialization. For example, MH: 12 store all the metadata structure of the folder identified as 12. Key for the file name, Value for the metadata
SDIR:DID	Hash	Folder structure, Key for the folder, Value for the folder name
DID	Long	Automatic growth of the folder identification

The storage structure of HDFS platform is the Sequence File stored and combined under basic directory, named after timestamp. The Sequence File uses the sequential storage structure so that we can quickly locate the contents of the small files through the file location Store Position recorded in metadata corresponding to File Name. And, Sequence File uses Block compression to reduce disk usage and increase transmission speed. Block compression is a series of records, that is, the small files here, organized together, unified compressed into a Block. Block information mainly store: the number of records contained in the block, the set of the length of each record Value, and the set of the value of each record Value.

2.2 Storage platform implementation

According to the above design proposal, based on the load cost model, the file platform is divided into two parts: basic processing and background processing. Users process basic file operation such as upload, modify and delete through the platform interacted with Redis and Sequence File; The timer combined with the basic operation triggers the event to invoke background processing to ensure the reliability and speed of the system.

When uploading a file, the received file is stored in the file cache RCF in Redis and the RCF Length of the file (RCFL) stored in the RCF is updated. Then, to determine the RCFL, if the length achieved the size for merge, a "merge file" message MF is send to the background processing module. When reading a file, the file will be returned directly if it exists in the upload buffer of Redis. Otherwise, the contents of the file will be read by the cache processing module and returned to the client. When deleting a file, first determine whether it exists in RCF. If true then delete it. Otherwise, the metadata of delete flag will be set to 0 and mark the file would be deleted.

2.3 Load Cost Model

As a complete system, not only to improve the efficiency of file storage, but also take into account the system load conditions. The load cost of existing server resources generally evaluated by the usage of independent CPU or memory. This statistics is not comprehensive. For example, high CPU usage will not affect the operation which only occupy high memory and disk I / O usage, and in actual use, the various types of resources requirements of the thread operation cannot be comprehensively evaluated either.

In order to make up for the shortage of resource statistics, this paper puts forward a load cost model which use the user's response time as an estimate criteria, based on the experimental analysis of statistical data to determine the cost of the formula, so to evaluate the effects of variety factors more reasonable. The process is: while the system is running normally, gather statistics and analysis the various types of resources, such as CPU, memory, disk I / O and other in real-time, record the response time of task processing threads and main customer service thread, to determine various factors. In this way, to avoid the lack of timeliness of traditional statistical estimates, the use of real-time computing can ensure reliable and comprehensive analysis with all kinds of environmental resources.

This paper combines the features of Sequence access and based on the GD-SIZE algorithm, calculate the cost H with the formula (1), and archive the small file cache replacement strategy.

$$H_i = N / S \tag{1}$$

The general GD-SIZE algorithm is: Each document in the buffer has a corresponding cost. When the document is brought into the buffer, the H value of the document is the reciprocal of the document size. When the replacement occurs, document which has the smallest H value H_{min} is swapped out, and the H value of the remaining document becomes the H value before the replacement minus H_{min}. According to the characteristics of Sequence File, reading the file in a single block may need to traverse many times. The value of H that GZ-SIZE algorithm used cannot actually reflect the cost of the document. The cost of the document has positive correlation with the traversed files number N for visiting the file. We can multiply the reciprocal of the file size S by N as the initial value of the GZ-SIZE algorithm, to achieve cache replacement.

3 Experiment

To establish the load cost model and determine the load cost formula, the influence of various factors on the response time of the main thread needs to be quantified. The coefficient of influence of the factor is obtained by the method of multiple linear regression analysis. In this system, CPU usage (C), memory usage (M), and disk I/O (D) have a major impact on performance, so they are used as dependent variables and uploading response time (T) as response variable, the multiple regression equation is expressed as:

$$T = k_1C + k_2M + k_3D \tag{2}$$

The specific operation is as follows: In the running nodes of the platform, execute multiple processes that have great impact to C, M and D to get different resource occupancy results, and gather the file upload time statistics. The results are as follows:

Table 2. Response time of the user request

T/ms	C/%	M/%	D/(Blk_wrtn/s)
49	13	33	17.03
59	15	31	184.00
92	63	21	188.00
55	23	22	0.00
57	12	21	0.00
73	36	32	116.00
51	45	43	8.00
68	89	21	32.00

Regression analysis of the results can calculate that k1 = 0.257, k2 = 0.332, k3 = 0.103. In order to enable the user get responding within 100ms, the response time T calculated with C, M, and D should be less than 100ms as the expected threshold of the load. In the experimental environment, the value of k1, k2, k3 is input to the running configuration. When the background message MF is received, the load threshold is calculated by the formula.

In order to eliminate the impact of unstable factors (such as speed), randomly selected 10 small files in the standard HDFS and the use of optimization modules in the file system to upload, and ultimately get the cost time shown below:

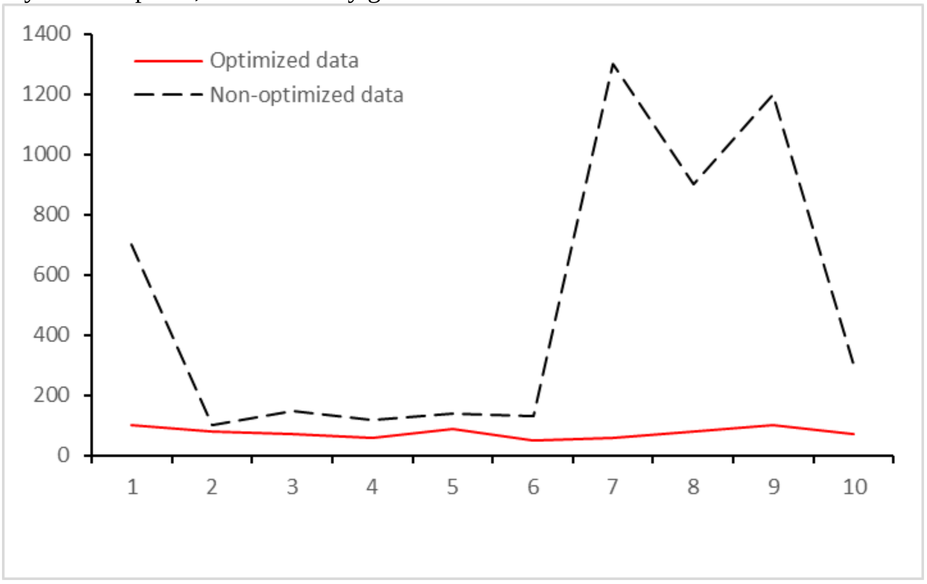

Fig. 2. File upload time

It can be seen that the time for file uploading is significantly reduced by batch merging of files, which is reduced from an average of 453.1 ms with traditional way to an average of 52.3 ms by 88.45%. In the file uploading in a batch file, the implementation changes from receiving files through original HDFS memory and writing the disk later to receiving files by Redis memory directly, no longer need to wait for the slow disk I / O operation of writing. The upload speed is significantly improved.

4 Conclusion

In this paper, we consider the storage method of small files in HDFS, design and implement the small files storage optimization based on reliable HDFS file system. Combining the Redis cache mechanism effectively reduces the memory usage of the NameNode node, the disk I / O is reduced compared to the traditional HDFS files merge method, speeding up the file uploading and acquiring speed in a large number of frequent file reads. It can be seen from the results of the experiment that the Redis-based HDFS file batch merge storage optimization method can improve memory utilization and speed up the file retrieval speed, and not affect the speed of file updating and querying, ensure the fast and reliable file operation and preservation.

References

1. Codd E F. A relational model of data for large shared data banks [J]. Communications of the ACM, 1970, 13(6): 377-387.
2. Michael Stonebraker. SQL databases v. NoSQL databases [J]. Communications of the ACM, 2010, 53(4): 10-11.
3. Karger D,Lehman E,Leighton T,et al.Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web[C]//ACM Symposium on Theory of Computing. New York, NY, USA: ACM, 1997:654-663.
4. Kubiatowicz J, Bindel D, Chen Y, et al. OceanStore: An architecture for global-scale persistent storage[C]// Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (2000). Boston, MA: ASPLOS, 2000:190-201.
5. PESTER M. Multidisciplinary conceptual aircraft design using CEASIOM [D]. Hamburg： Hochschule für Angewandte Wissenschaften Hamburg，2010.
6. Heath C, Gray J. OpenMDAO: Framework for Flexible Multidisciplinary Design, Analysis and Optimization Methods[C]// Aiaa/asme/asce/ahs/asc Structures, Structural Dynamics and Materials Conference, Aiaa/asme/ahs Adaptive Structures Conference, Aiaa. 2012.