Computer Science and Information Systems 15(3):533–548 https://doi.org/10.2298/CSIS171015021J

# An Optimized Method of HDFS for Massive Small Files Storage

Weipeng Jing[1,2], Danyu Tong[1], GuangSheng Chen[1],Chuanyu Zhao[2], and LiangKuan Zhu[1]

[1] College of Information and Computer Engineering, Northeast Forestry University, Harbin, China
weipeng.jing@outlook.com, {nefuchensy, kjc_chen}@163.com
[2] Heilongjiang Computing Center

**Abstract.** The development of the Internet-of-Things (IoT) and the Cyber-Physical System (CPS) has greatly facilitated many aspects of technological applications and development. This may lead to significant data growth, especially for small files. The analysis and processing of a large number of small files has become a crucial part of the development of IoT and CPS. Hadoop Distributed File Systems have become powerful platforms to store a larger amount of big data. However, this method has a number of issues when dealing with small files, such as substantial memory consumption and poor access. In this paper, a Dynamic Queue of Small Files (DQSF) algorithm is proposed to solve these problems. DQSF differentiates small files into different categories using an analytical hierarchal process that examines the performance of small files with different ranges across four indexes and determines the size of the dynamic queue according to the best system performance. Additionally, period classification is applied to preprocess the small files before storage, and the prefetching mechanism of the secondary index is used to process index tables. Experimental results show that this method could effectively reduce memory use and improve the storage efficiency of massive small files, which optimizes system performance.

**Keywords:** WSN, HDFS, massive small files, Dynamic Queue, Analytic Hierarchy Process.

## 1. Introduction

Wireless Sensor Networks (WSN) are distributed sensor networks in which end nodes monitor specific things and the accompanying data is transmitted back for the unified analysis process using wireless communications. Since WSN deployment costs have steadily decreased, it is widely used in many areas, such as environmental monitoring, medical manipulation, underwater sensing, interplanetary exploration, and others [2]. The large scale of a WSN can be divided into two aspects [3]. First, the sensor nodes are distributed over a wide area, such as in forest monitoring and forecasting [4]. Second, the sensors' density is large, which means a large number of sensors are densely arranged over a specific area [5]. This leads to a large data set and singular data being very small.

The Internet-of-Things (IoT) and the Cyber-Physical System (CPS) are based on WSNs. As the core technology of next generation networks, they are widely analysed by the government, academia and business circles. According to the definitions of ITU and PCAST, the IoT is an interconnected network [6]. It connects everything with the Internet in accordance with certain agreements through information sensing devices, exchanges information and communicates to achieve intelligent identification, positioning, tracking, monitoring and management [7-8]. A CPS is a multidimensional complex system that integrates computing, communications and the physical environment. It realizes real-time perception, dynamic control and information services in an engineered system through organic integration and depth cooperation of communication, computing and consumer electronics (3C) technology. IoT and CPS connect geographically distributed heterogeneous embedded devices through high-speed and stable networks that achieve information exchange, resource sharing and collaborative control [9]. They have broad market prospects and significant economic benefits. They are the inevitable trend of future network evolution. CPS is the theoretical core and technical connotation of IoT. IoT is the external manifestation of CPS's primary stage.

With the rapid development of information technology, IoT and CPS are widely used. They are closely related to people's life and social development and have wide applications for military affairs, including aerospace, military reconnaissance, intelligence grid system, intelligent transportation, intelligent medical, environmental monitoring, and industrial control. The data generated during the process will be immense.

Hadoop is an open source implementation of Google cloud computing [10]. As an open source-distributed computing framework, scholars and commercial applications can apply it to their research [11]. Hadoop has high scalability and can be built on any cheap hardware and normal operations will not be affected. In theory, it can arbitrarily increase the number of clusters according to the needs of the application. Additionally, it can provide users with a stable interface for operating the cluster.

HDFS is a distributed file system of Hadoop that possesses the advantages of Hadoop such as highly reliability, scalability and high fault tolerance [12]. HDFS is a master-slave distributed system [13] comprised of one Namenode and several DataNodes. The NameNode is responsible for the overall structure of the regulation, which stores all of the cluster index information, similar to the book directory structure information. DataNode is only responsible for storing real data and sending periodic heartbeat reports to the NameNode. This master-slave structure management style effectively simplifies the operation of the system from a certain sense, but the disadvantages are also obvious that the size of the NameNode limits to the expansion of the system. The number of DataNodes cannot be gradually increased to meet the needs of application. Especially when it encounters small files, the limitation becomes more obvious.

The block size in Hadoop 1.0 is 64 M, and it increased to 128 M in Hadoop 2.0. Data are stored on blocks, and metadata are saved on the NameNode. There are approximately 150 bytes of metadata information that must be kept in the NameNode and achieves 368 bytes by using three copies set in a file [14]. When storing small files, every file less than a block will occupy one block. This resulted in the phenomenon that when we store small files using HDFS, each one occupies a block of data, and the NameNode takes approximately 2 GB of memory space. If the small files are stored, it will take up

approximately 20 GB of memory space. That is a great challenge for the NameNode. The upload or read rate for small files is very poor. Therefore, an effective method is urgently needed that can solve the problems when HDFS stores massive small files.

In this paper, we propose a dynamic queueing small file method to solve data storage problems in IoT and CPS. The remainder of this paper is organized as follows. Section 2 introduces the related situation regarding IoT and CPS and the method for solving massive small files storage. Section 3 shows the method for text categorization and the analytical hierarchal process. Section 4 describes the method of dynamic queueing and the prefetching mechanism. Section 5 shows the experimental results and analysis. Finally, conclusions are presented in Section 6.

## 2. Related Work

The earliest WSN system was developed and put into use by the US military in 1998, and it has achieved great success. In China, the research and application of WSNs are almost synchronous with developed countries. It first appeared in 1999 at the Chinese Academy of Sciences Research conference. In recent years, WSNs have been widely used in commercial and civil areas [15-16]. Applications include environmental monitoring, medical manipulation, undersea searching, interplanetary exploration, military, and others [17-18].

The Internet and WSN laid the foundation for the development of IoT and CPS. Microsoft President Bill Gate first proposed the concept of IoT. After 2000, with the deep development of the Internet, the equipment and technology related with IoT have matured. In 2009, the United States proposed the Smart Planet, and IoT is taken as an important component for world development [19].The concept of CPS was first proposed by the American Natural Science foundation. In 2006, the United States clearly identified CPS as an important research project. So far, IoT and CPS have been integrated into the information and physical worlds. They will become a new strategic industry and the next turning point to promote economic and social development. Experts and scholars at home and abroad continue to study it. With continuous technological development, the amount of data generated by an application is also very large. However, the research on data storage processing is flawed [20].

Merging files to solve the problem of the storage of massive small files is both one of the most widely applied tools and an effective solution.

The methods of merging files in Hadoop include HAR [21], SequenceFile [22] and MapFile. The HAR solution uses the MR procedure of Hadoop to pack the files, which reduces memory usage but requires extensive time. The original small files require users to manually delete items and have other issues.

SequenceFile uses the <key, value> storage format, which reduces memory usage and improves efficiency. However, SequenceFile cannot ensure the orderly storage of documents, which means there are no maps between big files and small files. Therefore, it is very difficult to read files. MapFile is equivalent to two SequenceFiles where one stores data and the other one is for the index map, which solves the SequenceFile search problem. However, massive small files require a long index which will restrict query efficiency.

For the needs of industries and applications, a number of specific solutions have gradually emerged to solve the problem of massive small files storage. In [23], a new file storage strategy was proposed that files in the same geographical proximity were stored sequentially to reduce I/O access frequency for the continuity and relevance of WebGIS. However, it did not consider the establishment of indexing. For the characteristics of PPT and MP3 files, respectively, new ideas were proposed in [24-25] that merged the storage of files that are in the same course or chapter file and the generated secondary index (which is beneficial for reading). However, this scheme was targeted at small files that have clear boundaries. Therefore, the method has limitations.

We proposed a method that established a widget to use additional I/O for controlling the merging and access to small files in [26]. However, this approach is more complex. A way to queue merging based on SequenceFile was proposed in [27]. It set an optimal size for the queue at 400 and controlled merging storage by detecting the system load. However, small files of less than 4.35 M will not have the same queue to be adapted.

Therefore, in this paper, we propose a dynamic queue that is designed to process massive small files around the queue size of 400. We use file dependencies to make determinations while establishing a secondary index and prefetching policy for the same large merged files.

## 3. Ready Work for Optimize Storage

In this paper, we provide a solution to solve the problem of large memory storage occupancy, slow storage and reading speed when storing massive small files. Before uploading small files, we use a period classification algorithm to classify files, merge content-related files and store closed files. Additionally, we use an analytical hierarchal process to compute each factor's weight.

### 3.1. Period Classification Algorithm

The period classification algorithm was first proposed to solve the problem of repeated Chinese website recommendations [28]. This algorithm determined the degree of repetition or similarity between two text files. In this paper, the period classification algorithm will be used to classify the text files and store the related small files. File classification, as the premise of merging, is very important for the establishment of a file prefetching policy and the promotion of file reading.

The advantage of the period classification is to use symbols to define sentences. Then the data are classified. It is suitable for data distribution defined by symbols in IoT and CPS.

The period classification algorithm takes periods S, which appear in articles as the text characteristics. Its value is 10 characters before each period corresponding to. Take all characters for sentences of less than 10 characters as the characteristic value set. Each characteristic set corresponds to one document that contains all the periods. Then, determine the degree of repetition or similarity between two text files to achieve file classification. Suppose text A and text B correspond to the characteristic set of $a$ and $b$,

where $a$ is $a$ : $\{s_{a1}, s_{a2}, \ldots, s_{an}\}$ . Suppose there is similarity between them using Common Similarity (CS). Then,

$$CS(a,b) = \left| a \cap b \right| \Big/ \min\{|a|,|b|\} \quad (1)$$

where $|a|$ represents the number of period classifications in characteristic set $a$ . $|a \cap b|$ represents similar sentence numbers. Add 1 to $|a \cap b|$ when there are 5 similar characters between $s_{ai}$ , $s_{bj}$ and then judge whether they are similar to each other when $CS(a, b)$ is more than the threshold.

### 3.2. Investigating System Performance

The analytical hierarchal process is a method with multi-index comprehensive evaluation for solving the problem of changing performance with respect to quantity. In this paper, we calculate each respective weight according to the four parameters of the system's contributions during the processing of small files. Finally, quantify system performance is obtained. The concrete steps are as follows.

**Table 1.** Evaluation scale and the meaning

| Scale | Meaning |
|---|---|
| 1 | Comparison of two factors that have the same importance. |
| 2 | Comparison of two factors in which one is slightly more important than the other. |
| 3 | Comparison of two factors in which one is significantly more important than the other. |

Step one: Set the value of the evaluation scale $w_i$ for x. The original evaluation scale is numbered 1-9 where we decide to improve it. As each factor has a slightly different impact on system performance, each factor plays a decisive role in the overall analysis. Therefore, the differences between the importance of each one is not large. Therefore, set the three evaluation scale to construct the judgement matrix, as shown in Table 1.

Step two: Calculate the judgement matrix P. The judgement matrix directly reflects the recognized value of each factor and their importance in the practical application. According to the $w_i$ of step one, compare every two factors and then get the judgement matrix P.

$$P = \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{pmatrix} \quad (2)$$

$P_1$ , $P_2$ , $P_3$ , and $P_4$ in (2) represent the accessing time, merging time, the storage time, and the memory saved.

$$p_{ij} = {w_i}/{w_j} .(1 \le i, j \le 4) \tag{3}$$

$p_{ij}$ means the importance of factor $i$ to factor $j$ . The value of divisor is in table 2 and the illustration is in table 1.

Additionally,

1) $p_{ij} > 0.$

2) $p_{ij} = p_{ji}$

3) $p_{ii} = 1$

Step three: Check the consistency of the judgement matrix and correct for inconsistencies. The test for judgement matrix consistency evaluates its reasonableness. The calculation of test indicators $CI$ is as follows.

$$CI = {\lambda_{\max} - n}/{n-1} \tag{4}$$

$\lambda_{\max}$ is the maximum eigenvalue of the judgement matrix. $n$ is the number of factors, where $n = 4.$

$$CR = {CI}/{RI} \tag{5}$$

$RI$ is the average value of all random $CR$ . $RI$ is a fixed value. $CR$ is the consistency ratio. We say that the judgement matrix has satisfactory consistency when $CR \le 0.1$. The results that are obtained by reference [27] are calculated and corrected. Then, we get the following weight table:

**Table 2.** Weight of each factor

| **Factor** | Access Time ( $P_1$ ) | Memory Saved ( $P_2$ ) | Merge Time ( $P_3$ ) | Storage Time ( $P_4$ ) |
|---|---|---|---|---|
| **Weight** | 0.449 | 0.287 | 0.139 | 0.125 |

The features of HDFS are supported during storage and multiple accesses. Therefore, the access time is more important than the upload time. Additionally, the second one is memory saved due to restricted system expansion. Storage time has uncontrollable factors such as network conditions. Therefore, the importance of merge time is slightly greater than storage time [28-32].

# 4. Optimize Storage with DQFS

Creating a dynamic queue according to file size will guarantee that the merged files retain better system performance. By establishing a caching mechanism based on the characteristics of near-file access probability will greatly improve accessing speed.

## 4.1. DQFS Method

Small file storage optimization has been researched for many years. The merger proposal is one very effective method that is often used. The corresponding small files are combined to obtain a large file and then addresses the large files for storage. However, the number of small files that should be merged is not well defined. Bracket [27] provided a value of 400. However, a queue of size 400 cannot adapt to all ranges of small files. Therefore, dynamic queueing was proposed. Dynamic queueing makes different queue sizes for all ranges of small files. The process is as follows.

First, classify the small files before storage.

Second, choose an adopted size for queue according to the size of small files. The system will give the sign of when the queue is full and time has expired. The system will also give the sign of TF when the number of remaining small files is less than the queue's size.

Thirdly, the system will merge small files in the queue if each sign for the two appears to concurrently form big files and generate a secondary index. The secondary index will be introduction in section 4.2.

Finally, the big files are stored that are merged in HDFS.

## 4.2. Secondary Index and Prefetching Mechanism

Establish a secondary index for merged files. The first index saves the metadata messages in HDFS for big files, and the secondary index establishes the mapping between big and small files. The secondary index includes filename, fileindex, filelength and bigfilename. The format is shown in the following table 3.

**Table 3.** Secondary index

| filename | fileindex | filelength | bigfilename |
|---|---|---|---|
| a | 0 | n | x |
| … | … | … | … |

The sample in table 3 is a sm。all file named "a" with n bytes and a fileindex of "0", and it lies in big file "x".

Access the file through small file name to find the corresponding big file. Additionally, use fileindex and filelength to get the small file as the returned value.

Since the file prior to storage is classified, here we set the prefetching mechanism in accordance with file correlation. This arrangement assumes that when one small file is

accessed, it is highly probable that the next one with similar content will then be accessed. Put all the small files messages that are in the same big file with the small file access, which is convenient to search and can reduce I/O access frequency. It enhances storage and access efficiency.

## 5. Experiment

In this section, we will get the range of small files through experiments. To make the experiment convenient, we choose 4.35 M as the upper bound. We will get the optimal size of queueing for every range of small files, which is the dynamic queue.

### 5.1. Experimental Platform and Experimental Data

In this paper, we will use the Hadoop cluster that consists of five nodes in which one node is the master, and the remaining four are slaves. The host computer is a Shuguang I450-G10 tower server that utilizes an InterXeon E5-2407 quad-core 2.2 GHZ processor, 8 GB memory, and a 300-GB hard drive. The node system is Centos 6.4, and the JDK version is jdk-6u31- linux-i586. The Hadoop version is hadoop-2.5.2. We use 100 M/s Ethernet and the nodes use 1000 M/s Ethernet.

In experiment 1, we construct nine groups of data. Each group contains 10,000 small text files. The size of the small files of the nine groups is approximately 0.5 M, 1 M, 1.5 M, 2 M, 2.5 M, 3 M, 3.5 M, 4 M, and 4.5 M.

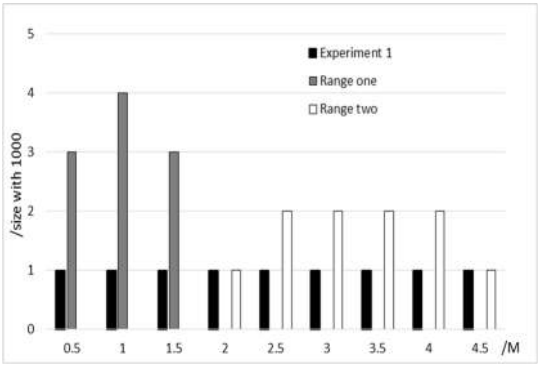


**Fig. 1.** Data distribution

We choose the files of 0.5 M, 1 M, and 1.5 M with the proportion of 3:4:3 in the data collection for range one. Range two uses 2 M, 2.5 M, 3 M, 3.5 M, 4 M, and 4.5 M with the proportion of 1:2:2:2:2:1. Every treatment has 10,000 small files. The reason for this assignment is so that each range of data is even and universal. The experimental data are shown in Figure 1.

### 5.2. Determine Range for Small Files

First, similar to the results obtained from the literature [27], our experiments set the size of the test queue as 400. Observe the four factors under the nine groups of data and receive system performance trends based on the analytical hierarchal and data standardization processes. System performance is shown in picture 2.

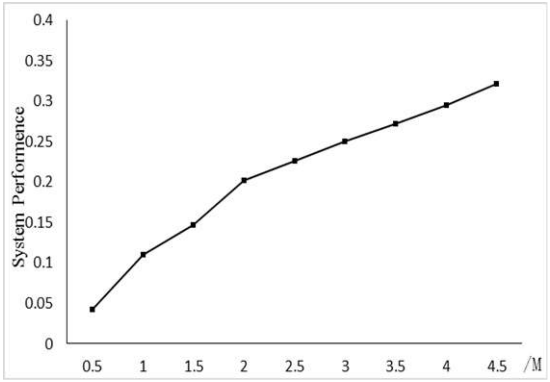


**Fig. 2.** System performance

The picture shows that small files in the 0-2 M range have similar system performance trends, while the performance of small files in the 2-4.5 M range experience linear growth. Data under the same trend showed similar characteristics. The four indexes of file reading, memory usage, combined efficiency and upload time for small files in each trend are explored. Divide small files less than 4.5 M into two ranges. One range is 0-2 M and the other is 2-4.5 M. The following results are based on the two ranges.

### 5.3. The Four Factors Situation

To achieve effective experimental results, each group of experiments uses the nine queues as independent variables. Queue size uses 400 as the centre and selects four at each side of 400. These queues are 200, 250, 300, 350, 400, 450, 500, 550, and 600 wide. Test the four indexes of file reading, memory usage, combined efficiency and upload time to get their values and the optimal alignment of each range.

#### 5.3.1 Access Files

Access 50 files 10 times over different time periods. The average access time for each file is used as the experimental data. Finally, get the access for the file over the two data ranges. The results are shown in Figure 3.

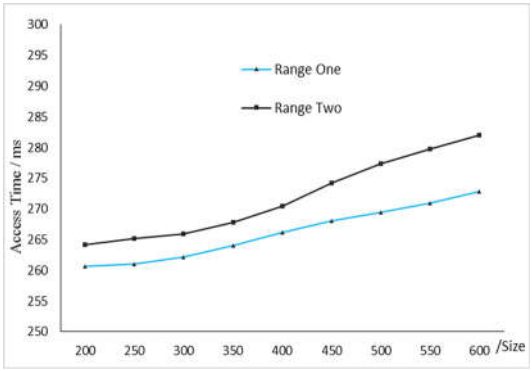


**Fig. 3.** Access time of the two ranges

The two sets of experiments are of different queue sizes and read 10000 small files. Experimental results show that the average access time in range one is 265.7ms, and the time in range two is 273.2ms. Since the data in range two is bigger than range one, it uses more time than range one. The average time for Hadoop with the HAR to read the 10000 small files is approximately 500ms. Therefore, the file reading efficiency is significantly improved after applying queue merging.

### 5.3.2 Memory Saved

Next, compute the memory saved when merging 10000 small files. Use nine queue sizes to merge files and store the merged big files into HDFS. The memory saved equals the storage time for unmerged files minus the storage time for merged files. The results are shown in Figure 4.

This saved memory is compared to the time it takes to directly store the data into HDFS. Range one has smaller data, so it is better than range two. After a size of 450, range one smooths out, while this occurs for range two after a size of 350. The experiment shows that the saved file memory is better than the scope of the two. It shows that the queue merging method will obtain better results with smaller files. Memory usage is one of the biggest problems encountered when storing small files. Therefore, memory savings under small file storage is essential.

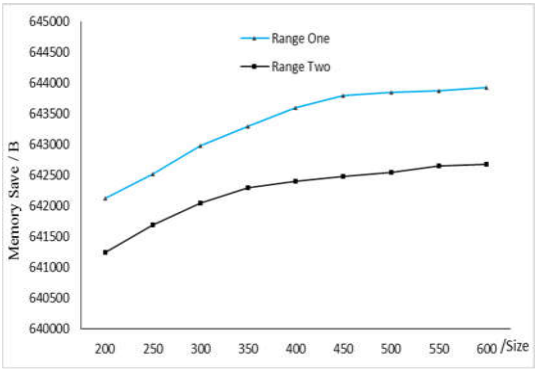


**Fig. 4.** Memory saved

### 5.3.3 Merge Time

We merge the small files according to the nine queue sizes. To ensure the validity of data, we compute the statistics 10 different times, and we take the average time as the result. These results are shown in figure 5.

Due to the bigger data of range two, its merge time is longer than for range one. When queue size increases, range one's time gradually decreases. However, the situation of range two obeys that of range one. This shows that big files are suitable for small queues and the small files are adapted to large queues. The average merge time for range one is 61ms and 80ms for range two.

### 5.3.4 Store Time

Store files that are merged from 10,000 small files into HDFS with different queues sizes. Every data group is conducted 10 times. Take the average value as the result, which is shown as follows.

The average storage time for merged files in range one is 8.6 minutes and for unmerged files is 18.5 minutes. Due to the bigger data of range two, the average storage time for merged files is 32.8 minutes and for unmerged files is 41.4 minutes because the total number of small files is 10,000. Different queue sizes have different merged files. The experiment proves that storage time is related to queue size and the total number of queues. Different sizes of small files achieve optimal results at different points of each queue.

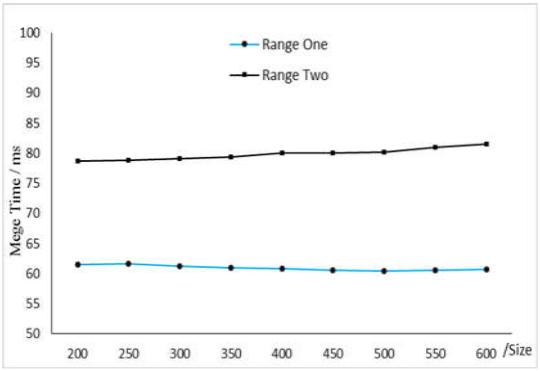


**Fig. 5.** Merge time

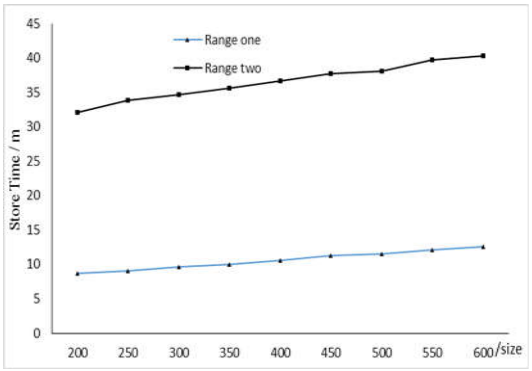


**Fig. 6.** Storage time

### 5.4. Result Analysis

Analyse the data for the four factors above with data standardization and use an inverse index to make analysis. This indicates that system performance is optimal when results are smaller. The situation is shown in the following figures.

We use an inverse index to analyse the data. System performance is optimal when it is smallest. Figures 7 and 8 show that performance first rises and then falls, and each arrives at the minimum value at different points. We obtain two different sizes around a queue size of 400. According to the experimental results above, range one has smaller files that correspond to an optimal queue of 450 and range two's optimal queue is 300.

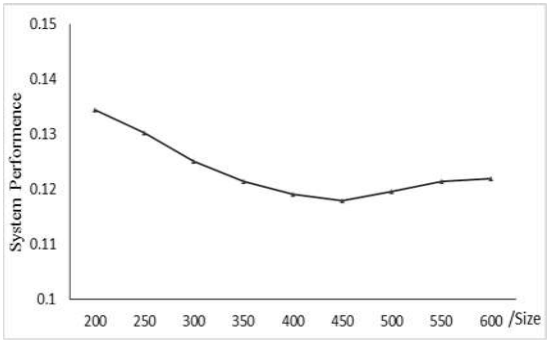


**Fig. 7.** System performance of range one

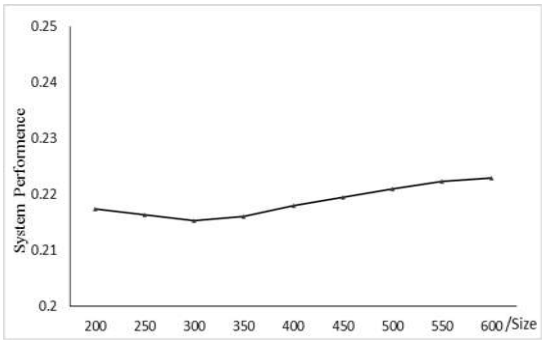


**Fig. 8.** System performance of range two

## 6. Conclusion

We aim to correct the problems of IoT and CPS in that HDFS is poor in dealing with massive small files generated by corresponding social network applications. We propose a method of dynamic queueing that uses an analytical hierarchal process to analyse system performance and addresses differing ranges for small files. We also classify the text files with a period classification algorithm. We choose suitable queues for different sizes of small files and then merge the files in the queue. We store merged files to save memory. Then, it generates a secondary index and uses a file prefetching strategy to improve the efficiency of file access.

Our next works will aim to optimize the algorithm of classified text to improve access time and improve the index storage method.

**Acknowledgments.** The work described in this paper is supported by National Natural Science Foundation of China (31770768), the Natural Science Foundation of Heilongjiang Province of

China (F2017001), the Fundamental Research Funds for the Central Universities (2572017CB32) and China Forestry Nonprofit Industry Research Project (201504307).

## References

1. S Liu, Z Pan; H Song*. Digital image watermarking method based on DCT and fractal encoding, IET Image Processing, 2017, 11(10) :815-821
2. V. Kumar. DESIGN OF A LOW COST WIRELESS SENSOR NETWORK: Low-cost, Low-power, Compact WSN[M]. Saarbrücken : Vdm Verlag Dr. Müller: 2010
3. Liu S, Fu W, He L, et al. Distribution of primary additional errors in fractal encoding method[J]. Multimedia Tools and Applications, 2017, 76(4): 5787-5802
4. Wolter P T, Townsend P A. Multi-sensor data fusion for estimatingforest species composition and abundance in northern Minnesota[J]. Remote Sensing of Environment, 2011, 115(2):671-691
5. Faheem M, Abbas M Z, Tuna G, et al. EDHRP: Energy efficient event driven hybrid routing protocol for densely deployed wireless sensor networks[J]. Journal of Network & Computer Applications, 2015, 58:309–326
6. Kafle V P, Fukushima Y, Harai H. Internet of things standardization in ITU and prospective networking technologies[J]. 2016, 54(9):43-49
7. S Liu, Z Pan, X Cheng*. A Novel Fast Fractal Image Compression Method based on Distance Clustering in High Dimensional Sphere Surface, Fractals, 2017, 25(4): 1740004
8. Peng L, Wang Q, Yu A. Internet of Things technology-based management methods for environmental specimen banks.[J]. Environmental Science and Pollution Research, 2015, 22(3):1612-9
9. Laibinis L, Klionskiy D, Troubitsyna E, et al. Modelling Resilience of Data Processing Capabilities of CPS[M]. Software Engineering for Resilient Systems. Springer International Publishing, 2014
10. Tom White. Hadoop: the definitive guide, 2nd edition[M]. Beijing: Tsinghua University Press, 2011
11. X. Fan, H. Song, X. Fan, and J. Yang, "Imperfect information dynamic stackelberg game based resource allocation using hidden Markov for cloud computing", IEEE Trans. Services Comput., vol. PP, no. 99, p. 1, Feb. 2016, doi: 10.1109/TSC.2016.2528246
12. DONG B,ZHANG Q, TIAN F, et al. An optimized approach for storing and accessing small files on cloud storage[J]. Journal of Network and Computer Application, 2012, 35(6): 1847-1862
13. Konstantin Shvachko, Hairing Kuang, Sanyjy Radia, et al.The Hadoop Distributed File System[C]. IEEE International Conference on Mass Storage Systems and Technologies (MSST).2010.1-10
14. Song H, Li W, Shen P, Vasilakos A. Gradient-driven parking navigation using a continuous information potential field based on wireless sensor network. Information Sciences, 408(C), 100-114, DOI information: 10.1016/j.ins.2017.04.042, OCT 2017
15. Dong Xicheng. Hadoop HDFS [EB/OL]. [2015-04-13]. http://dongxicheng.org/
16. Tom White. The Small Files Problem [EB/OL]. http: //blogcloudera. Com/blog/2009/02/the-small-files-problem
17. Bauer J, Siegmann B, Jarmer T, et al. On the potential of Wireless Sensor Networks for the in-situ assessment of crop leaf area index[J]. Computers & Electronics in Agriculture, 2016, 128:149-159
18. Lounis A, Hadjidj A, Bouabdallah A, et al. Healing on the cloud: Secure cloud architecture for medical wireless sensor networks[J]. Future Generation Computer Systems, 2015, 55 :266-277

19. Zaidi, Slim; El Assaf, Ahmad; Affes, Sofiene. Accurate Range-Free Localization in Multi-Hop Wireless Sensor Networks[J]. IEEE Transactions on Communications,2016,64(9), 3886-3900
20. Mirsadeghi M, Mahani A. Energy efficient fast predictor for WSN-based target tracking[J]. Annals of Telecommunications, 2015, 70(1):63-71
21. Yang XL, Shen PY, Zhou B. Holes detection in anisotropic sensornets: Topological methods. International Journal of Distributed Sensor Networks. 2012 Oct 23;8(10):135054
22. Fleisch E. What is the Internet of Things?: An Economic Perspective[J]. Economics Management & Financial Markets, 2010, 241(2):33
23. Laibinis L, Klionskiy D, Troubitsyna E, et al. Modelling Resilience of Data Processing Capabilities of CPS[M]. Software Engineering for Resilient Systems. Springer International Publishing, 2014
24. Hadoop[EB/OL]. apache.org/jira/browse/HADOOP-1687
25. SequenceFile[EB/OL]./wiki.apache.org/hado-op/SequenceFile
26. Xuhui Liu, Jizhong Han, Yunqing Zhong,et,al. Implementing WebGIS on Hadoop:A case study of improving small file I/O performan- ce on HDFS[C]. IEEE International Conference on Cluster Computing and Workshops. 2009.1-8
27. BoDong,JieQiu,QinghuaZheng,XiaoZhong, Jingwei Li, Ying Li .A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files[C]. IEEE International Conference on Services Computing. 2010.65-72
28. Zhao Xiaoyong, Yang Yang, Sun Lili, Chen Yu. Hadoop based storage architecture for mass MP3 file[J]. Journal of Computer Applications.2012,32(06): 1724-1726
29. Zhang Y, Liu D. Improving the Efficiency of Storing for Small Files in HDFS[C]. IEEE International Conference on Computer Science & Service System (CSSS).2012. 2239 – 2242
30. YU Si, GUI Xiaolin, HUANG Ruwei, ZHUANG Wei. Improving the Storage Efficiency of Small Files in Cloud Storage[J]. JOURNAL OF XI′AN JIAOTONG UNIV-ERSITY.2011.59-63
31. Wei Y, Yuan C, Huang Y. CCDet:An Efficient Detection Method for Large-Scale Duplicate Chinese Web Pages[J]. Journal of Computer Research & Development, 2013. 140-15
32. Weipeng Jing, Danyu Tong, Yangang Wang , Jingyuan Wang, Yaqiu Liu, Peng Zhao, MaMR:High performace MapReduce programming model for material cloud application, Computer Physics Communications, 2017, 211:79-8

**Weipeng Jing**, Received Ph.D. degree from Harbin Institute of Technology of China. He is currently an associated professor in the Northeast Forestry University, China. His research interests include modelling and scheduling for distributed computing systems, High performance computing and system reliability, cloud computing, spatial data mining. He has published more than 60 research articles in refereed journals and conference proceedings, including IEEE TIP, IEEE TSP, IEEE TOC, CPC, PUC, FGCS,etc. Besides, he served as an Publication chair of ICYCSEE (2016,2017,2018) and Wicon(2017),served as a Program Committee (PC) member for several popular international conferences, including IIKI 2018, CollaborateCom 2017, SC 2016, etc. He is now a member of the IEEE, ACM, and a member of the China Computer Federation (CCF).

**Danyu Tong** received the B.S. degree in Computer Science and Technology from Northeast Forestry University, Harbin, China, in 2015. She is currently pursuing the

master's degree with Northeast Forestry University. Her current research interests include image classification, and distributed computing.

**Guangsheng Chen** is currently Doctoral Supervisor and Professor with Northeast Forestry University, China. He is the member of national innovation methods research institute, executive director of education information technology council of Education Ministry. His research interests include biomass material prediction, intelligent detection of new composite materials and big data on forestry. He has published over 100 academic papers and one monograph. He is the corresponding author.

**Chuanyu Zhao** is a senior engineer. He is currently the technical director of Heilongjiang Computing Center, director of Provincial Key Laboratory of industrial process computer control simulation, leader of echelon of talents in the discipline of "chemical machinery and equipment", recipient of special allowance from provincial government, and postdoctoral research workstation of Heilongjiang Computing Center. Post doctoral cooperative mentor. Long-term engaged in traditional Chinese medicine, natural plant modern extraction and separation process equipment and its intelligent technology and process control technology in the process industry and other fields of interdisciplinary independent innovation research. A number of key technologies and major new products with leading domestic level and gaps in China have been completed. In the past five years, he has presided over the research and development of more than 20 national, provincial and municipal scientific research projects. As an actual inventor, he holds 2 inventions and 10 utility model patents. Many technologies and products have been transformed into industries. He has obtained many domestic leading scientific research achievements and won three second prizes for scientific and technological progress in Heilongjiang Province.

**Liangkuan Zhu** received his BS and MEng in Mathematics from the Bohai University, Jinzhou, PR China, in 2001 and 2004, respectively. He received his PhD in Control Science and Engineering from the Harbin Institute of Technology in 2008. Since 2008, he has been with the School of Electromechanical Engineering at Northeast Forestry University and is currently an associate professor. From 2013-2014, he is a visiting scholar in the Department of Mechanical and Industrial Engineering at Concordia University, Montreal, QC, Canada. His research interests include network based control, robust control, intelligent control and application.

*Received: October 15, 2017; Accepted: July 22, 2018.*