Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 5525009, 10 pages
https://doi.org/10.1155/2021/5525009

Research Article

Hadoop-Based Medical Image Storage and Access Method for Examination Series

Xin Huang[1,2] **Wenlong Yi**[3] **Jiwei Wang**[4] **and Zhijian Xu**[2]

[1]*School of Software, Jiangxi Normal University, Nanchang 330031, China*
[2]*Department of Computer Science and Technology, Tongji University, ShangHai 201804, China*
[3]*School of Software, Jiangxi Agricultural University, Nanchang 330045, China*
[4]*Department of Information, The 73th Group Army Hospital of P.L.A, Xiamen 361003, China*

Correspondence should be addressed to Wenlong Yi; yiwenlong@mail.ru and Jiwei Wang; 772073190@qq.com

Received 26 January 2021; Revised 22 February 2021; Accepted 9 March 2021; Published 25 March 2021

Academic Editor: Yi-Zhang Jiang

Under the background of electronic medical data, doctors use electronic images to replace the traditional film for diagnosis, and patients can view examination images at any time through various electronic means. The storage and frequent reading of massive data bring new challenges. Given the characteristics of the size and quantity of image files generated by different examination types, different merging strategies are proposed to improve the storage performance of the files; according to the characteristics of medical data with examination as the basic unit, a two-level model combined with medical imaging information is proposed. The indexing mechanism solves the problem that SEQ files cannot be read randomly without an index; given the time characteristics of data access, an improved 2Q algorithm is proposed to cache the prefetched files and the read files in different cache queues, which improves the efficiency of file reading. In the experimental comparison, the proposed algorithm surpasses the baseline method in storage and access performance.

1. Introduction

Smart medical is an important part of a smart city. The electronization, informatization, and intelligentization of hospitals can effectively improve the efficiency of the medical system. Massive electronic medical imaging data provide powerful data support for intelligent auxiliary diagnosis algorithms and also pose new challenges for multisource complex data storage and access [1–4]. In the new application scenario, doctors use electronic images to replace the traditional film for diagnosis, and patients can view the examination images at any time through various electronic methods (WeChat and AliPay). These queries are usually based on a certain examination. Measured by the size and quantity of the files, the medical image files produced by a certain examination should be classified as small data. According to our statistics shown in Table 1, common DICOM medical images such as CT (computed tomography) and MRI (magnetic resonance imaging) are only a few hundred KB in size, and their single examination usually has 100 to 300 images. The size of a single US (ultrasound) image is only a few hundred KB, and the number is relatively small. Although the file size of a single image of CR (computed radiography) and DR (digital radiography) can reach 4~10 MB, the medical image generated by one examination usually does not exceed 5. Figure 1 shows our statistical results of the size distribution of medical image files generated in one day's examination in a Shanghai tertiary hospital. Among them, the examination files less than 15 MB accounted for 75%, 15 M to 45 M accounted for 13.1%, between 45 M and 70 M accounted for 6.42%, and larger than 75 M accounted for 5.58%. Although the HDFS (Hadoop distributed file system) technology used in traditional PACS storage can easily store and manage petabytes of data, its original intention is how to compress the storage better, and the performance is not good in the scenario of massive random read and write.

Table 1: Statistics of data generated by a single examination of medical imaging.

Type	Single file (MB)	Numbers	Series files (MB)
CT	0.18	258	46.44
MR	0.12	124	1.59
US	0.53	3	1.59
CR	5.79	2	11.58
DX	4.56	1	4.56

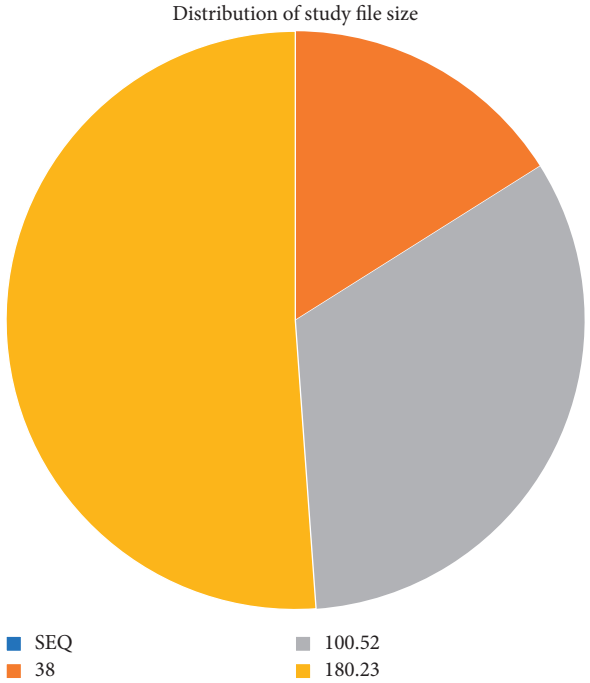

Figure 1: Medical file size distribution for examination.

In the application scenario of the patient-oriented query, PACS (picture archiving and communication systems) will face more frequent and fragmented query requests. The traditional PACS storage method has poor performance in this application scenario. There are three reasons: (1) files or directories in Hadoop need to generate corresponding metadata in the NameNode, and the NameNode in Hadoop is a single-point design, which easily causes the memory of the NameNode to be tight; (2) when a large number of small files make storage requests to HDFS, each small file storage will request the NameNode to allocate data blocks, which will cause huge pressure on the NameNode; and (3) when storing and accessing a large number of small files, it is necessary to frequently make RPC (remote procedure call) requests to the NameNode and DataNode, which puts a lot of pressure on Hadoop. It is not efficient to store and read a large number of small files in HDFS [5]. Each access must first obtain file information from the NameNode before accessing the corresponding DataNode. Therefore, for files with the same amount of data, small files will constantly access the NameNode and then read small files from one DataNode to another, which seriously affects the storage and access performance.

The current research mainly focuses on how to optimize the Hadoop cluster and does not specifically optimize the application scenarios where the hospital uses examination as the basic unit of storage and access [6–10]. This paper proposes improved storage and reading method based on Hadoop, which fully considers the characteristics of small medical image files, redesigns new file formats, uses different merging strategies for different types of medical image files, and optimizes 2Q (two-queue) algorithm to improve file access efficiency. The main contributions of this paper are as follows:

(1) We propose a merging strategy for different image files to improve the storage performance of files

(2) We propose a secondary indexing mechanism combined with a hierarchical model of medical imaging information to improve the performance of random access to small files

(3) We propose an improved two-queue algorithm that caches prefetched files and read files in different cache queues to improve file access performance

The rest of this paper is organized as follows. Section 2 reviews the related work for HDFS storage and access. Section 3 details the design of the proposed method. Sections 4 and 5 present and discuss the experimental settings and results, respectively. Finally, we draw the conclusion in Section 6.

2. Related Work

2.1. HDFS Storage and Access. Ghemawat et al. [11] improved the HAR technology to merge small files. The merged file is divided into three parts, MasterIndex, Index, and Data. The MasterIndex is stored in the NameNode, and the Index is used to find the merged large file. Index and Data are stored in the DataNode, and the Index is responsible for the index of the small file in the large file. This method effectively reduces the memory consumption of the NameNode and improves the storage efficiency of the NameNode metadata. Chen and Gong [12] proposed to merge small files into SequenceFiles in turn according to a merge algorithm. This method merges a large number of small files into a small number of large files, greatly reducing the number of metadata in the NameNode and improving the metadata retrieval query efficiency. Yu et al. [13] proposed a cloud storage system based on HDFS design, which uses SequenceFile to optimize the storage efficiency of small files and combines the multiattribute decision theory and experiments based on this theory to obtain the optimal way to merge files. Jiang et al. [14] proposed a method for data blocks in the DataNode to save the metadata information of small files, which reduces the amount of metadata in the NameNode and at the same time reduces the number of requests to the NameNode when the client reads and writes data and improves the overall Hadoop cluster file I/O performance. To solve the small file problem of the general distributed file system in cloud storage, Wang et al. [15]

proposed a file merging and prefetching strategy based on user access tasks to improve the PLSA model. Experimental results show that this strategy has a high prefetch hit rate, which can effectively reduce the load of the metadata server and the response delay of user requests. Liu et al. combined the relational database management system with HDFS, and the relational database was used as the starting point for small file access. There are two storage strategies in this method. (1) A large number of different data request sources make file storage requests in a short period. Small files are organized in a time-based manner and merged for this scenario. (2) For a long time but there is only one requesting source for file storage requests, for this scenario, the request source is used as the unit to organize the files. This approach makes full use of the access efficiency of relational databases and the advantages of Hadoop distributed storage architecture for storing large files [16]. Zhang et al. proposed the merging algorithm HIFM, which starts with the correlation between small files and small files and the directory structure between files and files in special scenarios and builds an index for the files in each directory, and each directory created higher-level index. At the same time, the preloading strategy of index files and data files is designed to improve the reading efficiency of small files [17].

2.2. Medical Big Data Storage and Access. Chahal et al. [18] used a fusion-based approach to develop a framework to balance accuracy and time and then proposed a hybrid fuzzy c-means and k-means method which was implemented using the Hadoop MapReduce platform to improve the scalability of the cluster. Ryu [19] proposed an efficiency model for expanding Hadoop clusters to execute MapReduce applications more economically, which can improve the cost efficiency of big data analysis. Li et al. [20] then used a decomposition-based multiobjective optimization memetic algorithm to improve the default strategy in the Hadoop distributed file system, which effectively improved the performance of HDFS. Li et al. [21] proposed a virtual file pool technology, which uses a memory-mapped file method to achieve an efficient data reading process and provides a data exchange strategy in the pool. Rathidevi and Parameswari [22] proposed a centroid-based clustering of the small files method, which placed related files in clusters to improve data access performance. Xiang et al. [23] proposed a Hadoop distributed file system storage strategy based on the response time of data nodes, and the simulation results show that this strategy can realize the distributed storage of big data and avoid the emergence of hot nodes.

In summary, the current research on medical imaging big data storage ignores the characteristics of clinical diagnosis as a unit of examination. Under the background of large-scale access applications, the efficiency is not high. The development of this research will help resolve such a research gap.

3. Methods

3.1. Medical Image Hierarchy Based on Examination Series. The medical image hierarchy based on the examination sequence follows the principle of "patient-examination-sequence-image": (1) a patient may have multiple examinations; (2) each examination may include one or more image series; and (3) each sequence has one or more images. Figure 2 shows the hierarchy diagram.

Among them, the uppermost layer of the structure is the patient layer, which is a collection of all medical examination documents of a certain patient. The second layer is the examination layer, which contains all the image files produced by a patient during a medical examination. This is also the data that doctors query most frequently. The third layer is the sequence layer. Since a certain medical examination of a patient may involve multiple parts and a variety of different images, the images generated by a certain part are saved as a corresponding sequence, and different series form a second-level directory. The bottom layer of the structure is the image layer. Each sequence directory stores its image, which is the basic unit of the storage structure.

3.2. Multistrategy Merging Method Combined with Secondary Index. According to the medical imaging hierarchy based on the examination sequence described in Section 3.1, the images under the same sequence have strong internal connections, and there is a strong correlation between different sequences in the same examination. However, the original SequenceFile after merging in Hadoop cannot reflect the logic between the data, so we propose a new MD (medical data) file container that considers the file type and the corresponding SMERGE (small file merge) model. To reduce the pressure on the NameNode, a secondary index mechanism is introduced to effectively speed up the indexing of small files.

3.2.1. Multistrategy Merger Method. Since our storage method is improved based on Hadoop, we still use 128 MB of storage space as the basic unit of merged files. The traditional SequenceFile merge method is to merge the files into a 128 MB SEQ file, but there is a high probability that the single examination file of a patient will be divided into two data blocks. In this case, a single examination of the patient requires a data read across data blocks. Our approach is to store a check in the same data block as much as possible. The SEQ merge method is shown in Figure 3.

According to the statistics in Table 1, we divide medical images into two categories. One is MR, CT, and other examination files. The images produced by such examination usually contain multiple sequences, and a single sequence contains multiple image files. In practical applications, doctors often check with a sequence as the basic unit; the other type is based on CR, DR, and examination files. The images produced by this type of examination are usually single or several files.

For the former, we merge with a sequence as the basic unit, and for the latter, we merge with examination as the

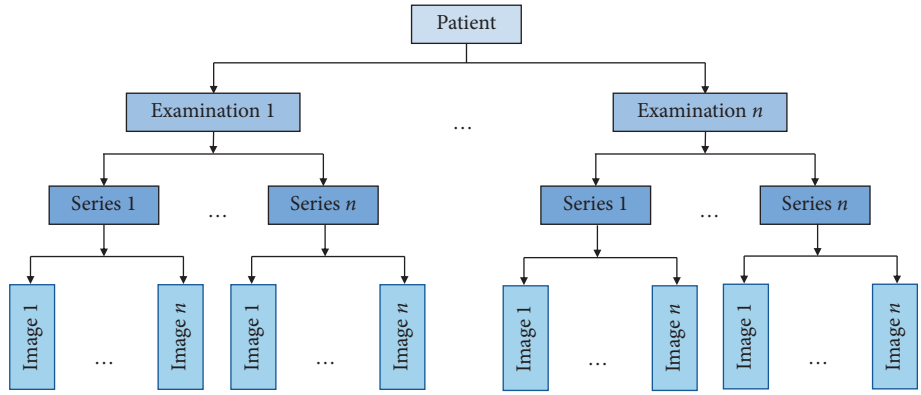

Figure 2: The hierarchy of medical image storage.

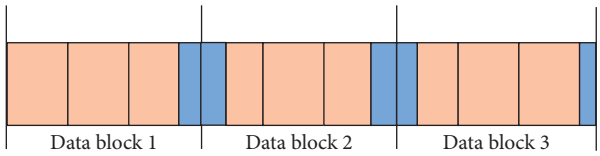

Figure 3: SEQ merge method.

basic unit. To ensure the same sequence or check the existence of the same storage unit, we do not adopt the strategy of filling the 128 MB storage space but adopt the strategy of approximately 128 MB space.

Figure 4 shows our merging strategy. The combined approximately 128 MB file is called the MD file container. Before the file container is added to the check file, it will be judged whether the capacity of the current file container plus the size of the check file exceeds the threshold, and if it exceeds the threshold, a new file container will be created. There are two reasons: the first reason is any file in Hadoop occupies a data block. For example, a 1 MB file will only occupy 1 MB of disk space, but it will still occupy a 128 MB data block. As close as possible to 128 MB can save a lot of data blocks; the second reason is to avoid dividing an examination file into different data blocks during storage.

Based on the above merge strategy, we designed a new merge model, SMERGE, whose structure is shown in Figure 5. The SMERGE process is divided into five steps:

(1) Scan all examination files of that day and preprocess them daily. Organize files by type and merge examination files that belong to the same type.

(2) According to the multipolicy merging of small files, the examination files were merged into the MD file container, and the examination files generated by different imaging devices adopted different merging strategies.

(3) Remove files from the examination files, in turn, put them into MD data files, and generate their indexes at the same time, and put their indexes in the MD index file container.

(4) Judge the capacity of the current file container and check whether the size of the file exceeds the threshold value. If it exceeds the threshold value, create a new file container.

(5) Finally, the generated MD file collection is stored in the HDFS cluster through the HDFS client, and the index file is put in the index pool.

The MD index file is divided into two parts: the file header and the file body. The file header contains the file flag, the name mapping table, and the synchronization flag point. The name mapping table records the small file name and the corresponding serial number in the form of key-value pairs. The file body is composed of several file records. A record is composed of three parts, which are the length of the file record, the serial number (fixed byte), and the file address. MD data files are also divided into two parts: file header and file body. The file header is the same as the MD index file. The file body is composed of several file records. A record is composed of three parts, which are the length of the file record, the serial number, and the file content. The collection of small files is merged into several MD data files and uploaded to HDFS, and the corresponding MD index files are stored in the index pool.

3.2.2. Secondary Index. The secondary index file structure is two levels. The first level index is generated by the examination date, examination type, and the serial number of large files. The second layer is the index file called SIndex (secondary index) file of each MD file. The SIndex file stores the offset value of the small file in the corresponding MD file. To speed up index search, a mapping table of serial numbers and small file names is added to the header of SIndex files and MD files. We use a combination of centralized storage and distributed storage to manage index files and data files, store the index files centrally in the index pool, and upload the data files to HDFS, thus realizing the separation of index files and data files.

3.3. Prefetch Cache Method Based on the Improved Two-Queue Algorithm

3.3.1. Two-Queue Algorithm. The 2Q (two-queue, 2Q) [24] algorithm consists of two queues, the A1 queue with FIFO (first input first output) buffering and the A2 queue with LRU (least recently used) buffering. The two buffering

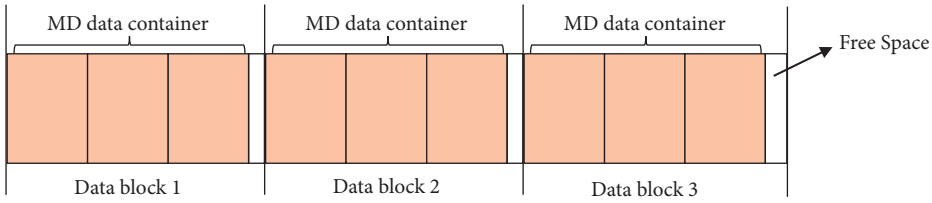

Figure 4: MD merge method.

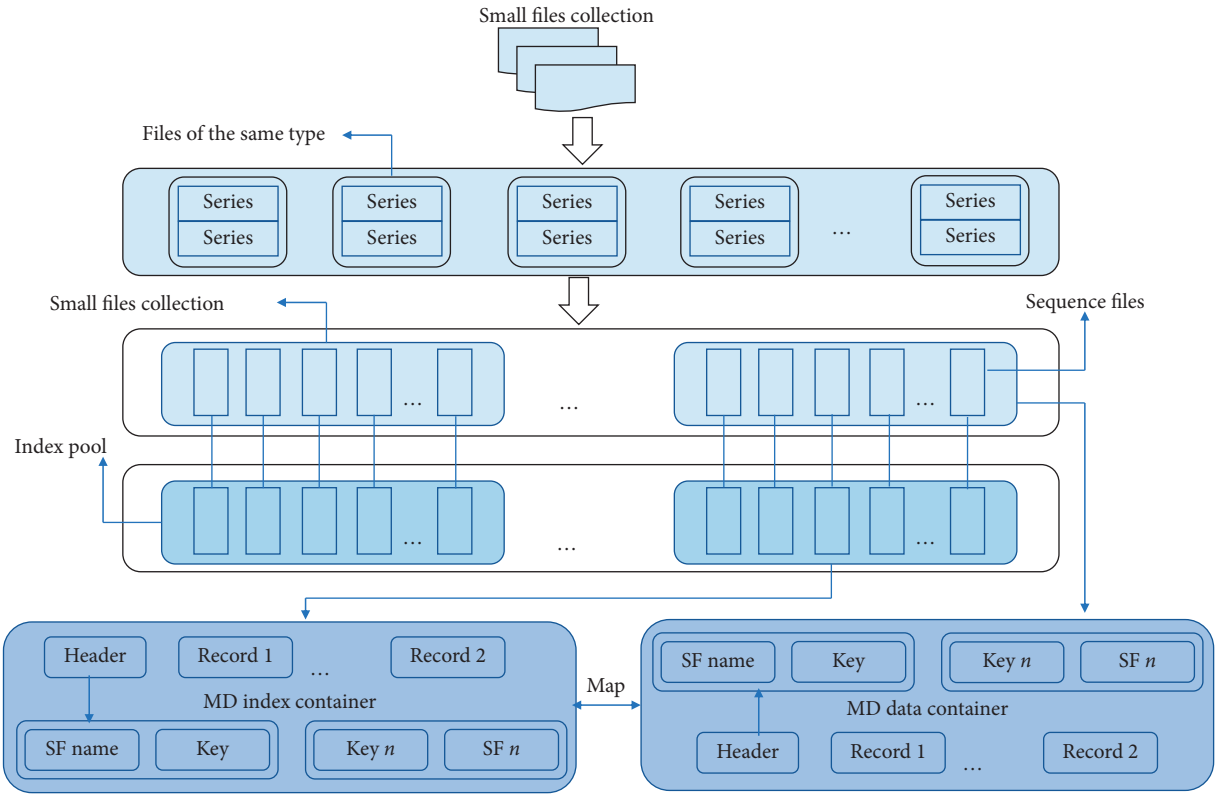

Figure 5: SMERGE model, SF name means sequence file name.

queues buffer and eliminate data according to their own buffering rules.

The flow of the algorithm is shown in Figure 6. Specifically, the algorithm is divided into four steps:

Step 1. When the data are accessed for the first time, it is directly added to the A1 cache queue. If the cache queue is full, the cache will be eliminated in the order of first in, last out.

Step 2. When the data are accessed again in the A1 cache queue, move the buffer in the A1 cache queue to the A2 cache queue, which is an LRU cache queue.

Step 3. When the data are accessed again in the A2 cache queue, move the data to the end of the linked list of the A2 cache queue, indicating that it has been accessed recently.

Step 4. When the A2 cache queue is full, the cache is eliminated according to the LRU algorithm.

3.3.2. Improved Two-Queue Algorithm. The access to medical imaging data by patients and doctors is often concentrated in the consultation phase, which results in most of the data being frequently accessed in a short period (in the consultation period) and rarely accessed after the consultation period. According to this feature, we have improved the 2Q algorithm. The improved 2Q algorithm splits the original A1 queue into two FIFO queues, A0 and A1. When the data are accessed for the first time, it enters the A0 queue. After a while, the eliminated data enter A1. If the data are accessed again in the A1 queue, it will enter the A2 queue. The advantage of this is to ensure that the data that have been repeatedly accessed in the short term stay in the cache as much as possible. The A2 queue is still an LRU queue. The flow chart of the improved 2Q algorithm is shown in Figure 7.

After the improvement, the write cache method is shown in Algorithm 1.

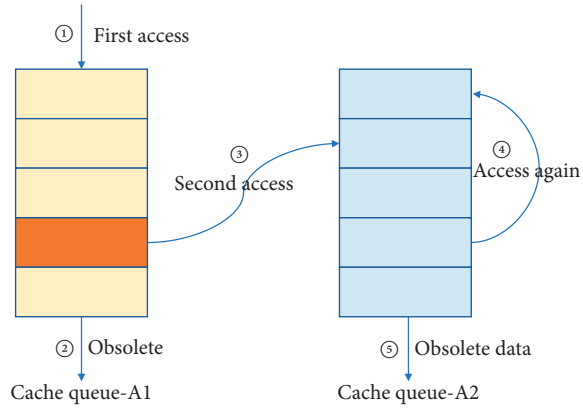

Figure 6: Two-queue algorithm flow.

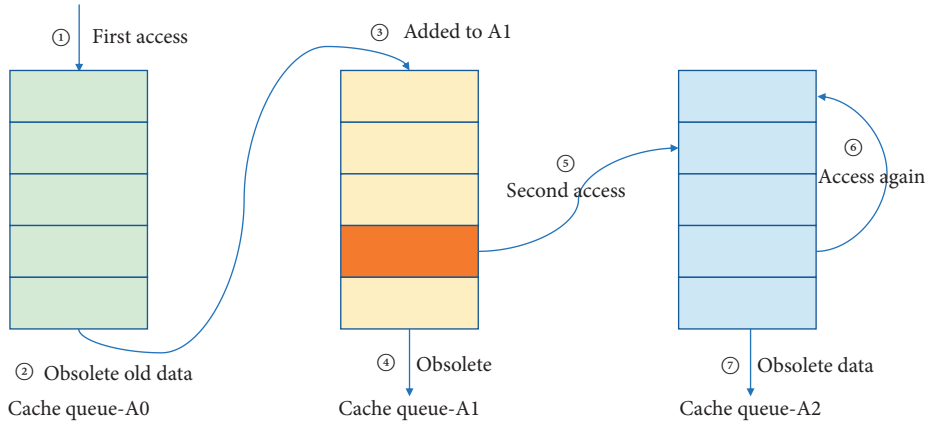

Figure 7: Improved two-queue algorithm.

```
    Input: data to be written X
(1)   begin
(2)     if A0 is not full
(3)         Insert data X at the top of |A0 in|
(4)     else if |A0 in| is full
(5)         Take out the bottom data Y of |A0 in|, and insert Y to the top of |A1 out|
(6)         if |A1 out| is full
(7)           Eliminate the data at the bottom of |A1 out|
(8)         endif
(9)         Insert X at the top of |A1 out|
(10)    endif
(11)  end
```

Algorithm 1

After the improvement, the read cache method is shown in Algorithm 2.

Before starting the cache, first set the cache address and the maximum capacity of the cache. The cache capacity initial ratio of A0, A1, and A2 is 2 : 2 : 4. The write operation of the cache is done using "DiskCache.Editor." The editor provides the entry editing function and provides the input and output stream objects of the file corresponding to the entry.

The read cache operation needs to provide a key, find the file corresponding to it according to the key, and then return the file stream of the file. Reading files is divided into four steps:

```
Input: data to be read X
(1)  begin
(2)    if X is in A2
(3)      Move X to the top of A2
(4)    else if X is in |A1 out|
(5)      Delete X from |A1 out| and move it to the top of A2
(6)    else if X is in |A1 in|
(7)      Do nothing
(8)    else//x is not in the cache
(9)      Request X to disk or network
(10)   end if
(11) end
```

Algorithm 2

(1) If the read object is in the A0 queue, return the read object directly

(2) If the read object is in the A1 queue, first return the read object and then move the read object from A1 to the head of the A2 queue

(3) If you read the object A2 queue, first return the object and then move the read object to the head of the A2 queue

(4) If there is no cache in the caches A0, A1, A2, request the file server to download the file and issue a write operation to the cache

A0, A1, and A2 adopt the data structure of LinkedHashMap. LinkedHashmap is composed of a Hashmap and a linked list. Hashmap is responsible for the storage of key-value content, and the linked list records the insertion order of key value, which can be FIFO or LRU. Because the order of all cache queues is recorded by a linked list, the time complexity of moving and deleting cache contents is O(1).

4. Experiments

4.1. Data Preprocessing. We select CT, US, and CR medical images generated by a hospital in Shanghai for ten consecutive days as experimental data. The data preprocessing is divided into four steps:

(1) Read and judge whether the file is a DICOM file.

(2) Analyze data elements in DICOM files, such as file UID, file name, check date, and file type.

(3) Generate file mapping table, record file UID, check date, file type, and serial number (serial number is generated during the merging process); large file naming rule is "date_file type_serial number.suffix"; the file mapping table is used to establish the mapping relationship between the small files to be merged and the large files after the merge.

(4) The examination documents are organized according to the two directory levels of examination date and document type based on examination documents.

In the end, we got 649,583 CT images, 2830 US images, and 3713 CR images.

4.2. Baseline. This experiment will be compared with Vanilla HDFS and SEQ methods. Among them, Vanilla HDFS directly transfers small files into HDFS without any processing, and SEQ uses SequenceFile to merge single-check files into one large file.

4.3. Details. The experimental network environment is Ethernet, and the communication speed between the hosts is 1.0 Gbps. The cluster contains 3 hosts, and the specific performance of different hosts is shown in Table 2:

5. Results

5.1. Small File Merge Result. Table 3 shows the results of merging the different method files. As shown in the table, for all 649,583 CT image files, the total number of files merged by the SEQ method is 2167 and the total number of files merged by the SMERGE method is 1043. Although the combined performance is excellent, the performance of the SMERGE method is still twice as good as that of the SEQ method. For all 2830 US image files, the total number of files merged by the SEQ method is 1040 and the total number of files merged by the SMERGE method is 37. For all 3713 CR image files, the total number of files merged by the SEQ method is 2183 and the total number of files merged by the SMERGE method is 176.

We also use BFR (big file ratio) to measure the effect of merging. BFR is the ratio of the number of large files after merging to the number of files after merging. Table 4 shows the detailed values of the BFR. It can be seen from the table that the ratio of the files merged by the SEQ method to large files is very low. For the US and CR medical image files, if the files are merged according to the single examination file, the "large files" obtained by the merger are still small files, while the files merged by the SMERGE method are almost all large files. The experimental results show that the SMERGE model can effectively merge small medical image files and greatly reduce the number of HDFS small- and medium-sized files, and almost all the merged files are large.

Table 2: Experimental host performance parameters.

Host	CPU	RAM (G)	Role
N1	Intel Xeon E6510	32	NameNode
N2	Intel Xeon E6510	32	NameNode
N3	Quad-Core AMD Opteron Processor 8378	16	DataNode

Table 3: Comparison of the number of files after the merger.

Number	Vanilla HDFS	SEQ	SMERGE
CT	649583	2167	1043
US	2830	1040	37
CR	3713	2183	176

Table 4: BFR value of the merged image file.

BFR	Vanilla HDFS (%)	SEQ (%)	SMERGE (%)
CT	0	19.75	71.43
US	0	0	100
CR	0	0	95.24

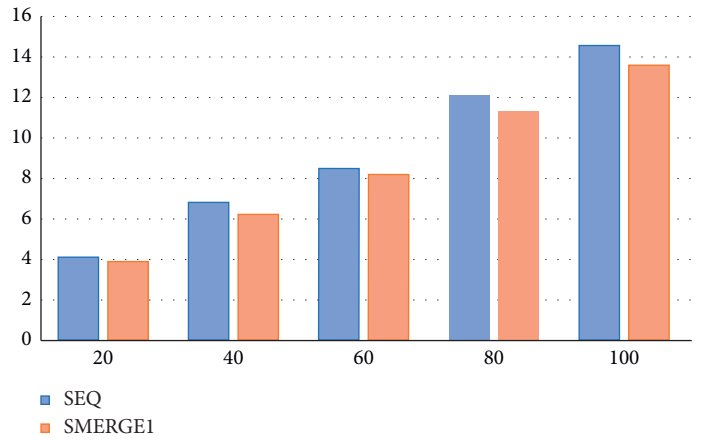

Figure 8: CR file reading speed comparison.

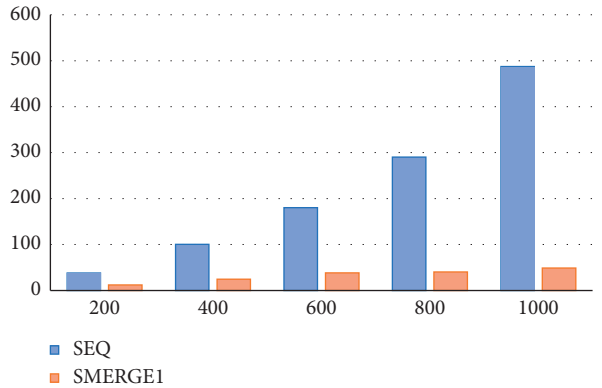

Figure 9: US file reading speed comparison.

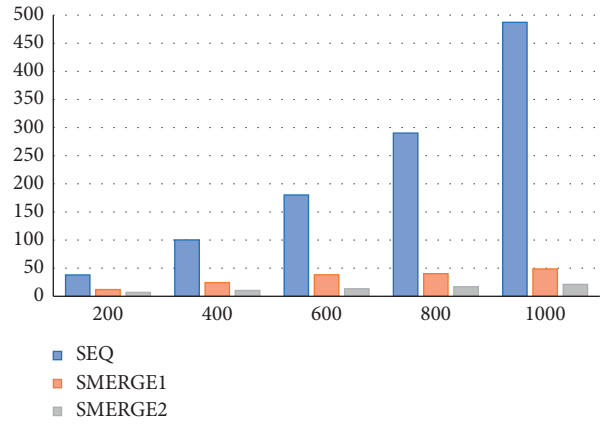

Figure 10: CT file reading speed comparison.

5.2. Small File Prefetch Cache Result. To eliminate experimental errors caused by network blockage or other uncertain factors, the same experiment was repeated 7 times before and after. Then, the maximum and minimum values were removed, and the average value of the remaining 5 results was obtained. The results of accessing different image files are shown in Figures 8–10.

For US and CR image files, the SMERGE method is slightly more efficient than the SEQ method in reading files. For CR file, average single file read speed is as follows: SEQ: 130 ms; SMERGE: 117.49 ms and for US file, average single file read speed is as follows: SEQ 36 ms; SMERGE: 43 ms, resulting in the difference in the US and CR file read time mainly because a CR file is about 5 MB and a US file is about 500 kB. For CT type images, the SMERGE method has a better performance in reading speed, and the average reading speed of a single file is as follows: SEQ: 434.31 ms, SMERGE1: 43.05 ms, and SMERGE2: 13.36 ms. SMERGE1 represents the SMERGE method using the original 2Q caching mechanism, and SMERGE2 represents the SMERGE method using the improved 2Q caching mechanism. The improved 2Q caching mechanism by adding the prefetch method greatly improves the efficiency of reading files.

6. Conclusion

Aiming at the low efficiency of small file storage when Hadoop processes small medical image files in a new scenario, this paper proposes an improved storage and reading method that fully considers the characteristics of small medical image files.Different merging strategies for different types of medical image files are used, and the 2Q algorithm to improve file reading efficiency is improved. The main work of this paper includes: different merging strategies are proposed for different image files to improve the storage performance of the files; a secondary index mechanism is proposed combined with the medical image information hierarchical model to solve the problem that SEQ files cannot be read randomly without an index the file problem; and an improved 2Q algorithm that caches prefetched files and read files in different cache queues is proposed, which improves the efficiency of file reading.

At the same time, there are still some areas for improvement in our research, and further research will be carried out on this basis in the future. (1) When adding a check file to the MD file container, consider the file size so that the actual file size contained in each MD file container is closer. (2) For the prefetch mechanism, the number of prefetched files needs to be dynamically adjusted according to specific scenarios.

Data Availability

The data used in this study were obtained from a tertiary hospital in Shanghai. It is due to the privacy of patients, currently, there is no plan to disclose the data. If necessary, data samples after desensitization can be provided.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Natural Science Foundation of China (grant no. 61762048) and the Science and Technology Research Project of Jiangxi Provincial Department of Education (grant no. GJJ200318).

References

[1] D. Jin, A. P. Harrison, L. Zhang et al., "Artificial intelligence in radiology," *Artificial Intelligence in Medicine*, pp. 265–289, 2020.

[2] J. Pavlopoulos, V. Kougia, I. Androutsopoulos et al., "Diagnostic captioning: a survey," 2021, https://arxiv.org/abs/2101.07299.

[3] X. Huang, F. Yan, W. Xu, and M. Li, "Multi-attention and incorporating background information model for chest x-ray image report generation," *IEEE Access*, vol. 7, pp. 154808–154817, 2019.

[4] C. Huang, X. Huang, Y. Fang et al., "Sample imbalance disease classification model based on association rule feature selection," *Pattern Recognition Letters*, vol. 133, pp. 280–286, 2020.

[5] X. Yaya and Z. Bi-Geng, "Research on medical image storage and retrieval system based on Hadoop," *Journal of Physics: Conference Series*, vol. 1544, no. 1, Article ID 012119, 2020.

[6] Q. A. Yao, H. Zheng, Z. Y. Xu et al., "Massive medical images retrieval system based on Hadoop," *Journal of Multimedia*, vol. 9, no. 2, p. 216, 2014.

[7] J. A. Rodger, "Discovery of medical big data analytics: improving the prediction of traumatic brain injury survival rates by data mining patient informatics processing software hybrid Hadoop hive," *Informatics in Medicine Unlocked*, vol. 1, pp. 17–26, 2015.

[8] X. Zhang and Y. Wang, "Research on intelligent medical big data system based on Hadoop and blockchain," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–21, 2021.

[9] K. Li, W. Ai, Z. Tang et al., "Hadoop recognition of biomedical named entity using conditional random fields," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 3040–3051, 2014.

[10] G. Jiang, "Research on medical data mining algorithm based on Hadoop platform in big data environment," *Machine Tool & Hydraulics*, vol. 2018, p. 18, 2018.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[12] J. Chen and F. G. Gong, "File merging strategy for optimization of distributed file system," *Journal of Computer Applications*, vol. 31, no. S2, pp. 161–163, 2011.

[13] S. Yu, X. L. Gui, R. W. Huang et al., "Improving the storage efficiency of small files in cloud storage," *Journal of Xi'an Jiaotong University*, vol. 45, no. 6, pp. 59–63, 2011.

[14] L. Jiang, B. Li, and M. Song, "The optimization of HDFS based on small files, broadband network and multimedia technology (IC-BNMT)," in *Proceedings of the 2010 3rd IEEE International Conference*, pp. 912–915, IEEE, Chengdu, China, 2010.

[15] T. Wang, S. H. Yao, Z. Q. Xu et al., "A small file merging and prefetching strategy based on access task in cloud storage," *Geomatics and Information Science of Wuhan University*, vol. 38, no. 12, pp. 1504–1508, 2013.

[16] X. J. Liu, Z. Q. Xu, and S. M. Pan, "A massive small file storage solution combination of RDBMS and Hadoop," *Geomatics and Information Science of Wuhan University*, vol. 38, no. 1, pp. 113–115, 2013.

[17] C. M. Zhang and J. W. Rui, An Approach for Storing and Accessing Small Files on Hadoop//Computer Applications and Software, 2012.

[18] P. K. Chahal and S. Pandey, "An efficient hybrid approach for brain tumor detection in MR images using Hadoop-map reduce," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and COMMUNICATIONS (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pp. 926–931, IEEE, Rhodes, Greece, 2020.

[19] W. S. Ryu, "Scaling of Hadoop cluster for cost-effective processing of MapReduce applications," *The Journal of the Korea Institute of Electronic Communication Sciences*, vol. 15, no. 1, pp. 107–114, 2020.

[20] Y. Li, L. Jiao, D. K. Saxena, Q. Zhang, Y. Wang, and M. Tian, "A new replica placement strategy based on multi-objective optimisation for HDFS," *International Journal of Bio-Inspired Computation*, vol. 16, no. 1, pp. 13–22, 2020.

[21] W. Li, C. Feng, K. Yu, and D. Zhao, "MISS-D: a fast and scalable framework of medical image storage service based on distributed file system," *Computer Methods and Programs in Biomedicine*, vol. 186, Article ID 105189, 2020.

[22] R. Rathidevi and R. Parameswari, "Performance Analysis of small files in HDFS using clustering small files based on centroid algorithm," in *Proceedings of the 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 640–643, IEEE, Coimbatore, India, 2020.

[23] M. Xiang, Y. Jiang, Z. Xia et al., "HDFS efficiency storage strategy for big data in smart city," in *Proceedings of the 2019 Chinese Automation congress (CAC)*, pp. 2394–2398, IEEE, Hangzhou, China, 2019.

[24] T. Johnson and D. Shasha, "2Q: a low overhead high performance bu er management replacement algorithm," in *Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 439–450, Santiago, Chile, 1994.