Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 23 Issue 2 Version 1.0 Year 2023
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Critical Analysis of Solutions to Hadoop Small File Problem

By Prof. Shwetha K. S. & Dr. Chandramouli H.

Abstract- Hadoop big data platform is designed to process large volume of data. Small file problem is a performance bottleneck in Hadoop processing. Small files lower than the block size of Hadoop creates huge storage overhead at Namenode's and also wastes computational resources due to spawning of many map tasks. Various solutions like merging small files, mapping multiple map threads to same java virtual machine instance etc have been proposed to solve the small file problems in Hadoop. This survey does a critical analysis of existing works addressing small file problems in Hadoop and its variant platforms like Spark. The aim is to understand their effectiveness in reducing the storage/computational overhead and identify the open issues for further research.

GJCST-C Classification: *LCC: QA76.585*

Strictly as per the compliance and regulations of:

Critical Analysis of Solutions to Hadoop Small File Problem

Prof. Shwetha K. S. [α] & Dr. Chandramouli H. [σ]

Abstract- **Hadoop big data platform is designed to process large volume of data. Small file problem is a performance bottleneck in Hadoop processing. Small files lower than the block size of Hadoop creates huge storage overhead at Namenode's and also wastes computational resources due to spawning of many map tasks. Various solutions like merging small files, mapping multiple map threads to same java virtual machine instance etc have been proposed to solve the small file problems in Hadoop. This survey does a critical analysis of existing works addressing small file problems in Hadoop and its variant platforms like Spark. The aim is to understand their effectiveness in reducing the storage/computational overhead and identify the open issues for further research.**

I. Introduction

Hadoop is an open source big data processing platform designed to process large volume of data. The data is kept in form of files in Hadoop distributed file system (HDFS). A map job is spawned on a java virtual machine (JVM) instance for each file in HDFS. The file data is copied to a memory block and the block is passed to map task. In addition, a object instance is created for each file in the Namenode of Hadoop to facilitate processing. When the file size is more than or equal to block size, maximum performance gain in achieved in terms of number of maps spawned and the meta data storage overhead at Namenode. In case of IoT applications, the data files are small (less than 2KB) and when these files are stored in HDFS for data processing, it affects the Hadoop performance [1-2]. On one hand, it drastically increases the storage overhead at Namenode for object bookkeeping [3]. On another hand it exhausts the computational resources by spawning multiple map tasks which only lasts for smaller duration to process small files. The time spent in bootstrapping the map task becomes higher than data processing time in case of small files. Various solutions have been proposed addressing the Hadoop small file problem. The existing solutions can be categorized as: (i) file merging solutions, (ii) file caching solutions, (iii) optimizing Hadoop cluster structure and (iv) Map task optimizations. In file merging solutions, pre-treatment of small files is done to form a big file and this big file is stored in HDFS. In file caching solutions, files are sent to a file queue, and when queue size crosses threshold files are sent to processing in a systematic manner. In Hadoop cluster structure optimization solutions, hierarchical memory structure is created combining cache and HDFS memory to reduce the overhead due to single name node. In map task optimization solution, number of JVM instances spawned for map tasks are reduced and shared.

This work does a critical analysis on various solutions in the above four categories of file merging, file caching, Hadoop cluster structure optimization and map task optimization. The effectiveness of each of the solutions in terms of storage and computation are analyzed and their open issues are identified. Based on the open issues, a prospective solution framework is designed and detailed.

II. Survey

Ahad et al [4] proposed a dynamic merging strategy based on the file type for Hadoop. Dynamic variable size portioning is applied to blocks and the file contents are fitted to blocks using next fit allocation policy. By this way large file is created and saved to HDFS. In addition, authors also secured the block using Twofish cryptographic technique. The solution reduced name node memory, number of data blocks and processing time. Merging was done only based on file types without considering the context and their semantic relation. Siddiqui et al [5] proposed a cache based block management technique for Hadoop as a replacement for default Hadoop Archives (HAR). A logical chain of small files is built and transferred to data blocks. In addition, efficient read/write on blocks was facilitated using block manager. Though the solution achieved more than 92% space utilization of data blocks, small files are merged only based on size, without considering the semantic relations and content characteristics. Zhai et al [6] built a index based archive file to solve the small file problem in Hadoop. The small files are merged to large file and metadata record is created to retrieve each file content. Meta data records are arranged into buckets. An order preserving hash is created over metadata records. The hash and the metadata records are in turn written to a index file. The index files helps to retrieve the file contents for processing. This method is able to save atleast 11% disk space but the solution access efficiency becomes

Author α: Ph.D Research Scholar Department of Computer Science & Engg., East Point College of Engineering and Technology, Bengaluru, Karnataka, India. e-mail: shwethaise.nhce@gmail.com
Author σ: Dr. Chandramouli H Professo Department of Computer Science & Engg., East Point College of Engineering and Technology, Bengaluru, Karnataka, India.
e-mail: hcmcool123@gmail.com

lower with large number of small files. Also the indexing does not support streaming inputs. Cai et al [7] proposed a file merging algorithm based on two factors of distribution of the files and the correlation of the file. Correlation between files is built based on their history of access and the highly correlated files are kept in the same block. Through experiments, author found that placing highly correlated files in same block improved the speed up. The correlation is not based on content characteristics so over a period of time, performance can reduce. Choi et al [8] integrated combinedfileinputformat and JVM reuse to solve the small file problem. Small files are combined till block size and passed to map task. JVM instances are reused for the map task , so they overhead of JVM bootstrap is minimized. Though the integration reduces the computational overhead, the approach combined files in order without considering their semantics. Also the memory buildup due to JVM reuse can crash the tasks due to inefficient memory management. Peng et al [9] combined merging and caching techniques to solve the small file problem. User based collaborative filtering is applied to learn the correlation between the files. Files with higher correlation are merged into single large file. Remote procedure call (RPC) requests to fetch the block information about the files are reduced by caching the access requests and looking into cache for the blocks before placing RPC requests. By this way, authors were able to reduce the file access time by 50% and increase storage utilization by 25% compared to default Hadoop. The scheme does not works well for streaming data, as the correlation model proposed in this work is not adaptive to streaming data. Niazi et al [10] proposed a new technique called inode stuffing to solve the small file problem. For small files, the metadata and data block are combined and decoupling is maintained only for large files. The approach is not scalable as it increases the metadata storage overhead at Namenodes. Jing et al [11] proposed a dynamic queue method to solve the small file problem. The files are first classified using the period classification algorithm. The algorithm calculates similarity score based on sentence similarity between two documents. The similar files are then merged to large file using multiple queues for specific file sizes. Authors also used file pre-fetching strategy to improve the efficiency of file access. Analyzing similarity between pairs is a cumbersome task for large number of files. Sharma et al [12] proposed a dual merge technique called Hash Based-Extended Hadoop Archive to solve the small file problem in Hadoop. The small files are merged using two level compaction. This reduces the storage overhead at Namenode and increase the data block space utilization at Datanodes. File access is made efficient using two level hash function. The proposed solution is atleast 13% faster compared to default Hadoop. The files were merged without considering the content characteristics and their semantics. Wang et al [13] combined merging and caching to solve the small file problem in Hadoop. Authors proposed a equilibrium merger queue algorithm to merge small files to Hadoop block size and then merged file is saved to HDFS. Indexing is built to access small files. To reduce the communication overhead between the client and Namenode for small file access, pre-fetched cache is used. With the cache, the number of RPC calls to name node is reduced. The memory consumption at Namenode drastically reduced in the proposed solution compared to default Hadoop Archives. Contents were merged without considering their content characteristics and semantic correlation. Ali et al [14] proposed a enhanced best fit merging algorithm to merge small files based on type and size. The merging is done till Hadoop block size is reached and merged file is saved to HDFS. Author found that merging improved Hadoop storage utilization by 64% but the file access time was higher in this work. Prasanna et al [15] compressed many small files into a zip file to the size of Hadoop data block and saved to disk. This increased the disk utilization of data nodes and name nodes. But the computational overhead in compressing stage and decompressing during processing is higher. Huang et al [16] addressed the small file problem for the case of images in Hadoop. A two level model was proposed specific to medical images. The images were grouped at first level based on series and next level based on examination. The grouped images are saved to data blocks in HDFS. Indexing and pre-fetching is done to done is reduce the access time for small image files. The pre-fetching algorithm did not have higher cache hit. Renner et al [17] extended the Hadoop archive to appendable file format to solve the small file problem. Small files are appended to existing archive data files whose block size is not completely used. Authors used first fit algorithm to select the data blocks. In addition indexing is done to facilitate faster access. Red black tree structure is used for indexing for efficient lookup. Though this scheme improved the data block utilization, appending is done without considering content characteristics and semantic similarity. Liu et al [18] proposed a file merging strategy based on content similarity. Files are converted to vector space features and correlation between the features is measured using cosine similarity. When cosine similarity is greater than threshold, files are merged. In addition authors used pre-fetching and caching to speed up the file access. Constructing a global feature space for streaming data is difficult and thus this approach is not suitable for streaming data.Lyu et al [19] proposed an optimized merging strategy to solve small file problem. The small files are merged based on size in such that way block size is fully utilized. In addition authors used pre-fetching and caching to increase the access speed. Only block size utilization was considered as the only criteria for

merging without considering content characteristics and semantic relations. Similar to it Mu et al [20] proposed an optimization strategy to maximally fill the existing Hadoop archive by appending small files. In addition author also used secondary index to speed up the execution of file access. But here too merging was done without considering content characteristics and semantic relation. Wang et al [21] used probabilistic latent semantic analysis to determine the user access pattern and based on it small files are merged to a large file and placed in HDFS. In addition author also improved the pre-fetching hit ratio based user access transition pattern. Both the strategies improvised the speed of access and data block utilization. But this scheme is not suitable for multi user environment as for each user, a merging order must be kept and this increases the storage overhead. He et al [22] merging the small files based on balance of data blocks. The aim was to increase the data block utilization. Merging did not consider content characteristics and their semantic relation. Fu et al [23] proposed an flat storage architecture to handle the small files. In this scheme, both files and meta data are collocated with meta size fixed for any number of small files. This is facilitates by meta data having only pointer to related information in its index. But the scheme is not suited for Hadoop as collocation causes higher access overhead for large files. Tao et al [24] merged small files to large file and built a linear hash to small files to speed up access. File size was the only criteria considered for merging. Bok et al [25] integrated file merging and caching to solve the small file problem. Author used two level of cache for small files, so that access requests to –

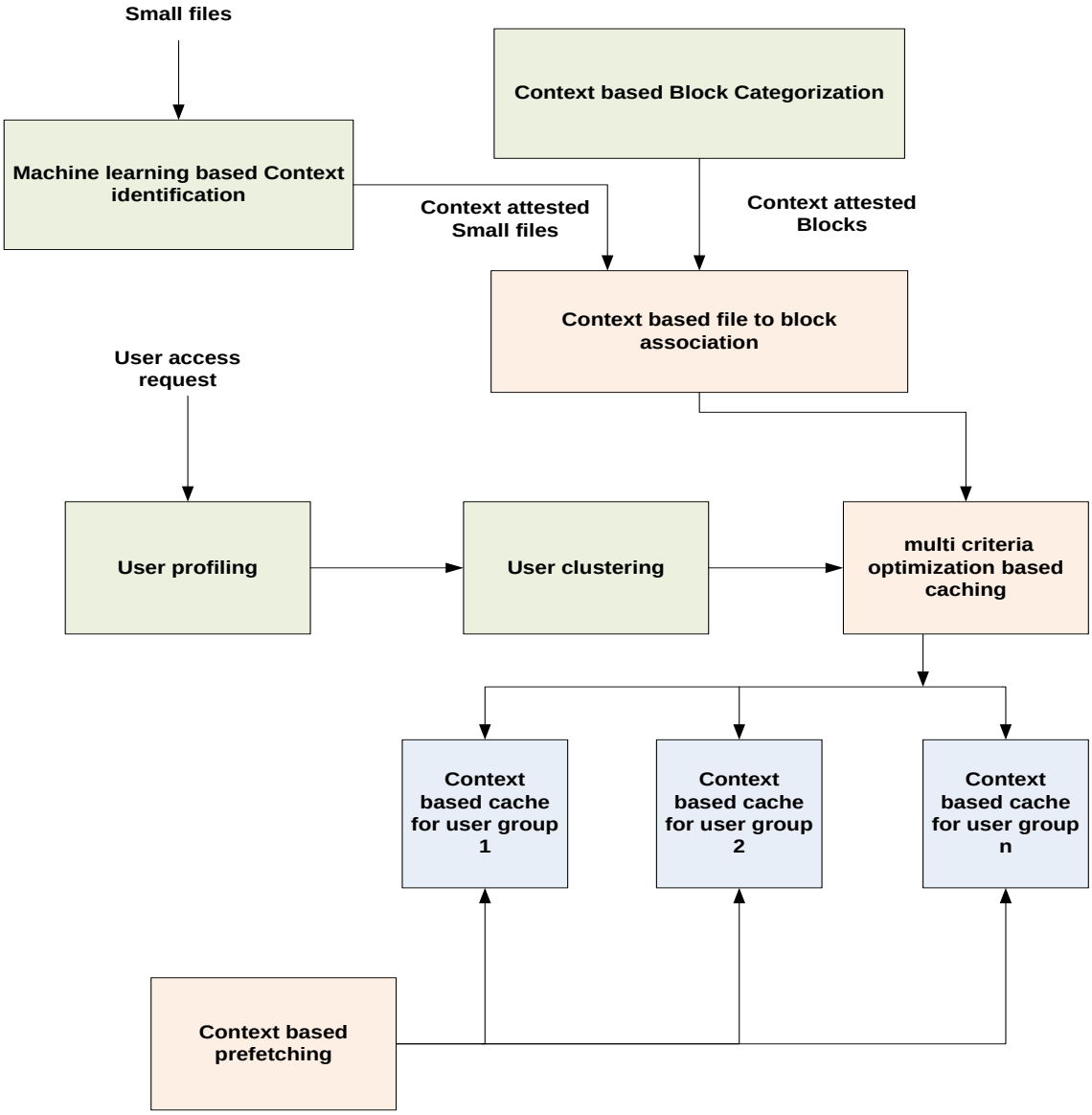

Figure 1: Research direction framework

Table 1: Survey Summary

Work	Solution for Small file Problem	Gap
Ahad et al [4]	dynamic merging strategy based on the file type	Merging was done only based on file types without considering the context and their semantic relation
Siddiqui et al [5]	cache based block management technique	small files are merged only based on size, without considering the semantic relations and content characteristics
Zhai et al [6]	a index based archive file with order preserving hash for speedup	Does not support streaming
Cai et al [7]	file merging algorithm based on two factors of distribution of the files and the correlation of the file	The correlation is not based on content characteristics
Choi et al [8]	integrated combinedfileinputformat and JVM reuse to solve the small file problem	memory buildup due to JVM reuse can crash the tasks due to inefficient memory management
Peng et al [9]	combined merging and caching techniques to solve the small file problem	The scheme does not works well for streaming data, as the correlation model proposed in this work is not adaptive to streaming data
Niazi et al [10]	Coupling both meta data and small file together.	The approach is not scalable as it increases the metadata storage overhead at Namenodes
Jing et al [11]	Files classified using the period classification algorithm and merged based on similarity	Analyzing similarity between pairs is a cumbersome task for large number of files
Sharma et al [12]	Hash Based-Extended Hadoop Archive to solve the small file problem	The files were merged without considering the content characteristics and their semantics.
Wang et al [13]	combined merging and caching to solve the small file problem	Contents were merged without considering their content characteristics and semantic correlation
Ali et al [14]	enhanced best fit merging algorithm to merge small files based on type and size.	file access time was higher in this work
Huang et al [16]	A two level model was proposed specific to medical images	The pre-fetching algorithm did not have higher cache hit
Renner et al [17]	Small files are appended to existing archive data files	Appending is done without considering content characteristics and semantic similarity
Liu et al [18]	File content based merging	Constructing a global feature space for streaming data is difficult and thus this approach is not suitable for streaming data
Lyu et al [19]	optimized merging strategy to solve small file problem.	Only block size utilization was considered as the only criteria for merging without considering content characteristics and semantic relations
Wang et al [21]	probabilistic latent semantic analysis to determine the user access pattern and based on it small files are merged to a large file	scheme is not suitable for multi user environment as for each user, a merging order must be kept and this increases the storage overhead
He et al [22]	merging the small files based on balance of data blocks	Merging did not consider content characteristics and their semantic relation
Fu et al [23]	flat storage architecture collocating metadata and file in same object	the scheme is not suited for Hadoop as collocation causes higher access overhead for large files
Tao et al [24]	merged small files to large file and built a linear hash to small files to speed up access	File size was the only criteria considered for merging
Bok et al [25]	integrated file merging and caching to solve the small file problem	The merging was based only on size without considering the content characteristics and semantic similarity

Namenode is totally minimized. Least recently used (LRU) mechanism is used to upgrade the cache. The merging was based only on size without considering the content characteristics and semantic similarity.

The summary of survey so far discussed is presented in Table 1.

III. Open Issues

From the survey, following three open issues are identified.

1. Context specific merging
2. Personalized access
3. Streaming support

Context specific merging: In most of the existing approaches, merging was based only on size. Merging did not consider user access or application contexts, content characteristics and their semantic relation. In applications like recommendations based on user comments, it is necessary to co-locate user comments related to specific product characteristics in same blocks for application speedup.

Personalized Access: In most of the existing caching strategies, caching was based on least recently used at a global context without considering the user access context. But it is important to consider user access context as each user access behavior is different. Caching on global context can provide better performance for some users and can give worst performance for other users. To solve this access time discrepancy among the users, personalized caching strategy must be employed.

Steaming Support: Most of the merging schemes does not handle the steaming data effectively. Streaming data content similarity cannot be computed effectively using vector space modeling and their merging can become ineffective. Merging based on streaming arrival patterns has not been considered in earlier works.

IV. Research Directions

Based on the open issues identified, a prospective framework for further research is presented in Figure 1.

The framework addresses three problem areas of context specific merging, personalized access and streaming support.

Context Specific Merging: It can be facilitated and made adaptive using machine learning. Based on the application contexts and inherent data characteristics the files to be merged can be found. Blocks can be categorized based on context and small files can be categorized based on context. Context based merging is the realized to merge files and blocks based on context similarity. Instead of flat context, hierarchical context can be learnt automatically from file summarization. File summarization strategies specific to file types can be proposed to identify the context to be associated with files and blocks.

Personalized Access: User can be clustered based on their content access patterns over a temporal duration and multiple caches can be maintained for each user group. Also the cache item management can be based on multi criteria optimization instead of LRU mechanisms. The items to pre-fetch can be identified based on context associated with files. By this way access speed up can be increased and optimized specific to each user group.

Streaming Support: To support streaming data, the context must be learnt dynamically in a light weight manner and association of small file to blocks must be done based on context. To learn context in a light weight manner, the streaming data characteristics and their arrival patterns must be used.

V. Conclusion

This survey made a critical analysis of existing solutions for small file problem in Hadoop. The solutions were analyzed in four categories of file merging solutions, file caching solutions, optimizing Hadoop cluster structure and Map task optimizations. Based on the survey, three open issues of context specific merging, personalized access and streaming support are identified. Prospective solutions to these three open issues were identified and a solution roadmap for further exploration in this area was documented.

References Références Referencias

1. Small size problem in Hadoop: http://blog.Cloudera.com/blog/2009/02/the-small-files-problem/
2. Solving Small size problem in Hadoop https://pastiaro.wordpress.com/2013/06/05/solving-the-small-files-problem-in-apache-hadoop-appending-and-merging-in-hdfs/
3. Bo Dong, Qinghua Zheng, Feng Tian, Kuo-Ming Chao, Rui Ma, Rachid Anane. (2012), An optimized approach for storing and accessing small files on cloud storage, Journal of Network and Computer Applications, 35 (2012) 1847-1862, Elsevier.
4. Ahad, Mohd & Biswas, Ranjit. (2018). Dynamic Merging based Small File Storage (DM-SFS) Architecture for Efficiently Storing Small Size Files in Hadoop. Procedia Computer Science. 132. 1626-1635. 10.1016/j.procs.2018.05.128.
5. Siddiqui, Isma & Qureshi, Nawab Muhammad Faseeh & Chowdhry, Bhawani & Uqaili, Mohammad. (2020). Pseudo-Cache-Based IoT Small Files Management Framework in HDFS Cluster. Wireless Personal Communications. 113. 10.1007/s11277-020-07312-3.
6. Zhai, Yanlong & Tchaye-Kondi, Jude & Lin, Kwei-Jay & Zhu, Liehuang & Tao, Wenjun & Du, Xiaojiang

& Guizani, Mohsen. (2021). Hadoop Perfect File: A fast and memory-efficient metadata access archive file to face small files problem in HDFS. Journal of Parallel and Distributed Computing. 156. 10.1016/j.jpdc.2021.05.011.
7. Cai, Xun & Chen, Cai & Liang, Yi. (2018). An optimization strategy of massive small files storage based on HDFS. 10.2991/jiaet-18.2018.40.
8. Choi, C., Choi, C., Choi, J. et al. Improved performance optimization for massive small files in cloud computing environment. Ann Oper Res 265, 305–317 (2018).
9. Peng, Jian-feng & Wei, Wen-guo & Zhao, Hui-min & Dai, Qing-yun & Xie, Gui-yuan & Cai, Jun & He, Ke-jing. (2018). Hadoop Massive Small File Merging Technology Based on Visiting Hot-Spot and Associated File Optimization: 9th International Conference, BICS 2018, Xi'an, China, July 7-8, 2018, Proceedings. 10.1007/978-3-030-00563-4_50.
10. S. Niazi, M. Ronström, S. Haridi, and J. Dowling, 'Size Matters : Improving the Performance of Small Files in Hadoop', presented at the Middleware'18. ACM, Rennes, France, 2018, p. 14.
11. Jing, Weipeng & Tong, Danyu & Chen, GuangSheng & Zhao, Chuanyu & Zhu, LiangKuan. (2018). An optimized method of HDFS for massive small files storage. Computer Science and Information Systems. 15. 21-21. 10.2298/CSIS171015021J.
12. V. S. Sharma, A. Afthanorhan, N. C. Barwar, S. Singh and H. Malik, "A Dynamic Repository Approach for Small File Management With Fast Access Time on Hadoop Cluster: Hash Based Extended Hadoop Archive," in *IEEE Access*, vol. 10, pp. 36856-36867, 2022
13. K. Wang, Y. Yang, X. Qiu and Z. Gao, "MOSM: An approach for efficient storing massive small files on Hadoop," *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, Beijing, China, 2017, pp. 397-401
14. A. Ali, N. M. Mirza and M. K. Ishak, "Enhanced best fit algorithm for merging small files," Computer Systems Science and Engineering, vol. 46, no.1, pp. 913–928, 2023.
15. L. Prasanna. Kumar, "Optimization Scheme for Storing and Accessing Huge Number of Small Files on HADOOP Distributed File System". International Journal on Recent and Innovation Trends in Computing and Communication, vol. 4, no. 2, Feb. 2016, pp. 315-9
16. Xin Huang, Wenlong Yi, Jiwei Wang, Zhijian Xu, "Hadoop-Based Medical Image Storage and Access Method for Examination Series", Mathematical Problems in Engineering, vol. 2021, Article ID 5525009, 10 pages, 2021.
17. Thomas Renner, Johannes Müller, Lauritz Thamsen, and Odej Kao. 2017. Addressing Hadoop's Small File Problem With an Appendable Archive File Format. In Proceedings of the Computing Frontiers Conference (CF'17). Association for Computing Machinery, New York, NY, USA, 367–372.
18. Liu, Jun. (2019). Storage-Optimization Method for Massive Small Files of Agricultural Resources Based on Hadoop. Journal of Advanced Computational Intelligence and Intelligent Informatics. 23. 634-640. 10.20965/jaciii.2019.p0634.
19. Y. Lyu, X. Fan, and K. Liu, "An optimized strategy for small files storing and accessing in HDFS," in Proc. IEEE Int. Conf. CSE, IEEE Int. Conf. EUC, Jul. 2017, pp. 611_614.
20. Q. Mu,Y. Jia, and B. Luo, ``The optimization scheme research of small files storage based on HDFS," in Proc. 8th Int. Symp. Comput. Intell. Design, Dec. 2015, pp. 431_434.
21. T. Wang, S. Yao, Z. Xu, L. Xiong, X. Gu, and X. Yang, ``An effective strategy for improving small _le problem in distributed file system," in Proc. 2nd Int. Conf. Inf. Sci. Control Eng., Apr. 2015, pp. 122_126
22. H. He, Z. Du, W. Zhang, and A. Chen, ``Optimization strategy of Hadoop small_le storage for big data in healthcare," J. Supercomput., vol. 72, no. 10, pp. 3696_3707, Aug. 2016
23. 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
24. W. Tao, Y. Zhai, and J. Tchaye-Kondi, "LHF: A new archive based approach to accelerate massive small _les access performance in HDFS", in Proc. 5th IEEE Int. Conf. Big Data Service Appl., Apr. 2019, pp. 40_48.
25. K. Bok, H. Oh, J. Lim, Y. Pae, H. Choi, B. Lee, and J. Yoo, ``An efficient distributed caching for accessing small files in HDFS," Cluster Comput., vol. 20, no. 4, pp. 3579_3592, Dec. 2017.