Research on General File Storage Scheme Based on Hadoop

Huimin Liu[1,a,*] and Huijie Liu[2,b]

[1]Beijing University of Technology, Beijing, China
[2]China Railway Taiyuan Group Co.,Ltd., Shanxi Taiyuan, China
a. 18935399040@163.com, b. 1136592603@qq.com
**corresponding author: Huimin Liu*

Keywords: Merging algorithm, correlation, storage of common files, cache replacement strategy.

Abstract: HDFS is Hadoop's underlying distributed file storage system, designed for large files. When accessing a large number of small files, the problem of low read and write rate and excessive memory load will be encountered. Therefore, HDFS is optimized for the problem of different size files. Firstly, the historical access log is analyzed, the relevance between files is acquired by using Apriori algorithm, the correlation probability model is established, and then a directed graph merging algorithm based on correlation is proposed. In order to solve the problem of low access rate caused by small file merging, prefetching strategy and LRU substitution strategy based on high heat were introduced. Experimental results show that this scheme can effectively reduce the metadata volume of NameNode, improve the memory utilization, and effectively improve the file read and write performance.

1. Introduction

With the development of information technology and the popularization of network, the data closely related to our life is growing rapidly in the form of explosive index. The era of big data has come. Cloud computing has been integrated into government public utilities, e-commerce, education, entertainment industry, medical and health care and other fields, and is affecting people's clothing, food, housing and transportation. According to the authoritative release of the 44th statistical report on the development of China's Internet published by CNNIC [1], by the end of June 2019, the number of Internet users in China had reached 854 million, and the Internet penetration rate had exceeded 60%. Take Headline and Tik Tok for example, according to the data announced by the company (byte dance) ,as of July 2019, the global total DAU (daily active users) of byte-dance's products exceeded 700 million, and the total MAU (monthly active users) exceeded 1.5 billion, of which Tik Tok DAU exceeded 320 million. As the most frequently used taxi platform in China, Didi processes more than 106 TB of track data and 4.78 PB of comprehensive data every day. People enjoy the achievements and convenience of the scientific and technological progress, the number of small files, such as pictures, files and videos, is growing exponentially. According to the latest white paper released by IDC (International Data Company) China's data circle will increase to

48.6 ZB by 2025, accounting for 27.8% of the world's total, becoming the largest data circle, in which a variety of small files are everywhere.

The Hadoop file system HDFS released by Apache Software Foundation is a parallel cluster and distributed file system, which can quickly process large data sets. At present, it has been researched and applied by well-known companies at home and abroad, such as Huawei, Tencent, Baidu, Alibaba, Didi, Facebook, China Mobile, etc. [2]. HDFS is a Java open source implementation of Google GFS, which provides the underlying support for distributed computing. It has the advantage of ignoring the hardware errors of machines, and can also migrate and upgrade clusters at a low cost. It is an ideal file storage system, but not a perfect file storage system. HDFS is designed to store and process large files conveniently. It adopts the blocking strategy to make large files transfer and store quickly among multiple machine clusters, so that each file occupies one or more data blocks [3]. But with the emergence of electronic social networking, entertainment new media, scientific research and computing, a large number of small files are generated. The volume of photos, audio, and files is usually between 30K and 8M, far smaller than the size of the data block in HDFS. Therefore, for massive small files, the problem of HDFS can be summarized into three points: (1) the memory pressure of NameNode is large. In the cluster, the metadata of each small file will occupy a NameNode. With the increase of the number of small files, the load of NameNode gradually increases and the utilization rate of internal storage space is low. (2) The efficiency of reading and writing files is low. In data interaction, frequent interaction between requester and NameNode increases the load of NameNode and reduces the efficiency of file reading and writing. (3) The efficiency of file retrieval is low. HDFS itself lacks prefetching mechanism, while small file storage presents a decentralized state. Although Hadoop introduces MapReduce, it does not significantly improve the read rate of HDFS.Be advised that papers in a technically unsuitable form will be returned for retyping. After returned the manuscript must be appropriately modified.

Therefore, how to build a general file storage system to improve system performance and user experience has been a hot issue in IT industry for a long time. The following paper takes a large number of small files as the research object, briefly expounds the research results at home and abroad, introduces in detail the improvement scheme of establishing a general file storage system, and builds a simulation test platform for comparative experiments, and analyzes and summarizes the experimental results.

2. Related Work

In view of the impact of HDFS on the working efficiency of small files, there are some research achievements in this aspect at home and abroad, which are mainly divided into two strategies: one is the general small file storage strategy, and the other is the small file HDFS improvement scheme for specific scenarios.

Hadoop has three self-research strategies for small files [4]: Hadoop Archives (HAR), Sequence File, and Map File. HAR effectively stores small files into HDFS by means of filing, packaging and archiving several small files containing metadata information and content of small files into a single HAR file and then accessing the small files. The Sequence File is a serialized combination of a large number of small files into a large File using key-value keys and supports segmentation and block compression. Map File is a Sequence File after sorting, which USES index as the data index of the File to record the Key value of the data File and the offset of the data. All three contain disadvantages: once the file is set up, HAR cannot be modified, compression and automatic deletion are not supported, and merging takes a long time. The Sequence File has no index of its own, so it traverses globally during retrieval. Map File consumes a portion of memory to store the index data.

Article [5], NHAR scheme is proposed by improving HAR. Its idea is to hash the metadata of small files to improve the metadata management of NameNode, and then to establish indexes.

In the research field, most scholars focus on optimizing the amount of metadata, then merging small files into large files for storage. The processing idea is mainly from two aspects: one is the preprocessing of files, what is the benchmark of the association between files, how to set the benchmark to make files with large association can be selected; the other is the small file merging algorithm, which can improve the memory utilization rate for the files meeting the Association, so as to reduce the overhead of nodes and save magnetism Disk storage space.

In terms of file preprocessing, article [6, 7] all use the filtering method based on file extension. The former also uses encryption technology to encrypt files, and then merges small files according to the size of files. And the latter uses pipeline merging to improve the efficiency of memory. The preprocessing based on scene use usually adopts the method of classification. For a large number of logs generated in the computer, article [8] uses Java coding technology to transfer out to XML file, and then preprocesses according to the device name. Article [9] is classified and merged based on the Department of management documents, which is of great significance to the storage and research of national cultural resources in China. Article [10] analyzes user's behavior through small file access log and merges according to the access frequency between files, which is more accurate than other preprocessing methods. Generally, the user will make a request again for the first file in a unit time, so that the file will have a clustering range. In the unit range, merging and storing the files with high correlation can not only improve the efficiency of reading and writing, but also reduce the system load. Therefore, this paper uses the historical access log as the basis of relevance.

After Hadoop 2.0, if the file size exceeds 128M after merging, the metadata will increase and the load of NameNode will increase. Otherwise, the space will be wasted. In reference [11], a Tetris merge algorithm based on balanced data block is proposed for small file merging, which aims to make the merged large files evenly distributed. In reference [12], a strategy of setting up multiple NameNode and dynamically dividing ring metadata is proposed. By starting two main nodes and hash algorithm at the same time, metadata is evenly distributed on NameNode. In reference [13], an optimization algorithm OMSs based on MapFile is proposed. From the worst point of view, small files are combined into a large file according to the worst fit strategy. In order to improve the performance of cluster and reduce the reading time of data, a heterogeneous aware replica deletion scheme is proposed in reference [14]. In reference [15], a metadata management method based on log merging tree and flat directory is proposed, which uses optimization measures such as memory mapping file and boolean filter to improve the ability of data storage, and uses flat directory to enhance the integrity and operability of metadata in the access process.

After small files are merged, there is usually a problem of low reading efficiency. Domestic and foreign researchers have also made relevant research and improvement. Article [16] proposes to build a secondary index based on the creation time and file type, and create a prefetching and cache module based on the commonly used files of users, which effectively improves the utilization of system space. Article [17], a data prefetching scheme scheduling aware is proposed to improve the access of non-local map task's age data, and the performance of hybrid cloud in data locality is significantly improved. For the problem that the application program can only access the data set once and reclaim the memory, article [18] proposes a heuristic algorithm that uses the standard form of integer linear programming to obtain the optimal solution and the job scheduling information to prefetching the data. This mechanism can effectively reduce the time of job execution in heterogeneous environment. All of these literatures make the performance of data prefetching and the caching mechanism of the system significantly improve the reading performance of files.

Through the analysis of the achievements of various researchers, we can see that the performance of small files cannot be improved without the combination algorithm and prefetching

caching mechanism. Therefore, it is necessary to improve and upgrade HDFS and design a general file storage system based on Hadoop.

3. Optimization of General File Storage Scheme Based on Hadoop

The optimization scheme of general file storage file system is divided into four parts: the first step is to judge the size of the file, and then deal with it separately; There are two parts in the second step :on the one hand, continuing to use the strategy of HDFS file storage for the large file; on the other hand, analysing the small file according to its historical access log and establishing the association probability model; the third step is to use the proposed merging algorithm based on directed graph. The fourth step is to build a general file storage system based on Hadoop by using cache replacement strategy to build a cache mechanism.

3.1. Establishment of Correlation Probability Model

In general, people will be interested in related items, and then businesses will use this mentality to build a binding model. In file access, if we use "bundle storage" reasonably in unit time, we can effectively save storage resources and reduce the space occupation of system storage.

Apriori [19] algorithm is a frequent item set algorithm for mining Boolean association rules. Through the double standards of file confidence and support, the associated files in unit time can be acquired at one time. By using this algorithm to filter the association of small files, the memory consumption of NameNode can be reduced and the access efficiency of small files can be improved.

First of all, the log files are counted. Due to the large number of files, it is impossible to identify all the file information. The simulation data of whether small files are accessed in unit time is shown in Table 1.

Table 1: Simulated experimental data.

UID	File1	File2	File3	File4
1	0	1	0	1
2	1	1	1	0
3	1	0	1	1
4	1	1	0	0
5	1	0	0	0
6	1	1	1	1

In order to analyze the correlation between File1 and File2, make file1 A and file2 B, and the probability that file1 and File2 are accessed in unit time can be expressed by support degree, as shown in equation 1, and the support degree of file1 to File2 is 0.5 through calculation.

$$\text{support} = P(AB) \tag{1}$$

In unit time, after file1 file is requested to be accessed, the probability that File2 file is also accessed can be expressed by confidence, as shown in equation 2, the confidence degree of file1 to File2 can be calculated as 0.6.

$$\text{confidence} = P(B|A) = \frac{P(AB)}{P(A)} \tag{2}$$

If we give min_confidence of 0.3 and min_support of 0.5, we can show that file1 and file2 are strongly related, and file2 can be used as an associated file. Reasonable determination of confidence and support is of great significance. If P(B|A) is very high, but P(AB) is very low, it means that these two documents appear rarely at the same time. Even if the confidence is higher, it cannot be used as a reference. Therefore, it is reasonable to use Apriori to set the minimum confidence and support.

3.2. Association-based Directed Graph Merging Algorithm

By using the Apriori algorithm, we can filter out the small file set that meets the correlation standard, but it is only limited between two files. If two small files are stored in each DataNode, the space utilization is still low, and the repeated storage of small files in each DataNode will also affect the retrieval rate of small files, which cannot solve the problem substantially. Based on the association relationship between multiple files, we propose a small file merging algorithm based on Association directed graph.

Definition 1: the digraph is <V,E>， as shown in Figure 1, which set V and E are limited, and V is not an empty set, including V as node,, its elements in the set are v_1,v_2,v_3,v_4,$v_{\dots}$, as shown in equation 3; E is an edge set, and the elements in the set are directed edges, as shown in equation 4; E is the set of edges between the nodes in V, and is a multiple subset of Cartesian product V xV.

$$V = \{v_1, v_2, v_3, v_4, v_{\dots}\} \tag{3}$$

$$E = \{\langle v_1, v_2\rangle, \langle v_1, v_3\rangle, \langle v_1, v_{\dots}\rangle, \langle v_2, v_1\rangle, \langle v_2, v_4\rangle, \langle v_2, v_{\dots}\rangle, \langle v_{\dots}, v_1\rangle, \langle v_{\dots}, v_2\rangle\} \tag{4}$$

After file (v_1) is triggered, the access probability of associated files (v_2, v_3, v_4, $v_{\dots}$) are taken as the weight of the directed graph <V, E>, and $P(e_{1,2}+e_{1,3}+e_{1,4}+e_{1,\dots})<1$.

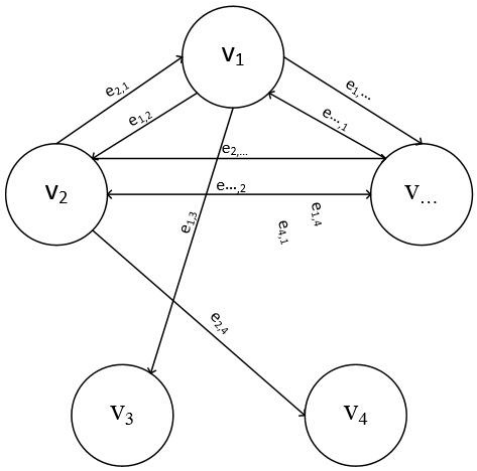

Figure 1: Directed graph based on node correlation degree.

Taking digraph <V, E> as an example, the flow chart of the association based digraph merging algorithm is shown in Figure 2. In the process of small file association merging, if v_1 is de associated with v_2, but it does not mean that $e_{1,2}$ must be higher than $e_{2,1}$. In addition, if a node is deep enough, the parent node cannot be found. Therefore, the files to be merged cannot be deleted.

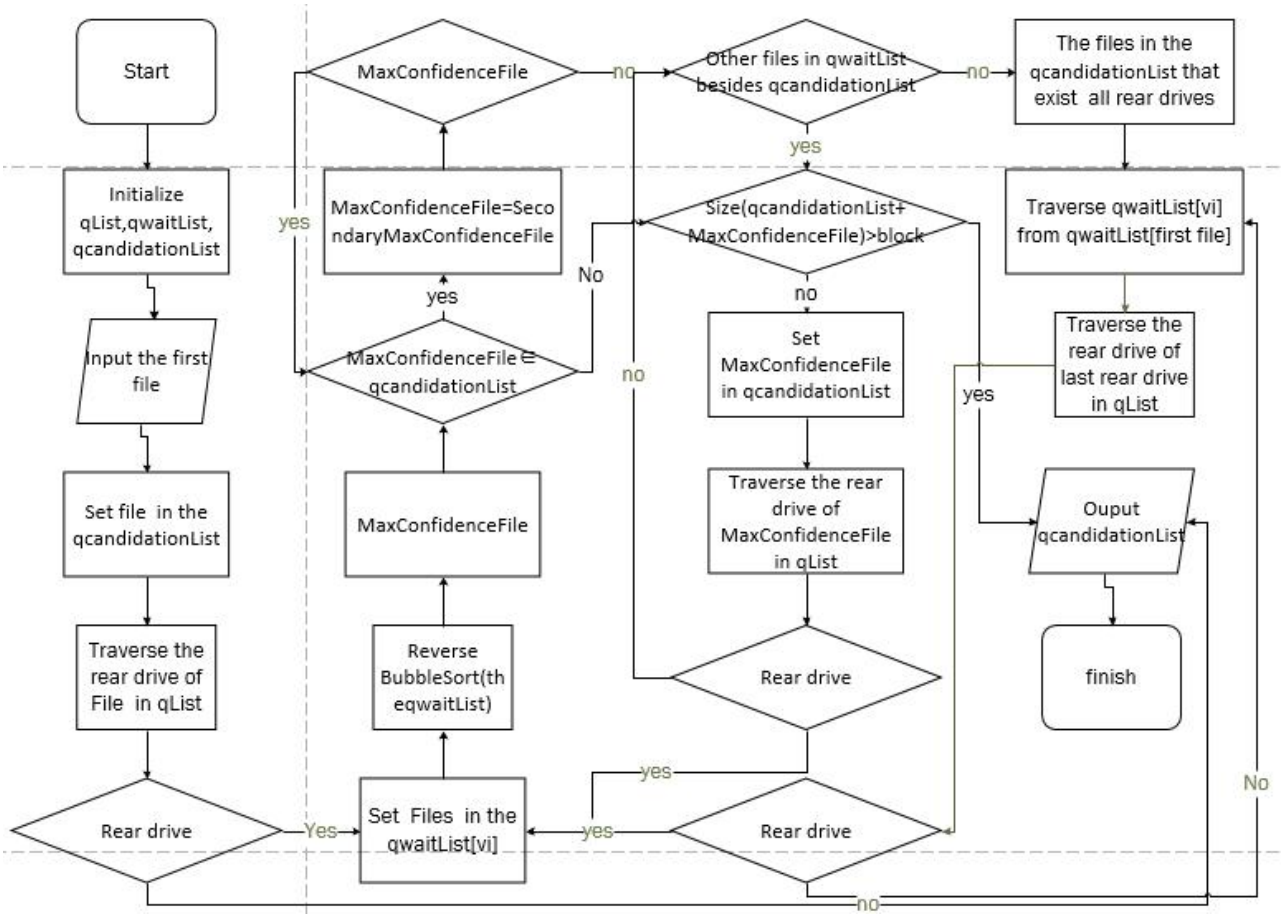

Figure 2: A flow chart of merge algorithm based on relevance directed graph.

(1) Initialize qlist (small file set), qwaitlist (small file set with confidence), qcandidationlist (ready merge small file queue), turn to 2;

(2) Select a file as the input item, store it into qcandidationlist, file is used as the precursor, traverse the backdrive of this file in qlist, if there is a backdrive, jump to 3; otherwise, turn to 11;

(3) The found files are stored in the qwaitlist, and the qwaitlist is named by the precursor, turning to 4;

(4) According to the confidence degree, perform the reverse bubble sorting on qwaitlist (Due to the advantages of bubble sorting: strong stability, space complexity, and time complexity are relatively the lowest). Select the first small file as the maxconfidence file, and turn to 5;

(5)To determine whether maxconfidentiefile has been included in the qcanditionlist. If yes, turn to 6.;Otherwise, turn to 7. In addition to the first merge of input files, this judgment is generally required.

(6) Loop through qwaitlist to find the small file with the next highest confidence level, and then turn to 7 if the maxconfiidentefile does not exist in qcanditionlist; In the end, if we didn't find one that met the requirements, turn to 9.

(7) Judge whether the size (qcanditionlist + maxconfidentiefile) is greater than block (128M). If it is greater than 128M, do not update the qcanditionlist and turn to 11; otherwise, turn to 8.

(8) Save maxcondensefile into qcandidationlist, and then use maxcondensefile as the precursor to traverse the qlist to find the successor. If there is rear drive, turn to 3; otherwise, turn to 9.

(9) QwaitList named after the trigger file does qwaitList exist in addition to the existing file of qcandiationlist,, if so, turn to 7; if not, turn to 10;

(10) In the qcandidate list, the precursors of all the rear-drive nodes are queried, the rear-drive of the precursors are traversed in turn, and each rear-drive is queried whether there is still a rear-drive as a precursor. If so, turn to 3; if not, iterate until there are no new small files or memory greater than 128M;

(11) Output qcandidationlist.

Using this algorithm, we can filter out the small files to be merged, with two exceptions: memory overflow, and all small files satisfying the confidence are put into the qcandidationlist.

3.3. Merge and Index Mechanisms

Traverse qcandidationlist according to the confidence between small files, the contents of small files are merged by mergefile (), and the merged file is named NameNode according to the time the file was written.

In HDFS, the client obtains the metadata by visiting NameNode, and then obtains the DataNode by offset to obtain the small files in the data block. The access to each small file must interact with NameNode, which increases the load on the system. Therefore, building an appropriate index mechanism can reduce the consumption of NameNode. What we use is to build an index file for the named large file, store it in the block, and record the mapping relationship between the small file and the large file, as well as the volume size and offset of the small file. This method can effectively reduce the number of interactions with NameNode and improve the performance of the system.

3.4. Prefetching and Caching Mechanisms

We use the merge algorithm and index mechanism to reduce the memory consumption, but affect the read performance of the system. The HDFS is designed based on the mode of "write once, read many times". Access information can be obtained by directly accessing the DataNode. There is still a problem of hot data. On the basis of the original HDFS, we also added the process of association calculation and small file merging, which has a more delayed impact on the user's file reading. Therefore, we introduced prefetching and cache machine System.

The prefetching mechanism we use is to directly return the information stored in the cache to the user if it hits the cache after the user sends out the request; if not, send a request command to HDFS to transmit the information returned by HDFS to the client. At this time, what we need to do is to package the small files associated with the request file into the cache. If so, when the access frequency is relatively high, its correlation file will also be accessed, so as to improve its access speed. In this case, due to the space limitation of cache, not all related files can be cached, so how many files with high correlation can be prefetched becomes a problem. For example, When customers are shopping at the mall, the time for customers to choose goods plus the time for sales to provide related services cannot exceed the patient waiting time of customers, otherwise, customers will be lost. We define that the time to hit the file from the cache is time (cache), the time to directly access the file from HDFS is time (HDFS), and the time to prefetch each file is time (prefetching). Then, the time for the user to request and get the file from HDFS once and cache the related file is as shown in equation 5.

$$\text{time(read from HDFS)} = \text{time(HDFS)} + \sum_{i=1}^{n} (\text{time(prefetching)} \times P_i) \tag{5}$$

The waiting time of the user is time (wait). The number of caches that can be cached is shown in formula 6, 7.

$$\text{time(read from HDFS)} \leq \text{time(wait)} \tag{6}$$

$$\text{time(HDFS)} + \sum_{i=1}^{n} (\text{time(prefetching)} \times P_i) \leq \text{time(wait)} \tag{7}$$

Through equation 7, we can find that the number depends on Pi， which can ensure that the system provides high-quality prefetching performance.

Cache is only a temporary tent and can be replaced at any time. Therefore, clearing cache in time can clear redundant data and ensure real-time data. In general, the Least Recently Used (LRU) algorithm will adopt a higher frequency, and clear the data that has not been accessed for a long time and has few accesses from the cache. However, we analyze the internal data access log of the HDFS cluster of Yahoo [20], It is found that 89.61% of the data blocks will have the last access within 10 days after their upload, and the probability of data block access is almost zero. When it comes to real-time and large amount of data, this algorithm is not perfect. Based on the real-time and limited cache, we choose the high-hot and commonly used associated file cache, which can improve the file access speed and cache hit rate.

4. Experiment and Result Analysis

The experimental environment adopts a Hadoop cluster of a NameNode and two DataNodes, all of which are configured on Intel i5 CPU 2.3GHz, 4GB memory and 20TB disk. Our operating system is RedHat 7.0, Hadoop version is 2.7.5, JDK version is 1.7, HDFS data block size is 128M, and the number of copies is 3. By comparing the HDFS, Har and our improved HDFS (IHDFS), the evaluation criteria are volume of metadata used, average file read time and upload time.

In order to ensure the authenticity and effectiveness of the experiment, 80000 files are selected in this paper, with size distribution ranging from 100k to 150m, mainly small files, especially those under 5m. The number of selected files and the volume of small files are shown in Figure 3. We tested the performance of each of the 10000,20000,400000, and 800000 files for the experimental subjects.

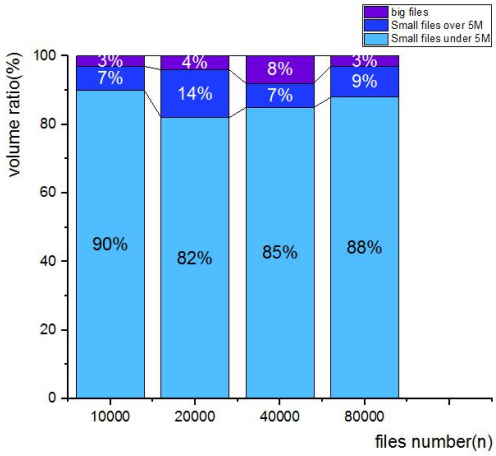

Figure 3: File number and volume distribution scale graph.

In this paper, the directed graph merging algorithm is based on the correlation of small files, which uses the historical access log of coding to small files to analyze and summarize into excel, and then combines the Apriori algorithm to realize which small files are accessed together with high

probability. We use enumeration trial and error method to set the minimum confidence level of small files to 0.3, and the support level should not be less than 0.5, then we filter out small files that conform to confidence > 0.3 & & support > 0.5.

4.1. Experiment on File Memory Consumption

The original HDFS, HAR and IHDFS were used to carry out 5 groups of experiments respectively, each group of experiments was carried out for 3 times, and the average value was taken as the evaluation basis. The volume of metadata is shown in Figure 4, with the x-coordinate representing the number of files and the y-coordinate representing the volume of metadata in the NameNode.

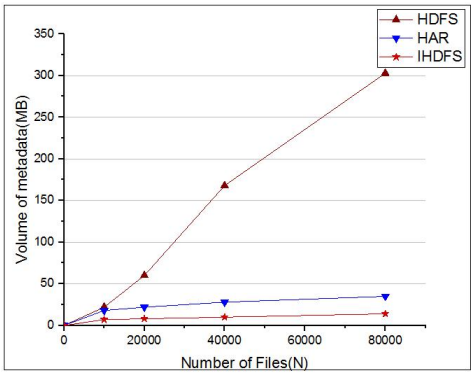

Figure 4: Volume of metadata in the NameNode

It can be intuitively seen from the above figure that with the increase of the number of files, the number of small files increases sharply, and the memory consumption of HDFS becomes more and more serious; HAR does reduce memory consumption by taking advantage of archiving, while IHDFS consumes the least memory. The reason is that small file merging reduces the metadata of NameNodes, thus reducing memory consumption and improving space utilization.

4.2. Experiment with File Reading Time

The original HDFS, HAR and IHDFS were used to conduct five experiments, each of which was conducted three times. The average value of the data was selected and divided by the total number of files. That is the Average Read Time per File (ARTPF), as shown in equation 8. The relationship between the number of files and ARTPF is shown in Figure 5. The abscissa represents the number of files, and the ordinate represents the average time reading files.

$$\mathrm{ARTPF} = \frac{\sum_{i=1}^{n} \mathrm{read\ time}_i}{n} \tag{8}$$

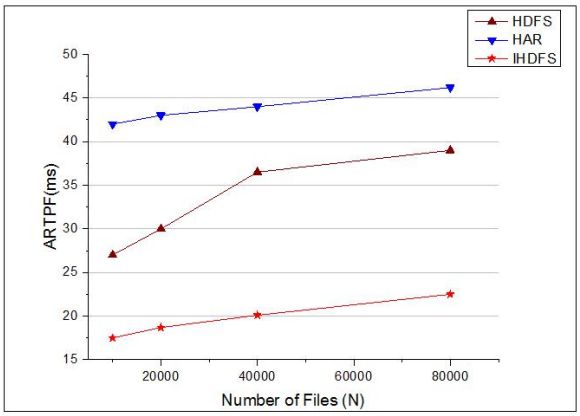

Figure 5: Average read time per file.

It can be seen from the figure that HAR scheme consumes the most time when reading every file, because it has no index mechanism and the retrieval is slow. As the number of files increased, the reading time of HAR and HDFS increased significantly, but the reading time of IHDFS was only half that of HDFS. This is because we added the prefetching and cache strategies, and the adoption of HDFS can improve the reading performance of the system and ensure the robustness of the system.

4.3. Experiment with File Uploading Time

Use HDFS, HAR and IHDFS respectively conducted a series of five experiments, the experiment done three times in each group to choose, the Average of the sum of the File Upload Time divided by the total number is the Average File Upload per Time (AUTPF), as shown in the equation of 9, the number of files and AUTPF relations as shown in Figure 6, abscissa represent the number of files, the consumption of ordinate represents the average upload file time.

$$\mathrm{AUTPF} = \frac{\sum_{i=1}^{n} \mathrm{upload\ time}_i}{n} \tag{9}$$

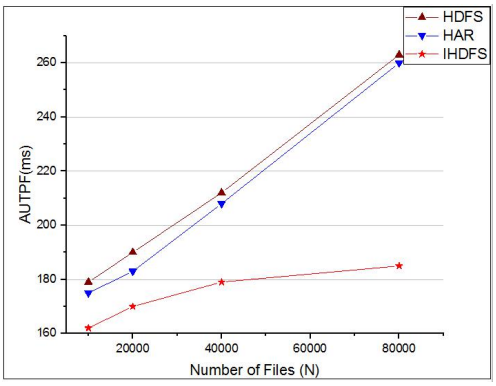

Figure 6: Average upload time per file.

The unit file upload time can reflect the upload performance of the scheme. According to the information in the figure, the storage performance of HDFS and HAR is close, while the performance of IHDFS is obviously better than theirs. As the number of files increased, the upload time of files increased, but the IHDFS remained at 0.185s and tended to be stable.

5. Conclusions

In this paper, the problems of high memory and low access efficiency of massive small files are studied, including three modules: establishment of correlation probability model, merging of small files and prefetching of small files. By taking the relevance and volume of files as parameters, a association-based directed graph merging algorithm is proposed, an indexing mechanism is established based on access time and offset, and the LRU substitution strategy is improved by taking the high heat as the influence factor, so as to improve the cache hit ratio. Combined with the high processing capacity of HDFS for large files, IHDFS becomes a storage scheme for common files. Experiments show that IHDFS can effectively merge small files, reduce the pressure of NameNode, and improve the cache hit ratio by using the pre-fetch strategy to determine the number of cache files, thus optimizing the file reading performance. It is a pity that the correlation algorithm proposed in this paper does not consider the type of file, and it is not sure whether the combination algorithm can achieve the optimal performance under the circumstance of different file types. If the effect is not expected, how to optimize it will be the future research work.

References

[1] Yu Zhaohui. The 44th statistical report on the development of China's Internet network released by CNNIC [J]. Cyberspace military-civilian integration, 2019 (09):30-31.)

[2] Zhang yifeng. Research and implementation of massive small file processing strategy based on HDFS [D]. Southeast university, 2018.

[3] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, Samir Belfkih, Big Data technologies: A survey, Journal of King Saud University - Computer and Information Sciences, Volume30, Issue 4, 2018, Pages 431-448, ISSN 1319-1578.

[4] Fang guodong. Research on voice storage and retrieval of network intercom based on HDFS [D]. Huaqiao University, 2019.

[5] Li X, Han J, Thong Y, Ital. Implementing Websites on Hadoop: A case study of improving small file I/O performance on HDFS [J].2013:1-8.

[6] Mohd Abdul Ahad, Ranjit Biswas. Dynamic Merging based Small File Storage (DM-SFS) Architecture for Efficiently Storing Small Size Files in Hadoop [J]. Procedia Computer Science, 2018,132.

[7] Devi. A distributed efficient storage method for file resources across HDFS cluster [J]. Electronic design engineering, 203, 27 (21): 14-17 + 22.

[8] Cheng xiaorong, li yujin, li zijun, liu yuchen. Research on optimization method of small and medium file storage in network security equipment linkage system [J]. Computer knowledge and technology, 2015, 11 (35): 10-11.

[9] Liu yunyu, liu yan. Research on the storage of national folk cultural resources based on cloud platform [J]. Science and technology vision, 2019 (01): 86-88.

[10] Li tie, yan cairong, huang yongfeng, song yalong. Optimization method of small file access for Hadoop distributed file system [J]. Computer applications, 2014, 34 (11): 3091-3095 + 3099.

[11] He, H., Du, Z., Zhang, W. et al. Optimization strategy of Hadoop small file storage for big data in healthcare. J Supercomput 72, 3696–3707 (2016).

[12] Bhang Y,Li D.Improving the efficiency of storing for small files in hdqrs[C].Computer Science & Service System (CSSS), 2012 International Conference on. IEEE, 2012:2239-2242.

[13] SETHIA D, SHEORAN S, SARAN H. Optimized Map File based Storage of Small files in Hadoop[C]//Ieee/acm International Symposium on Cluster, Cloud and Grid Computing. 2017: 906–912.

[14] Ciritoglu Hilmi Egemen,Murphy John,Thorpe Christina. HaRD: a heterogeneity-aware replica deletion for HDFS. [J]. Journal of big data, 2019, 6 (1).

[15] Wang kun.Research on small file storage mechanism for Hadoop [D]. Beijing University of posts and telecommunications, 2018.

[16] Luo qing. Optimization and implementation of small file storage problem based on HDFS under Hadoop platform [D]. Huazhong University of science and technology, 2019.

[17] Chunlin Li, Jing Zhang, Yi Chen, Youlong Luo. Data prefetching and file synchronizing for performance optimization in Hadoop-based hybrid cloud [J]. The Journal of Systems & Software, 2019,151.

[18] C. Chen, T. Hsia, Y. Huang and S. Kuo, "Data Prefetching and Eviction Mechanisms of In-Memory Storage Systems Based on Scheduling for Big Data Processing," in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 8, pp. 1738-1752, 1 Aug. 2019.

[19] Ji, Q., Zhang, S. Research on sensor network optimization based on improved Apriori algorithm. J Wireless Com Network 2018, 258 (2018).

[20] GreenHDFS: towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster. Kaushik R T, Bhandarkar M. International Conference on Power Aware Computing and Systems. 2010.