A Metadata Management Method Base on Directory Path Code

Hui Cai
Logistical Engineering University of P.L.A, Chongqing, China
Email: caihui_cool@126.com

Shenglin Li
Logistical Engineering University of P.L.A, Chongqing, China
Email: 492739390@qq.com

Zhaoxia Wang
Logistical Engineering University of P.L.A, Chongqing, China
Email: weimo1234@sina.com

Xingchen Li
Logistical Engineering University of P.L.A, Chongqing, China
Email: since0701@163.com

Lichuan Huang
Ministry of Supplies & Pol Chengdu Military Region, Chengdu, China
Email: lichuan_85@aliyun.com

***Abstract*—Metadata distribution is important in mass storage system. Sub-tree partition and hash are two traditional metadata distribution algorithms used in file system. But they both have a defect in system scalability. This paper presents a new metadata management method, Directory Path Code Hash ("DPCH"). This method is to store directory and file metadata separately, and effectively solving the unbalanced metadata distribution and access hot point problems in Sub-tree partition and the excessive reading times and large metadata migration amount after directory property modification in hash algorithm. The experiment indicates that this method proposed significantly outweighs other algorithms in terms of throughput rate, metadata distribution, reading times, etc.**

***Index Terms*—metadata distribution, binary code, file system, directory path code hash(DPCH)**

I. Introduction

With the rapid development of scientific computing and network application, the data size of network information is increasing constantly and PB-grade mass data storage system plays a more and more important role. As a new storage structure, object-based storage adopts the current network technology and storage technology and provides a foundation for mass data storage system. The core idea of this object-based mass data parallel storage system is to store the metadata and data of files separately and manage them in a distributed cluster mode. According to relevant studies, many operations in mass data storage systems are for metadata only. 75.4% of read operations requires the use of metadata, accounting for 49% of total time and space consumption; 82.2% of write operations requires the use of metadata, accounting for 63.5% total time and space consumption [1-2]. Therefore, the metadata management efficiency determines the performance of the whole storage system, and the realization of high performance, high reliability, load balance and expansion of metadata management are the hot topics of current study on mass data storage system.

Regarding to the structure of metadata management system, exsiting researches mainly divided into two methods, they are centralized metadata servers and distributed metadata servers. The centralized metadata servers means the single decisive metadata data server node exists in the storage network, the interaction only exsits between other metadata servers and the decisive metadata server. Though the structure of this method is relatively simple, it has an obvious bottleneck. To decrease the happening rate of this bottleneck, we could optimize the processing paths requested by metadata, which could, to some extent, meet the requirements of the system scale expansion. While the distributed metadata servers indicates no single decisive server does not exsit in the storage network, and all metadata server could intercommunicate. Structurally speaking, there is no bottleneck, but it is relatively complicated which need a consistent preservation.

From the metadata distribution strategy, researches are mainly include Sub-tree partition and Hash algorithm. The research findings on Sub-tree partition are LOCUS [Popek 1986], AFS [Morris 1986], CODA

doi:10.4304/jsw.9.9.2443-2453

[Satyanarayanan 1990], Sprite [Ousterhout 1988], Dynamic Sub-tree partition [Weil 2004], etc. The research findings on Hash algorithm are Vesta [Corbett 1996], InterMezzo [Braam 1999], RAMA [Miller 1997], Lustre [Braam2002], Dynamic Hashing [Li2006], etc.

This paper, based on previous studies, proposes a new metadata management method based on path object codes called Directory Path Code Hash. This method is to store directory and file metadata separately, perform hash distribution through binary coding of directory paths, and introduce the concepts of bucket partition, comprehensive access authority and main and secondary mapping tables, effectively solving the access hot point and large metadata migration amount after directory property modification.

II. Relevant Studies

Metadata distribution is an important study aspect of metadata management. Reasonable data distribution may bring forth high retrieval efficiency, balanced load and good system expansibility. Metadata is the data for data description, and is small in size. However, with the constant expansion of system size and application degree, metadata size will increase gradually. Therefore, a good metadata distribution strategy may maximize the use of the whole cluster resources, realize the uniform load distribution between different metadata servers, and improve the performance of the whole storage system [3-4]. The current study mainstreams may be classified into two types; i.e., sub-tree partition [5] and hash algorithm [6].

A. Sub-tree Partition

Sub-tree partition, proposed by Popek et al [7] from Massachusetts Institute of Technology in 1986, could be classified into static sub-tree partition and dynamic sub-tree partition. Static sub-tree partition is to distribute all subdirectories or certain subdirectories at a lower level to different metadata servers for storage. The advantages include that the metadata distribution is simple and it could maintain a traditional hierarchical file directory structure and the disadvantages include that it could not effectively partition the load between different metadata servers. When metadata becomes a hot access point, it will become the performance bottleneck of the whole system and the computing consumption during directory traversal process is large. Dynamic sub-tree partition is to entrust different sub-trees in the hierarchical directory structure of file systems to different metadata servers. The advantages include that the partition granularity is smaller, the method is more flexible and the dynamic load balance is realized according to the load of metadata server; and the disadvantages include that the consumption of directory traversal is large, the repeat cache of prefix directory information reduces the utilization rate and hit rate of Cache, and large amount of metadata migration is required when a sub-tree is re-entrusted.

B. Hash Algorithm

A hash algorithm is also called static hash algorithm, which was proposed by Corbett et al [8] from IBM in 1996. The basic idea of a hash algorithm to hash a certain key value in files, such as file name or path, and then uniformly distribute the files to different metadata servers according to the hash results. The best advantage of this method is the realization of load balance, avoiding the bottleneck problem that a certain directory becomes a hot access point. This algorithm also has disadvantages. First, hash algorithm destroys the original hierarchical directory structure, which is not good for the search of instructions similar to IS; actually, the system expansibility is poor in that the output range will also be determined when a certain hash function is determined, and mass data migration will be required if output range needs to be expanded and hash function needs to be changed. Second, the file rename operation is not well supported. After a file is renamed, a large amount of data between metadata servers needs to be re-distributed.

C. Lazy Hybrid Algorithm

The Lazy Hybrid (LH) algorithm was proposed by Brandt et al [9] from University of California, Santa Cruz in 2003. It combines two types of algorithms, i.e., tree structure and Hash algorithm, use full pathname of files to compute hash value and then define the storage position of metadata in Metadata Lookup Table (MLT) with the hash value obtained as the index. The advantage is that it integrates the directory access authority into the metadata of every file, thus reducing the consumption of directory path traversal. The disadvantage is that the re-computation of hashed value will generate the migration and upgrading of a large amount of metadata, thus not reducing the consumption of system and not realizing the goal of distributed directory renaming.

III. Directory Path Code Hash

A. Basic Idea

By analyzing the studies and exiting problems of metadata management, this paper proposes a new metadata management method, Directory Path Code Hash (“DPCH”). The method differs from other traditional file system metadata management methods in that it has the following features. First, the partition granularity is different. The hash codes adopted in traditional file systems are for files, but this file hash algorithm has manage defects, which are described in Article II hereof; statistics indicates that the operation number of directory only accounts for less than 10% of the total file number. Therefore, the paper adopts the directory-based codes, which differs from traditional file systems in partition granularity, significantly reducing the complexity of operation. Second, the storage method is different. In traditional file systems, file metadata is used to store the access properties of files and the storage position of data block. From the perspective of management, data access is controlled by file property, while data property is controlled by directory path property. Therefore, this

paper classifies file metadata into directory property and file property, and store and management them separately. Directory property includes the absolute directory path of files and the access control authority of directories; file property includes the access properties such as file name, file type, creation time, modification time and file size. This storage mode is the further deepening of object-based storage structure, eliminating the metadata upgrading and migration caused by directory modification, and this separation storage mode may better reduce the directory path traversal consumption and disk I/O times. Third, the coding type is different. According to the first feature, the paper adopts the coding method based on directory path. However, the directory-based coding in existing documents and studies mostly adopts the full-path name as the main key value of hash codes, the biggest problem of which is that the directory renaming operation may require a large amount of file migration. Therefore, this paper proposes a new coding method, the basic idea of which is to perform binary transcoding of directory name according to creation time and then take it as a hash major key. That is, a mapping layer is added between directory full-path name and hash major key value, thus avoiding the metadata migration caused by directory renaming operation. Fourth, the concept of a bucket partition is introduced. For the partition granularity based on directory coding, we introduce the concept of bucket partition to overcome the metadata distribution and load balance problems caused by the uncertainty of file quantity under directory. That is, every path code could store only a certain quantity of metadata; and when the metadata quantity under a certain path code, a new path code will be created.

B. System Model

The DPCH system model realized in this paper is composed of the following parts: client, client file system (CFS), directory management server (DMS), metadata server (MDS) and object storage device (OSD). The working principle of this system model is similar to the tripartite transmission framework structure commonly used in object-based storage systems, but the study object is different slightly. The object-based storage studies the storage and management of files, while this paper studies the storage and index management of metadata. The structure of the system model is shown in Figure 1.

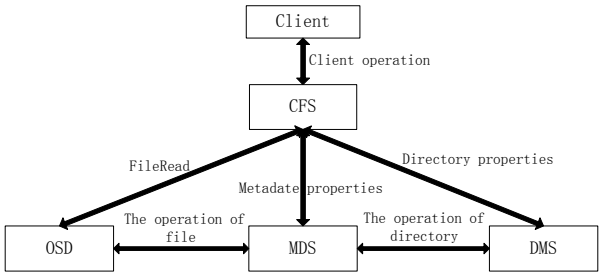

Figure 1. DPCH System Model

CFS is mainly responsible for providing hierarchical directory structure and metadata management system interface for upper application, and supports the upper user operation to execute complete file operations such as creation, deletion and modification.

DMS stores all directory paths, file names under directory, father directory, existing file quantity of bucket and binary encodes of path, and is mainly responsible for the management of directory path properties such as path coding and query, creation, deletion and modification.

MDS stores all metadata information and is mainly responsible for the storage and management of all metadata under directory, including the query, creation, upgrading and deletion of metadata. The cooperation of MDS and DMS provides uniform naming space and rules for the whole storage system, controls the client's access authority to the storage system, and is responsible for directory hierarchical partition and authority management as well as relevant operations on metadata.

OSD is responsible for the storage of bottom data and the supply of object-based storage interface service.

IV. METADATA DISTRIBUTION

In the metadata management system proposed in this paper, metadata management is divided into two independent parts; i.e., directory management server and metadata server, so as to separate the directory information and the file metadata under directory. Directory information mainly comprises the authority distribution and hierarchical management of directory, and file information mainly includes the contents of file metadata. Directory management server is specifically responsible for directory coding, directory traversal and authority management tasks, minimizes the cache occupancy, applies more cache on file metadata and reduces the disc I/O consumption of metadata read and right, thus alleviating the management task of file metadata in metadata server and enabling it to focus more on the management of file data.

A. Improved Metadata Distribution Algorithm

Basically, metadata distribution algorithm based on path object coding adopts Hash algorithm, but has two features. First, the hash object is directory, not file; second, the hash content is binary codes after path transcoding, not directory path.

We may express the improved metadata distribution algorithm with the following three formulas:

pathcode=f（path） （1）

result=Hash（pathcode）错误!未找到引用源。 （2）

错误!未找到引用源。 MDS_Location=Search（result） （3）

In Formula (1), f(x) is the function used for the coding of directory path name. We specify fixed-length binary code as the directory path name, with length as N and formalization as 000…000 (N figures). Therefore, the new equative directory under a certain path may be expressed respectively according to the creation time [000…000], [000…001], [000…010], [000…011], [000…100], [000…101], [000…110], [000…111]…. The absolute path of directory after coding is expressed as

000…000\000…001\...\000…010\..., so the only code of any directory in file system formed at creation could be obtained by this analogy.

This algorithm may be expressed as:

```
CreatCode (path)  /path is a new directory under current directory/
if (PathCode_Father (path) != null)   /path's father directory exists/
 {
  CodeID++;    /under current directory, create directory code ID plus 1/
  PathCode (path);    /code the path according to the CodeID value/
                /path's absolute path code is path's upper directory/
                /code plus   "\Code (path)" /
 }
else    /path's father directory does not exist/
 {
  PathCode (path) = 000…000;  /path's code is N-digit full 0 binary code/
                /path's absolute path code is N-digit full 0 binary code/
 }
```

In Formula (2), Hash (x) is the hash function used by this algorithm, the function value is the path code in Formula (1), and result is the result obtained after path code hash.In Formula (3), Search (x) is the distribution function from has result to MDS_Location, while MDS_Location is the distribution position of metadata under this directory in MDS cluster. Therefore, Formula (3) could also be considered as the mapping function from hash directory code value in query operation to metadata MDS cluster distribution position.

B. Storage Object Distribution

This paper separately stores the metadata directory and files; thus, the storage objects include directory information and file information respectively. When the Client has new write-in contents, new metadata will be written in DMS and MDS. For directory information, DMS is responsible for the management and maintenance, and the structure includes the absolute path of directory, the binary code of directory, father directory information, names of all files under this equative directory (excluding the directory name under this directory) and bucket volume information. See the following figure:

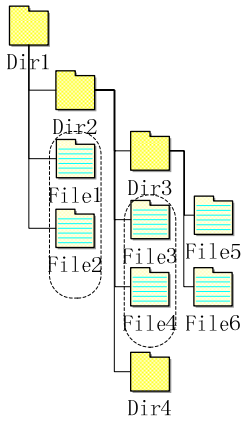

Figure 2. Distribution of Directory Storage Object

Dir1 contains the content such as Dir1's absolute path, binary code 000…000, name file1 and file2 under this directory and bucket volume information. Since Dir1 has no father directory, there is no father directory information.Dir2 contains the content such as Dir2's absolute path, binary code 000…000\000…000, file3, file4, bucket volume information and father directory Dir1's information.

For file information, DMS and MDS are jointly responsible for the management and maintenance. To be more specific, DMS is responsible for recording the names of files under the corresponding directory, and MDS is responsible for storing the metadata of the file. The distribution of file metadata in MDS is determined by the improved metadata distribution algorithm as mentioned in Article III. First, it looks up the corresponding binary codes according to the directory of the files, hash the binary codes, then apply the hash result into the distribution function, and finally determine the actual storage position of file metadata in MDS.

During the actual application process of system, we set a fixed upper volume limit for bucket. For example, we specify the upper volume limit of bucket as 50, the bucket volume information in every directory code in DMS the actual file quantity and bucket status value respectively. When there are new write-in contents in this directory, DMS will first check the bucket status value under this directory code. If the status value is not full, the file name after successful creation will be stored in this directory, and the actual file quantity will increase by 1.If the status value is full, the binary code of the directory will be created, and then the file name will be written in the new directory code. This indicates that the same directory in DMS may have several binary code records, that is, the contained file metadata exists in the different positions of MDS. This way may effectively avoid the problems of unbalanced distribution of metadata and load balance.

C. Metadata Storage

Metadata storage is mainly classified into directory storage and file storage.

For directory, the position of directory in DMS and relevant information need to be determined. First, Client sends a request on directory creation to DMS, which will check whether this directory exists. If this directory exists,

DMS will return the existence information to Client; if this directory does not exist, DMS will check whether the father directory of this directory exists. If the father directory exists, DMS will create a new directory, perform binary coding on the new directory according to Formula (1) in Article III, perform binary transcoding, set the volume information of the directory bucket, save the father directory information, and finally return the successful creation information to DMS. If DMS finds that the father directory of the new directory does not exist, will look up the maximum matching father directory of this directory, then circularly construct the directory according to the maximum matching father directory and the method of level-by-level creation, and finally return the successful creation information to Client.

For file, the following two aspects need to be determined: first, directory structure of file storage, i.e., the position of file directory in DMS, since the tree structure of file directory is mainly completed and maintained by DMS; second, the position of file metadata in MDS, i.e., the specific storage position of metadata. Therefore, first, Client needs to send the file directory look-up request to DMS, which will look up in the existing directory. If DMS finds the path of the directory and the bucket volume under this directory code is not full, DMS will hash the binary codes of the path and look up the storage ID of metadata in MDS according to the hash results and metadata distribution algorithm, and then return to Client which then sends the write application to the corresponding MDS. After the write operation of metadata is completed, MDS sends the success information to DMS, which will add the file name under corresponding directory and increase the actual bucket volume under this directory code by 1. If DMS finds that the bucket volume under this directory code is ready full, it will create a new directory code record, perform the binary coding on the directory according to the algorithm in Formula (1) in Article III, reset the bucket volume information and father directory information, hash the binary code of the directory, look up the storage ID of metadata in MDS according to the hash results and metadata distribution algorithm, and then return to Client which then sends the write application to the corresponding MDS. After the write operation of metadata is completed, MDS sends the success information to DMS, which will add the file name under corresponding directory and increase the actual bucket volume under this directory code by 1. If DMS does not find the directory, it will create a new directory according to the method above and then complete the file creation operation.

V. METADATA EXPRESSION

A. Directory Index Entry

We learn from the above introduction that the metadata management method adopted in this paper differs from traditional metadata management methods in that directory path information is independently managed and stored by DMS, the service is provided by one or several servers (constituting a server cluster) according to storage size, and the only directory binary code corresponding to MDS address is distributed to all directory paths through binary transcoding of path, thus guaranteeing the one-to-one correspondence of binary code and MDS address. The directory index entries of DMS include:

DirectoryPath: refers to the path name of the directory;

DirectoryID: refers to the binary code of the directory;

FatherDirectory: refers to the father directory path name of the directory;

DircetoryAC: refers to the self access control authority property of the directory;

DircetoryAC F: refers to the comprehensive access control authority property of the directory;

FileName: refers to the names of all files under the directory;

FileNum: refers to the quantity of files under the directory;

BC: refers to whether the quantity of files under the directory is full.

The directory index entry is determined only by DirectoryID, and the file metadata under this directory is distributed to different metadata servers according to the hashed value of Directory. The codes may, according to the system storage size, control the directory volume by adjusting the digits of binary codes. This way may avoid the migration and upgrading of metadata under this directory caused by directory modification. FatherDirectory refers to the upper father directory of the directory, and is used for maintaining the tree structure of directory. DirectoryAC is used for controlling user's access authority. Filename is used to express the file name under the directory code. After the introduction of bucket, a certain directory may have several codes and each code may store different files. FileNum and BC is about the information volume of bucket, with FileNum referring to the quantity of files in the bucket and BC referring to whether the bucket is full.

When accessing a certain file, it will first look up the DircetoryAC of the file directory path in DMS and determine whether the user has corresponding access authority. If the user has no access authority, it will reject the access; on the contrary, it will look up the file's ID in MDS through the binary code of the directory and return it together with DircetoryAC to Client, which will store the directory cache obtained to local Cache and then obtain file metadata in corresponding MDS. The positioning of directory index in local Cache is to increase the metadata access efficiency. In default, the user will access the local Cache for the first time. This is because the possibility of the same user accessing the same directory successively is very high according to principle of locality. In the following two circumstances, Cache will expire. First, the user detects that the cache has expired; second, the directory index that the user accesses changes.

The following is the comparison taking the file access process according to static sub-tree segmentation as an example and assuming the access file is /usr/src/test/hello.c.

The access process of static sub-tree segmentation includes: (1) GetAttr (usr); (2) LookUp (src); (3) GetAttr (src); (4) LookUp (test); (5) GetAttr (test); (6) LookUp (hello.c); (7) GetAttr (hello.c) and (8) Read (hello.c). The first seven steps is the process of obtaining metadata, Step (8) is to read the actual data of file.

The access process of the method adopted in this paper includes: (1) GetAttr (/usr/src/test/); (2) LookUp(hello.c); (3) GetAttr (hello.c) and (4) Read (hello.c). The first three steps is the process of obtaining metadata, Step (4) is to read the actual data of the file.

According to the above derivation, we may get the general case of the access process of the two methods. When the number of file layers to be accessed is n, it takes 2n+2 times to access the actual metadata through the way of static sub-tree segmentation. The first 2n+1 times are to obtain metadata, and the time is to read the metadata; the method adopted in this paper is irrelevant with the number of file directory layers, and it takes only four times to read the metadata, with the first three times obtaining metadata and the last time reading metadata. Clearly, when n>1, the method adopted in this paper has a significant higher file access rate over the static sub-tree segmentation.

B. Directory Object

In a traditional file system, the directory file content and file metadata are stored separately, and file metadata may be obtained through several times of disk I/O. The imbedding of file metadata into directory files may increase the efficiency of access [10]. The directory object referred to in this paper is classified based on the binary code of directory path and according to the number of files, and includes all metadata of files. It is similar with the directory files in traditional file systems, but differs from traditional directory file that contains only file names and index numbers and obtains index number and then reads the metadata. For the directory object adopted in this paper, when the directory object is accessed, the metadata could be obtained simultaneously.

Directory objects are uniformly managed by MDS, and each object is comprised of several fixed-number entities and file metadata entries. The file quantity of directory object is determined by bucket size, while metadata entry contains all metadata of the directory object properties. The structure of the directory object is shown as follows:

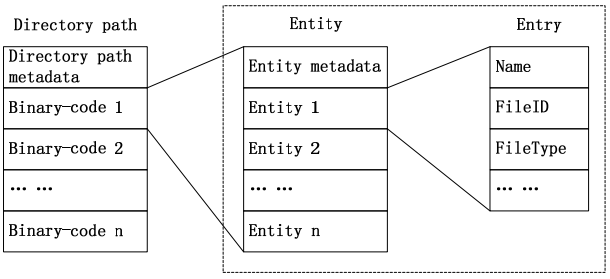

Figure 3. Structure of Directory Object

The metadata of directory object mainly includes the followings:

DirectoryID: corresponding to the DirectoryID in DMS and used for examining the binary code information of directory.

EntityID: indicating the ID of entity object inside the directory object. It is the only identifier of the entity inside the directory object, expressed by integer N, and adjustable according to bucket volume. For example, 16-digit identifier may support 65536 entities.

EntityFlag: indicating the current status of entity object;

Entry: indicating the entrance information of file metadata in file object;

Name: indicating the name of file;

FileID: indicating the globally uniform file ID of file objects;

FileType: indicating the type of file metadata;

FileFlag: indicating the current status of file metadata;

FileAC: indicating the access authority of file metadata.

Other basic properties of file metadata include mode, uid, gid, size, atime, ctime, mtime, etc.

On one hand, the use of directory object changes the inefficient linear method of positioning file metadata through level-by-level traversal index in traditional file systems and significantly reduce the disk I/O times required by metadata reading; on the other hand, the use of directory object increases the efficiency of MDS. In traditional file system, in order to avoid large volume problem, MDS needs to partition the directory volume [11] apart from the storage of metadata. The partition of directory volume will be handed over to DMS for management. In this way, MDS could be responsible for metadata storage in a more focusing and efficient manner.

VI. METADATA ACCESS

A. Setting of Access Authority

The paper classifies the access authority of a file or directory as self access authority and comprehensive authority, where self access authority refers to the own access authority of the file of the directory; that is, each file or directory has its own access authority, and is recorded in the corresponding metadata of MDS. While the comprehensive access authority is only available in directory as we define. The comprehensive access authority is the intersection of the self access authority of directory and other upper directory authority and is recorded in corresponding directory index in DMS.

This classification method mainly solves two problems. First, it reduces the directory traversal times when access authority is determined. In traditional directory-based access control methods, when the access authority of a certain file or directory needs to be determined, the user's access authority could finally be determined after the traversal of all directories under the directory path. With the introduction of comprehensive access authority, when a new directory is to be created, the intersection of the self access authority of new directory and the comprehensive access authority of upper directory is taken, then the comprehensive access authority of the new directory is obtained and recorded in the corresponding directory indexes (the self access authority

and comprehensive access authority at the first level of directory are the same). When a new directory is to be created, this method could be used recursively. In this way, when the access authority of a certain directory needs to be determined, it only needs to access the comprehensive access authority under the directory index, without the need of the traversal of all upper directories in the directory path. Second, it avoids the upgrading of a large amount of metadata resulted from the change to the access authority of directory. In a traditional file system, the access authority is saved in file metadata and distributed in each MDS, such as Lazy Hybrid method. When the access authority of a certain directory is changed, all metadata under this directory must be changed, which will result in the upgrading operation of a large amount of metadata. If the volume of file system is large, this will be a heavy task and cause the corresponding metadata compliance problems. This paper classifies the access authority into self access authority and comprehensive access authority. The file metadata of MDS only stores the self access authority, while the comprehensive access authority is stored in DMS. Therefore, when the access authority of directory is changed, it only needs to upgrade the comprehensive access authority property in corresponding directory index in DMS, without the need of changing the metadata information in MDS.

The following figure illustrates the while construction process of access authority:

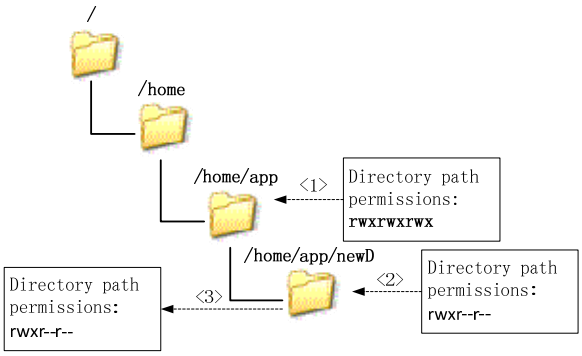

Figure 4. Construction of Directory Comprehensive Access Authority

According to Linux file system rules, the access authority of the directory is expressed with 9 characters, with the left three characters indicating the authority of the owner, the middle three characters indicating the authority of users at the same group with the owner and the right three characters indicating the authority of other users. The characters r, w and x means read, write and execute respectively. Assume a user creates a new directory /root/app /newD under /root/app:

(1) First, obtain the comprehensive access authority property of directory /root/app; assume it is rwxrwxrwx, and then the user has the authority to create a new directory under this directory.

(2). Second, the user sets the self access authority of the new directory newD; assume it is rwxr--r--.

(3) Last, take the intersection of the comprehensive access authority of upper directory /root/app and the self access authority of the new directory new D, and obtain the comprehensive access authority of directory /root/app / newD as rwxr--r--.

B. Metadata Positioning

The metadata positioning operation mainly adopts the method of combining hashed and mapping tables. Hash algorithm mainly includes the three algorithms in Article III, which are established in DMS. The directory binary codes and the mapping table corresponding to MDS needs to be established in DMS according to the results from hash algorithm. The mapping table is classified into main mapping table and secondary mapping table, with the first one being the mapping established with the value from hash algorithm and MDS according to directory binary code and the latter one being the mapping established directly with the directory binary code and MDS. The MDS found through main mapping table is called the main MDS of a certain directory path, while the MDS found through the secondary mapping table is called the secondary MDS of a certain directory path. In default, the first lookup of metadata positioning is to look up the MDS entrance of the secondary mapping table and then, if there is no corresponding entrance, look up the MDS entrance of the main mapping table. The establishment of the main and secondary mapping tables is shown in Table 1 and Table 2.

TABLE 1.
MAIN MDS MAPPING TABLE

Value	IP address of MDS
0	192.168.1.1
1	192.168.1.2
2	192.168.1.3
3	192.168.1.4
4	192.168.1.11
5	192.168.1.1

TABLE 2.
SECONDARY MDS MAPPING TABLE

Value	IP address of MDS
1001	192.168.1.1
2001	192.168.1.2
3008	192.168.1.3
1008	192.168.1.4
…	…

The advantages of classifying the mapping table into main mapping table and secondary mapping table include that the MDS load is balanced and that no certain MDS will become a hot access point. The main mapping table is established when a new directory is created. It is determined according to the hashed value of the binary code of directory path. According to the characteristic that the binary code is globally unique and will not change, the main MDS of any directory is unchanged, regardless of the change to directory property. Under

normal load circumstances, all metadata access will be responded by the main MDS. In addition, according to the processing capacity difference of each MDS, more entrances may be allocated to the main mapping table of the MDS with strong capacity, so that it could undertake more metadata tasks. For example, the processing capacity of the IP 192.168.1.1 in Table 1 is twice the capacity of 192.168.1.11, and then the mapping entrance in the main mapping table is also twice the capacity of 192.168.1.11.

In practical situations, unbalanced load or even more extreme cases may be unavoidable. Therefore, we balance the load through the way of establishing mapping tables. The specific method is to set a counter and timer in the directory object of MDS, while the access frequency (counter/timer) indicates the access enthusiasm of directory object, then MDS transmit through the heartbeat mechanism the directory information and access frequency to DMS, which will select the MDS with lightest load as the secondary MDS of the directory according to the set access frequency threshold after receiving the information, and insert the mapping information into the secondary mapping table. If the access frequency of a certain directory in the secondary mapping table also reaches the threshold, then MDS will continue to add directory mapping relations from the secondary mapping table. When the metadata read entrance is to be selected, it is required to select the lightest entrance. In order to guarantee the conformity of metadata, the secondary table is only responsible for the read operation of metadata, while the write operation must be treated by the main mapping table.

VII. Simulation and Implementation

In order to verify the effectiveness of DPCH proposed in this paper, the paper compares this method and several typical metadata management methods through experiment, which utilizes 10 computers with LINUX operation system (i5-3570K processor, 4GB memory and 500GB hard disk) and connected through gigabit Ethernet. The prototype system uses JAVA language and Iozine as performance testing tool.

The experiment is consisted of five parts; i.e., file creation, file deletion, file reading, metadata distribution and metadata migration.

A. File Creation

The file creation method is to create a 0 word length file, because a 0 word length file does not requires the involvement of storage node server and could be completed by MDS independently, thus reducing the impact of other links on metadata' processing capacity and enabling the test results to be more accurate. Then, the file creation efficiency will be summarized, with throughput rate as the performance evaluation index, i.e., the number of files created in a unit time.

The experiment includes balanced access load and concentrated access load, with the former one referring to the file creation of Client under different directories and the latter one referring to the file creation of Client under the same directory. The experiment uses 10 clients, 3 DMS nodes, 7 MDS nodes and 7 layers of directory tree for new files, with each directory containing 7 sub-directories and 50 files.

The following figures show the experiment results of the two circumstances and the performance comparison with other metadata management methods. Figure 5 shows the file creation throughput rate of balanced access load, and Figure 6 shows the file creation throughput rate of concentrated access load.

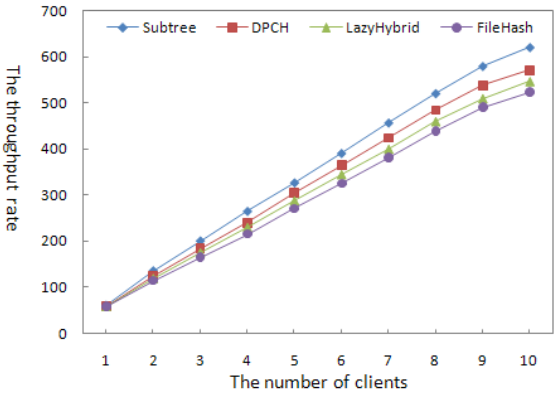

Figure 5. File Creation Throughput Rate of Balanced Access Load

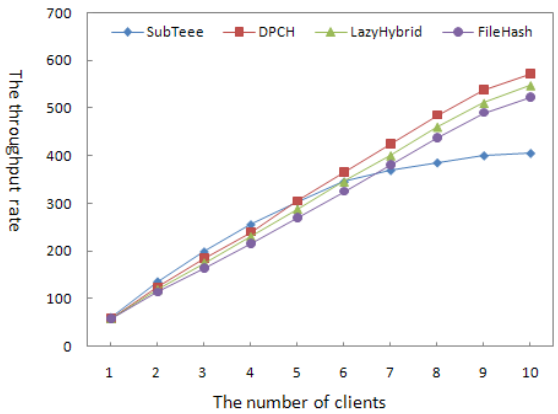

Figure 6. File Creation Throughput Rate of Concentrated Access Load

According to Figure 5, under balanced access load, the Sub-tree partition has the best throughput rate because this algorithm processes the sub-tree directories in the same MDS, thus enabling all metadata requests under this directory to be completed in the same MDS, reducing the disk I/O times of MDS and increasing the file creation rate. The FileHash algorithm has the poorest throughput rate, since this algorithm breaks the traditional directory hierarchy structure concept of file systems and distributes files through HASH evenly to each MDS, thus directly causing massive MDS access requests and causing the files under the same directory to be distributed in different MDS. This algorithm needs the traversal of several MDS, significantly increasing the disk I/O times for accessing MDS. LazyHybrid algorithm has improved throughput rate over the FileHash algorithm, but is basically a file-based hash algorithm. It only makes some improvements on file pre-treatment and Cache. The algorithm proposed in the paper has a larger partition granularity than that of the files, but is not completely the

same with the Sub-tree partition. The creation process involves the directory binary coding and hash distribution function, thus the throughput rate under balanced access load is lower than that of Sub-tree partition but higher than that of FileHash algorithm and LazyHybrid algorithm.

According to Figure 6, under concentrated access load, the Sub-tree partition has an obvious low throughput rate. This is because the files under the same directory will increase with the increase of client nodes, which will directly result in the load increase of MDS storing the directory, thus causing it to be the access hot point and finally the access bottleneck and reducing the file creation throughput rate. The other three algorithms adopt Hash distribution, and could still better process the access hot point problem under concentrated access load. The file creation throughput rate has only a minor difference with that under balanced access load, and the throughput curve also has only slight change with the increase of clients. It is particularly worth mentioning that the algorithm proposed in this paper takes directory code as hash unit, with partition granularity larger than that of files, but it is different from traditional directory structure. Through the bucket partition and the two mechanisms, i.e., main mapping table and secondary mapping table, this algorithm may enable the file metadata to be evenly distributed in each MDS, and the file creation throughput rate is better than that with FileHash algorithm and LazyHybrid algorithm.

B. *File Deletion*

The file deletion method is to delete the files based on the new files created above, and the file deletion throughput rate is also taken as the evaluation index. The experiment environment is the same with that of file creation operation above. The following figures show the file deletion throughput rate of the two circumstances and the performance comparison with other metadata management methods.

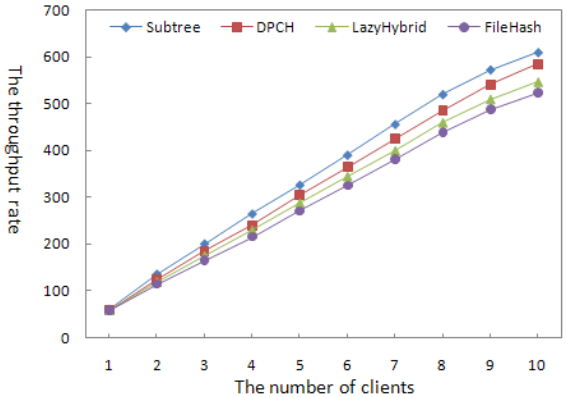

Figure 7. File Deletion Throughput Rate of Balanced Access Load

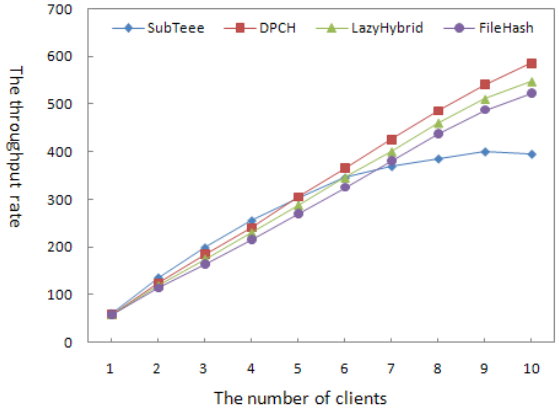

Figure 8. File Deletion Throughput Rate of Concentrated Access Load

According to Figure 7, under balanced access load, the experiment results are basically the same with that in previous description. The Sub-tree partition has the best file deletion throughput rate, followed by the algorithm adopted in this paper, LazyHybrid algorithm and FileHash algorithm in order. The reasons for the performance differences of these algorithms are the same with the previous description and will not be repeated here.

The same conclusion could also be obtained from Figure 8. Under the Sub-tree partition, an access bottleneck will be formed with the increase of clients, resulting in the decrease of throughput rate. For the other algorithms adopting hash distribution, the throughput rate will be reduced significantly with the increase of clients.

C. *File Reading*

File reading mainly refers to the file reading operation from disk to memory. In file metadata operation, reading operation accounts for a considerable proportion, thus the file reading time is an important index to evaluate the performance of different algorithms. The experiment data is 10000 files randomly distributed under 1000 directories. The method adopted is to test the total reading times of each algorithm under the same file quantity and in the same reading order. The following figure shows the reading times of several algorithms.

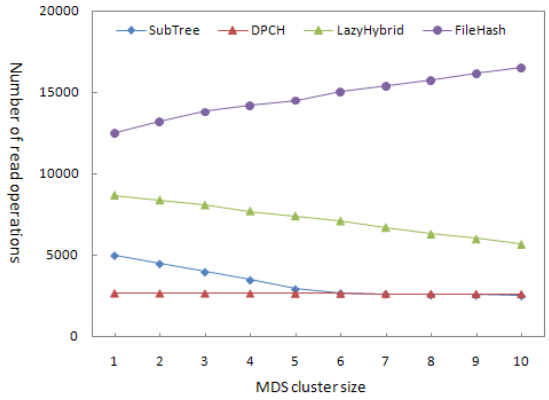

Figure 9. Reading Times of Different MDS

According to Figure 9, the algorithm adopted in this paper and the Sub-tree partition have the least reading time, followed by the LazyHybrid algorithm and

FileHash algorithm in order. The main reason for this difference is due to the different Cache hit rate. The basic idea of the algorithm adopted in this paper and the Sub-tree partition is to distribute metadata into different MDS according to the file location in directories. The only difference of the two algorithms is the processing mode. This partition may optimize the directory storage locality and reduce the overlapping of prefix directory in different MDS, thus increasing the Cache hit rate and reducing the reading times. Particularly, the algorithm adopted in the paper also introduces the concept of comprehensive access authority, which may further reduce the reading times enquired by directory traversal. LazyHybrid algorithm and FileHash algorithm determine the metadata distribution in MDS according to the hash value of full-path file names, which results in the distribution of metadata under the same directory into different MDS and ignores the directory storage locality and prefix directory MDS overlapping, thus reducing the Cache hit rate and increasing the reading times. LazyHybrid algorithm also adopts the method of storing directory access authority, thus having less reading times than FileHash algorithm.

D. Metadata Distribution

Metadata distribution mainly refers to the distribution of metadata in each MDS. The experiment method is to store, in the same order, 10000 files in 10 MDS with same size. The figure below shows the metadata distribution in different MDS in different algorithms.

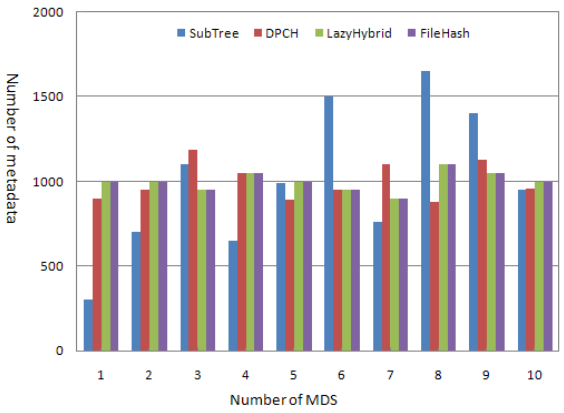

Figure 10. Metadata Distribution in Different Algorithms

According to Figure 10, LazyHybrid algorithm and FileHash algorithm have well balanced metadata distribution, with minor difference of metadata quantity in each MDS. This is because the two algorithms take files as the partition granularity and evenly distribute each metadata into each MDS through hash algorithm. It is followed by the algorithm proposed in the paper. Since the partition granularity is larger than files, the algorithm proposed in the paper may have less balanced metadata distribution in certain extreme circumstances compared with the previous two algorithms. The Sub-tree partition has the poorest metadata distribution, with obvious difference of metadata quantity in each MDS.

E. Metadata Migration

Metadata migration refers to the migration of metadata in each MDS when the user modifies directory properties (path, directory name, etc.). The experiment method is, based on the previous description, to draw 1-10% of all directories at random. The figure below shows the metadata migration after the modification of directory property in centralized algorithm.

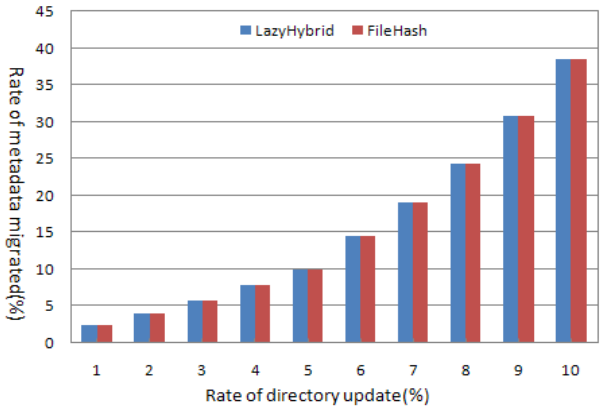

Figure 11. Metadata Migration after Modification of Directory Proportion

According to the above figure, LazyHybrid algorithm and FileHash algorithm have the largest metadata migration amount. With the increase of directory modification amount, the metadata migration amount may reach 38.5%. This is because the two algorithms adopt hash distribution based on file full-path name. The modification of directory property results in the migration of a large amount of metadata in MDS. The algorithm proposed in the paper performs hash distribution after binary coding of path names. The modification of directory property will not affect the metadata distribution in MDS and has no metadata migration. The Sub-tree partition also only needs to modify a single directory property, and it involves no migration of metadata.

F. Experiment Summary

According to the simulation experiment results above, a comprehensive comparison on different metadata management performances is shown in the following Table 3. The evaluation adopts five rating points, with 1 as the poorest and 5 as the best.

TABLE 3
PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS

	Creation throughput rate	Deletion throughput rate	Reading times	Metadata distribution	Metadata migration
Sub-tree partition	3	3	5	2	5
DPCH	4	4	5	4	5
LazyHybrid	3	3	2	5	2
FileHash	3	3	2	5	2

According to the table above, the metadata management method proposed in the paper shows good performance in all aspects, i.e., throughput rate, reading

times, metadata distribution and metadata migration, and has the highest comprehensive points.

VIII. Conclusion

Based on the study on different metadata management methods, this paper proposes improvements for existing problems as well as the Directory Path Code Hash(DPCH). This method is to store directory and file metadata separately, perform hash distribution through binary coding of directory paths, and introduce the concepts of bucket partition, comprehensive access authority and main and secondary mapping tables, effectively solving the unbalanced metadata distribution and access hot point problems in Sub-tree partition and the excessive reading times and large metadata migration amount after directory property modification in LazyHybrid algorithm and FileHash algorithm. The experiment indicates that the method proposed in this paper significantly outweighs other algorithms in terms of throughput rate, metadata distribution, reading times, etc.

Acknowledgment

Fund project: the army logistics key scientific research program funded projects of PLA (BS211R099).

References

[1] Roselli D, Lorch J, Anderson T. A comparison of file system workloads [C]// Proceedings of the 2000 USENIX Annual Technical Conference, San Diego. Boston: USENIX Association, 2000: 41–54.
[2] Grorge A, Garcia J, Kim K. Distributed parallel processing techniques for adaptive sonar beamforming[J]. Journal of Computation Acoustics, 2002, 10(1):1-23.
[3] Weil s a, Pollack K T, Brandt S A, et al. Dynamic metadata management for petabyte-scale file systems[C]. ACM/IEEE Conference on Supercomputing, DC, USA, 2004:4.
[4] National Information Standards Organization. Understanding Metadata. [2012-01-01]. http://www.niso.org.
[5] Gongye Zhou, Qiuju Lan, Jincai Chen. A dynamic metadata equipotent subtree partition policy for mass storage system //Proc of the Japan-China Joint Workshop on Frontier of Computer Science and Technology. 2008.
[6] Walter A, Francesco G, Marco M. Indexing and retrieval of multimedia metadata on a secure DHT. Informatica, 2009,33(1):85-100.
[7] Popek G J, Rudisn G, Stoughton A, et al. Detection of mutual inconsistency in distributed system[J]. IEEE Transactions on software engineering, 1986,12(11):1067-1075.
[8] Corbett P F, Feitelso D G. The Vesta Parallel File System[J]. Transactions on Computer Systems, 1996, 14(3): 225-264.
[9] Brandt S A, Lan Xue, Miller E L, et al. Efficient Metadata Management in Large Distributed File Systems[C]//Proceedings of the 20th IEEE NASA Goddard Conference on Mass Storage Systems and Technologies. [S. 1.]: IEEE Press, 2003: 290-298.
[10] Ganger GR, Kaashoek MF. Embedded inodes and explicit groupings: Exploiting disk bandwidth for small files. In: Proc. Of the 1997 USENIX Annual Technical Conf. Anaheim: USENIX, 1997.1−17.
[11] Litwin W, Neimat MA, Schneider DA. LH—A scalable, distributed data structure. ACM Trans. On Database Systems, 1996,21(4):480−525.
[12] Tong Yang. Mass Data Analysis and Forecasting Based on Cloud Computing. Journal of Software, Vol. 7, No. 10, 2012:2189-2195.
[13] Xindong You. ARAS-M: Automatic Resource Allocation Strategy based on Market Mechanism in Cloud Computing. Journal of Computers, Vol. 6, No. 7, 2011:1287-1296.

Hui Cai, born in 1983, Ph. D. His research orientations are cloud and P2P computing.

Shenlin Li, born in 1964, professor, Ph.D. His research interests include Logistic Informationization and cloud computing.

Zhaoxia Wang, born in 1973, Ph.D. Her research direction includes cloud and grid computing.

Xingchen Li, born in 1986, Ph.D. His main research interests are data processing and intelligent interaction of the IOT.

Lichuan Huang, born in 1985, M.S. His main research interests are oil gas station management, and oil Informationization.