Merging Small Files Based on Agglomerative Hierarchical Clustering on HDFS for Cloud Storage

Khin Su Su Wai, Julia Myint, Tin Tin Yee
University of Information Technology, Yangon, Myanmar
khinsusuwai@uit.edu.mm, juliamyint@uit.edu.mm, tintinye@uit.edu.mm

Abstract

Hadoop distributed file system (HDFS) was originally designed for large files. HDFS stores each small file as one separate block although the size of several small files is lesser than the size of block size. Therefore, a large number of blocks are created with massive small files. When the large number of small files is accessed, NameNode often becomes the bottleneck. The problem of storing and accessing large number of small files is named as small file problem. In order to solve this issue in HDFS, an approach of merging small files on HDFS is proposed. In this paper, small files are merged into a larger file based on the agglomerative hierarchical clustering mechanism to reduce NameNode memory consumption. This approach will provide small files for cloud storage.

1. Introduction

Cloud computing has become increasingly popular as the next infrastructure for hosting data and deploying software and services. Distributed file system is the foundation of cloud computing and provides reliable and efficient data storage for upper applications.

Today most popular storage system for cloud computing such as Google File System (GFS) and Hadoop Distributed File System (HDFS) are widely used and well known. However, HDFS is more light-weighted and open-source platform.

Hadoop is a software framework which is an open source that supports big data in distributed environment. It has two major components HDFS and Map Reduce. HDFS has master-slave architecture, with a single master called the NameNode and multiple slaves called DataNodes. NameNode manages the metadata and regulates client accesses. The metadata is maintained in the main memory of the NameNode to ensure fast access to the client, on read/write requests. DataNodes provide block storage and service read/write requests from clients, and perform block operations by contacting with NameNode.

The consumption of memory in NameNode is decided by the number of files stored in HDFS. Each file requires 150 bytes of memory space to store metadata in NameNode. DataNode is responsible for saving the real and replicated data. Each file is split into several blocks with the size of 128MB. The DataNode keeps on sending the heartbeat signal to the NameNode at regular intervals to indicate its existence in the system. The heart-beat consists of DataNode's capacity, used space, remaining space and some other information.

There are huge numbers of small files in the area of biology, climatology, energy, e-learning, e-business and e-library. A small file is a file whose size is less than the HDFS block size. Although the size of a file is less than the HDFS default block size, HDFS creates it as one block. When the large number of small files is stored, HDFS is inefficient because of high memory usage and unacceptable access cost. Hadoop cannot provide optimal performance for small files processing. In this paper, file merging will be presented to overcome small file problems in HDFS.

The rest of the paper is organized as follows. Section 2 is an illustration of related works with the proposed topic. Section 3 is the background theory of the proposed system. The proposed system is presented in section 4. The evaluation results are described in section 5. Finally, the paper is concluded in section 6.

2. Related Works

MENG Bing and et al. [1] provided a solution to reduce NameNode memory consumption, by TLB-Map File. TLB-MapFile merges massive small files into large files by MapFile mechanism to reduce NameNode memory consumption and add fast table structure (TLB) in DataNode, and to improve

retrieval efficiency of small files. A challenging work is to build up suitable TLB refresh cycle.

Zhipeng Gao and et al. [2] defined Logic File Name (LFN) and proposed the Small file Merge Strategy Based LFN (SMSBL). SMSBL is a new idea and a new perspective on hierarchy; it improves the correlation of small files in the same block of HDFS effectively based different file system hierarchy. This system solved small file problem in HDFS and has appreciable high hit rate of prefetching files. The proposed system needs to combine SMSBL with other great solution to improve performance of HDFS.

A new structure for HDFS (HDFSX) is presented by Passent M EIKafrawy and et al. [3] to avoid higher memory usage, flooding network, requests overhead and centralized point of failure of the NameNode. In the other word, the performance analysis of the systems is needed to be developed.

Tao Wang and et al. [4] defined a user access task. The correlations among the access tasks, applications and access files are constructed by the improved PLSA, and the research object is transferred from file-level to task-level. Then, an effective strategy is proposed to improving small file problem in distributed file system. The strategy merges small files in term of access tasks and selects a prefetching targets based on the transition of the tasks. This strategy reduces the MDS workload and the request response delay.

Parth Gohil and et al. [5] focused on a MapReduce approach to handle small files. This approach improves the performance of Hadoop in handling of small files by ignoring the files whose size is larger than the block size of Hadoop. This also reduces the memory required by NameNode to store these files. So, it requires very less amount of memory than original HDFS but it requires some more memory than HAR and Sequence.

Extended Hadoop Distributed File System (EHDFS) is used by Tanvi Gupta and et al. [6]. This paper focuses on increasing the 'efficiency' of the indexing mechanism for handling 'Small files' on HDFS. This also added the concept of 'Avatar node' that eliminates the single point of failure.

Kyoungsoo Bok and et al. [7] proposed a distributed cache management scheme that considers cache metadata for efficient accesses of small files in Hadoop Distributed File Systems (HDFS). The proposed scheme can reduce the number of metadata managed by a NameNode. Many small files are merged and stored in a chunk. It also reduces unnecessary accesses by keeping the requested files using clients and the caches of data nodes and by synchronizing the metadata in client caches according to communication cycles.

Yonghau Huo and et al. [8] used additional hardware named SFS (Small File Server) between users and HDFS to solve the small file problem. This approach includes a file merging algorithm based on temporal continuity, an index structure to retrieve small files and a prefetching mechanism to improve the performance of file reading and writing.

3. Background Theory

Hierarchical clustering is a widely used data analysis tool. The idea is to build a binary tree of the data that successively merges similar groups of points. This tree provides a useful summary of the data. Hierarchical clustering only requires a measure of similarity between groups of data points. A hierarchical classification can be illustrated in several ways. The result of hierarchical clustering is a tree-based representation of the objects, which is also known as dendrogram. The dendrogram is a multilevel hierarchy where clusters at one level are joined together to form the clusters at the next levels. This makes it possible to decide the level at which to cut the tree for generating suitable groups of a data objects.

In data mining and statistics, hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types: agglomerative and divisive. In this system, the agglomerative hierarchical clustering is focused to merge many small files. A measure of dissimilarity between sets of observations is required in order to decide which clusters should be combined. In most methods of hierarchical clustering, this is achieved by use of an appropriate matric and a linkage criterion which specifies the dissimilarity of sets as a function of the pairwise distances of observations in the sets.

Agglomerative hierarchical clustering is a clustering algorithm that builds a cluster hierarchy from the bottom-up. It starts by adding a cluster for each of the data points to be clustered, followed by iterative pair-wise merging of clusters until only one cluster is left at the top of the hierarchy. The choice of clusters is decided to merge at each iteration base on a distance metric. An agglomerative clustering algorithm is described in the single-linkage

clustering. The single-linkage clustering is the minimum distance between elements of each cluster.

The dissimilarity values between one file and another have to be calculated in advance. In this paper, the Euclidean distance measure is used to cluster the small files. The clustering is based on distance matrix. Only the half of the matrix is needed because the distance between objects is symmetric. The Euclidean distance measure is:

$$d(i,j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \cdots + |x_{ip} - x_{jp}|^2)} \quad (1)$$

This formula defines data objects i and j with a number of dimension equal to p. The distance between the two data objects d(i,j) is expressed as given the above formula x_{ip} is the measurement of object i in dimension p.

4. Proposed System

HDFS has been adopted to support the Internet applications because of its reliable, scalable and low-cost storage capability. It is a file system that supports for cloud storage. However, it would not be able to handle storage and access performance when processing a huge number of small files. In this system, small files will be clustered into a large file to reduce NameNode memory consumption. The small file is a file, whose size is less than 75% of default block size (128MB). The proposed system architecture is shown in figure 1.

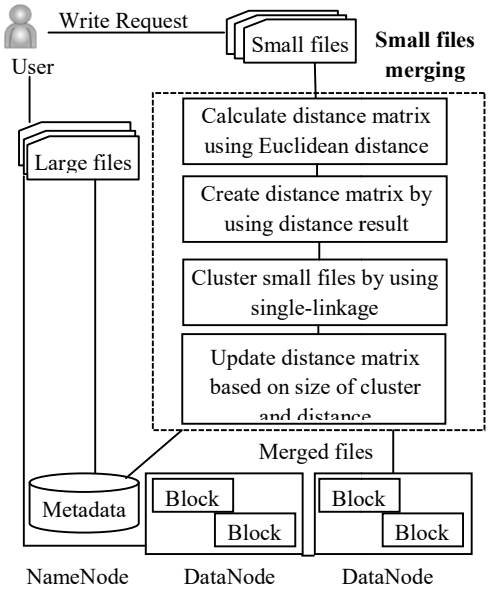

Figure 1. Proposed system architecture

In the proposed system, the files will be ignored to merge if they are already larger than the threshold value. This file is directly operated in the original HDFS. The default threshold for this system is set to (0.75) 75% of default block size (128 MB). If the user accesses the files, the system will check the size of file.

If the file is a small file, the system will put it in the hierarchical structure according to the proposed small files merging algorithm. There are four main objectives to implement the small files merging. Firstly, the distance matrix between small files is calculated by using Euclidean distance. Secondly, the distance matrix is created by using the previous distance results. Next, the small files are clustered by using single-linkage. Finally, the distance matrix is updated depend on the size of cluster and distance.

The small files are nested in larger cluster of files. These larger clusters are joined until its size is less than the default block size. Thus, a small file belongs to many clusters depend on its size and threshold value.

4.1. File Merging Operation

In this system, file merging operation is proposed in Hadoop. The small files are merged into a large file. NameNode only maintains the metadata of merged files and does not save the original small files. File merging reduces the length of the metadata of many small files. The data access and latency can be improved. The proposed algorithm bases on agglomerative hierarchical clustering.

The proposed technique focuses on managing metadata in NameNode. This technique will solve memory consumption on NameNode. The different small files may have different size. So, the proposed algorithm will overcome the issues of concerned with NameNode according to the size of small files. The size of small files is the important factor to merge these files. The size of cluster should less than or equal to default block size. The proposed algorithm is as follow.

Algorithm 1: small files merging Algorithm

Input: Small files S = {F_1, F_2, F_3, …, F_n}
Output: Cluster hierarchies
C= {C_1, C_2, … , C_m}
Each cluster consists of set of small files
Method:
(1) **For** each small file F_i ∈ S **Do**

(2) **For** each small file $F_j \in S$ **Do**
(3) Calculate Euclidean distance matrix between small files $D_e(F_i, F_j)$ by using (1)
(4) End for
(5) End for
(6) Create distance matrix (C, S, D_e)
(7) C= {{F} | F ∈ S}// initial each file cluster
(8) **While** sizeOfEachCluster |C| < 128MB **Do**
(9) {C, C'}= min $D_e(F_i, F_j)$ {C_i , C_j}∈C: $C_i \neq C_j$)//choose cluster by single-linkage
(10) **If** (sizeOfCluster |C| + sizeOfCluster |C'|) <= 128MB **Then**
(11) C=({C } U { C'})//clustering
(12) End if
(13) Update distance matrix (C, S, D_e)
(14) End while
(15) **Return** (C)

4.2 Calculation steps

The sample input of small files is shown in table 1 to trace the different sizes of small files in the proposed small files merging algorithm. The Euclidean distance matrix of small files is shown in table 2. The following are the calculation steps.

Table 1. Sample input size of small files

File	Size(MB)
F1	40
F2	10
F3	50
F4	20
F5	60
F6	30

$d(F1, F2) = \sqrt{(|40 - 10|^2} = 30$
$d(F1, F3) = \sqrt{(|40 - 50|^2} = 10$
$d(F1, F4) = \sqrt{(|40 - 20|^2} = 20$
$d(F1, F5) = \sqrt{(|40 - 60|^2} = 20$
$d(F1, F6) = \sqrt{(|40 - 30|^2} = 10$

$d(F2, F3) = \sqrt{(|10 - 50|^2} = 40$
$d(F2, F4) = \sqrt{(|10 - 20|^2} = 10$
$d(F2, F5) = \sqrt{(|10 - 60|^2} = 50$
$d(F2, F6) = \sqrt{(|10 - 30|^2} = 20$

$d(F3, F4) = \sqrt{(|50 - 20|^2} = 30$
$d(F3, F5) = \sqrt{(|50 - 60|^2} = 10$
$d(F3, F6) = \sqrt{(|50 - 30|^2} = 20$

$d(F4, F5) = \sqrt{(|20 - 60|^2} = 40$
$d(F4, F6) = \sqrt{(|20 - 30|^2} = 10$

$d(F5, F6) = \sqrt{(|60 - 30|^2} = 30$

Table 2. Euclidean distance matrix of six small files (|C|=6)

	F1	F2	F3	F4	F5	F6
F1	**0**	30	10	20	20	10
F2	30	**0**	40	10	50	20
F3	10	40	**0**	20	10	20
F4	20	10	20	**0**	40	10
F5	20	50	10	40	**0**	30
F6	10	10	20	20	30	**0**

The minimize pairs are F1-F3, F2-F4, F3-F5 and F4-F6 at distance 10. Firstly, F1 and F3 are merged into a single cluster among them called "F1-F3". Then the distance from this new compound cluster is computed to all other files.

Size of cluster (F1-F3) = 90
$d(\mathrm{F1-F3}, F2) = \sqrt{(|90 - 10|^2} = 80$
$d(\mathrm{F1-F3}, F4) = \sqrt{(|90 - 20|^2} = 70$
$d(\mathrm{F1-F3}, F5) = \sqrt{(|90 - 60|^2} = 30$
$d(\mathrm{F1-F3}, F6) = \sqrt{(|90 - 30|^2} = 60$

Table 3. After merging F1 with F3 into a new cluster (|C|=5)

	F1-F3	F2	F4	F5	F6
F1-F3	**0**	80	70	30	60
F2	80	**0**	10	50	10
F4	70	10	**0**	40	20
F5	30	50	40	**0**	30
F6	60	10	20	30	**0**

The minimize pairs are F2-F4 and F2-F6 at distance 10. So, F2 and F4 are merged into a new cluster called "F2-F4". Then the distance from this new compound cluster is computed to all other files and clusters to get a new distance matrix.

Size of cluster (F1-F3) = 90
Size of cluster (F2-F4) = 30

$d(\mathrm{F2-F4}, \mathrm{F1-F3}) = \sqrt{(|30 - 90|^2} = 60$

$d(\mathrm{F2-F4, F5}) = \sqrt{(|30-60|^2} = 30$

$d(\mathrm{F2-F4, F6}) = \sqrt{(|30-30|^2} = 0$

Table 4. After merging F1 with F3 and F2 with F4 into a new cluster (|C|=4)

	F1-F3	F2-F4	F5	F6
F1-F3	**0**	60	30	60
F2-F4	60	**0**	30	0
F5	30	30	**0**	30
F6	60	0	30	**0**

Now, the minimize pairs is F2-F4-F6 at distance 0. So, F2-F4 and F6 are merged into a cluster called "F2-F4-F6". Then the distance from this new compound cluster is computed to all other files and clusters to get a new distance matrix.

Size of cluster (F1-F3) = 90

Size of cluster (F2-F4-F6) = 60

$d(\mathrm{F2-F4-F6, F1-F3}) = \sqrt{(|60-90|^2} = 30$

$d(\mathrm{F2-F4-F6, F5}) = \sqrt{(|60-60|^2} = 0$

Table 5. After merging F1 with F3 and F2 -F4 with F6 into a new cluster (|C|=3)

	F1-F3	F2-F4-F6	F5
F1-F3	**0**	30	30
F2-F4-F6	30	**0**	0
F5	30	0	**0**

Now, the minimize pairs is F2-F4-F6-F5 at distance 0. So, F2-F4- F6 and F5 are merged into a cluster called "F2-F4-F6-F5". Then the distance from this new compound cluster is computed to all other files and clusters to get a new distance matrix.

Size of cluster (F1-F3) = 90

Size of cluster (F2-F4-F6-F5) = 120

$d(\mathrm{F2-F4-F6-F5, F1-F3}) = \sqrt{(|120-90|^2} = 30$

Table 6. After merging F1 with F3 and F2 -F4-F6 with F5 into a new cluster (|C|=2)

	F1-F3	F2-F4-F6-F5
F1-F3	**0**	30
F2-F4-F6-F5	30	**0**

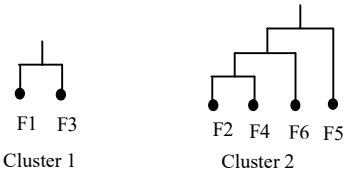

Figure 2. File merging dendrogram of six small files

Finally, the file merging dendrogram of small files is shown in figure 2. According to the proposed algorithm, the sample input from table 1 will have two clusters output.

5. Evaluation

The proposed algorithm will be implemented and tested in simulated cloud environment. In this paper, the strategy of small files merging will reduce the usage of metadata stored on NameNode. The processing time will also be optimized by comparing the original Hadoop and existing approach. Client will achieve better performance on accessing small files into HDFS.

To evaluate the small file merging algorithm, Web Yahoo Web Scope dataset [10] is used. This dataset consists of Language Data, Graphic and Social Data, Rating and Classification Data, Competition Data, Advertising and Market Data and Computing System Data. Among them, small files sizes are used to trace the proposed method.

In the proposed approach, file size having greater than threshold which is ignored to merge. In the evaluation, the main memory usage is measured by different file ranges. The metadata of each file consumes 150 bytes of the memory of Namenode.

Firstly, there are two clusters for 6 files. The size of these files range from 10MB to 60MB. The size of cluster 1 and cluster 2 are 90MB and 120MB respectively. So, the memory of NameNode is only needed 300 bytes to store the metadata of two clusters. The original HDFS takes 900 bytes to store the metadata of 6 small files into NameNode.

Secondly, there are three clusters for 8 files. The size of these files range from 14MB to 81MB. The size of cluster 1, cluster 2 and cluster 3 are 97MB, 90MB and 81MB respectively. So, the memory of NameNode is only needed 450 bytes to store the metadata of three clusters. The original HDFS takes 1200 bytes to store the metadata of 8 small files into NameNode.

Finally, there is only one cluster for 11 files. The size of these files range from 14KB to 14MB. The size of cluster is 50.857MB. So, the memory of NameNode is only needed 150 bytes. The original HDFS takes 1650 bytes to store the metadata of 11 small files into NameNode. If the numbers of small files increase, the memory of NameNode usage increases. It is depicted in figure 3.

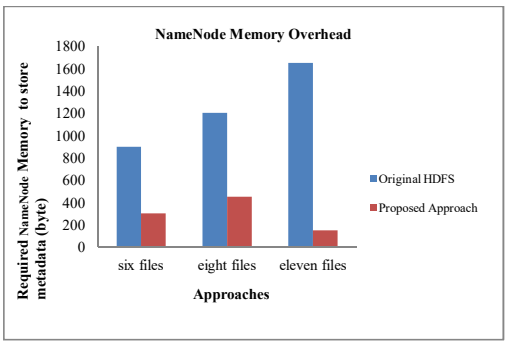

Figure 3. NameNode Memory Consumption of different file sizes

6. Conclusion

In this paper, the small file problem of HDFS is focused. By eliminating the number of small files into a large file by the proposed agglomerative hierarchical merging approach, the memory consumption of NameNode in HDFS is reduced. The performance of the proposed system can be better than the original HDFS. Furthermore, this paper provides a new way of solving for Hadoop in storage of massive small files. As a future work, many experiments have to be done in order to get the efficiency of proposed algorithm on HDFS. Moreover, a prefetching mechanism will be considered as a future work to retrieve small files from the merged large file when the clients access these small files.

References

[1] B. Meng, W.B. Guo, G.S. Fan and N.W. Qian, "A novel approach for efficient accessing of small files in HDFS: TLB-MapFile". In Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on (pp. 681-686).

[2] Z. Gao, Y. Qin and K. Niu, "An effective merge strategy based hierarchy for improving small file problem on HDFS". In Cloud Computing and Intelligence Systems (CCIS), 2016 4th International Conference on (pp. 327-331).

[3] P.M. ElKafrawy, A.M. Sauber, and M.M. Hafez, "HDFSX: Big data Distributed File System with small files support". In Computer Engineering Conference (ICENCO), 2016 12th International (pp. 131-135).

[4] T. Wang, S. Yao, Z. Xu, L. Xiong, X. Gu and X. Yang, "An effective strategy for improving small file problem in distributed file system". In Information Science and Control Engineering (ICISCE), 2015 2nd International Conference on (pp. 122-126).

[5] P. Gohil, B. Panchal, and J.S. Dhobi, "A novel approach to improve the performance of Hadoop in handling of small files". In Electrical, Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference on (pp. 1-5).

[6] T. Gupta, and S.S. Handa, "An Extended HDFS with an AVATAR NODE to handle both small files and to eliminate single point of failure". In Soft Computing Techniques and Implementations (ICSCTI), 2015 International Conference on (pp. 67-71).

[7] K. Bok, J. Lim, H. Oh, and J. Yoo, "An efficient cache management scheme for accessing small files in Distributed File Systems". In Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on (pp. 151-155).

[8] Y. Huo, Z. Wang, X. Zeng, Y. Yang, W. Li and C. Zhong, "SFS: A massive small file processing middleware in Hadoop". In Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific (pp. 1-4).

[9] G. Prasad, "An Efficient Approach to Optimize the Performance of Massive Small Files in Hadoop MapReduce Framework", 2017.

[10] https://webscope.sanbox.yahoo.com.