J Supercomput (2016) 72:3696–3707
DOI 10.1007/s11227-015-1462-4

Optimization strategy of Hadoop small file storage for big data in healthcare

Hui He[1] · Zhonghui Du[1] · Weizhe Zhang[1] · Allen Chen[2]

Published online: 17 June 2015

Abstract As the era of "big data" comes, the data processing platform like Hadoop was born at the right moment. But its carrier for storage, Hadoop distributed file system (HDFS) has the great weakness in storage of the numerous small files. The storage of numerous small files will increase the load of the entire colony and reduce efficiency. However, datasets such as genomic data and clinical data that will enable researchers to perform analytics in healthcare are all in storage of small files. To solve the defect of storage of small files, we generally will merge small files, and store the big file after merging. But the former methods have not applied the size distribution of the file, and not further improved the effect of merging of small files. This article proposes a method for merging of small files based on balance of data block, which will optimize the volume distribution of the big file after merging, and effectively reduce the data blocks of HDFS, so as to reduce the memory overhead of major nodes of cluster and reduce load to achieve high-efficiency operation of data processing.

Keywords HDFS · Storage of small files · Algorithm of merging of small files

1 Introduction

In 2012, the big data started to rise in China and attract attention, while 2013 has been widely regarded as the "First Year of Era of Big Data" [1]. According to the data of

✉ Weizhe Zhang
wzzhang@hit.edu.cn

Allen Chen
allen.chen.joggle@gmail.com

1 School of Computer Science and Technology, Harbin Institute of Technology, Harbin, HL, China

2 Lynbrook High School, San Jose, CA 95129, USA

ZDNET [2], the total amount of data produced in China in 2013 has exceeded 0.8 ZB (equaling to 800 million TB), which is two times of that in 2012 and equals to the total amount of data of the world in 2009. It's estimated that in 2020, the total amount of data produced in China will be 10 times of that in 2013, exceeding 8.5 ZB.

Hadoop has irreplaceable advantages in scalability, robustness, calculated performance and cost, and has become the main big data analysis platform of the current Internet enterprise in fact [3]. Hadoop conducts data storage by Hadoop Distributed File System (hereinafter referred to as HDFS). However, since Hadoop uses the data of information unit of all data blocks in clusters for storage in major nodes of Namenode, the operating pressure of nodes of Namenode will sharply increase when it stores a large amount of small files with volume smaller than 5 MB, which will need more memories for storage of information metadata; when processing large amount of small files, more MapReduce tasks need to be established, while the interaction and communication of the large amount of MapReduce tasks will increase the overhead of CPU, which will greatly reduce the operating efficiency of the whole Hadoop cluster.

This article proposes a method for merging of small files based on the balance of data block, to optimize the storage of small files in HDFS and the data processing procedure, so as to achieve high-efficiency operation of system.

2 Related work

Massive small files have become a recognized challenge in the industrial circles and academic circles [4]. Generally, files with volume within 5 MB are called small files. The current file systems, including local file system, distributed file system and object-based storage system, are all designed for large files. For example, XFS/EXT4, Lustre, GlusterFS, GPFS, ISLION, GFS and HDFS are all mainly targeting on the large files in implementation strategy of metadata management, data layout, stripe design and cache management, while the application of them in numerous small files has been greatly reduced in performance and storage efficiency, even not able to work on that.

The application of small files has become more and more common in practice, such as social network site, e-commerce, broadcasting and TV, online video, and high-performance computing. Now several classical application situations are given as examples. Facebook, the famous social network site, stores over 60 billion pictures, and promotes Haystack [5] which is custom-made and optimized, aiming at storage of large number of small pictures. Taobao is currently the biggest C2C e-commerce website, which stores more than 20 billion pictures with the average volume of 15 KB. It also promotes the TFS [6] file system aiming at optimization of small files to store pictures, and opens sources. The optimization of FastDFS [7] targeting at small files is similar with TFS. National University of Defense Technology has conducted optimization of Lustre, designed and realized the Cache freestanding structure of small files, Filter Cache, which will increase the measures of I/O cache of small objectives, based on the originally existing data path by extending data path at OST of Lustre, so as to improve the performance of Lustre [8].

About the optimization of storage of small files of HDFS, an article [9] proposes a new kind of file merging method, which has optimized the I/O performance of

the system to some extent, and improved the performance of distributed file system. Directing at the storage of small files in HDFS, literature [10] has proposed particular algorithm for the processing of small files before storage in HDFS and retrieval after storage, which improves the efficiency of storage and access of small files in HDFS. Literature [11–13] has put forward a method, which is, storing small files in HDFS after merging them into a single large file. In other words, the volume of file after merging is the sum of volumes of all the small files. And this can effectively reduce the storage space and calculation of CPU of system. However, it will not further optimize the distribution of volume of small files and volume of file block in HDFS. Aiming at these defects, this paper has proposed a method for merging of small files based on balance of data block, to optimize the storage of small files in HDFS, so as to achieve high-efficiency operation of system.

3 Defects and solutions of small file storage in HDFS

Hadoop distribution file system (hereinafter referred to as HDFS) performs well in processing large-volume data file. HDFS will divide the input data file into data block if 64 MB. The collected Namenode nodes will store the information metadata in all data blocks in the cluster, and Datanode nodes will store entities of all data blocks, each information metadata occupying 150 bytes at Namenode nodes. These data blocks will be invoked and processed by MapReduce framework.

Hadoop has in low efficiency when processing large amount of small files (generally from 10 KB to 5 MB). These small files will seriously affect the performance of Hadoop. Firstly, the storage of large amount of small files in Hadoop will increase the internal storage space for storing information metadata of data blocks in Namenode, further to limit the entire performance of this cluster. For example, for a large file of 1 GB, HDFS will divide it into 16 data blocks of 64 MB, and the storage space spent is only 2.4 KB. However, for 10,000 files of 100 KB, totally 1 GB, the storage space of Namenode will increase to 1.5 MB, increasing by over 600 times; secondly, small files smaller than the block will still occupy 64 M, and more MapReduce tasks need to be established when processing large amount of small files. The interaction and communication among the MapReduce tasks will increase the overhead of CPU.

For the disadvantages of Hadoop when processing large amount of small files, generally we will merge the small files into a large file and then input it into HDFS. This kind of strategy has advantages as below:

At first, it reduces a lot of metadata and lightens the burden of Namenode nodes. By storing lot of small files into a big file, and changing large amount of data of small files into data of large file, it reduces quantity of files, so as to decrease quantity of metadata of Namenode, increase the efficiency of retrieval and query of metadata, reduce the I/O operation delay of reading and writing of files, and save plenty of time for data transmission. The proportion of overhead of metadata of discretized small files is big, which will greatly reduce the metadata and directly result in the significant improvement of performance. The large file after merging is stored on the disk file system, and it will significantly decrease the pressure of disk file

system in metadata and I/O, which will improve the storage performance of each node.

Secondly, it increases data locality, and improves storage efficiency of HDFS. In the disk file system or distributed file system, the metadata of file and data are stored at different positions. After adopting merging storage, the metadata of small files and data will be stored in large file, which will greatly enhance the data locality inside each small file. During merging small files, we can use the spatial locality, temporal locality and relevance to adjacently store the small files that may be visited in a row in the large file, so as to enhance the data locality between small files. This has directly reduced the random I/O ratio on disk, and transformed it into I/O orderly, which can efficiently increase the storage efficiency of small files.

Thirdly, it has simplified the I/O access flow at nodes in Hadoop. By using merging storage of small files, the I/O access flow has greatly changed, which mainly presenting at the disk file system at storage nodes. When the disk file system reads or writes a small file, the biggest system overhead is the system call when opening files, which needs to find path, analyze components of name of path, and transform them into the internal representation in kernel of the corresponding files. This process occupies many system overheads, especially the files under deep content. But through merging, many small files will share a large file, and the opening of file is transformed into many deviating positioning operations of data with small overheads. We only need to position the corresponding positions in the large files according to index, and there is no need to establish the relevant VFS data objects in kernel, which saves most of the original system overheads.

Thus, after HDFS divides the large file formed through merging into data blocks of 64 M, the information metadata of data blocks stored at Namenode nodes will be greatly reduced, effectively decreasing the overhead in internal storage space. Meanwhile, it can improve the operating efficiency of MapReduce tasks processing data blocks at Datanode nodes.

4 Design and implementation of small files merging based on balance of data block

4.1 Basic idea of algorithm

The existing methods of small files merging based on the volume of files will generally set a threshold value of buffer area. It will constantly accumulate files in queues while conducting traversal on small files, and when the total volume exceeds the threshold value, the merging and storage will be conducted on the file cluster in the queue of buffer area. The process is shown as Fig. 1. However, these kind of methods will take “overflow of volume of file” as condition for merging, and ignore the defect of uneven volumes of files, so it will finally cause “uneven” volumes of large files formed through merging, and the waste of internal storage space at the Namenode in cluster of Hadoop to some extent. At the same time, the uneven distribution of volume of files is also harmful for the high-efficiency operation of parallel computing of MapReduce framework.

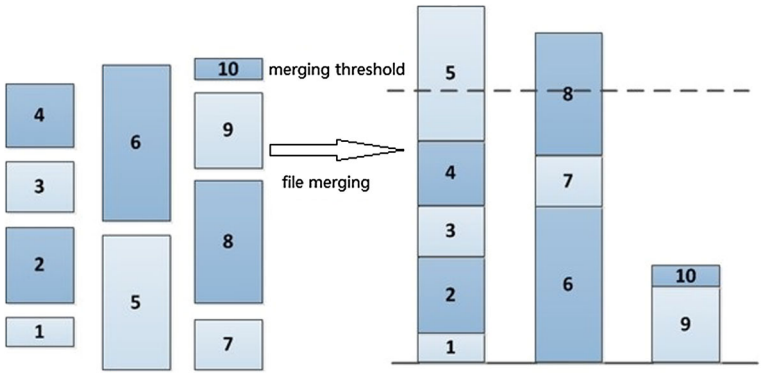

Fig. 1 Process of existing merging algorithm of small files

This paper proposes a method for small files merging based on balance of data block, of which the core idea is to evenly distribute the small files into large files according to the volume of small files, and transform the condition for file merging from "overflow of volume of file" into "approach to criticality" to ensure that the large file formed through merging will not include redundant blocks after dividing in HDFS. It has reduced the load of memory at Namenode nodes to some extent, and meanwhile, the even distribution of volume of files is helpful for the efficiency of parallel computing of MapReduce. Since the merging strategy of small files is similar with that of filling gaps in Tetris, this algorithm is called Tetris Merge algorithm, in short, TM algorithm.

4.2 Algorithm design

Firstly, we will introduce the data structure used in this algorithm. This paper classifies the queues in the algorithm into two categories, file merging queue and tolerance queue. Several merging queues of files in total are used to store the file collection for merging. When the total volume of files in queue reaches the conditions for merging, pack the file sets and merge them into HDFS, and then empty the queue; in the meantime, several tolerance queues are used to store the files with relatively big volume that "accidently" appear for buffering and ensuring the even distribution of volume of files after merging. The two kinds of queues can be switched to each other, and the strategies and conditions for switching will be introduced later in this paper.

Next, we will introduce the execution process of algorithm. The process is shown as Fig. 2. The algorithm contains two stages, file merging and post processing.

The execution process of the algorithm is as below:

1. Establish m file merging queues, q_{fl}, and n tolerance queues, q_{tl}, according to threshold value of file merging (generally $n < m$).
2. Conduct traversal all small files for merging, f, select a file merging queue and conduct enqueuer. The selecting principle is to select the queue with smallest total

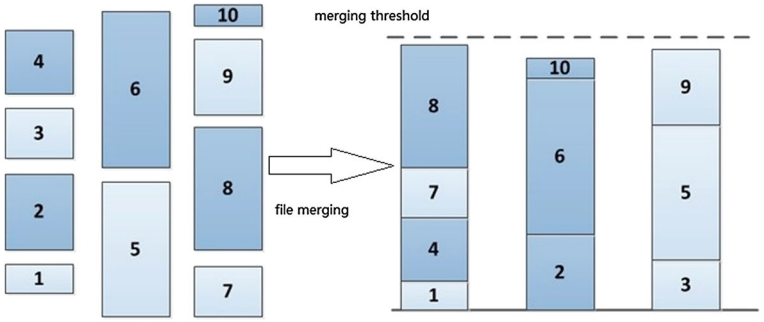

Fig. 2 Effect of file merging through TM algorithm

volume of files (queue with biggest free space), $q_{\max}$, in the existing queues. If the total volume of file after this file entering the queue is smaller than the threshold value of merging, then enter it normally; otherwise, it will enter exception handling.

3. In the exception handling, the entering of file into the queue will make the total volume exceed the threshold value of merging. If the total volume of files in this queue, $q_{\min}$, has exceeded 95 % of the threshold value of merging at this time, then merge the files in $q_{\min}$ and output, and then empty the queue. Enter the files for entering this queue into the queue; otherwise, if it does not exceed 95 % of the threshold value, then it proves that the files for entering the queue is a relatively large file, and this queue will be inserted into tolerance queue (if tolerance queue existing). At this time, this tolerance queue is changed into file merging queue, and will participate into the selection of merging queue in the next round.
4. Rule of emptying of queue. Before emptying the queue mentioned in Step 3 above, we must judge the conditions. When the quantity of queues of file merging at present is bigger than m at the beginning, no file shall be inserted after emptying this queue, but it will directly change into an empty tolerance queue.
5. The four steps are stages of the entire merging process. When the traversals of small files for merging are finished, leave the files not being merged in the existing set of queue, and post processing will start. The post processing mainly adopts monofile queue, and will merge the left files in queue.

4.3 Implementation of algorithm

Pseudocode of core implementation of TM, full named Tetris Merge algorithm, is described as below:

The large files formed through merging of small files will adopt form of SequenceFile supported by Hadoop. About the storage structure, SequenceFile mainly consists of multi Records following one Header. Header mainly contains Key classname, Value classname, storage and compression algorithm, metadata user defines, and others. Besides, it also includes some synchronized marks for quick positioning of the boundary of record, as shown in Fig. 3.

Algorithm TM(Tetris Merge):
Input: To merge small file sets named ***FileList***, merging threshold named ***MergeLimit***.
Output: After the merger of large files named ***MergedFile***.
1. Initialization: q_{fl}, q_{tl};
2. For each file f in ***Filelist***. If ***Flielist***==NULL, Then goto step 10;
3. Select max freespace queue q_{max} in q_{fl};
4. If f's size less remain size of q_{max} then go step5,
   Else goto step 6;
5. Push f into q_{max}, goto step 2;
6. Calc min freespace queue q_{min} remain size S_{min};
7. If S_{min}/ ***MergeLimit*** > 0.95 or q_{tl} is empty goto step 8,
   Else goto step 9;
8. Merge files in q_{min} to ***MergeFile***, push q_{min} into q_{tl};
9. Select a queue q_t from q_{tl}, push f into q_t, push q_t into q_{fl};
10. Merge remain files to ***MergedFile*** in queues;

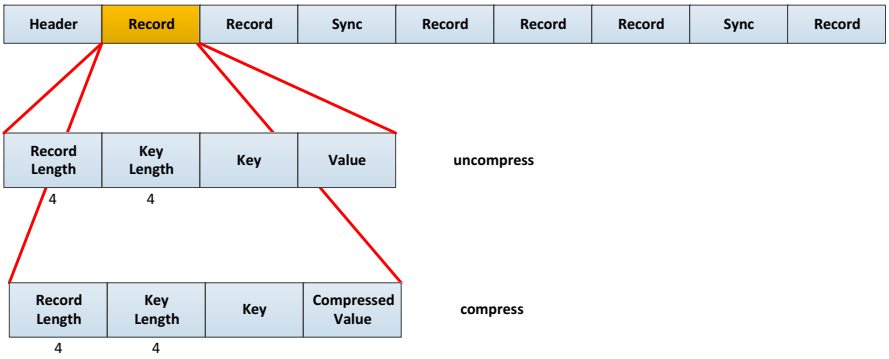

Fig. 3 File layout of SequenceFile

5 Experiments

5.1 Experimental environment

Experiments in this paper uses two servers to constitute Hadoop cluster, respectively at Namenode and Datanode, and the hardware environment is as shown in Table 1.

Edition of Hadoop is 1.2.1, and the edition of operating environment of Java is 1.6. The quantity of copies is set at 2, and size of HDFS data block is 64 MB under system default.

The collection of small files for testing data includes 4294 files, totally 10.12 GB. These files are all small files of different formats, and volumes of them are different from less than 100 to 64 KB, unequal in size. Figure 4 has shown the distribution of quantity and volume of file collections, among which small files with volume below 5 MB account for 97.71 % of the total quantity of files, while the files of volume between 5 and 64 MB are mostly used for observing the effect of file merging.

Table 1 Experimental environment

Namenode	
Operating system	CentOS 6.2
CPU	AMD Opteron(TM) 8Core 1.4 GHz *32
Memory	32 GB
Hard disk	320 GB
Datanode	
Operating system	CentOS 6.2
CPU	AMD Opteron(TM) 8Core 1.4 GHz *32
Memory	32 GB
Hard disk	320 GB

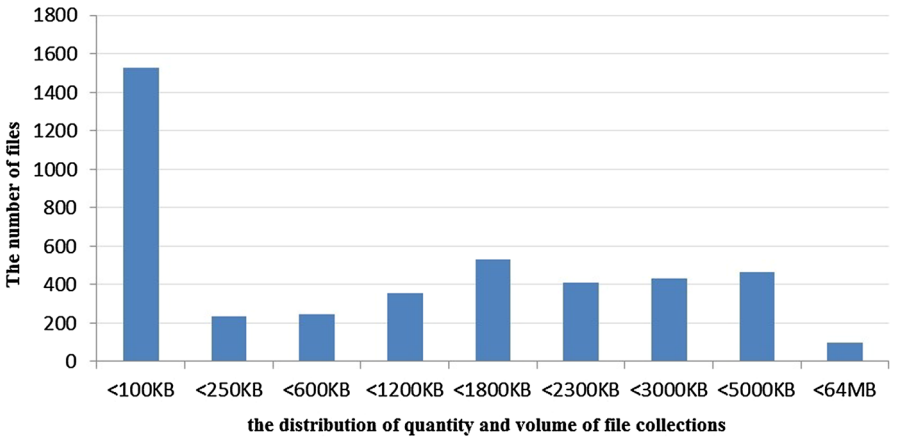

Fig. 4 Distribution of quantity and volume of file collections

5.2 Contrast experiment on time consumption of importing files into HDFS

Import files into HDFS by three different methods including TM algorithm, and record the time consumption. In monofile merging algorithm, the size of files is 128MB, the same as the threshold value of TM algorithm. Table 2 indicates the comparison of time consumption of importing files into HDFS by three methods. Figure 5 presents time consumptions by three methods in the form of histogram.

Table 2 Comparison of time consumption of importing small files

Algorithm of file import	Quantity of files/volume	Quantity of file after merging	Time consumption (s)
Normal import	4294/10.12 GB	4294	567
Monofile merging algorithm	4294/10.12 GB	82	249
TM algorithm	4294/10.12 GB	84	252

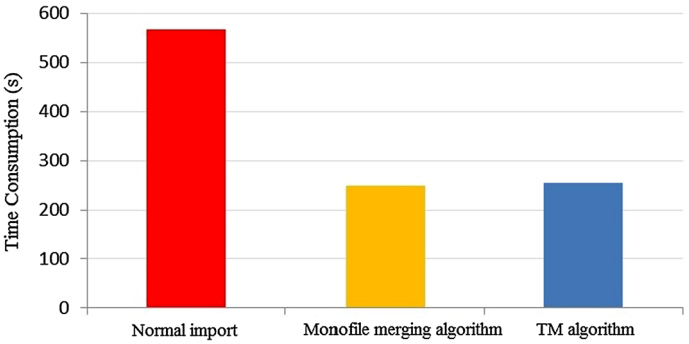

Fig. 5 Comparison of time consumption of importing small files

Through the comparison test, it can be concluded that in terms of time consumption of file import, TM algorithm is a little worse than monofile merging algorithm. The reason for this is that the triggering conditions of "file merging" of two algorithms are different, and the quantities of files after merging are different. The quantity of merged files by TM algorithm will not be less than that by monofile merging algorithm, so the time consumption is slightly inferior.

5.3 Comparison of memory consumption at Namenode

HDFS will divide the imported data into data blocks of 64 MB under default. Then the system will store the metadata of all data blocks on Namenode, and store all data blocks at Datanode. At Namenode, the information metadata of each data block will occupy about 150 bytes.

Among the tested data file collections, the total volume is 10.12 GB, and there are 4294 files with volume not smaller than 64 MB. If no operation is done on them, HDFS will establish 4294 data blocks. To store the information metadata of these data blocks, Namenode will consume 644,100 bytes. However, in the monofile merging algorithm proposed in literature [11], since redundant data blocks will be left when the file formed through merging (128 MB) divides the data blocks, it can not ensure the least memory consumption. TM algorithm will optimize according to the features of volume of data blocks in HDFS, to ensure no redundant data blocks will be divided from the files formed through merging, so the data blocks after dividing will be better in memory consumption at Namenode than the former two.

Table 3 indicates the influence of three methods to import files on the quantity of divided data blocks, and it finally causes the comparison of memory consumptions at Namenode. Figure 6 indicates the memory consumption at Namenode by three methods in the form of histogram.

We can see from the comparison test that the memory consumption of large files produced through monofile merging algorithm and TM algorithm at Namenode have been significantly improved when compared to the importing without any processing. In the meantime, TM algorithm considers the influence of volume of HDFS data block

Table 3 Memory consumption at Namenode

Algorithm of file import	Quantity of data blocks after merging	Memory consumption at Namenode (byte)
Normal import	4294	644,100
Monofile merging algorithm	245	36,750
TM algorithm	168	25,200

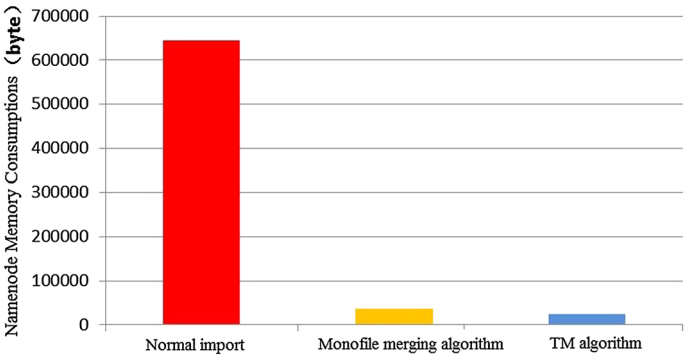

Fig. 6 Comparison of memory consumptions at Namenode

on the division of data block, and significantly reduces the quantity of data blocks when large file is divided when compared with monofile merging algorithm, and it finally greatly reduce the memory consumption at Namenode. In this experiment, the memory consumption at Namenode reduces by 31.4 %, compared to other monofile merging algorithm.

5.4 Comparison of file data processing speed

The original intention of Hadoop is to process files of large volume. Processing large amount of files with small volume will reduce the performance of Hadoop. There are totally 4294 small files of total 10.12 GB in the rest data file collection. If they are separately processed, it will take a long time.

Test experiment is to separately process the data imported into HDFS by three algorithms by using the same Chinese word count (Chinese words segmentation and then count), MapReduce program, compare the time consumption and get the processing speed. Table 4 indicates the comparison of time consumptions of processing by MapReduce on the produced data by three methods, and reflects the processing speed; Figure 7 shows the comparison of speed of data generation by three methods through MapReduce in the form of histogram.

We can see from the test comparison that the time consumption of processing either the data generated by monofile merging algorithm or TM algorithm has been greatly reduced through MapReduce processing when compared to normal import, that is, the

Table 4 Comparison of processing speed of MapReduce

Algorithm used for importing data	Time consumption of map stage (s)	Time consumption of reduce stage (s)	Total time consumption (s)
Normal import	13,993	333	14,326
Monofile merging algorithm	1190	119	1309
TM algorithm	1035	101	1136

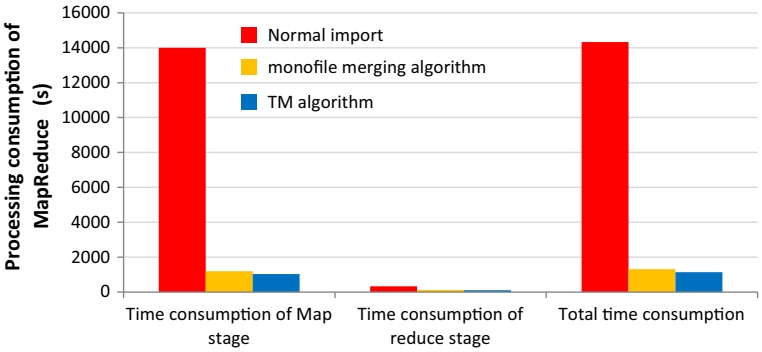

Fig. 7 Comparison of data processing speed by MapReduce

processing speed has been greatly improved. In the meantime, the processing speed of TM algorithm is much higher than that of monofile merging algorithm, with respective increase of 15.0, 17.8 and 15.2 % at Map stage, Reduce stage and the total process. The reason of the result is because the division of data generated by TM algorithm in HDFS is more reasonable, and the quantity of divisions is less, so as to further reduce the time consumption of I/O of system, and improve the speed of data processing.

Through the comparison tests, we can see that TM algorithm is only slightly worse than the monofile merging algorithm proposed in literature [11] in time consumption of data import, but it has great improvement in memory consumption at Namenode and processing speed of the generated data, which proves the efficiency of the algorithm.

6 Conclusions

This paper puts forward an optimized strategy for file merging and storage by aiming at the defects of Hadoop distributed file system on storage and processing of small files. The method is to conduct optimized grouping of the files for merging by using the distribution of volume of files, and it has greatly reduced the memory consumption at the major nodes in collections of Hadoop; it has improved the efficiency of data processing by MapReduce. Compared to the monofile merging algorithm, the effect has been improved by 15 %.

Firstly, this paper summarizes the existing solutions of storage of small files. Secondly, it analyzes the defects on storage of small files of Hadoop distributed file system. Based on the regularities of volume distribution of small files, we propose the small

file merging algorithm based on balance of data block, and give integrated algorithm. Finally, it conducts experimental analysis on the proposed algorithm, and proves that the algorithm it proposes can reduce the memory consumption at the major nodes of collections and improve the efficiency of data processing of collections.

The future work may be proceeded on the merging test about the similarities and differences of file types, and studying the influence on efficiency of data processing after finding the same file type and merging them into large file. In the meantime, the influence of different data sets on the experimental results can be tested.

Acknowledgments This work was supported in part by the National Basic Research Program of China under Grant No. G2011CB302605. This work is partially supported by the National Natural Science Foundation of China (NSFC) under Grant Nos. 61173145, 61472108.

References

1. Manyika J, Michael C, Brad B, Jacques B, Richard D, Charles R, Angela HB (2011) Big data: the next frontier for innovation, competition, and productivity. McKinsey Global Institute, Washington, pp 129–137
2. First website on cloud computing of China. http://www.zdnet.com.cn. Accessed 20 Feb 2010
3. Apache Hadoop. http://hadoop.apache.org. Accessed 10 Oct 2012
4. Yu L, Chen G, Wang W et al (2007) Msfss: a storage system for mass small files. In: 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2007. IEEE, Melbourne, Australia, pp 1087–1092
5. Beaver D, Kumar S, Li HC et al (2010) Finding a needle in Haystack: Facebook's photo storage. OSDI 10:1–8
6. Taobao File System. http://tfs.taobao.org/
7. Liu X, Yu Q, Liao J (2014) FastDFS: a high performance distributed file system. ICIC Express Lett Part B Appl Int J Res Surv 5(6):1741–1746
8. Qian Y, Yi R, Du Y et al (2013) Dynamic I/O congestion control in scalable Lustre file system. In: IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), 2013. IEEE, Lake Arrowhead, USA, pp 1–5
9. Mohandas N, Thampi SM (2011) Improving Hadoop Performance in Handling Small Files. In: Abraham A, Lloret Mauri J, Buford JF, Suzuki J, Thampi SM (eds) First International Conference, ACC 2011, Kochi, India, July 22–24, 2011, Proceedings, Part IV. Communications in Computer and Information Science, vol 193. Springer, Berlin, Heidelberg, pp 187–194
10. Grant M, Saba S, Wang J (2009) Improving metadata management for small files in HDFS. In: International Conference on Cluster Computing and Workshops, CLUSTER '09. IEEE, New Orleans, USA, pp1–4
11. Yan CR, Li T, Huang YF, Gan YL (2014) Hmfs: efficient support of small files processing over HDFS. Algorithms Archit Parallel Process Lect Notes Comput Sci 8631:54–67
12. Zhang WZ, Lu GZ, He H, Zhang QZ, Yu CL (2015) Exploring large-scale small file storage for search engines. J Supercomput. doi:10.1007/s11227-015-1394-z
13. Zhang WZ, He H, Ye JW (2013) A two-level cache for distributed information retrieval in search engines. Sci World J. 2013:Article ID 596724 (2013). doi:10.1155/2013/596724