Research on Cloud-Based Mass Log Data Management Mechanism

Fengying Yang
School of Information Engineering, Huanghuai University, Zhumadian, China
Email: yfyinghh@yeah.net

Huichao Liu and Zhanping Zhao
School of Information Engineering, Huanghuai University, Zhumadian, China
Email: yfyinghh@yeah.net

***Abstract*—In this paper, we study the file management mechanism of large-scale cloud-based log data. With the rise of big data, there are more and more the Hadoop-based applications. Log analysis is an important part of network security management, but the existing network log analysis system can't deal with huge amounts of log data, or only use offline mode which with a longer response delay. Therefore, building the online Hadoop-based log processing system is necessary. However, how to effectively manage vast amounts of log data have become the key problems of such system. To this end, this paper puts forward a new hierarchical file archiving (HFA) mechanism which can realize the hierarchical and sorted storage of massive amounts of log data. In addition, some feasible methods for the mechanism are also proposed. Through the HFA mechanism, the traditional log analysis mode and Hadoop-based offline analysis mode can be combined to achieve the online Hadoop-based log analysis system, which have good scalability that can effectively store and handle the massive log data, and faster response speed for user request to meet the requirements of online processing. The feasibility and effectiveness of the HFA mechanism have been verified by the experiment of a small log process system.**

***Index Terms* —Network security, Log analysis, Hadoop, File management, Hierarchical file archiving mechanism, Online system**

I. INTRODUCTION

With the rise of big data, the number of Hadoop-based application and research continues to increase [1-6], and the application areas also keep expanding [7-9]. Conventional log data processing models cannot deal with the massive amount of log data for large network systems, so Hadoop-based log analysis systems are emerged [10-14]. However, almost all of the current Hadoop-based log analysis systems use offline processing mode, which means that the raw log data should be merged and clean in advance into the large data files which already have the clear structure and reasonable layout, and the analysis system only need to focus on business processes, and don't need to care about the complex log file management issues. Moreover, the response time of log analysis to user request is also not strict.

But in order to build online Hadoop-based log analysis system, the more complex situations will be faced [15]. On the one hand, the requirements of response time to process user requests in online log analysis system will be higher. The system must efficiently handle the various query and analysis requests in time or even in real-time. In addition, the response contents generated by system also should include the processing of the latest log data. On the other hand, log data are the stream data. Therefore, online log system must timely receive numerous log records from various log sources, and meanwhile, split the data stream into separated log file for appropriate size to store in the HDFS. For a large network system, the amount of log data that need to be store can be up to tens of TB or PB level [16]. If the size of log file is set too small, a large number of small files will be produced which could seriously impact the performance of Hadoop. Otherwise, if the size is too large, the update cycle of log file will be very long, that could result in the response for user request cannot contain the latest data.

Based on the above analysis, the online Hadoop-based log analysis system must have the high performance and massive storage capacity. Moreover, it also must have a valid file organization and management mechanism with good scalability. On the one hand, the scale of small file should be controlled reasonably in order to reduce the impact on system performance [17]. On the other hand, some effective methods should be taken for organizing the historical log data to meet the needs of business process, and remaining the relatively stable for file system structure when increasing the amount of log data to ensure the stability of system performance. Therefore, build a good organization and management mechanism of the log data is the key for online Hadoop-based log analysis system.

Hadoop platform also provides some mechanisms and tools for the processing of small files, such as HAR, SequenceFile and CombineFileInputFormat et al. But they all have some limitations. In addition, the literatures [18-21] have introduced some solutions for some kinds of specific application environment (such as WebGIS, Bluesky courseware delivery system and the Chinese font engineering). These works solve the problem of small file by adding a small file processing module above the

doi:10.4304/jcp.9.6.1371-1377

original HDFS file system. The design idea of the small file processing module is to archive some related small files into one large file firstly, and then create an index file for these small files for quick access.

In this paper, a hierarchical file archiving mechanism is proposed for the log data management based on the characteristic of online log analysis system, and some feasible methods of this mechanism are also present. The basic idea of hierarchical file archiving mechanism is to save the log data into different files which have different levels that are consistent with the access frequency of the log data. The data with higher access frequency will be saved in a relatively smaller file; otherwise, the larger file will store the data which has less access frequency. Thereby, this will form a pyramid and inverted pyramid hierarchy for log files and data storage structure respectively. The hierarchical file archiving mechanism can effectively solve the problem of small files through reducing the file size and maintaining reasonable file number to organize the global data space. The system which using this mechanism can maintain the good scalability, and improve the system performance and user experience.

Organization of the rest of this paper is as follows. The Hadoop environment is briefly reviewed in Section II. In Section III, the hierarchical file archiving mechanism and some feasible methods are proposed. The implement architecture of the mechanism is introduced in section IV. In Section V, a simple experiment for the mechanism is taken, and the results are given. Finally, the work is concluded in Section VI.

II. Hadoop File System

The Hadoop is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models [14]. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Hadoop is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers. Hadoop includes many important modules, but the basis of which are HDFS (Hadoop Distributed File System), which is a distributed file system that provides high-throughput access to application data, and MapReduce, which is a system for parallel processing of large data sets.

A. Read and Write Operation of HDFS

Hadoop Distributed File System (HDFS) is the default and widely used file system of the Hadoop platform. The main operation supported by HDFS are read, write and append, while the alter operation is not provided.

The procedure to read a file is indicated in Fig. 1, which include two main steps:

1) Client sends the read command to NameNode. If the file is not existence, error information will be returned. Otherwise, the blocks corresponding to the file and its position in DataNode will be sent to Client.

2) After receive the position information of the blocks, Client will connect to different Datanode, and get the data in parallel.

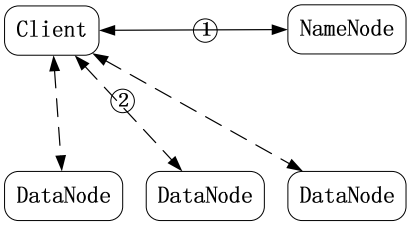


Figure 1. Read operation of HDFS.

Fig. 2 shows the procedure of the write operation, in which four steps are included:

1) Client sends the write file command. NameNode checks the file to see if the file exists. If it is, the error information will be return directly. Otherwise, NameNode will send a useful DataNode node list to Client.

2) Client splits the file into some blocks, and sends each block concurrently to different DataNode. When the sending complete, Client send message to NameNode and DataNode.

3) When receiving the message from Client, NameNode send a confirmation message to DataNode.

4) After receiving the confirmation messages from NameNode and Client, DataNode submits the write operation, and completes the file storage process.

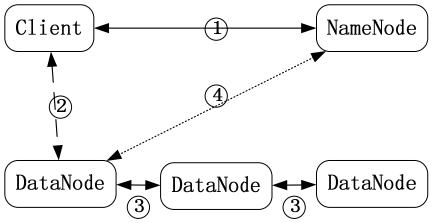


Figure 2. Write operation of HDFS.

B. Hadoop File Merging Mechanism

Hadoop also offers several file merging solutions for the problem of small file process. They are Hadoop Archive, Sequence file, CombineFileInputFormat and Append operation [17, 22].

Hadoop Archive or HAR is a new file system above the Hadoop file system, and also an efficient archiving tool that can package a number of small files into a HAR file and store it in some whole HDFS blocks. HAR mechanism can save the storage of HDFS, reduce the usage of memory in NameNode, and support the transparent access for those small files. Creating a HAR file is actually to run a Mapreduce job on a Hadoop cluster. In addition, once a HAR file is created, it cannot be changed. It must be re-created when add or remove the contents of the HAR file.

Sequence file consists of a series of binary Key/Value pairs. Hence the large numbers of small files can be merged into one large sequence file by setting the file name to the Key, and save the contents of the small file to the Value.

CombineFileInputFormat is a new format of Inputformat, used to combine multiple small files into a single split when running a Mapreduce job. In addition, the store locations of these small files are also taken into account for using.

The HDFS append operation has been redesigned and implemented in Hadoop 1.0 to fix some defects existed in previous versions. The new HDFS append operation is more complex for involving many interactions between Client, Namenode and Datanode. From the Client perspective, appending new data to an existed file need to call the append operation of *DistributedFileSystem* firstly, which will return an *FSDataOutputStream* stream object, namely *out*. Then the function *out.write*() should be called to append data to the file. Finally, the function *out.close*() should be executed to close the file.

C. Existed Small Files Handling Systems

Hadoop platform is designed for large data processing, and there is no universal system-level solution for the problem of small files so far. However, the small file problem is existed widely in many application fields, and some solutions have been proposed in some works. Literature [18] utilizes the data dependencies feature in WebGIS system to merge the small files which stored in adjacent location into one large file, and then creates an index file for these small files for quick access. Literature [19] proposed an HDFS small file storage solution base on the characteristics of Bluesky system. First, the files that belong to the same courseware will be archive into a large file in order to improve the storage efficiency of small files in HDFS. Second, the two-level pre-fetching mechanisms are proposed to enhance the reading efficiency of small files, which includes both the index file pre-fetching and data file pre-fetching. When using index file pre-fetching to access a file, the index file corresponding to the block which contains the small files will be loaded into memory firstly, so that the Client no longer need to interact with NameNode when access the other files. When using data file pre-fetching, all the files concern to the same courseware will be loaded into memory, so the access speed will be significantly improved when access the other files.

These works attempt to add a small file processing module base on the original HDFS to process the small files. Firstly, the module merges many small files which have some relevance into a large file. And then an index file is created for these small files for quick access. The mergence can not only increase the storage efficiency, but also improve the speed of application processing.

III. HIERARCHICAL FILE ARCHIVING MECHANISM

A. File Management Requirements

If the file stored in HDFS is not closed, the existence of the file cannot be observed by Client. But the user response in online log analysis system requires the processing of latest log information. This makes the online log analysis system must cut the log stream in time when receiving log data, and form separate log files which contains the received small amount of log information in short time. This approach can significantly reduce the data update cycle, and improve the real-time effect of log analysis. The timeliness of the response is undoubted a very important indicator for an online log analysis system, but it also will produce a large number of small files.

Too many small files would seriously affect the performance of Hadoop. Hadoop is designed to handle the large files, and the file storage unit is block which default size is 64M. If the size of file is less than the default size of block, the file will still take up a separate block for storage. Therefore, Excessive number of small files will significantly reduce the system storage efficiency. In addition, the metadata information of each file is stored in Namenode and resides in memory. If the number of small files is too many, the file namespace and memory cost will be very large in NameNode, and the processing cost for each request will be very time consuming. The Hadoop may also be crash due to the memory limitations in some case. If the problem of small file cannot be solved effectively, to build the online log analysis system is pointless.

Using the file merge methods provided by Hadoop to solve the small file problem is not a feasible idea. On the one hand, these files merge methods have some drawbacks which could result the merge cost or file access cost is too high. On the other hand, the primitive merge idea of these methods is also not suitable for the application features of online log analysis system. In which the distribution of log data is too concentrated will also produce some serious system performance issues. In the online log system, not all businesses are required of all data sets. Instead, only a few operations need to manipulate large data sets. Large number of operations is focus on the current or recent data. The longer of the historical data, the access frequency of the log data will be lower. If the data is too centralized, each business will involve a lot of useless data processing, which is bound to improve the system processing performance.

Therefore, the appropriate file management mode must be selected to solve the problem of small file by organizing the system data effectively, and make the system not only has better scalability, but also have better performance to meet the needs of user response time.

B. The Ideas of Hierarchical Archiving

Hierarchical file archiving mechanism is proposed based on the application characteristics of online log analysis system. This mechanism need to firstly set the different archiving levels in accordance with certain archiving criteria. When the lower-level file reaches the archiving standard, the mechanism will be started to merge the lower-level files into one upper-level file, and empty the old files meanwhile. When after a certain time, the lower-level files could reach the archiving standards again, and the archiving program will also be started. The similar methods will be used to archive the upper-level files to more high-level file when the file archiving standards is arrived. So after the cycle of step by step, the finished system file structure and storage structure will form a pyramid and inverted pyramid structure, shown in

Figure 3. Log analysis system splits log stream by the first-level file standard, and forms the bottom-level log file. Within the file structure of the system, the higher of the level, the less of the files number, and vice versa, thereby a positive pyramid structure is formed. However, in the storage structure, the lower level, the less amount of stored data, so an inverted pyramid structure is formed.

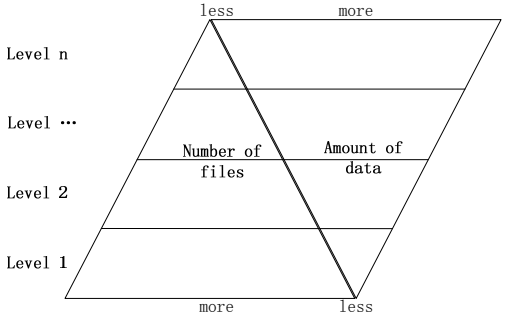


Figure 3. File and storage structure.

A good file archiving mechanism should have the following characteristics:

1) File merging frequency must be low. If the file merges too often, it will increase the burden of the system undoubtedly.

2) Average file merge cost should be small. This requires reducing the bigger cost merge operation as far as possible.

3) File system should have good scalability. Files number in the system should remain relatively stable, could not have a greater change with the increase of the amount of log data.

4) File structure should meet the needs of application. The partitions of file should fit the application needs of the data set, and minimize the chance of cross-file operations.

C. Hierarchical File Archiving Mechanism

Since the log data structure is relatively simple, just consisted by the text log record, so the log file has better divisibility. Therefore, the mergence or reorganization for log files has greater flexibility, which only needs to keep the integrality of the log record. For different application needs, it can be specified by different hierarchical archiving standards, such as archiving by content attributes, by file size, by files number, by time attribute of log data, by comprehensive multiple factors and so on.

1) Archiving by files size

First of all, the standards of file size should be set for different levels of the archiving mechanism. Log system splits log stream and forms log file according to the first-level archive file size. When the data size of lower-level file is greater than or equal to the higher-level file size standard, then the system will start archiving process and merge the lower-level files into a upper-level file. For example, setting the first-level file size is 64MB, the file size of the second and third level is 1GB and 1TB respectively. When the number of first-level file reaches 16 (16 * 64MB = 1GB), the system starts the archiving program to form a second-level file. When the number of 1GB file reaches 1024, they will be merged into the third-level file.

Hierarchical archiving by the file size has the advantage that it can translate the large data process in log analysis system into a series of sub-operation. When processing a query, the system can be firstly carried out the sub-operations on lower-level file. Because the small data size of lower-level file, the system can complete and return the response in a shorter time. While the user to process the returned results, the system could start a new search for higher-level file. This enables the user processing and system processing in parallel. Furthermore, if the user has found the result of the needed, the subsequent sub-operations should be abort. Using this file organization model can not only ensure that the system can respond faster, but also to process all log data gradually.

Achieving by file size can guarantee that the file size in every level is constant. For the predictable data size, the maximum amount of files needed to maintain by system can be easily calculated. Therefore, the scalability of the file system is very good, and the maximum number of files changed very little as the data size increasing. With this mechanism, the archiving opportunity of system performing is uncertain. When the log flow is large, the period to formation equal size file will be short, and the times for file archiving will be more. And vice versa, the chance of archiving file will be decreased when the data flow is small. But because of the network log flow has its periodicity, and the amount of log data files with different level is different, the higher-level file will have the smaller effect by the changing of data flow.

2) Archiving by number of files

Firstly, the desired levels and the number of files standard for each archiving level need to be set. When the number of current-level files accumulates to the archiving standards for next-level files, system will start the file archiving operations to form a higher-level file. Assume that the number of files for every archiving level is set to 400, when the number of files reaches 400 for each level, the lower-level file will be archived into an upper-level file.

Archive by the number of files has the similar effect with the archiving by file size. The system processes each file step by step, which can ensure the response speed, and improve the user experience.

However, there are some differences between the two methods: archiving by the number of files can ensure the regularity of archived files within the same time interval. But it cannot guarantee the uniformity of size of each merged file. When the log flow enlarges, the size of the log file archived within the interval will be very large. On the contrary, the size of the file may be smaller. But this effect mainly occurred in the low-level file, the more upper-level file the smaller of the influence on it.

3) Archiving by log time

The frequencies of use of log data will become less with its time growing. Users are always taking more interested in current or recent log data, even though the historical data also is valuable. Therefore, we can make different

archiving time standards according to different application environments. Equidistant time standards are one of the feasible methods, such as one week, two weeks, three weeks, etc. Non-equidistant time standard is another good choice, such as one day, one week, one month, one quarter and more than one year and so on. In the latter for instance, system archives the accumulated data within one day to form a daily log file. The weekly archived data was formed by merging the log data accumulated in a week. And so on, log files within different time periods can be created.

Time is an important attribution for the log query. Therefore, the data within some time interval is more suitable for some application. Archiving by time could allow the system to process user request confined to the specific data set, rather than the all of the data, which can significantly improve system performance.

4) Archiving by log priority

The log messages generated by various kind of network equipment are all have the priority tag, typically which are debug, informational, notice, warning, critical, alert, emergency and so on. Log message with different priority represents different important level which could result in the different probability of being requested or processed in practice. Using this method, the log messages with same priority will archive into the same log file. Therefore, when processing the user requests, only the interested datasets need to access, which can reduce the amount of log data, and in favor of improving the process efficiency.

5) Archiving by comprehensive strategies

Two or more strategies can be used together to form a more careful and reasonable file archiving mechanism. For example, the log message priority and time can be combined to form a new archiving method which first to merge log messages with different priority into different log files. Then achieve each log file by different level time standard. The log time and file size method also can be combined, the low-level file can be archived with file size standard, and the formed high-level log file can be processed by time standard.

The general principle of selecting multiple strategies is beneficial to show the advantages of each strategy, and overcome their disadvantages. Comprehensive strategies can not to undermine the scalability of the log storage system, and should be helpful to improve system performance.

D. Solving the Archiving Singularity

Archiving singularity refers to the lower-level archiving operation triggers on a series of upper-level archiving operations in the archiving process. In this case, the archiving operation would rise like the dominoes extension. Concentrated archiving operation will undoubtedly bring greater impact on system performance, while resulting in major changes of the file structure. This could deviate from the original design of hierarchical archiving mechanism.

The time delay mechanism can be taken to resolve the archiving singularity problem. One way is to set the delay interval for continuous archiving operation, and the successive archiving operations must be conducted outside the specified delay interval. This separates the archiving operation artificially. Delay interval should reference the average interval of each archiving level, it is best to set the number coprime with all archiving levels. Another method is to set the system to perform high-level archiving in spare time. The probability of upper-level archiving is already very low, so perform the archiving mechanism when the system is idle can avoid the archiving singularity problem effectively.

IV. The Implement Architecture

In order to apply this mechanism better in practice, this section discusses the implementation architecture of the hierarchical file archiving mechanism in detail. Fig. 4 shows the simple deployment architecture of the Hadoop-based log analysis system. There are three major characters in this architecture: log source, log server and Hadoop cluster. The range of log source is very wide, which could be the computers (such as workstation, application server et al.), network devices (such as router, switcher, firewall and many others) and so on. During the running of the log source, some events may be occurred. The log process on log source captures the event and forms the log message. Then, these log messages will be sent to log server in the form of stream. The number of these log messages from different log source will be very huge. So, the log server should put them into different queue. Then, these log messages will be do some pretreatment, such as format conversion, removing the useless field, adding some default fields and so on. Then the log will be saving to a local log buffer file. When the local log file reaches the predefined conditions, they will be sent to the Hadoop cluster, saved into the DataNode and indexed in NameNode.

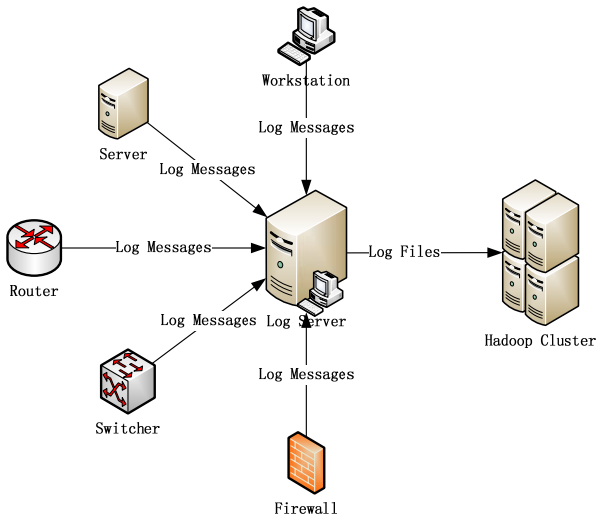


Figure 4. The deployment architecture of log system.

Obviously, the log server occupies the core position in the whole system. We should understand how log server works. Fig. 4 indicates the process procedure of log

information within the log server. In order to store a log message to the cloud system, there are seven steps that need to be performed.

Step1: log server uses some threads to receive the log message, and put them in the appropriate queue.

Step2: log server does some pretreatment, such as discarding the useless or repeated log information, adding or removing some fields and so on. Finally, the log information also should be classified.

Step3: log server stores the classification data into the local log buffer file, which according to bottom level of the hierarchical file archiving mechanism.

Step4: log server checks the size of the saved log buffer file. If the file meets the predefined size, then the flow continues, otherwise, the flow will go to step 6.

Step5: merges the saved local log files into a single log file, and save it to Hadoop cluster, and then delete all the local log buffer files.

Step6: check the conditions of the hierarchical file archiving mechanism. If the conditions are met, step 7 will be taken to perform the hierarchical file archiving mechanism; otherwise, step 8 will be performed.

Setep7: run the MapReduce programs on the Hadoop cluster. Archive the low-level log file to the upper-level log file, and then delete namespace of the low-level files in Namenode.

Step8: check whether the exit condition is met. If so, the flow will be exited. Otherwise, the flow will be go to setep1, and start the next loop.

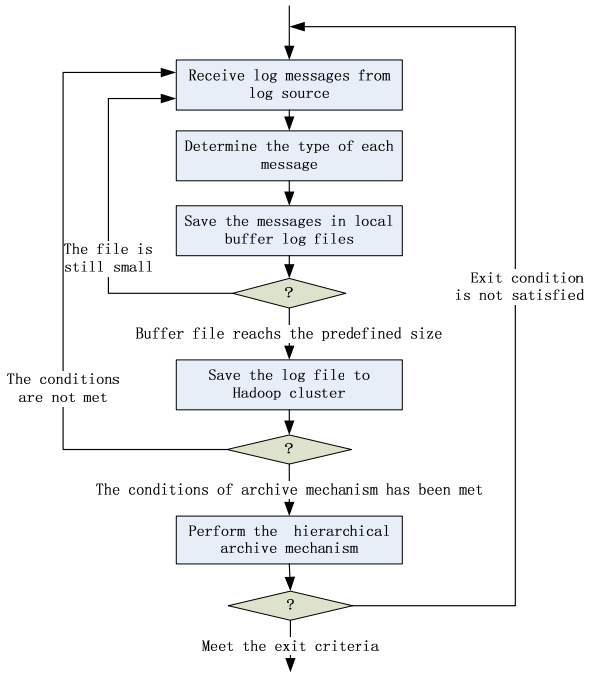


Figure 5. Log processing flow of log server

V. Experimental Analysis

In order to verify the feasibility of proposed hierarchical file archiving mechanism, here take a simple experiment, in which the 1T log data was processed to compare the processing performance in a hierarchical and non-hierarchical system. Hierarchical mechanism used the method of archiving by file size, and the archiving standards for level 1 to level 4 were set to 64MB, 1GB, 16GB and 256GB. The amount of log data allocated for each level are 1GB, 15GB, 240GB and 768GB. The hardware platform contains 4 servers, one acts as the NameNode, the remaining 3 servers are the DataNode. The data processing time in hierarchical and non-hierarchical case are shown in Table I and Table II.

From Table I and Table II, it can be seen that, the total data processing time of the system with hierarchical file archiving mechanism is slightly more than the system with non-hierarchical mechanism for the same amount of log data. But the response time of system with hierarchical file archiving mechanism is in the large lead of the other system, which is a crucial indicator for the online log analysis system. In addition, since most user requests do not need to access the entire data set, so the processing time of hierarchical archiving data could far fewer than non-hierarchical data in practice.

TABLE I.
PROCESSING COSTS

Item/data	Non-hierarchical	Hierarchical			
File size	1T	64M	1G	16G	256G
Data size	1T	1G	15G	240G	768G
Files number	1	16	15	15	3
Time	12760	63	547	4923	9716

TABLE II.
PERFORMANCE COMPARISON

Item/data	Non-hierarchical	Hierarchical
Total processing time	12760	15249
First response time	12760	63

VI. Conclusion

Aim at the application characteristics and needs of the online log analysis system, this paper analyzes the necessity of using the hierarchical file system to manage the log data, and proposes the hierarchical file archiving mechanism and some specific archiving methods. Furthermore, one or more suitable archiving methods can be selected to form the more complete solution in the practical application. Simulation experiments show that after taking the hierarchical file archiving mechanism, the system response time can be significantly shortened. Hierarchical archiving mechanism is also beneficial to the system for selecting the appropriate data sets to process, which can avoid useless data operate, and improve the processing speed and system performance.

Acknowledgment

This work was supported by Henan Province Science and Technology R&D Program (No.: 122102310474), Henan Province Basic and Frontier Technology Research Projects (No.: 122300410071) and Zhumadian Municipal Science and Technology Research Projects (No.: 11314).

REFERENCES

[1] T. A. Sang. "Log-based approach to make digital forensics easier on cloud computing," *Proc. of the 2013 3rd Inter. Conf. on Intelligent System Design and Engineering Applications, ISDEA2013*, pp. 91-94, 2013.

[2] D. Wang, J. Chen, W. B. Zhao. "A Task Scheduling Algorithm for Hadoop Platform," *Journal of Computers*, vol. 8, no. 4 (2013), pp. 929-936, Apr 2013.

[3] S. B. Ren, D. Muheto. "A Reactive Scheduling Strategy Applied On MapReduce OLAM Operators System," *Journal of Software*, vol. 7, no. 11 (2012),pp. 2649-2656, Nov 2012.

[4] S. Chen, N. An, L. Li, Y. W. Wu, W.M. Zheng, L. Sun. "Human dynamics revealed through log analytics in a cloud computing environment," *Lecture Notes in Computer Science* , vol. 7923 LNCS, pp. 58-63, 2013.

[5] J. H. Lee, M. W. Park, J. H. Eom, T. M. Chung. "Multi-level intrusion detection system and log management in cloud computing," *Inter. Conf. on Advanced Communication Technology, ICACT*, pp. 552-555, 2011.

[6] R. R. Bhandari, N. Mishra. "Encrypted IT auditing and log management on cloud computing," *Inter. Journal of Computer Science Issues*, vol. 8, no. 55-1, pp. 302-305, Sep. 2011.

[7] M. Cheng, H. P. Chen. "Weblog Mining Based on Hadoop," *Computer Engineering*, vol. 37, no. 11, pp. 37-39, 2011.

[8] F. Y. Yang, H. C. Liu. "Research in HDFS based on Campus Network Environment," *Proc. of 2011 International Conference on Image Analysis and Signal Processing*. Wuhan, China, 2011, pp. 648-652

[9] C. B. Huang, J. L. Wang, H. J. Deng, J. Chen. "Mining Web Logs with PLSA Based Prediction Model to Improve Web Caching Performance," *Journal of Computers*, vol. 8, no. 5 (2013), pp. 1351-1356, May 2013.

[10] J. G. Lou, Q. Fu, Y. Wang, J. Li. "Mining dependency in distributed systems through unstructured logs analysis," *Operating Systems Review (ACM)*, vol. 44, no. 1, pp. 91-96, 2010.

[11] M. Philippe, N. Syed et.al. "A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures," *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 1510-1517.

[12] W. Xu, L. Huang et.al. "Detecting large-scale system problems by mining console logs," *Proc., 27th International Conference on Machine Learning*. Haifa, Israel, 2010, pp. 37-44.

[13] H. Y. Yu, D. Sh. Wang. "Mass log data processing and mining based on Hadoop and cloud computing," *Proceedings of 2012 7th International Conference on Computer Science and Education*, Melbourne, Australia, 2012, pp. 197-202.

[14] J. Therdphapiyanak, K. Piromsopa. "Applying Hadoop for Log Analysis toward Distributed IDS," *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, Kota Kinabalu, Malaysia, 2013.

[15] W. Zhou, J. F. Zhan, D. Meng, Zh. H. Zhang. "Online Event Correlations Analysis in System Logs of Large-Scale Cluster Systems," *Proceedings, International Conference on Network and Parallel Computing*. Zhengzhou, China, 2010, LNCS6289, pp. 262-276.

[16] C. J. Jiang. "The Function of Log Analysis in Network Security," *New Technology of Library and Information Service*, vol.20, no. 12, pp. 58-60, 2004.

[17] X. C. Dong "HDFS small file problems and solutions", *http://dongxicheng.org/mapreduce/hdfs-small-files-solution/*.

[18] W. B. Chen, X. J. Zhang, L. LI, J. Tang. "A distributed system of log analysis based on Hadoop," *Journal of Guangxi University (Natural Science Edition)*, no. S1, pp. 339-342, 2011.

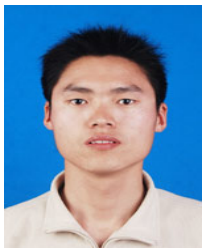
[19] G. M. Hu, L. Zhou, L. X. KE. "Research on Hadoop-based Network Log Analysis System," *Computer Knowledge and Technology*, no. 22, pp. 6163-6164+6185, 2010.

[20] A. Q. Song. "Design and completment of Hadoop-based log analysis system," *Beijing: China University of Geosciences*, 2012

[21] The Apache Software Foundation, "HDFS 0.21 Documentation," http://hadoop.apache.org/hdfs/docs/r0.21.0/.

[22] White T. *Hadoop: The Definitive Guide, 2nd Edition.* O'Reilly Media / Yahoo Press, 2010.

Fengying Yang was born in 1978. She received the M.S. degree in computer technology from Wuhan University in 2011. She is a lecturer at Huanghuai University, Zhumadian, China. Her research interests include cloud computing and computer network.



Huichao Liu was born in 1982. He received the M.S. degree in computer technology from Wuhan University in 2011. He is a lecturer at Huanghuai University, Zhumadian, China. His current research interests are cloud computing and intelligent computation.

Zhanping Zhao was born in 1965. He received his Ph. D. degree in probability and mathematical statistics from Yunan University in 2008. He is a professor, master tutor in department of economic management of Huanghuai University, Zhumadian, China. His current research interests include bayesian inference and computer techenolog.