Computer Communications 157 (2020) 124–131

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Optimized data storage algorithm of IoT based on cloud computing in distributed system

Mingzhe Wang, Qiuliang Zhang [*]

China Academy of Railway Sciences, Beijing 100081, China

A R T I C L E I N F O

Keywords:
Cloud computing
Data access
Storage
Hash algorithm
Network topology

A B S T R A C T

The existing Internet of Things(IoT) uses cloud computing data access storage algorithms, that is, the hash algorithm has defects of low data processing efficiency and low fault tolerance rate. Therefore, HDFS is introduced to optimize cloud computing data access storage algorithms. HDFS is first used to optimize the data access storage architecture according to problems of data access storage architecture in the Internet of Things, in which factors of data access storage distribution in the IoT are fully considered, and hash values are used to optimize the configuration of data access information storage locations, so that data access storage distribution strategy can be optimized. Then, the topology of the IoT is optimized, and data block size is also optimized with effect algorithm. Finally, the design of file storage is optimized. Through simulation experiments, it is proved that the optimized cloud storage method has obvious performance advantages in file read and write speed as well as memory usage. Compared with the traditional hash algorithm, optimization algorithm proposed in the paper greatly improves file upload and download efficiency, data processing efficiency and fault tolerance rate, which fully demonstrates that the proposed cloud computing data access storage optimization algorithm is more superior.

1. Introduction

Cloud computing is mainly based on the Internet technology and provides users with services, uses, and interactions. Generally, dynamic and proliferative and virtualized resources are provided through the Internet. Among them, “Cloud” is a metaphor for the Internet and the network, which is mainly an abstract representation of the Internet and infrastructure. There are many definitions for cloud computing, and most of them are accepted by the American Institute of Technology. Today, cloud computing is a payment-based model that provides convenient, available, on-demand network access, and it only required less effort to interact a bit with the vendor if access to a configurable pool of computing resources is needed. Besides, cloud computing has the large quantity of advantages, such as ultra-large scale, virtualization, high reliability, versatility, high scalability, on-demand service, low cost and so on. Moreover, cloud computing can provide storage services in addition to computing services. Due to its multiple advantages, cloud computing is widely used in many fields. Many enterprises use cloud computing to store their own data access information, which has certain privacy, but there are still potential dangers. Therefore, data access storage optimization in cloud computing has become one of the key research topics among scholars today [1–3].

As one of the many services provided by cloud computing, cloud storage which is widely welcomed because of its high efficiency, flexibility and pay-as-you-go allows users to store and share data on the platform. However, data stored in cloud platform is in an uncontrollable domain, and data owner (DO) loses the control of data, which takes great risk on security. When users want to obtain the data through cloud platform after encryption data are uploaded to the cloud platform, they have own the access rights. That is why it is important to establish a reasonable access policy for the cloud storage platform. If traditional public key encryption mechanism is adopted when DO uploads data with encryption operation, DO needs to perform a data encryption operation for each user which not only increases the computing burden on DO side, but also cloud service provider (CSP) has to repeat the operation every time after sharing the stored data. The workload is huge. To solve above problems, an attribute-based encryption scheme (ABE) which uses identity features of users as attributes to select some or all the attributes to encrypt and decrypt data was proposed by Sahai and Waters. ABE allowed DOs to share data without using public keys of others, which reduced computation amount, but cannot prevent collusion attacks among multi-user. Moreover, based on ABE scheme, Key-Policy Attribute Based Encryption (KP-ABE) scheme where DO constructed access structure in user key, and attribute set was utilized to perform encryption operations on data was proposed by Goyal et al. in which Fine-grained access control and flexibility in managing user rights could be achieved, but the solution required updating all of the user's keys in key revocation. Besides it,

* Corresponding author.
E-mail address: zhangqiuliang816@163.com (Q. Zhang).

https://doi.org/10.1016/j.comcom.2020.04.023
Received 24 February 2020; Received in revised form 27 March 2020; Accepted 13 April 2020
Available online 16 April 2020
0140-3664/© 2020 Published by Elsevier B.V.

based on ciphertext and simplifying key revocation process, Bethencourt et al. proposed Cipher-text Policy Attribute Based Encryption (CP-ABE), whose computational overheads were much more compared with KP-ABE scheme [4–6].

Cloud data sharing service designed to enhance privacy protection was proposed by Sabrina et al. which converted the access control structure into a binary access control tree. However, due to structural constraints, the flexibility of the solution is limited and the storage efficiency is not significantly improved.

Based on data mining technology and genetic algorithm, Karimi et al. proposed a combination of QoS-aware services in cloud computing whose requirements must be dynamically implemented, and it requires a trade-off between the best performance state and the execution speed of service portfolio. In order to achieve this goal, the combination of methods in previous studies has been used to maximize and optimize results in the shortest possible time. However, as the number of services was increasing, and the search space for problems was expanding, there was no enough efficiency for traditional methods to combine the required services in reasonable time. So, the genetic algorithm is adopted to globally optimize the service level agreement. In addition, service clustering is used to reduce the search space of the problem, and according to its history association rules are utilized for the composite service to improve the service composition efficiency. Shila et al. [7] proposed a secure mobile storage system called cloud computing storage system. Although these literatures have made certain sharing of cloud computing and data optimization, cloud computing models mainly provide cloud computing services with a set of dedicated and expensive machines, which can result in huge investments in capital expenditures and ongoing costs. Cloud computing is a revolutionary paradigm with efficient resource utilization and advanced manageability, which provides services from data storage and processing to software computing resource process on the network. Nowadays, popular cloud computing model includes a set of dedicated and expensive machines which provide cloud computing services and result in huge investments in capital expenditures and ongoing costs. The cost-effective solution is to leverage the capabilities of ad hoc clouds consisting in distributed and dynamically undeveloped local resources, and the paper adopts a distributed system solution. It can be further divided into static and mobile clouds. Static clouds utilize computing resources that are underutilized by general-purpose machines, and mobile clouds utilize idle computing resources of mobile devices. However, the dynamic and distributed nature of ad hoc clouds poses a challenge to system management [8].

As far as the existing research is concerned, data access storage algorithms in cloud computing have the defects of low data processing efficiency and low fault tolerance, which cannot meet the needs of today's society for data access storage. Therefore, HDFS which refers to a distributed file system, which is highly fault-tolerant and can be connected to inexpensive machines is introduced to design data access storage optimization algorithm in cloud computing. Meanwhile, HDFS can also provide high-throughput data access, which is ideal for storage applications of data access information. The application of HDFS can greatly improve the data processing efficiency and fault tolerance rate of IoT data access storage optimization algorithm, which provides more effective technologies for the storage of IoT data access information.

2. Data mining structure in large cloud storage system

Cloud computing implements a componentized Web 2.0 application system and a cloud computing-based information platform. In order to achieve resource access and data mining for large cloud storage system, it is necessary to build an overall model at the beginning. Then through construction of resource access in large-scale cloud storage system, information exchange and resource integration of multi-source data are realized, and multi-dimensional business services and multi-function control are provided as well. Finally, cloud storage system under digital information service is monitored by the multi-channel data query and network data through the memory management mechanism, which can improve the efficiency and security performance of network data access and scheduling. What is more, in order to establish the dynamic balance of the QoS access system and serve in the digital information of the MAC layer, the overall model of data mining framework in Hadoop platform is constructed in the paper, which is shown in Fig. 1 [9].

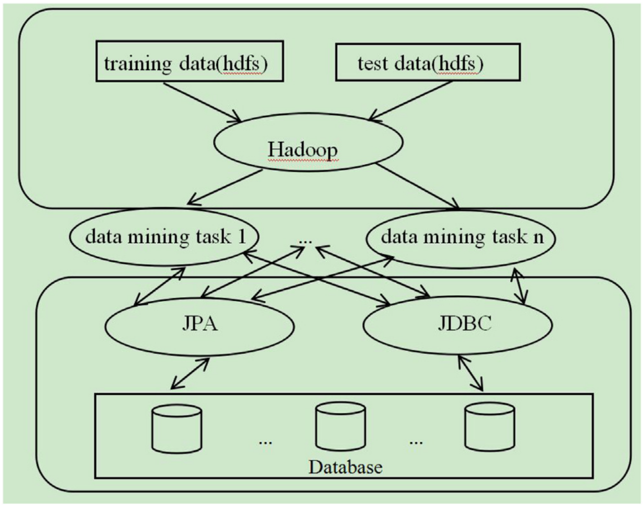

Fig. 1. Data mining framework in Hadoop platform.

In Fig. 1, JDBC (Java Database Connectivity) represents a random read and write operation interface under large cloud storage system, which provides standardization tools for reading and writing programs in cloud storage system when resource scheduling is performed, and it has parallel processing capability. Resource integration for large cloud storage systems with cloud computing as the core. However, frequent large-data query operations in cloud storage system result in increasing load pressure on the cached data in parsing engine, and scheduling design [10] in resource access will be required. For data mining task 1 to task n, when $y(n)$ is the Gaussian time sequence that each element is independently generated from Gaussian pseudo-random number in large cloud storage system query response, $A = \{A_{1,} A_{2,} \ldots, A_m\}$ is the attribute of data classification, and the maximum trust value is 1, the longer the interval is, the smaller the impact of the trust value on current situation will be in data mining and resource scheduling design combined with the data in the framework diagram, and the time-decreasing function is $T_{sim} \in (0, 1]$. Moreover, the obtained edge inverse vector is used to represent the rank of the original data $x(n)$, and the state reorganization of the multi-source data queried by digital information service under cloud storage system in parallel is achieved through amplitude-adjusted Fourier transformation. Then, the cloud storage system is built with the help of Hadoop to mine massive high-dimensional data flows. Finally, distributed code execution and the KD tree data indexing method are used to distribute task code to multipath monitoring node. In the paper, based on the limit separation theorem of auto-correlation function, the cache permutation function is constructed in this paper to establish the dynamic balance in the KD tree, and the obtained cache permutation function is as follows [11,12].

$$MTTA = \sum_{i,j,l=1,1,1}^{M,n,N} d_{ijl} \cdot Q\left(d_{ij}\right)^{-1} \cdot T\left(s_l\right) / (N-1) \tag{1}$$

where d_{ijl} is the state parameter of resource scheduling in cloud storage system with equal spacing, $Q\left(d_{ij}\right)$ is the linear program-controlled branch vector of data block d_{ij}, and $T\left(s_l\right)$ is the time function of node movement in cloud storage system. In a word, data mining in large cloud storage systems is realized through design structure model

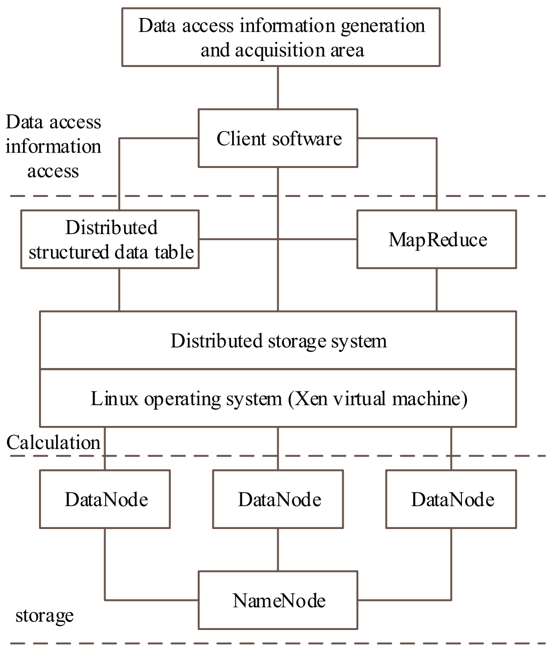
Data access information generation and acquisition area
Data access information access
Client software
Distributed structured data table
MapReduce
Distributed storage system
Linux operating system (Xen virtual machine)
Calculation
DataNode
DataNode
DataNode
NameNode
storage

Fig. 2. Optimization diagram of data access storage architecture.

mentioned above, which provides an accurate data foundation for resource scheduling access.

3. Optimize algorithm design for IoT data access storage

Data access information is critical for each domain or device. Customers and devices can be analyzed based on the data access information to ensure customer satisfaction and device security. Therefore, data access information needs to be stored, and IoT data access storage system has following characteristics.

(1) Practical performance. According to different data information, data port needs to be provided, and the complete monitoring system is used for real-time monitoring, which maintains close contact with the outside world and ensures good application experience of users.
(2) Low cost. IoT data access storage system expansion equipment is simple, and maintenance and economic costs are getting lower and lower with the continuous development of technology.
(3) Scalablity. The number of servers for the IoT data access storage systems can reach hundreds, and the greater the number is, the more scalable the storage system will be.

All mentioned above have put higher requirements on the IoT data access storage algorithm [13]. However, existing data access storage algorithms cannot meet these requirements, so they need to be optimized accordingly. The specific process is shown below.

3.1. Optimization of data access storage architecture

In order to improve the data processing efficiency of storage algorithm, HDFS is first introduced to optimize the data access storage architecture first according to the characteristics of the IoT data access information, the advantages of cloud computing, and the requirements of data access storage. Data access storage architecture is optimized as shown in Fig. 2 [14–18].

As shown in Fig. 2, after the data access storage architecture is optimized, it is mainly divided into three parts, which are the information access of data access, calculation and storage. The client software for accessing data access information can generate access information and acquire functional areas. What is more, the computing core is a distributed storage system with Linux operating system, which Xen virtual machine whose key part is the storage part. The HDFS cluster managed by the NameNode correspondingly stores the data access information, and it can store and manage massive data access information.

IoT data access information can be transferred to HDFS file or stored in database by the IoT data access information access and calculation part according to different requirements, both of which can realize the persistent storage of the data access information. Meanwhile, HDFS referenced at the same time can greatly reduce the delay of data access and increase user experience [19].

3.2. Strategy optimization in data access storage distribution

Based on the above optimized data access storage architecture, the hash algorithm is adapted to re-optimize the data access storage distribution strategy.

In the process of data access information storage, HDFS is introduced to perform distributed storage for data access information. However, a key issue is how to distribute data access information on IoT nodes, namely the distribution strategy of data access storage which has a direct and large impact on data processing efficiency.

In order to improve the efficiency of data processing, the hash algorithm with high performance and low maintenance cost on data structure is used to optimize the data access storage distribution strategy, which can greatly reduce the failure of IoT nodes and data access information migration caused by the increase of nodes.

3.2.1. Factors affecting data access storage distribution

After the data access information is accessed, it can be divided into multiple data blocks and distributed on IoT nodes according to certain rules. It is found through research that the main factors affecting the distribution of data access storage are mainly divided into three. The details are shown below.

(1) Load balancing. In the process of storage distribution, data access information should be evenly distributed as evenly as possible on the nodes to maintain the load balance of the nodes, so that the data access information can be processed in parallel to improve the data processing efficiency. If the storage distribution is not balanced, there will be a “data skew” in data access information. What is worse, it will be impossible for nodes to perform parallel processing, which will have a greater impact on data access information storage.

(2) Node failure. For HDFS clusters, it is a frequent problem that its nodes are out of order. Therefore, optimization algorithm in data access storage needs to be much greater fault tolerance, so that data access information can be stored in a balanced manner.

(3) Storage operation performance. The effect of data access storage is directly affected by operation performance. In general, only network transmission operation is considered. If the operation on network transmission is simple, communications among nodes in the execution process of data access storage algorithm can be greatly reduced, and the compression efficiency of data access information can be effectively improved [20–23].

According to the above analysis and correlation among data access information, the hash algorithm is applied to map and store the related data on the same node, and the related data access information is aggregated, so that query and analysis on subsequent data access information is facilitated [24,25][1].

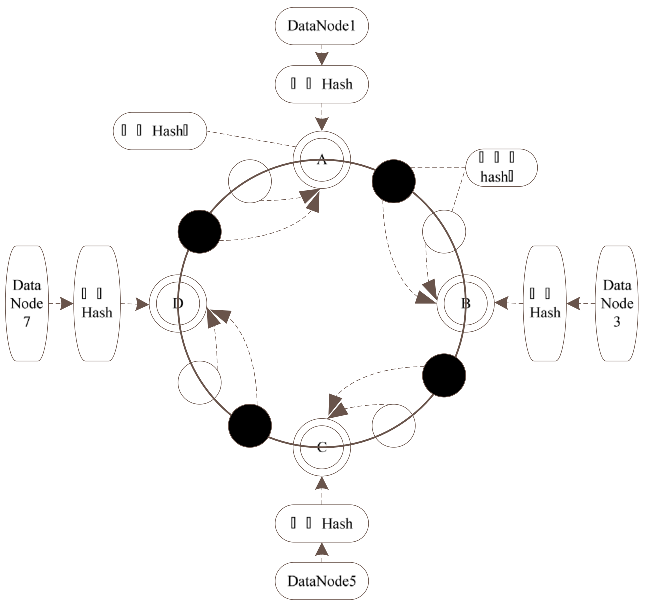

Fig. 3. Schematic diagram of hash algorithm.

3.2.2. Hash value configuration for data access information storage location

The schematic diagram of the hash algorithm is shown in Fig. 3.

As it can be seen from Fig. 3, the hash algorithm is divided into four areas: module A, module B, module C, and module D, which correspond to DataNode1, DataNode3, DataNode5, and DataNode7. The loop of hash algorithm is completed by calculating the hash value [26,27].

Since the hash algorithm loop, distribution strategy in data access storage is optimized. The specific steps are as follows.

First, relevance of data access information and the number of redundant copies need to be set, and the calculation formula of correlation degree on data access information is

$$\delta = \frac{\sum_{i=1}^{n} x_i * \alpha}{n^2 + 1} \tag{2}$$

where δ represents correlation degree among data access information, α indicates the number of redundant copy that is set to 3 according to related literature, and n is the total number of data access information.

Second, node hash value in HDFS cluster is accordingly calculated and configured into hash loop interval. The calculation formula on nodes hash value is expressed as

$$\omega = \int_{t=1}^{m} \sqrt[2]{p} \otimes \beta \tag{3}$$

where ω represents nodes hash value, p is the node, m indicates total number of nodes, and β represents the calculation parameter of nodes hash value [28].

Then, according to the correlation of data access information, data hash value is correspondingly calculated, and the calculation formula is expressed as

$$\mu = \frac{\int_{i=1}^{n} \sqrt[2]{x} \otimes \delta}{n^2} \tag{4}$$

where μA represents the hash value of data access.

Finally, the storage location of data access information is configured according to the obtained hash value of the nodes and data, and the configuration result is obtained as

$$f(x) = \prod_{i=1,t=1} {}^{\omega}/_{\mu} * \chi \tag{5}$$

where χ represents configuration parameter.

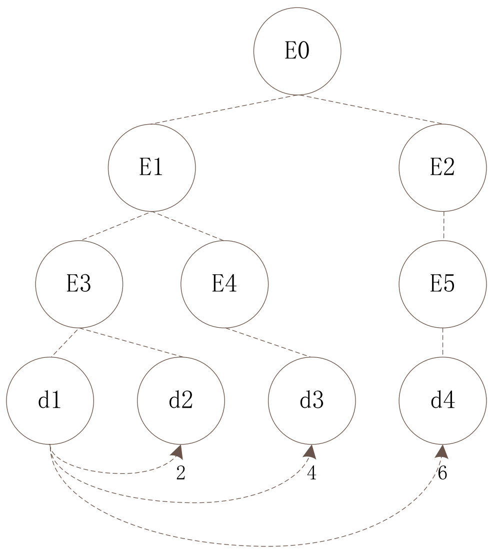

Fig. 4. Optimization diagram of IoT network topology.

3.3. Optimization on IoT network topology

Since it is stored by cloud computing in storage process of the data access, the network topology will have a great impact on the storage effect. A good network topology can greatly improve the efficiency of data access storage and speed up data access storage. Existing IoT data access storage algorithm network topologies have major problems. Therefore, the scheduler network topology is used for optimization. The specific process of optimization is as follows [29].

The nodes in the network topology are mainly arranged in a tree, and each node(such as d1, d2,…) is connected to the switching node (such as E0, E1, E2, …)of the computer. The optimization diagram of network topology is shown in Fig. 4.

As it is shown in Fig. 4, the distance between each node and the switch is the shortest after IoT network topology is optimized, which can greatly improve the speed of data access information access and storage, and advance the performance of data access information storage. What is more, optimization in network topology can be achieved preferably.

3.4. Optimization on data block size

Storing data access by using HDFS, actually, divides access data into data blocks for distributed storage. Therefore, it is seen that the data block size also directly affects the data access storage. In order to make the data access storage larger and faster, the effect algorithm is used to optimize data block size.

Optimization formula for data block size is

$$effect(\gamma) = f(x)\wp \frac{trans\_time}{trans\_time - seek\_time} \tag{6}$$

where $trans\_time$ indicates the access time of data access information and $seek\_time$ B is the storage location configuration time of data access information [25,30].

According to the above formula, the optimal size of the data block is obtained, and data access information is correspondingly stored based on it.

Through the above process, the optimization of data access storage in cloud computing is realized, which can greatly improve the data processing efficiency and fault tolerance of data access storage, and provide more advanced technical support for data access storage and management.

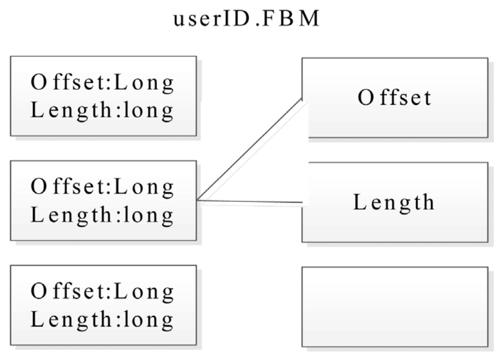

Fig. 5. Mapping relationship between FBM files and user files.

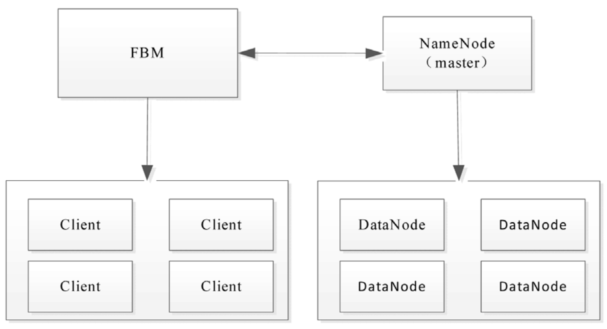

Fig. 6. Schematic diagram of file optimization scheme.

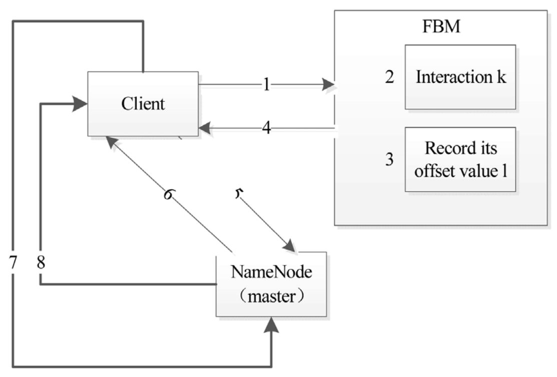

Fig. 7. File writing process diagram.

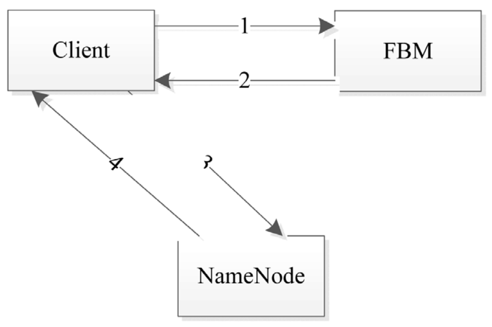

Fig. 8. Process of reading files.

4. Optimization methods of file storage

4.1. Optimization scheme design of file storage

Original HDFS architecture and block management mode are maintained to introduce the meta data information server File-Block-Mappin(FBM) for storing IoT user space. What is more, FBM records the file size and the offset value of user file stored in HDFS, and any user on HDFS has its own user file and stores all its files. Besides it, each file name corresponds to user ID, whose mapping relationship is shown in Fig. 5, and file optimization is shown in Fig. 6.

The optimized file scheme mainly accesses customers and data nodes through FBM and master nodes respectively. The improved effect based on the parallel management scheme can shorten the response time of the system. And the file meta data information of a storage system occupies most of the entire system file meta data information, but it occupies a small amount of storage memory. Therefore, this type of file storage system is used as optimization object.

4.2. Read and write process of file optimization program

(1) File writing process is shown in Fig. 7.

Client sends a request, ① File request j is written, which is not interacted with NameNode, but ② k is interacted with FBM. Then, ③ offset value l is recorded, and ④ FBM is marked as record success m. Meanwhile, ⑤ Client sends write stream of user files to NameNode, ⑥ which returns this stream to Client. At this time, ⑦ Client interacts with NameNode to perform actual data write operation p, and ⑧ new file writes to the ending q.

(2) Process of user reading file is shown in Fig. 8.

Client interacts with FBM and NameNode in a two-way method, and the three sides can communicate with each other fully. First, Client requests to read file name ①, then interacts directly with FBM to obtain meta data information ②. Moreover, Client requests to read input stream ③ of user file according to ②, and file returned from NameNode is read, which is presented to Client ④.

(3) IoT users delete files without directly deleting original file data, but store their meta data information in FBM, for which delete flag is set.

4.3. Problems caused by file optimization solution

IoT users repeatedly adding and deleting operations will produce a lot of fragmentation. Therefore, a threshold is set, which is equal to the ratio of file fragmentation to all files. Given that the threshold is exceeded, system will sort out fragment. In other words, a new user file will be created for the user, the user's original file will be copied, and the corresponding FBM index file will be established for the new file as well. Moreover, the new file is named the original user file, and the corresponding new index file is also renamed original index file instead of the original file.

4.4. Implementation on file optimized storage

In order to optimize storage of small files, the SmallFileStatus class is introduced into small file meta-data information based on the original HDFS, which adds the offset and size attributes of small files in user files based on FileStatus. Moreover, the core class of small file storage optimization design is FBM class, its core attributes is fileMap that belongs to HashMap type. Additionally, key–value pair is the file name whose key is small file, and the value is its meta-data information,

When the system starts, FBM loads each user's text file information and small file index entries. Then, the system loads user meta-data information into memory so that client can locate the file information by the file path. The disadvantage is that too much meta-data information is loaded in the memory, which may lead to the FBM memory overhead to be too high. To solve the problem of persistent storage of FBM, relational database is adopted to store related meta data information

Table 1
Experimental system configuration.

	Configuration	Type
IoT server	CPU	Intel Xeon W-3175X 28-core 3.1G
	RAM	32GDDR4*4
IoT node	CPU	Intel i7-9700K Core Eight Core 3.6G
	RAM	16GDDR4*2
	hard disk	Seagate Galaxy Exos 7E88 TB 256 MB 7200 RPM

Table 2
Experimental parameter settings.

User scale	Number of files created	File size (w)
1000	100	10
2000		20
3000		40
4000		60
5000		80
6000		100

in the database space, which has the advantages of easy deployment and reduces FBM memory consumption will be reduced, but relational databases cannot be deployed in a cluster. In addition, a single point of limitation may make the database prone to performance bottlenecks to a certain extent. Besides it, NoSQL non-relational database, such as MongoDB database can be used to store file and folder information. This database can be deployed in a distributed cluster, which has certain difficulties in deployment, but MongoDB database can more effectively solve the problem of single point limitation.

When new IoT users register, they will have their own user files. User files slowly accumulate to a certain level and become large files. Therefore, in an HDFS cluster, when Na-meNode node only has one userID.file meta data information, memory consumption of the NameNode will be greatly reduced.

When the user performs delete a operation, a delete flag 1 is set in userID.FBM, and userID.file is retained. In other words, when the user wants to access the file of the IoT, the userID.FBM positioning mark will be used to indicate that the file does not exist. This file still exists in HDFS that cannot be accessed by index, which is fragmented area. Besides, a new file is added at the end of the user file. This process continues to operate, and the fragmentation of user files is increased, which leads the storage efficiency of the entire HDFS cluster to decrease. Therefore, corresponding fragmentation management methods are adopted to perform fragmentation management on user files.

The above threshold is set to 20%, and a fragment calculation is performed for each deletion, while Compared with the total file size, as long as the sum of file size in meta data information with deletion flag 1 is greater than 20%, disk management program will be run to write meta data information corresponding to deletion flag bit other than 1 into temporary file. Meanwhile, storage entity of user files are written to temporary files which are renamed and replaced to original file.

5. Performance test of optimization algorithm of IoT data access cloud storage

The above process realizes the design of the cloud storage optimization algorithm for IoT data access, but further verification is required to determine whether it can solve the problems of the existing algorithms needs to be further verified. Therefore, a simulation experiment is designed to compare the performance of data access storage optimization algorithm in the cloud computing, which mainly reflects algorithm performance by the data processing efficiency and fault tolerance.

5.1. Environment configuration

Test experiment for file optimized storage only improves NameNode, which does not involve DataNode. Therefore, HDFS is constructed on a physical machine, and a simulation experiment scenario is constructed in OPNET Modeler, which is equipped with 3 IoT server nodes and connected to communication equipment for experiments operating. The system configuration is shown in Table 1.

In the course of the simulation comparison experiment, in order to ensure the accuracy of the experimental results, the external parameters are kept the same during the experiment. The specific experimental results analysis process is shown below.

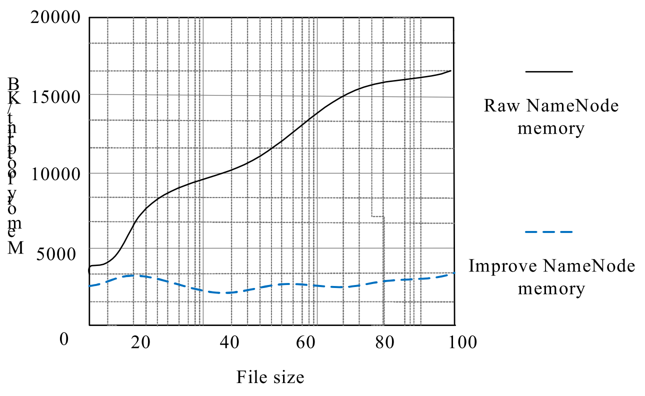

Fig. 9. Comparison of NameNode memory before and after file optimization.

5.2. Result analysis

In order to evaluate the performance of the method of adding the FBM optimization files to the IoT of, 6 sets of comparative experiments are performed on the improved HDFS and the original HDFS. The experimental parameters are shown in Table 2.

5.2.1. Namenode process memory comparison

The memory size of the NameNode process is recorded during the experiment, and the experimental results are shown in Fig. 9, in which the abscissa represents the file size and the ordinate represents the memory size of the NameNode process whose unit is KB.

As can be seen from Fig. 9, the NameNode process itself has a certain memory consumption. During file storage, when the file size is the same, the memory of the original NameNode memory is larger than that of the improved system NameNode. Moreover, with the increase in the number of file size. The original NameNode memory consumption is very fast, but the improved system NameNode memory consumption is relatively slow and there is no significant increase trend. The experiments show that in terms of memory consumption, especially when the file size is more than 40 kb, the memory footprint effect is better, the performance of the optimization scheme proposed in the paper is better than the original scheme, and name space of the HDFS cluster is extended as well.

5.2.2. Comparison on file upload efficiency

File reading and writing scale are set to 1000, 3000, 5000, 10 000 and 15 000, respectively; and time consumed by file reading and writing is tested. The two solutions before and after the improvement are compared in terms of creating files without writing data. The comparison results are shown in Fig. 10.

It can be seen from Fig. 10 that under the same meta data information, the difference between the two schemes in the previous period is not obvious. As the meta data information increases, the writing speed of the two schemes for creating files slowly decreases. However, when the HDFS improved HDFS optimization scheme is used to write Na-me Node metadata information, as the file size increases, the write speed increases, but the increase speed is slow, which is about 50% lower than

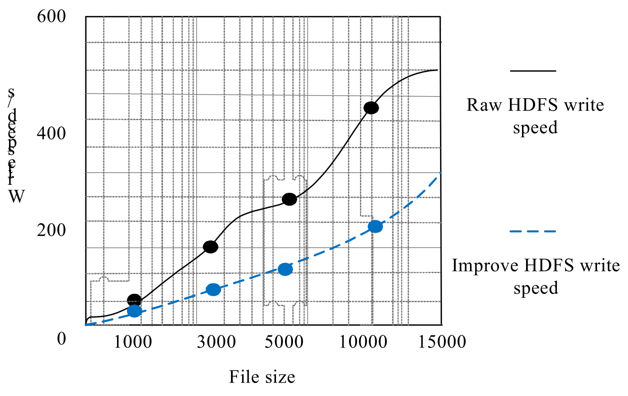

Fig. 10. Comparison of time consumption for creating a file (without writing data).

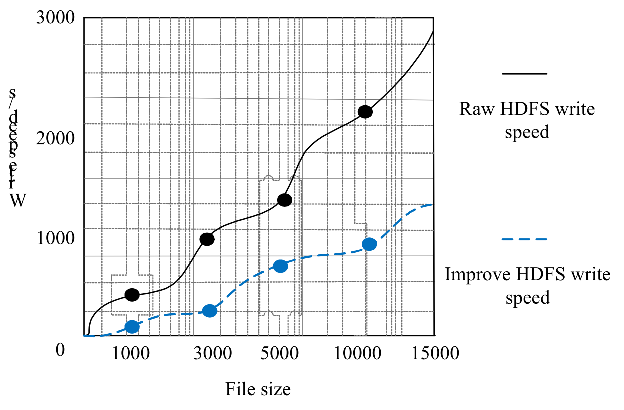

Fig. 11. Comparison of the time consumption of creating a file (writing data).

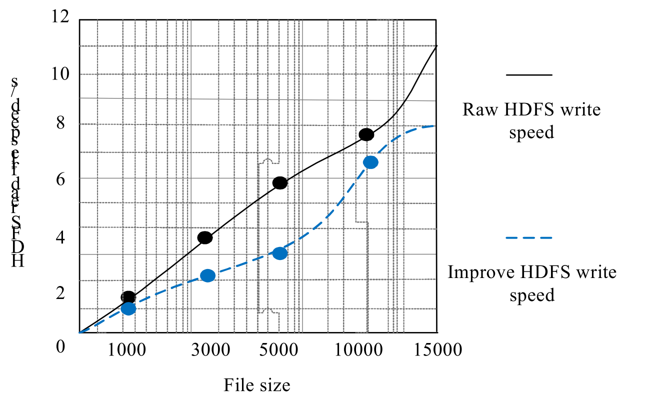

Fig. 12. Comparison of time consumption for reading files (without downloading data).

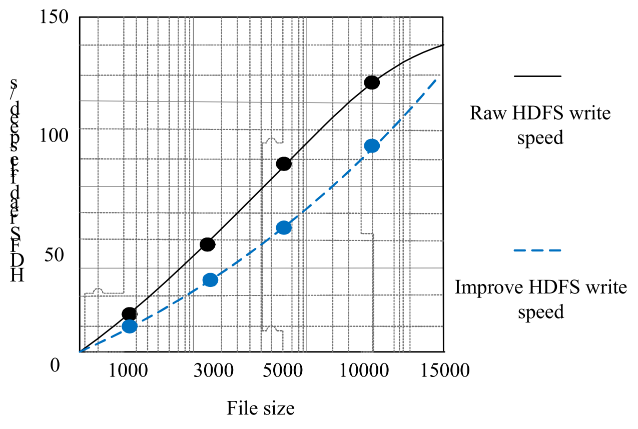

Fig. 13. Comparison of time consumption for reading files (downloading data).

the raw method. The speed advantage of creating files is very obvious when writing Na-me Node meta data information.

The above experiment is the speed of creating files. In order to test the speed of writing real data blocks, the following comparisons are made.

As can be seen from Fig. 11, with the increase of actual data information, the speed of the two schemes in the actual data writing slowly decreases. When the actual data size is the same, optimization solution by adding FBM consumes more time to write the actual data than original solution, since HDFS does not support the operation of appending files.

5.2.3. *Comparison on file download efficiency*

In addition, the speed of reading meta data information of the two schemes is compared with each other, and file size is the same as that of Section 5.2.2. The speed of not downloading data is shown in Fig. 12, and the speed of downloading data is shown in Fig. 13.

As can be seen from Figs. 12 and 13, the improved scheme where FBM is introduced can improve the speed of HDFS reading files to a certain extent. In summary, the improved method proposed in the paper, namely the method where FBM data structure is introduced, the solution of merging files into large files is feasible and better than the original HDFS method.

5.2.4. *Comparative analysis on data processing efficiency*

The data processing efficiency directly reflects the efficiency of the algorithm. Generally speaking, the higher the data processing efficiency is, the better the performance of the algorithm will be. The comparison of data processing efficiency obtained through experiments is shown in Table 3.

It is shown in Table 3 that when the times of experiment is small, only 20 times, the data processing efficiency of the existing algorithm

Table 3
Comparison on data processing efficiency.

Experiment times	Existing algorithm	Optimization algorithm
20	56%	89%
40	64%	88%
60	50%	79%
80	49%	76%
100	67%	85%
120	73%	81%
140	62%	91%
130	53%	90%
180	69%	99%
200	61%	93%

is 56%, and data processing efficiency of the optimized algorithm can reach 89%. Moreover, with the experiment times increases, if the number of experiments reaches 200, the data processing efficiency of the existing algorithm is 61%, and the data processing efficiency of optimized algorithm can reach 93%. Obviously, the data processing efficiency of optimized algorithm is much higher than that of the existing algorithm with different experiment times. What is more, the optimized algorithm can achieve the maximum processing efficiency, which is 99% after 180 experiments.

5.2.5. *Comparative analysis on fault tolerance*

Since node faults occur frequently in HDFS, a greater fault tolerance can improve the storage performance of the algorithm. The comparison of the fault tolerance obtained through experiments is shown in Fig. 14.

It can be seen from Fig. 14 that the fault tolerance of existing algorithm ranges from 19% to 60%, and the optimized algorithm has the minimum fault tolerance of 34% and the maximum of 96.12%.

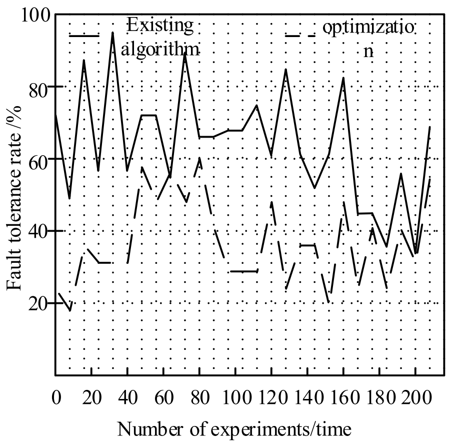

Fig. 14. Comparison on Fault Tolerance.

Therefore, the fault tolerance of the optimized algorithm is much higher than that of existing algorithms.

According to the experimental results mentioned above, optimization algorithm of IoT data access storage in cloud computing proposed in the paper greatly improves the data processing efficiency and fault tolerance, which fully demonstrates that the optimization algorithm of IoT data access storage in cloud computing proposed in this paper has better performance.

6. Conclusion

Optimization algorithm of IoT data access storage in cloud computing proposed in the paper greatly improves the data processing efficiency and fault tolerance rate, and provides more advanced technical support for the storage and management of data access. The experimental results are analyzed by constructing simulation experiment scenes in OPNET Modeler. Moreover, the experimental results show that IoT data optimized by algorithm proposed in this paper is superior to the original state in terms of transmission speed, system resource occupation and response time. In addition, the maximum IoT data processing efficiency of optimized transmission can reach 99%, and the maximum fault tolerance rate of optimized algorithm can reach 96.12%.

Due to the setting of the parameters in the simulation experiment process, there is a certain deviation between the experimental results and the actual results, but the deviation does not effect the overall contrast trend. Therefore, the experimental results obtained are objective, and it has been fully applied in railway 12306 website. In order to obtain more accurate experimental data and results, further research and optimization of the IoT data access storage optimization algorithm in cloud computing are needed.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Mingzhe Wang: Conceptualization. **Qiuliang Zhang:** Data curation.

References

[1] Chen Guangsheng, Cheng Yiqun, Jing Weipeng, Cloud computing DBSCAN optimization algorithm based on KD tree partitioning, Comput. Eng. 43 (4) (2017) 21–27.

[2] Y.E. Lunqiang, Simulation of data stream storage load balancing optimization in cloud computing, Comput. Simul. 35 (10) (2018) 256–259.

[3] Hongwen Hui, Chengcheng Zhou, Shenggang Xu, Fuhong Lin, A novel secure data transmission scheme in industrial internet of things, China Commun. 17 (1) (2020) 73–88.

[4] Fuhong Lin, Yutong Zhou, Xingshuo An, Ilsun You, Kim-Kwang Raymond Choo, Fair resource allocation in an intrusion-detection system for edge computing: Ensuring the security of internet of things devices, IEEE Consum. Electron. Mag. 7 (6) (2018) 45–50.

[5] Jingtao Su, Fuhong Lin, Xianwei Zhou, Xing Lv, Steiner tree based optimal resource caching scheme in fog computing, China Commun. 12 (8) (2015) 161–168.

[6] Li Jingwei, Sun Bo, Cloud computing task scheduling optimization algorithm based on IDEA fusion taguchi method, Control Eng. 24 (2) (2017) 458–466.

[7] Ye Lunqiang, Simulation of data stream storage load balancing optimization in cloud computing, Comput. Simul. 35 (10) (2018) 256–259.

[8] Jin Yu, Design of big data compatible storage system based on cloud computing environment, Mod. Electron. Technol. 42 (1) (2019) 24–27.

[9] Luo Siwei, Hou Mengshu, Niu Xinzheng, et al., Replica placement algorithm based on immune optimization strategy, J. Univ. Electron. Sci. Technol. China 46 (5) (2017) 741–746.

[10] Zhang Yanmin, Optimization analysis of discontinuous data path mining in cloud computing environment, Comput. Simul. 35 (8) (2018) 148–151.

[11] Wang Wei, Design of cloud computing data optimization storage system, Comput. Knowl. Technol. 13 (13) (2017) 26–27.

[12] Tu Junying, Li Zhimin, Design of unstructured big data storage system under cloud computing, Mod. Electron. Technol. 41 (1) (2018) 173–177.

[13] Li Bin, Li Qiming, Optimization of chameleon hash authentication tree for data storage security in cloud computing, Microelectron. Comput. 35 (6) (2018) 7–12.

[14] Ma Zitai, Cao Jian, Yao Yan, Workflow scheduling method using auction example and considering intermediate data storage strategy in cloud environment, J. Comput. Integr. Manuf. Syst. 23 (5) (2017) 983–992.

[15] Sa Rina, Cloud computing resource scheduling scheme based on ant colony particle swarm optimization algorithm, J. Jilin Univ. Sci. 55 (6) (2017) 1518–1522.

[16] H. Books, Hephaestus Books. Articles on Network File Systems, Including: Andrew File System, Google File System, Lustre (File System), Distributed File System (Microsoft), DCE Di, Hephaestus Books, 2011, pp. 102–104, (09).

[17] N. Lester, J. Zobel, H. Williams, Efficient online index maintenance for contiguous inverted lists, Inf. Process. Manage. (42) (2006) 916–933.

[18] D.J. Md, J. Ahmad, M. Mukri, The methodology on statistical analysis of data transformation for model development, Int. J. Stat. Appl. 2 (6) (2012) 7–11.

[19] L. Ogiela, Semantic analysis and biological modelling in selected classes of cognitive information systems, Math. Comput. Modelling 58 (5–6) (2013) 1405–1414.

[20] A. Alieldin, J. Tordsson, E. Elmroth, An adap-tive hybrid elasticity controller for cloud infrastructures, in: IEEE Network Operations and Management Sym-posium, 2012, pp. 204–212.

[21] B.B. Nandi, A. Banerjee, S.C. Ghosh, Elasticcloud resource management with dynamic SLA: A Saa SPerspective, in: Proceedings of the 13th IFIP/IEEE International Symposium on Integrated Network Manage-ment, 2013.

[22] C.G. Yang, Z.R. Wang, W. He, et al., Development of a fast transmission method for 3D point cloud, Multimedia Tools Appl. 77 (19) (2018) 25369–25387.

[23] C. Moreno, M. Li, A progressive transmission technique for the streaming of point cloud data using the Kinect, in: 2018 International Conference on Computing, Networking and Communications (I CNC). [S.l.]: [s.n.], 2018, pp. 593–598.

[24] M. Levoy, The digital michelangelo project: creating a 3D archive of his sculptures using laser scanning, in: Proceedings of the 2nd International Conference on 3-D Digital Imaging and Modeling, 1999, [2018-07-08], https://graphics.stanford.edu/papers/digmich-eva99/.

[25] Z.S. Rusinkiewic, M. Levoy, QSplat: A Multiresolution Point Rendering System for Large Meshes, SIGGRAPH 2000, ACM, New York, 2000, pp. 343–352.

[26] Gong Zhen, Research on Point Cloud Processing Method of 3D Laser Scan Technology, China University of Geoscience, Wuhan, 2017.

[27] Zhang Hongwei, Lai Bailian, Features of 3D laser scanning technology and it's application prospects, Bull. Surv. Mapp. (S1) (2012) 320–322, 337.

[28] G.G. Shi, X.H. Dang, X.G. Gao, Research on adaptive point cloud simplification and compression technology based on curvature estimation of engery function, Rev. Fac. Ing. 32 (4) (2017) 336–343.

[29] C. Moreno, M. Li, A progressive transmission technique for the streaming of point cloud data using the Kinect, in: 2018 International Conference on Computing, Networking and Communications (I CNC). [S.l.]: [s.n.], 2018, pp. 593–598.

[30] M. Levoy, The digital michelangelo project: creating a 3D archive of his sculptures using laser scanning, in: Proceedings of the 2nd International Conference on 3-D Digital Imaging and Modeling, 1999, https://graphics.stanford.edu/papers/digmich-eva99/.