IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 19, 2018, accepted January 15, 2019, date of publication January 21, 2019, date of current version February 8, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2893882

A Small File Merging Strategy for Spatiotemporal Data in Smart Health

LIAN XIONG[1], YUANCHANG ZHONG[2], XIAOJUN LIU[3], AND LIU YANG[1]

[1]School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[2]College of Communication Engineering, Chongqing University, Chongqing 400044, China
[3]School of Transportation, Huanggang Normal University, Huanggang 438000, China

Corresponding authors: Lian Xiong (xionglian@cqupt.edu.cn) and Yuanchang Zhong (zyc@cqu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61801072, in part by the Chongqing Science and Technology Commission under Grant cstc2018jcyjAX0344, and in part by the Natural Science Fund of Hubei Province under Grant 2018CFB597.

ABSTRACT With the rapid development of smart health, the health sensors and wearable devices bring huge amounts of small files of spatiotemporal data, which are distributed in different servers, and affect the I/O performance of the system seriously. There are many methods to solve the problems of a small file, but most of them are applied to specific applications. However, due to the influence of user access behavior and data type, these methods are not effective when applied to sensors data in smart health. In this paper, a novel small file merging strategy for smart health is proposed. By analyzing the features of health sensors data and the preferences of user access, the strategy uses spatiotemporal clustering for the historical user access information. Then, weights are applied to these clusters based on the access density to determine the access-related spatiotemporal range. Finally, the spatiotemporal range is used to merge small files. The experimental results show that the merging strategy is simple but efficient, and it can effectively reduce user access delay for small files of spatiotemporal data in smart health.

INDEX TERMS Smart health, small file merging, spatiotemporal data, spatiotemporal clustering.

I. INTRODUCTION

Among recent advancements in technology, cloud computing, the internet of things and wearables are widely applied in a smart health. As a result, massive spatiotemporal data with three basic attributes, location, time, and type, will be produced, and distributed stored on different servers, such as health sensors data, inspection information system (LIS) data, medical image data, electronic medical/health record (EMR/EHR) data. Such data are usually characterized by small size, wide variety, large amount, high redundancy, and dynamic growth over time, and are typically small files of spatiotemporal data [1], [2].

Research shows that in application services containing large amounts of small files, users' requests for access to small files account for more than 90% of the total number of requests, but the data they request is less than 10% of the total amount of data. The I/O performance of the system is severely restricted by the large number of small files [3].

However, nowadays the mainstream of distributed file system (Figure 1), such as Hadoop Distributed File System (HDFS), are suitable for large files in metadata management, data layout, cache management and other

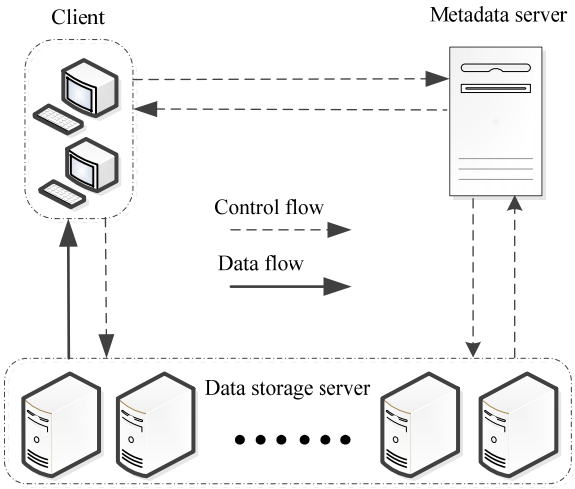


FIGURE 1. Distributed file system and access mechanism.

implementation strategies. All of them are focused on storage and access to a large amount of data stream, improving system throughput rather than response time. There are three main reasons for this phenomenon. (1) High memory usage

in metadata server (MDS): each small file in MDS is represented as an object and stored in memory, large amounts of small files occupy MDS memory. (2) Frequent Client-MDS communication: when Client read or write each small file, it must to establish a communication connection with MDS to acquire metadata information, which consumes the limited communication bandwidth of the system. (3) Cross node transmission of files: when an application needs to access a large number of small files, it normally causes lots of seeks, and lots of hopping from data storage server (DSS) to DSS to retrieve each small file.

Scholars have put forward many solutions to the problem of small files. Among them, small file merging, as a micro data layout mechanism, is widely used because of its high flexibility and excellent performance. But most of the existing merging strategies are targeted at particular application scenario and data, such as tile data in WebGIS, PPT files in educational websites, electronic health records (EHR) in healthcare, etc. However, due to the impact of user access preferences and data characteristics, these methods are not effective when applied to small files of health sensors data in smart health. In fact, there is no general method of merging, the merge strategy must match the application to improve the system performance [4].

In this paper, we propose a merging strategy for small files of spatiotemporal data in smart health based on spatiotemporal related of user access. The merge strategy first parameterizes the user access information, extracts the spatiotemporal attributes, and uses spatiotemporal clustering algorithm for each attribute to form a cluster. Then, calculate the access related spatiotemporal range of each cluster based on the density of the attribute point within the cluster. Finally, the access related spatiotemporal range is used to merge small files.

The rest of this paper is organized as follows: Section 2 presents the current research on the issue of handling small files. Section 3 introduces the calculation of the access related spatiotemporal range and its use in merging small files. Section 4 presents and discusses the performance evaluation results of our merging strategy. Finally, Section 5 briefly summarizes our findings and concludes the paper.

II. RELATED WORKS

At present, there are few studies on the storage of small files of sensor data in smart health, but many research results in other application and other types of small files. This research can be divided into two categories: one is changing the storage and management mechanism of small files in the system, the other is to research the merging strategy of small files.

A. CHANGING THE STORAGE AND MANAGEMENT MECHANISM

Ma *et al.* [5] proposed a novel distributed file system built over distributed tabular storage, HVFS, which uses extendible hash to index metadata, log-structured storage format and columnar storage, and could manage billions of small files and support highly concurrent accesses. However, this method needs to change the files storage format and the index mode, and it cannot run on the existing distributed file system. Zhang *et al.* [6], [7] proposed a distributed cloud storage system for small files based on P2P, where a central route node is introduced to improve the resource query efficiency, and clients can also cache the routing information. Obviously, when the system has a large number of small files and high concurrent user access, the central node is the system's performance bottleneck. Fu *et al.* [8], [9] abandoned the hierarchical file management model of the traditional file system and designed a flat lightweight file system called FlatLFS to improve the performance of the whole system. The flat storage architecture can effectively improve the continuous reading efficiency of files, but it is weak for file retrieval or random reading.

Considering the layout and access features of small files in a storage system, Zhao *et al.* [10] attempted to store in the logical continuous space of physical disks as far as possible, they use a cache to act as MDS and improved the utilization rate of cache by using simplified file information nodes. However, the capacity of the cache limits the number of small files stored in the system. Elkafrawy *et al.* [11] present a new structure for HDFS (HDFSX) to avoid higher memory usage, flooding network, requests overhead and centralized point of failure. This method can reduce the load of MDS, but it did not consider the access related between files. Therefore, applying this method to different application, the effect will be very different. Bok *et al.* [12] optimized the access efficiency of small files in a distributed file system from the perspective of cache management. Their method can reduce the volume of metadata to manage in the MDS by combining and storing multiple small files in a block, and reduce unnecessary accesses by keeping the requested files using clients and the caches of data nodes. However, this method does not take into account the impact of user access behavior on cache performance.

Cheng *et al.* [13] increased the I/O efficiency of small files in the system by establishing a small file index and using distributed cache. This method can effectively improve the reading performance of files, but the construction cost and complexity of the system are increased. In order to solve the problem of performance degradation of search engines owing to large-scale small file storage in original pages, Zhang *et al.* [14] improved the file compression algorithm in the EXT3 file system; Meanwhile, they design an original page oriented file organization structure and a read-write query tree to store the large-scale small files which need no modification. Their method can effectively reduce the search response time and disk space waste, but it cannot improve the efficiency of small file access.

In general, changing the storage and management mechanism can solve small file problems effectively, but it needs to rebuild the file system or add additional cache nodes on existing file system, which increases the construction cost and complexity of the system.

B. SMALL FILE MERGING

Yu *et al.* [15] took into account the reading time, merge time and memory occupancy of small files, and used multi-attribute decision theory to merge small files into large files by using a sequence file technology. However, their method did not consider the access related between files, so they can only merge the files which user have been accessed. Wang *et al.* [16] analyzed the relationship among the access tasks, application, and access files, and proposed a file merging and prefetching strategy based on the improved probabilistic latent semantic analysis (PLSA) model for user access tasks. This method is also unable to merge files that users have not visited. Xiong *et al.* [17] proposed a merging strategy for small files of spatiotemporal data in smart city based on the user access rules and spatiotemporal attributes. This method can effectively merge all small files by mining user access rules to build feature templates, but it only considers the logical state of spatiotemporal attributes when mining user access rules. Therefore, the feature template cannot reflect all user access rules, and reduce the merging performance. Liu *et al.* [18] combined the application characteristics and user access features in WebGIS, merging small files of neighboring geographic locations (MNGL), and build index for each file. Due to the particularity of GIS data, this method does not apply to other types of data.

Dong *et al.* [19], [20] take into account the relevance and locality of access between PPT files, merged files of the same courseware into a large file, and implemented two level prefetching when reading files. Because of the inherent access dependencies between PPT files belonging to the same courseware, this approach is not very applicable in other scenarios. Zhang and Rui [21] considered the related between small files and the directory structure of data, merging small files, and generated a hierarchical index. This method can effectively improve the search and sequential read performance of small files, but cannot meet other forms of access requirements. Gao *et al.* [22] propose the small file merge strategy based on logic file name (SMSBL) to enhance small file access performance. SMSBL improves the related of small files in the same block of HDFS effectively based different file system hierarchy. However, SMSBL does not take into account the access related between files, therefore, applying this method to different scenarios, the result is quite different.

To solve the defect of storage of small files in healthcare, He *et al.* [23] proposed a method for merging of small files based on balance of data block, called Tetris Merge algorithm (TM). The TM will optimize the volume distribution of the big file after merging, and effectively reduce the data blocks of HDFS. Though TM can reduce the memory overhead of major nodes of cluster and reduce load to achieve high-efficiency operation of data processing, but the efficiency of reading files is still not good enough. Dang *et al.* [24] proposed a novel Hash-Based File Clustering Scheme (HBFC) to distribute store and retrieve electronic health records (EHR) efficiently in cloud environments. The HBFC utilizes hashing to distribute files into clusters in a control way and it utilizes P2P structures for data management. Experimental results show that HBFC scheme is effective in handling big health data that comprises of a large number of small files in various formats, and user can retrieve and access data records efficiently. However, this method cannot be applied to distributed file systems with master-slave structure, such as HDFS.

From the above studies, we can observe that compared with changing the storage and management mechanism, small file merging is simple and efficient. However, it is not very well to apply existing methods to small files of sensor data in smart health directly, because most of them are targeted at specific application and different data types, it can only merge the files which user has been accessed, or simply use spatiotemporal adjacent merging. But in smart health, the health sensor data is typical spatiotemporal data and user access has obvious spatiotemporal related.

III. SMALL FILE MERGING STRATEGY

The merging strategy mainly consists of three parts: The first part is to explain the basic principles of small file merging algorithm, the second part is calculate the access related spatiotemporal range, and the third part is use the spatiotemporal range to merge small files.

A. PRINCIPLE

In a smart city, users access small files of spatiotemporal data through system predefined application services. Therefore, compared with the normal small files, the users accesses have obvious spatiotemporal locality and related. However, the existing merge algorithm does not take into account this spatiotemporal characteristic, so the merging is not very efficient.

Obviously, if some small files belonging to a certain spatiotemporal range are frequently accessed by users, these small files have an access related. Inspired by this, we try to mine access related small files from historical user access information by using spatiotemporal cluster, calculate the access related spatiotemporal range, and finally, use the access related spatiotemporal range to merge small files.

In addition, considering the impact of user access behavior and preferences, the access related spatiotemporal ranges of different types of attributes of small files are also different. Therefore, we classify each type attribute of small files based on the historical user access information, and calculate their access related spatiotemporal range for each attribute type. The proposed merging algorithm in this paper includes four steps:

Step 1: Separate the access request sequence for the type attribute of small files from the historical user access information.

Step 2: Using spatiotemporal cluster to cluster the small file access request information which contain the same type attribute, to obtain the access related spatiotemporal range.

Step 3: Merge small files of the same type attribute by using the access related spatiotemporal range.

Step 4: Loop steps (1-3) until all attribute types of small files are merged.

B. ACCESS RELATED SPATIOTEMPORAL RANGE

Assume that the set of small files of spatiotemporal data in a smart city is $F = \{f_1, f_2, \cdots, f_n\}$. The request sequence of historical user access of small files in a smart city is $A = (a_1, a_2, \cdots a_n)$, where, each request a_i, $1 \leq i \leq n$ corresponds to a small file of spatiotemporal data f_i, $1 \leq i \leq n$.

According to the definition of small files of spatiotemporal data, every small file contains location attribute l, time attribute t and type attribute s. Therefore, any small file can be represented by its three basic attributes (l, s, t). In order to mine the access related spatiotemporal range from the user's access request sequences $A = (a_1, a_2, \cdots a_n)$, we parameterized the representation and spatiotemporal attribute extraction for A to obtain the spatiotemporal attribute sequences,

$$\begin{aligned} A &= (a_1, a_2, \cdots a_n) \\ &= ((l_1, s_1, t_1), (l_2, s_2, t_2), \cdots, (l_n, s_n, t_n)) \end{aligned} \quad (1)$$

Then, the access request sequence which contains the type attribute s_i is,

$$\begin{aligned} A^{s_i} &= \left(a_1^{s_i}, a_2^{s_i}, \cdots, a_m^{s_i}\right) \\ &= \left(\left(l_1^{s_i}, s_1^{s_i}, t_1^{s_i}\right), \left(l_2^{s_i}, s_2^{s_i}, t_2^{s_i}\right), \cdots, \left(l_m^{s_i}, s_m^{s_i}, t_m^{s_i}\right)\right) \end{aligned} \quad (2)$$

1) LOCATION ATTRIBUTE

The essence of the spatiotemporal range of the location attribute is to calculate the geographic space range occupied by access related small files. Therefore, we used the spatiotemporal cluster algorithm to cluster the location attributes contained in access request sequence A^{s_i}. Then, according to the density of the location-attribute points in each cluster, the weighted average calculation of the spatial scope of the clusters was conducted to obtain the average cluster radius, which is the spatiotemporal range of the location attribute.

Agglomerative NESting (AGNES) is a agglomerative hierarchical cluster algorithm for large-scale data sets [25], In the clustering process, AGNES assumes each object as a cluster and merges similar clusters. The cluster-merging process is repeated until all the objects are eventually merged into a cluster. Figure 2 shows the clustering schematic of data object $\{a, b, c, b, d, e\}$.

Assume that the location attribute of small files in a smart city is represented by two-dimensional latitude and longitude coordinates (x, y), the set of location attributes contained in access-request sequence A^{s_i} is,

$$\begin{aligned} L^{s_i} &= \left\{l_1^{s_i}, l_2^{s_i}, \cdots, l_m^{s_i}\right\} \\ &= \left\{\left(x_1^{s_i}, y_1^{s_i}\right), \left(x_2^{s_i}, y_2^{s_i}\right), \cdots, \left(x_m^{s_i}, y_m^{s_i}\right)\right\} \end{aligned} \quad (3)$$

The goal of clustering is to enlarge and compress the similarities of the same and different cluster objects, respectively,

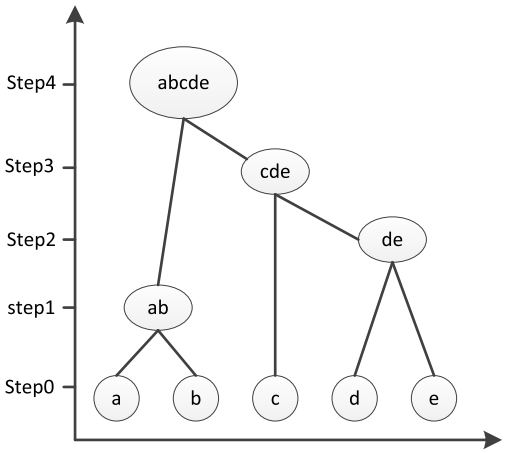


FIGURE 2. Schematic of clustering.

as much as possible. Therefore, a key problem of clustering is how to measure the similarity between two clusters. In this study, the group average connectivity was used to define the similarity between two clusters by calculating the average distance. The closer the average distance, the higher is the cluster similarity.

Suppose cluster C_m contains a set of location-attribute points $C_m = (l_1, l_2, l_3, \cdots)$, cluster C_n contains a set of location-attribute points $C_n = \left(l'_1, l'_2, l'_3, \cdots\right)$, and the elements between the two clusters do not intersect. Then, the distance of any two location-attribute points $l = (x, y)$ and $l' = (x', y')$ is,

$$d\left(l, l'\right) = \sqrt{(x - x')^2 + (y - y')^2} \quad (4)$$

The average distance of group average connectivity between clusters C_m and C_n is,

$$d_{avg}\left(C_m, C_n\right) = \frac{1}{N_m N_n} \sum_{\in C_m} \sum_{l' \in C_n} d\left(l, l'\right) \quad (5)$$

where $N_m = card\left(C_m\right)$ represents the number of location-attribute points in cluster C_m, and $N_n = card\left(C_n\right)$ represents the number of location-attribute point in cluster C_n.

According to the principle of the AGNES algorithm, if the number of clusters is not specified in advance, the AGENS will merge all the clusters until the whole object is merged into a cluster. Obviously, if the clustering algorithm merges all the clusters, it will not be able to find the access-related spatiotemporal range of the location attribute from the original location attribute coordinates. Therefore, we set average distance $\lambda_l^{s_i}$ between all coordinate point in L^{s_i} as the termination condition of clustering,

$$\lambda_l^{s_i} = \frac{2}{N_l^{s_i}\left(N_l^{s_i} - 1\right)} \sum_{l, l' \in L^{s_i}} d\left(l, l'\right) \quad (6)$$

where location-attribute points $l, l' \in L^{s_i}$, and $N_l^{s_i} = card\left(L^{s_i}\right)$ represents the number of location-attribute points in the L^{s_i}. If the average distance between clusters $C_m^{s_i}$ and $C_n^{s_i}$ is $d_{avg}\left(C_m^{s_i}, C_n^{s_i}\right) \geq \lambda_l^{s_i}$, they cannot be merged into a cluster.

When all the clusters cannot be merged, the clustering process ends.

Here, we describe how to use the AGNES algorithm to cluster location-attribute sequence L^{s_i} of access-request sequence A^{s_i}; the aim is to obtain the access-related spatiotemporal range of the location attribute of the small file with type attribute s_i. The clustering process is as follows:

Step 1: Consider every location-attribute point of location-attribute sequence L^{s_i} as a cluster.

Step 2: Calculate the average distance between each cluster, and merge the two nearest clusters. If the average distance between clusters $C_m^{s_i}$ and $C_n^{s_i}$ is the least, the clusters are merged into new cluster $C_k^{s_i} = C_m^{s_i} \cup C_n^{s_i}$.

Step 3: Loop step (2) until the average distance between the two clusters is larger than predefined distance threshold $\lambda_l^{s_i}$; finally, the clustering algorithm ends.

Assuming that after the clustering, the cluster set is $C_L^{s_i} = \left\{C_1^{s_i}, C_2^{s_i}, \cdots, C_{K_l}^{s_i}\right\}$. Then, for clusters $C_k^{s_i}, 1 \le k \le K_l$, the radius of the space range is,

$$R_l\left(C_k^{s_i}\right) = \frac{2}{N_m\left(N_m - 1\right)} \sum_{l \in C_k^{s_i}} \sum_{l' \in C_k^{s_i}} d\left(l, l'\right) \tag{7}$$

Next, this set of clusters is used to calculate their average space range. Obviously, a very dense location-attribute point in a cluster implies that this space is a hot area visited by users. Therefore, we further weighted the radius of the space range for each cluster according to the user-access heat, that is, the greater the density of the coordinate point in each cluster, the greater is the value. Then, for clusters $C_k^{s_i}, 1 \le k \le K_l$, the radius of the space range weighted by the access density is,

$$\bar{R}_l\left(C_k^{s_i}\right) = \frac{M_k^{s_i}}{N_l^{s_i}} R_l\left(C_k^{s_i}\right) \tag{8}$$

where $M_k^{s_i} = card\left(C_k^{s_i}\right)$ represents the number of location-attribute points in cluster $C_k^{s_i}$, $N_l^{s_i} = card\left(L^{s_i}\right)$ represents the number of location-attribute point in location-attribute sequence L^{s_i}, and the $R_l\left(C_k^{s_i}\right)$ represents the radius of the space range of cluster $C_k^{s_i}$. Finally, the average radius of the weighted space ranges of all clusters in set $C_L^{s_i}$ was calculated, and the location-attribute merging range of the small file with type attribute s_i is,

$$R_L^{s_i} = \frac{1}{K_l} \sum_{k=1}^{K_l} \bar{R}_l\left(C_k^{s_i}\right) \tag{9}$$

2) TIME ATTRIBUTE

The time attribute spatiotemporal range is used to calculate the time interval occupied by small access related files. Therefore, the method is the same as computing the location attribute spatiotemporal range. The spatiotemporal clustering algorithm is used to cluster the time attributes contained in access request sequence A^{s_i}. Then, according to the density of the time attribute points in each cluster, the weighted average of the time interval of the clusters is calculated to obtain the average cluster radius, which is the time attribute spatiotemporal range.

Assume that the set of time attributes contained in request sequence A^{s_i} is,

$$T^{s_i} = \left\{t_1^{s_i}, t_2^{s_i}, \cdots, t_m^{s_i}\right\} \tag{10}$$

The first step of clustering is to define the similarity between clusters, where each cluster is a set of time attributes. We define the similarity between clusters by calculating the average time interval. The closer the average time interval, the higher the cluster similarity.

Suppose cluster C_m contains a set of time attribute points, $C_m = (l_1, l_2, l_3, \cdots)$, cluster C_n contains a set of time attribute points, $C_n = \left(l_1', l_2', l_3', \cdots\right)$, and the elements of the two clusters do not intersect. Then, the time interval between any time attribute points, $t, t' \in T^{s_i}$, is,

$$d\left(t, t'\right) = \left|t - t'\right| \tag{11}$$

The average time interval between clusters C_m and C_n is,

$$d_{avg}\left(C_m, C_n\right) = \frac{1}{N_m N_n} \sum_{t \in C_m} \sum_{t' \in C_n} d\left(t, t'\right) \tag{12}$$

where $N_m = card\left(C_m\right)$ and $N_n = card\left(C_n\right)$ represent the number of time attribute points in clusters C_m and C_n, respectively.

As same as the cluster location-attribute sequence L^{s_i}, we must set a termination condition for the AGNES algorithm to cluster time-attribute sequence T^{s_i}. We set the average time interval $\lambda_t^{s_i}$, between all coordinate points in T^{s_i} as the termination condition of clustering,

$$\lambda_t^{s_i} = \frac{2}{N_t^{s_i}\left(N_t^{s_i} - 1\right)} \sum_{t, t' \in T^{s_i}} d\left(t, t'\right) \tag{13}$$

where time attribute points $t, t' \in T^{s_i}$, and $N_t^{s_i} = card\left(T^{s_i}\right)$ represents the number of time attribute points in T^{s_i}. Clearly, if the average time interval between clusters $C_m^{s_i}$ and $C_n^{s_i}$ is $d_{avg}\left(C_m^{s_i}, C_n^{s_i}\right) \ge \lambda_t^{s_i}$, they cannot be merged. The clustering process ends when all clusters cannot be merged.

The clustering process of a time attribute sequence is the same as location attribute clustering. Assuming that after clustering, the cluster set is $C_T^{s_i} = \left\{C_1^{s_i}, C_2^{s_i}, \cdots, C_{K_t}^{s_i}\right\}$. Then, for cluster $C_k^{s_i}, 1 \le k \le K_l$, the time interval of the time attribute is,

$$R_t\left(C_k^{s_i}\right) = \frac{2}{N_m\left(N_m - 1\right)} \sum_{t \in C_k^{s_i}} \sum_{t' \in C_k^{s_i}} d\left(t, t'\right) \tag{14}$$

The time interval for each cluster is weighted according to the density of the time attribute point, and the value increases with density. For cluster $C_k^{s_i}, 1 \le k \le K_l$, the time interval weighted by access density is,

$$\bar{R}_t\left(C_k^{s_i}\right) = \frac{M_k^{s_i}}{N_t^{s_i}} R_t\left(C_k^{s_i}\right) \tag{15}$$

where $M_k^{s_i} = card\left(C_k^{s_i}\right)$ represents the number of time attribute points in cluster $C_k^{s_i}, N_l^{s_i} = card\left(L^{s_i}\right)$ represents the

number of time attribute point in time attribute sequence L^{s_i}, and $R_l\left(C_k^{s_i}\right)$ represents the radius of the time interval of cluster $R_l\left(C_k^{s_i}\right)$. Finally, the average weighted time interval of all clusters in set $C_T^{s_i}$ is calculated, and the time attribute merging range of a small file with time attribute s_i is,

$$R_T^{s_i} = \frac{1}{K_t}\sum_{k=1}^{K_t}\bar{R}_t\left(C_k^{s_i}\right) \tag{16}$$

C. SMALL FILES MERGING

Assume that the set of small spatiotemporal data files have the type attribute s_i, set of location attributes $L^{s_i} = \left\{l_1^{s_i}, l_2^{s_i}, \cdots, l_m^{s_i}\right\}$, and set of the time attributes $T^{s_i} = \left\{t_1^{s_i}, t_2^{s_i}, \cdots, t_m^{s_i}\right\}$. Then, we use the spatiotemporal range $R_L^{s_i}$ and $R_T^{s_i}$ to merge the small files with the type attribute s_i.

The principle of merging is based on the geographical area. That is, the small files that belong to the same location attribute space are merged first, subsequently, the small files of the next location attribute space range are merged. The steps for this process are as follows:

Step 1: Create a large file.

Step 2: Set the most advanced (earliest) time attribute t_u in the set of time attribute T^{s_i} as the reference point. Then, find the time attribute point whose time interval is less than or equal to $R_L^{s_i}$ and form a set $Range\_t_u$.

Step 3: Set any location attribute l_v in the set of location attributes L^{s_i} as the reference point. Then, find the location attribute point whose distance is less than or equal to the $R_L^{s_i}$ and form a set $Range\_l_v$.

Step 4: Merge the small files into the large file with time attribute in $Range\_t_u$ and location attribute in $Range\_l_v$.

Step 5: If the total size of the merged file is larger than the predefined large file, jump to step (6), else, jump to the step (7).

Step 6: Delete the time attribute involved in the merge $T^{s_i} = T^{s_i} - Range\_t_u$, and sequentially repeat till the file size of the merged is less than the larger file.

Step 7: Delete the time attribute been involved in the merge $T^{s_i} = T^{s_i} - Range\_t_u$. Loop steps (2-5) until all the small files in the location attribute set $Range\_l_v$ and the time attribute set T^{s_i} are merged into the large file.

Step 8: If the total size of the merged file is still smaller than the predefined large file, then delete the location attributes $L^{s_i} = L^{s_i} - Range\_l_v$, and reset the set of time attributes T^{s_i}. Execute the steps (2-5) once more.

Step 9: Loop steps (1-8) until all small files are merged with type attribute s_i.

Based on the above merging steps, we can continue to calculate the access related spatiotemporal range of other types of attributes, and merge small files of the current attribute type. Thus, we can merge small files of spatiotemporal data of all types of attributes in a smart city.

IV. EXPERIMENTAL

In this section, we will first introduce the performance evaluation metrics for our merging algorithm. Subsequently, the experimental environment and data will be described. Finally, we will present and discuss the results of the experiments.

A. EVALUATION METRICS

Three indexes were used in the experiment to evaluate the performance of the merge algorithm: MDS memory usage, total average write time, and total average response time. These indexes are defined as follows:

- MDS memory usage: the memory consumption of MDS when storing a small file set.
- Total average write time: the total average write time for user write a small file to HDFS through the Client.
- Total average response time: the total average response time for user read a small file to HDFS through the Client.

B. EXPERIMENTAL ENVIRONMENT AND DATA

The experimental data was obtained from the Wuhan smart city network application demonstration platform, which includes 14 types of sensors located in different regions. Each data type of the sensor varies in size between 3.2 KB and 5.8 KB. The sensors have been collecting data since 1 January 2010, and currently provides 20 types of predefined applications to the public. The data storage system is HDFS with 5 nodes. Each node has memory of 2 GB and the data block size is the default 64 MB. The sensors are connected through a 1000 M Ethernet switch.

In order to get the historical user access information, we obtained the user access logs in the server between 1 October 2017 and 1 June 2018. After processing the logs, we generated 9,763,286 file access requests. The size of these small files was approximately 39.62 GB. The proposed algorithm is compared with the original HDFS, the improved sequence file merging strategy in [15], the merging small files of neighboring geographic locations (MNGL) in [18], and the small file merge in healthcare based on balance of data block, the Tetris Merge algorithm (TM) in [23].

C. EXPERIMENTAL RESULTS

1) MDS MEMORY USAGE

This experiment tests the memory usage in MDS when the system stores 5000, 10000, 15000, 20000, 25000, and 30000 small files. The results are shown in Figure 3.

It can be observed that the memory usage of the MDS is approximately 4.2 MB without storing any files, and as the number of files increase, the memory usage increase linearly. This is because in HDFS, both small and large files are represented as an object, store in MDS and occupies 150 bytes of memory space. The more files the system stores, the more memory usage in MDS. In addition, the original HDFS memory usage is largest, because there is no small file merge operation, so the system has the largest number of small files. Sequence file, MNGL, TM and our algorithm have merged small file, multiple small files are merged into one large file, so memory usage is very small.

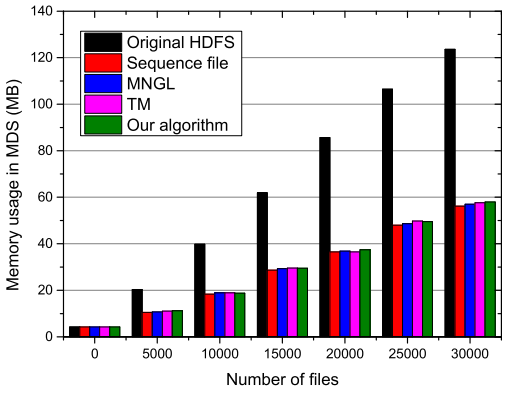


FIGURE 3. Comparison of memory usage in MDS.

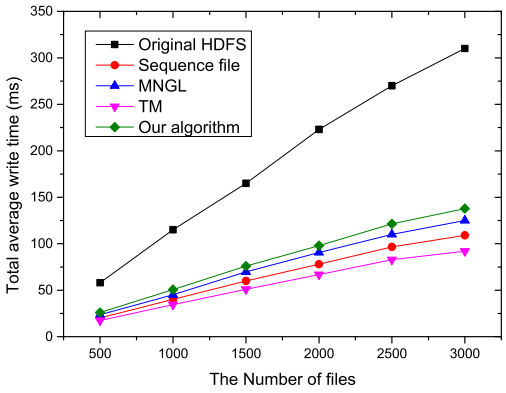


FIGURE 4. Comparison of average write time.

2) AVERAGE WRITE TIME

The experiment tests the total average write time when 500-3000 small files are written by Client, the results is shown in Figure 4.

It can be observed that the total average write time increased with the number of files increased. Simultaneously, because the Client must establish a communication connection with MDS to apply for storage space and create metadata information when writing every small file to HDFS, so the original HDFS has the longest time. But in other three methods, small files are merged into large files by Client at first. Then, a communication will be established with MDS to apply for storage space and create metadata information. Therefore, the number of communication with Client and MDS can be reduced greatly. In addition, TM has the shortest time, because it only considers file size when merging small files. Sequence file technology needs to convert the original small file into Sequence file, and then uses the key-value format for merging storage. MNGL and Our algorithm need to consider the file size, the spatiotemporal attributes and access related, then used spatiotemporal range is used to merge them into large files. Hence, the writing performance is not as good as TM and Sequence file.

3) AVERAGE RESPONSE TIME

The purpose of merging files is to reduce the user access delay. The experiment tests the total average response time of

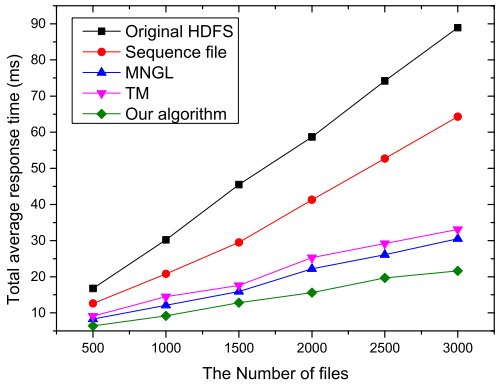


FIGURE 5. Comparison of average response time.

the system when 500-3000 small files are accessed by Client, and the result is shown in Figure 5.

It can be observed that the total average response time increased with the number of files increased. Simultaneously, because the Client must establish a communication connection with MDS to obtain metadata information when access every small file form HDFS, so the original HDFS has the longest time. And by merging small files, the Sequence file and TM technology can reduce the number of communicate between Client and MDS to a certain extent. However, neither of them considers the access related between files, so when users access an application which containing a large number of small files, they usually need to hop in different DSS to get each small file. The merging strategy of MNGL and our algorithm take into account the access related among files. On the one hand, it reduces the number of communication connection between Client and MDS. On the other hand, it solves the problem of hopping DSS. However, the MNGL only considers a simple mode of the user's continuous access to the adjacent geographic location. Therefore, for those small files that are not geographically adjacent but still access related. The problem of frequent hopping in different DSS still exists, which increases average response time for user access.

V. CONCLUSIONS AND FUTURE WORK

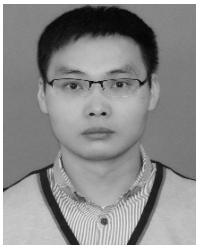
Merging can effectively solve the problem of large number of small files, but the effects of different merging strategy are different due to user access behavior. There is no merge strategy which is fit to all application scenarios and data type. In this paper, we proposed a merging strategy for small spatiotemporal data files in smart health. This method takes advantage of the spatiotemporal locality and related of user access, can effectively improve the efficiency of file reading and reduce user access delay.

In our future works, we will focus on optimizing the calculation of clustering, and dynamic incremental updating the access related spatiotemporal range, to merge small files more efficiently.

REFERENCES

[1] D. R. Li, J. J. Cao, and Y. Yuan, "Big data in smart city," *Sci. China Inf. Sci.*, vol. 58, no. 10, p. 108101, Oct. 2015.

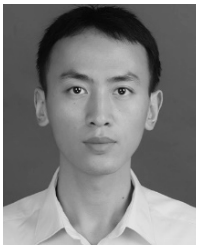
[2] L. Xiong, Z. Xu, H. Wang, S. Jia, and L. Zhu, "Prefetching scheme for massive spatiotemporal data in a smart city," *Int. J. Distrib. Sensor Netw.*, vol. 2, no. 1, pp. 1–12, Jan. 2016.
[3] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, and D. E. Long, "File system workload analysis for large scale scientific computing applications," in *Proc. IEEE/12th NASA Goddard Conf. Mass Storage Syst. Technol.*, Adelphi, MD, USA, Apr. 2004, pp. 1–6.
[4] B. S. Kumar, P. K. Ranjitham, K. R. Karthekk, and J. Gokila, "Survey on various small file handling strategies on Hadoop," in *Proc. Int. Conf. Commun. Electron. Syst.*, Coimbatore, India, Oct. 2016, pp. 1–4.
[5] C. Ma, D. Meng, and J. Xiong, "Dawning nebula distributed file system HVFS: For large scale small file access," *J. Chin. Comput. Syst.*, vol. 33, no. 7, pp. 1481–1482, Jul. 2012.
[6] Q.-F. Zhang, W.-D. Zhang, W.-J. Li, X.-Z. Pan, and Y. Shen, "Cloud storage system for small file based on P2P," *J. Zhejiang Univ. (Eng. Sci.)*, vol. 47, no. 1, pp. 7–8, Jan. 2017.
[7] Q. Zhang, X. Pan, S. Yan, and W. Li, "A novel scalable architecture of cloud storage system for small files based on P2P," in *Proc. IEEE Int. Conf. Cluster Comput. Workshops*, Beijing, China, Sep. 2012, pp. 41–47.
[8] S. Fu, X. Liao, C. Huang, L. Wang, and S. Li, "FlatLFS: A lightweight file system for optimizing the performance of accessing massive small files," *J. Nat. Univ. Defense Technol.*, vol. 35, no. 2, pp. 120–126, Feb. 2013.
[9] S. Fu, L. He, C. Huang, X. Liao, and K. Li, "Performance optimization for managing massive numbers of small files in distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3433–3448, Dec. 2015.
[10] Y. Zhao, X. Xie, Y. Cai, G. Wang, and L. Liu, "A strategy of small file storage access with performance optimization," *J. Comput. Res. Develop.*, vol. 49, no. 7, pp. 1579–1586, Jul. 2012.
[11] M. P. Elkafrawy, M. A. Sauber, and M. M. Hafez, "HDFSX: Big data distributed file system with small files support," in *Proc. IEEE Int. Comput. Eng. Conf. (ICENCO)*, Cairo, Egyp, Feb. 2016, pp. 131–135.
[12] K. Bok *et al.*, "An efficient distributed caching for accessing small files in HDFS," *Cluster Comput.*, vol. 20, no. 4, pp. 3579–3592, Dec. 2017.
[13] W. Cheng, M. Zhou, B. Tong, and J. Zhu, "Optimizing small file storage process of the HDFS which based on the indexing mechanism," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Big Data Anal.*, Chengdu, China, Apr. 2017, pp. 44–48.
[14] W. Zhang, W. Lu, H. He, Q. Zhang, and C. Yu, "Exploring large-scale small file storage for search engines," *J. Supercomput.*, vol. 78, no. 2, pp. 2911–2923, Aug. 2016.
[15] S. Yu, X. Gui, R. Huang, and W. Zhuang, "Improving the storage efficiency of small files in cloud storage," *J. Xi'an Jiaotong Univ.*, vol. 10, pp. 65–72, Jun. 2011.
[16] T. Wang, S. Yao, Z. Xu, and L. Xiong, "A small file mer-ging and prefetehing strategy based on access task in cloud storage," *Geomatics Inf. Sci. Wuhan Univ.*, vol. 38, no. 12, pp. 1504–1508, Dec. 2013.
[17] L. Xiong, Z. Xu, T. Wang and X. Gu, "On the store strategy of small spatiotemporal data files in cloud environment," *Geomatics Inf. Sci. Wuhan Univ.*, vol. 39, no. 10, pp. 1252–1256, Oct. 2014.
[18] X. Liu, J. J. Han, Y. Zhong, C. Han, and X. He, "Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS," in *Proc. IEEE Int. Conf. Cluster Comput. Workshops*, New Orleans, LA, USA, Sep. 2009, pp. 1–7.
[19] B. Dong, Q. Zheng, F. Tian, K.-M. Cao, R. Ma, and R. Anane, "An optimized approach for storing and accessing small files on cloud storage," *J. Netw. Comput. Appl.*, vol. 35, no. 6, pp. 1847–1862, Jun. 2012.
[20] B. Dong, J. Qiu, J. Q. Zheng, X. Zhong, J. Li, and Y. Li, "A novel approach to improving the efficiency of storing and accessing small files on Hadoop: A case study by powerpoint files," in *Proc. IEEE Int. Conf. Services Comput.*, Miami, FL, USA, Jul. 2010, pp. 65–72.
[21] C.-M. Zhang, J.-W. Rui, and T.-T. He, "An approach for storing and accessing small files on Hadoop," *Comput. Appl. Softw.*, vol. 29, no. 11, pp. 95–100, Nov. 2012.
[22] Z. Gao, Y. Qin, and K. Niu, "An effective merge strategy based hierarchy for improving small file problem on HDFS," in *Proc. Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, Beijing, China, Aug. 2016, pp. 327–331.
[23] H. He, Z. Du, W. Zhang, and A. Chen, "Optimization strategy of Hadoop small file storage for big data in healthcare," *J. Supercomput.*, vol. 72, no. 10, pp. 3696–3707, Aug. 2016.
[24] D. T. Dang, D. Hoang, and P. Nanda, "A novel hash-based file clustering scheme for efficient distributing, storing, and retrieving of large scale health records," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Tianjin, China, Aug. 2016, pp. 1485–1491.
[25] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*, 3rd ed. Beijing, China: Machine Press, 2012.



LIAN XIONG was born in Huanggang, Hubei, China, in 1985. He received the B.S. degree in telecommunications engineering from the North University of China, Taiyuan, China, in 2008, the M.S. degrees in communication and information system from the Taiyuan University of Technology, Taiyuan, in 2011, and the Ph.D. degree in communication and information system from Wuhan University, Wuhan, China, in 2016. He is currently a Lecturer with the School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications. His research interests include data mining, distributed storage, and the Internet of Things.

YUANCHANG ZHONG received the B.S. degree in electrical engineering from the Changchun University of Science and Technology, Changchun, China, in 1988, and the M.S. degree in communication engineering and the Ph.D. degree in mechanical electronic engineering from Chongqing University, Chongqing, China, in 2002 and 2009, respectively, where he is currently a Professor with the College of Communication Engineering. His current research interests include wireless sensor networks, MEMS, and RFID technology.

XIAOJUN LIU received the Ph.D. degree in communication and information system from Wuhan University, Hubei, China. He is currently an Associate Professor with the Department of Electronic and Information, Huanggang Normal University, Huanggang, Hubei, China. His research interests include computer networks, information processing, and cloud computing.



LIU YANG received the B.S. degree in electronic information science and technology from the Qingdao University of Technology, Shandong, China, in 2010, and the Ph.D. degree in communication and information systems from the College of Communication Engineering, Chongqing University, Chongqing, China, in 2016. He is currently a Lecturer with the Chongqing University of Posts and Telecommunications. His research interests include wireless sensor networks and image processing.