Optimizing the Storage of Massive Electronic Pedigrees in HDFS

Yin Zhang[1], Weili Han[1,2], Wei Wang[1],Chang Lei[1]
1. Software School, Fudan University, Shanghai, China
2. DNSLAB, China Internet Network Information Center, Beijing, China
{08302010001, wlhan, 08302010083, 10212010010}@fudan.edu.cn

Abstract—**Benefiting from trustworthily tracking of the processes in the production, processing, storage, transportation and sale phases, an electronic pedigree system becomes an important technology of the Internet of Things. In an electronic pedigree system, small-sized but huge volume of electronic pedigrees in the XML format will be generated, stored, and retrieved. Unfortunately, study of these massive electronic pedigrees' storage in an electronic pedigree system, which is in the form of small XML files, is rarely concerned. We, therefore, try to leverage Hadoop to solve the storage problem of massive electronic pedigrees, by the optimization of storing and accessing massive small XML files in HDFS. First, all correlated small XML files of the same envelope are merged into a larger file to reduce the metadata occupation at NameNode. Second, a prefetching mechanism and a remerging mechanism are used to improve the efficiency of accessing small XML files. Finally, we implement a prototype to evaluate the effectiveness and efficiency comparing with the origin HDFS. The results show that the optimized approach is able to reduce the memory consumption of NameNodes by up to 50%, improve performance of storing by up to 91%, and accelerate accessing by up to 88% in Hadoop.**

Keywords- electronic pedigree; small XML files; HDFS

I. INTRODUCTION

Pedigree can refer to the lineage or genealogical descent of people, whether documented or not, or of animals, whether purebred or not [17]. Drawing lessons from the concept of pedigree of people and animals in the real world, the electronic pedigrees [14][15][16][25], which contain the traces of objects in the Internet of Things (IoT for short), can be recorded, signed and verified, then used to anti-counterfeit [14][15]. An electronic pedigree system, which uses digital signatures to ensure the integrity of electronic pedigrees, is an important technology of the Internet of Things. In the system, an electronic pedigree storage server (EPSS for short) is required to manage electronic pedigrees and permit consumers to access electronic pedigrees which they search for. The EPSS can be deployed with an EPCIS [20], or alone. In the latter deployment, the EPSS usually stores massive data, and can be applied to assure the food safety [25].

When the technology of the electronic pedigree is used in various scenarios, from original scenario such as drug anti-counterfeit, to more application scenarios such as food [25] and other agricultural products, massive electronic pedigrees need to be generated, retrieved, and verified on a daily basis. Small and XML formatted are the rest two key features of these electronic pedigrees. For example, when a product is produced, only one piece of *Initial Pedigree* [25] will be generated and signed, in a KB-sized file. Thus, it is a challenge for operators to store the small-sized but huge volume of electronic pedigrees in the XML format.

HDFS (Hadoop Distributed File System) is an excellent distributed file system that is designed to reliably store very large files on the compute nodes, providing very high aggregate bandwidth across the cluster [1]. HDFS is designed so that node failures can be automatically handled in Hadoop. In addition, HDFS is implemented in the master-slaves architecture. HDFS stores file metadata and file content data separately. HDFS stores file metadata on a master server, called NameNode. File content data are stored on slave servers, called DataNodes. Both the NameNode and DataNodes communicate with each other based on TCP/IP protocols. To meet requirements of performance, scalability and reliability while processing large data sets, the EPSS leverages HDFS to store massive electronic pedigrees majorly in the form of XML files.

However, HDFS is originally designed for storing large files with streaming data access patterns [5], thus stores small files inefficiently. Storing amounts of small files in HDFS will result in high memory usage of NameNode and huge access cost. Meanwhile, in an electronic pedigree system, electronic pedigrees are usually small with a size of tens of KB around. The total electronic pedigree data sets in the EPSS have a large scale with massive XML files while sizes of single files are very small. Therefore, storing and managing massive electronic pedigrees becomes a big challenge to HDFS.

In this paper, we propose an approach to optimize performance of storing and accessing massive small XML in Hadoop, to reduce the metadata occupation at NameNode and to improve the efficiency of accessing small XML files. Taking the characteristics of electronic pedigrees' relevance and serialization among small XML files into consideration, some features are added such as grouping the same envelope and prefetching mechanism. At a high level, the approach is to combine small XML files which belong to the same envelope into a large one to reduce the file number and build internal index for each large file and to use prefetching mechanism to speed up accessing small XML files. The basic file operations such as reading, writing and deletion are supported. We finished the prototype, and the experiment results show that the optimized approach can achieve remarkable improvement in the performance of storing and accessing massive small XML files in HDFS.

The rest of this paper is organized as follows: Section II describes the background and motivation of this paper. Section III introduces the approach to optimize the storage of massive electronic pedigrees in HDFS in detail. Section IV evaluates the proposed approach. Section V investigates the related work. And Section VI concludes the work in the paper, and introduces the future work.

II. BACKGROUND AND MOTIVATION

A. HDFS

Hadoop is an open-source software framework developed for reliable, scalable, distributed computing and storage [1]. The Apache Hadoop Project contains two important parts. One is a distributed file system to support storage and the other is a framework to support the analysis of massive data sets [2]. Hadoop Distributed File System (HDFS), which is an open-source implementation inspired by GoogleFS, is the primary storage system used by Hadoop applications [4]. In recent years, HDFS has become a popular file system running on clusters [6], and is widely used to support many Internet applications as the infrastructure of file storage. For example, Facebook uses HDFS to store copies of internal log and dimension data sources, and uses it as a source for reporting and analytics and machine learning in two major clusters, one with 1,100 machines and the other with 300 machines. Adobe currently has about 30 nodes running HDFS, used in several application areas like social services [7].

HDFS typically consists of one NameNode, which is the master, multiple DataNodes, which are slaves, and several HDFS clients. The NameNode maintains a namespace tree and the mapping of file blocks to DataNodes in RAM [2], while DataNodes maintain the file content. The file content is split into fix-size blocks (typically 64 megabytes) and each block of the file is independently replicated at multiple DataNodes (typically three). A DataNode sends a block report to the NameNode for identifying block replicas in its local machine. Besides, DataNodes inform the NameNode of their active states and availability of the block replicas they host by sending heartbeats reports.

HDFS can support basic file operations: write, read, append and delete. When a HDFS client adds data to HDFS, it firstly sends a write request to the NameNode for a new block. The NameNode will assign a block with a unique block ID and a list of DataNodes to host replicas of the block. After receiving feedback from the NameNode, the HDFS client will send files to the DataNodes with packets. At last the HDFS client and DataNodes will notify the NameNode to update new block's metadata information. Once the operation finishes, the file will be closed and the content of the file cannot be changed or removed until reopening the file. Hadoop Distributes File System is of one single-writer, multiple-reader model [2]. When a HDFS client reads data from HDFS, it sends a read request to the NameNode for the requested file. And the NameNode will send the list of file blocks and the locations of each block replica to the HDFS client. The HDFS client tries the closest replica first to read the file content. When a HDFS client deletes data in the HDFS, it sends a deleting request to the NameNode. The NameNode will look for the blocks which belong to the file and remove the metadata only, but the real data will be deleted later which can be utilized in the configuration.

However, HDFS is originally designed to store large files rather than massive small files [2]. When storing massive small files on HDFS, the performance bottlenecks exist in HDFS so that HDFS designers encourage HDFS users to create larger files. For example, when 458,000 small files whose sizes range from 32 KB to 3,796 KB were stored on HDFS, about 184.86 MB memory of NameNode was occupied. It can be deduced that 24 million files will consume 16 GB memory of NameNode [12].

B. Electronic Pedigree System

As is shown in Figure 1, EPCglobal [19] proposes the concept and standard of electronic pedigree which offers the trustworthy assurance for generating data in RFID-enabled systems. An electronic pedigree system can be easily used to trustworthily track the processes in the production, processing, storage, transportation, sale, and even consumption phases. Extended from the standard of EPCglobal, a pedigree can record, in the electronic form, the traces of the foods and agricultural goods from the point of production and contains information about all transactions that the foods and agricultural goods undergo until they reach the end user [25]. Electronic pedigrees with different types contain different attributes and parameters. Attribute '*lot number*' records the lot number of the goods which contains the item corresponding to the electronic pedigree; attribute '*item serial number*' records the serial number of the item corresponding to the electronic pedigree; attribute '*pedigreeID*' records the unique identifier of the electronic pedigree. The electronic pedigrees are usually small, with sizes ranging from tens of KB to hundreds of KB. And each electronic pedigree is of a nested architecture. An EPSS is responsible for storing massive electronic pedigrees and searching for the specific electronic pedigree. Two ways can be used to store electronic pedigrees in the EPSS: the first is the one-by-one exportation by a production system; and the second is the enveloped exportation by a production system. Enveloped exportation means that several electronic pedigrees can be enveloped as a whole and be exported together to the EPSS. While a production system exports electronic pedigrees, electronic pedigrees which need to be exported are selected, the envelope with abstract information is generated and the envelope and the selected electronic pedigrees are bound and saved as a whole. Note that, the envelope around many electronic pedigrees with the internal index file is an extended feature of the EPCglobal standard.

Characteristics of electronic pedigree are summarized as follows:

- Some attributes of electronic pedigrees are serialized, such as pedigreeID, item serial number, lot number and so on. These attributes can be used to access electronic pedigrees. Thus these serialized attributes should be abstracted during optimization.
- Once electronic pedigrees are generated, they will not be changed. But these electronic pedigrees will be retrieved frequently.
- Electronic pedigrees are uploaded in two different ways. Electronic pedigrees may be uploaded one by one, or an envelope may be uploaded, which contains a

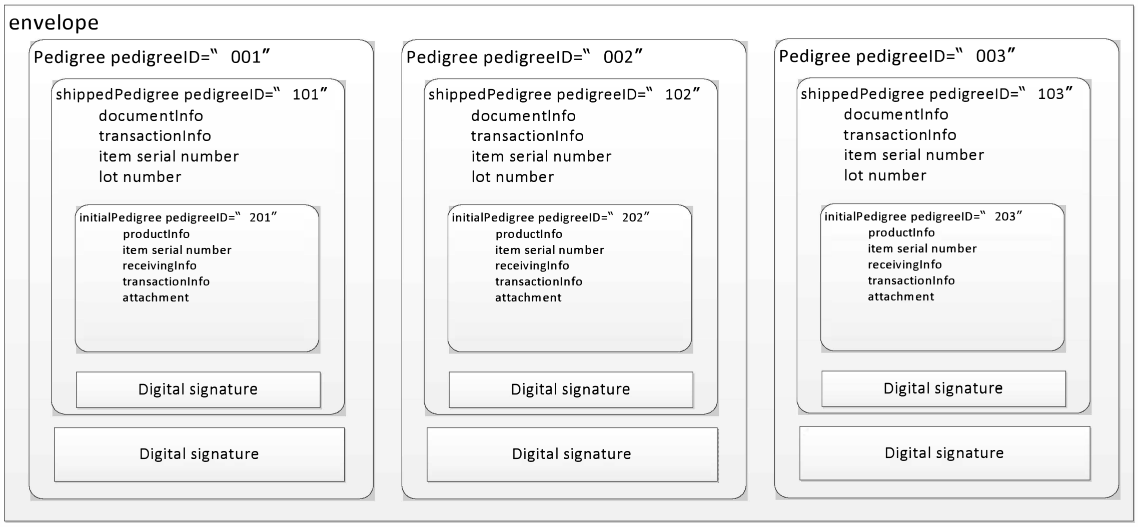

envelope
Pedigree pedigreeID=" 001"
shippedPedigree pedigreeID=" 101"
documentInfo
transactionInfo
item serial number
lot number
initialPedigree pedigreeID=" 201"
productInfo
item serial number
receivingInfo
transactionInfo
attachment
Digital signature
Digital signature
Pedigree pedigreeID=" 002"
shippedPedigree pedigreeID=" 102"
initialPedigree pedigreeID=" 202"
Pedigree pedigreeID=" 003"
shippedPedigree pedigreeID=" 103"
initialPedigree pedigreeID=" 203"

Figure 1. Structure of an envelope and pedigrees

lot of electronic pedigrees. Electronic pedigrees which come from one source or by one envelope are correlated. The goods which they are on behalf of are always in the market in the same period. When a consumer accesses an electronic pedigree, there is great probability that its correlated electronic pedigrees will be accessed by other consumers of the same area in the next period.

- Electronic pedigrees will not run out of date, but the frequency of accessing electronic pedigrees will turn down with time passing by. Considering the freshness date of goods, nobody will bother to ask about most old electronic pedigrees. Therefore, after specific time, electronic pedigrees can be merged again into bigger files.

C. *Motivation*

This paper is motivated to propose an optimization solution to store massive but small XML files in HDFS, using an electronic pedigree system [25] as the typical application scenario.

Three reasons which result in small files problems of HDFS are concluded, when the EPSS leverages HDFS to store electronic pedigree files. First, high time cost is caused by the metadata management of HDFS. In HDFS, each file has intended number of replicas, and each replica has its own metadata. The metadata mechanism of HDFS requires all the DataNodes which contains replicas and the NameNode to keep metadata consistent. Thus, it is necessary to communicate between the NameNode and all the machines which contain replicas. When the electronic pedigree system stores massive electronic pedigrees in HDFS, the majority of time is spent in managing metadata instead of file content transferring. Second, high memory cost is caused by large amount of files in HDFS. In HDFS, all the namespace and block addresses are located in the memory of the NameNode. No matter how small the files are, the more files stored in HDFS, the higher memory occupation in NameNode's memory. Moreover, the number of files and available block addresses are limited by the size of the NameNode heap. Third, high storage cost is caused by the storage mechanism of HDFS. In HDFS, the HDFS client needs to send writing request to the NameNode when storing an electronic pedigree. This behavior cannot be avoided for each file's storage. For massive electronic pedigree storing requirements every day, the HDFS client has to contact with the NameNode quite a lot of times.

Therefore, storing and reading massive but small XML files such as electronic pedigrees in the EPSS normally causes lots of requirements and seeks among DataNodes to deal with each small file if using HDFS straightly [13]. In order to meet the electronic pedigree system's requirements of electronic pedigree I/O performance, we design a prefetching mechanism and a merging mechanism as a middle-tier for HDFS at the application level, and characteristics of the file in special applications have to be considered.

III. SMALL FILE TUNING APPROACH FOR HDFS

A. *Overview of our approach*

The basic idea of our approach can be concluded in three aspects: (1) merging small XML files into big files to reduce the number of files in HDFS; (2) prefetching part of XML files into EPSS client to improve the speed of accessing files; (3) remerging part of early big files to reduce the number of files in HDFS further and speed up accessing files. During the process of designing this approach, characteristics of electronic pedigree are considered fully and our designs are able to meet them. First, considering the correlation among files of a same envelope, or of adjacent serialized codes, these files are gathered and merged into a big file to store in HDFS. Second, based on the access probability among correlated files, a prefetching mechanism is designed. Third, based on the curve of frequency of accessing electronic pedigrees, a remerging mechanism is designed.

The EPSS of electronic pedigree system utilizes the proposed approach. And its framework can be divided into three layers. The User Interface layer is used to interact with users, which provides interfaces of functions like uploading electronic pedigrees, and searching for specific electronic pedigree. The Business layer is the most important part of the EPSS, which provides business models, such as file merging, web container, file mapping, prefetching cache and file remerging. The Persistence layer is used to store electronic pedigrees using HDFS.

B. *File merging*

After electronic pedigrees are uploaded to the EPSS, the correlated electronic pedigrees are merged into a bigger file. There is an internal index file in each big file. An internal index file contains many items. And each item contains the name, offset and size of each XML file in the big file. It is stored in the beginning location of a big file. From the perspective of HDFS, the internal index file is part of file content and will not produce additional overhead for metadata in the NameNode. Using internal index files is helpful to manage the information of XML files in a big file and to verify the existence of the XML file when accessing it.

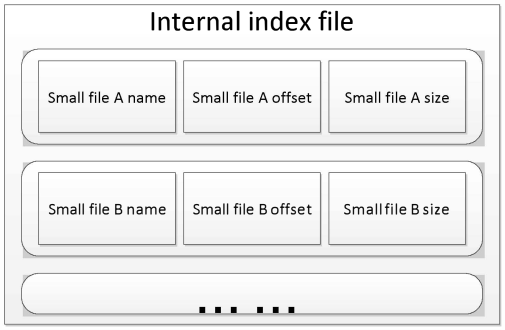


Figure 2. The structure of an internal index file

Considering the serialization of some electronic pedigree's attributes and two approaches of uploading electronic pedigrees to the EPSS, four strategies are adopted to decide what correlated electronic pedigrees are and to guarantee the efficiency of accessing XML files.

The first strategy is to decide the correlation of electronic pedigrees on the basis of electronic pedigree attribute '*lot number*'. The second strategy is to decide the correlation of electronic pedigrees on the basis of electronic pedigree attribute '*item serial number*'. The third strategy is to decide the correlation of electronic pedigrees on the basis of electronic pedigree attribute '*pedigreeID*'. The fourth strategy will be introduced later. These three strategies are similar, except according to different electronic pedigree attributes, which are all serialized. Which strategy will be chosen is decided by the attribute that the EPSS provides for user to search for specific electronic pedigrees.

In these three strategies, the file merging process has following steps.

Step 1: each correlated electronic pedigree's name and size are prepared.

Step 2: the internal index file is established. Since each correlated electronic pedigree's size has been known, each correlated electronic pedigree's offset can be calculated.

Step 3: the internal index file is written into a bigger file, and then all the correlated XML files are merged in turn.

The fourth strategy is to consider all the electronic pedigrees in the same envelope as correlated electronic pedigrees. Since each envelope contains an envelope internal index file, the name and size of each electronic pedigree in the envelope have been recorded.

In the fourth strategy, the file merging process has following steps.

Step 1: the internal index file is parsed to get the name and size of each electronic pedigree in the envelope.

Step 2: the internal index file is established. Since each correlated electronic pedigree's size has been known, each correlated electronic pedigree's offset can be calculated.

Step 3: the internal index file is written into a bigger file, and then all the correlated XML files are merged in turn.

The choice of these four strategies will also influence the result of prefetching XML files.

C. *File mapping*

When a reading request happens, what the user needs is a small XML file. However, HDFS stores merged files in the persistence layer, so a HDFS client has to convert the small XML file reading request to the merged file reading request and then send the merged file reading request to HDFS. Therefore, the mappings between small XML files and the corresponding merged file should be stored in the business layer.

A general solution is to build a global mapping table, which items are records between each small XML file and its merged file. The global mapping table is stored in the HDFS client and keeps a persistent file in disk. Even though the HDFS client meets problem and restarts, the mapping table will not be lost. Based on the global mapping table, renaming electronic pedigrees are proposed to reduce the size of the global mapping table. Considering that *lot number*, *item serial number* and *pedigreeID* are all serialized, these three attributes are used to rename correlated electronic pedigrees. Then renamed electronic pedigrees whose names are coherent are merged into a larger one. As a result, a range of serialized names can be abstracted for each merged one.

However, in an electronic pedigree system, this solution is not enough. Users usually search for an XML file by certain attributes such as *pedigreeID* rather than the small XML file's name. So before querying the global mapping table between small XML files and merged files, the requested attribute should be converted to the small XML file reading request. It means that there must be an indexing table between attributes and small XML files, too. In addition, more than one attributes can be used to search for an XML file. If setting up only one indexing table, too much information will reduce the efficiency of accessing the XML file. A better approach is to establish an indexing table for each attribute which can be used to search for.

E-pedigree file name(var)	Start(Long)	End(Long)	Merged file name(var)

Figure 3. Data structure of global mapping table

D. *File operations*

In this section, basic file operations of the EPSS will be introduced, including storing, and reading. For non-repudiation of electronic pedigrees, we do not redesign the deleting file operation. And in an electronic pedigree system, the EPSS follows a single-writer, multiple-reader model of HDFS. Therefore, the updating file operation is also not considered.

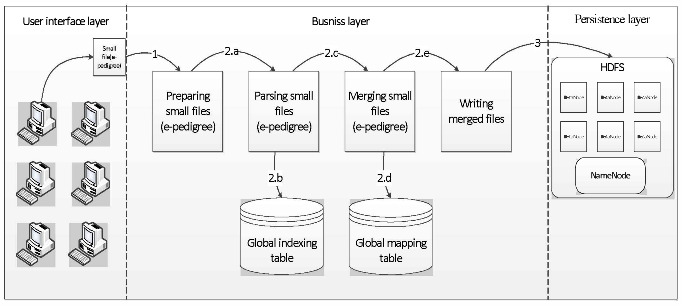


Figure 4. Storing process in our approach (e-pedigree refers to electronic pedigree)

As is shown in Fig. 4, the storing process consists of preparing XML files, file merging process and storing merged files into HDFS. The details are described as following steps:

Step 1: preparing XML files. Electronic pedigrees are uploaded to EPSS either one by one or by envelopes.

Step 2: file merging process. a) Parsing small files. b) Establishing global indexing table. c) The internal index file is established. According to the length of each correlated files, the offset of them can be calculated. Then correlated electronic pedigrees are merged into a big file in turn. d) Establishing the global mapping table. e) Finishing preparing merged files.

Step 3: storing into HDFS. HDFS client sends a writing request of the merged file to HDFS. This step is the same as a normal writing process in HDFS.

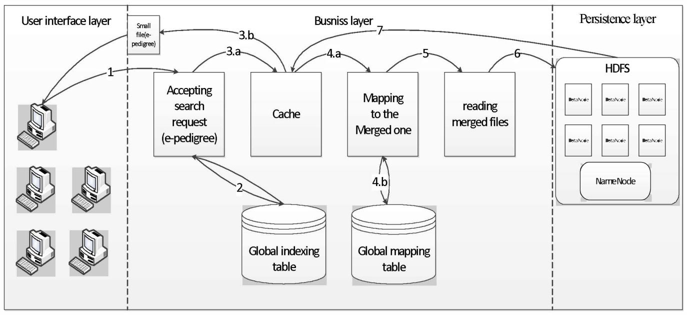


Figure 5. Reading process in our approach

As is shown in Fig. 5, the reading process will be triggered when users search for some specific electronic pedigrees. The reading process mainly includes querying the small XML file information by index, mapping the requested electronic pedigree to the merged file, getting the merged file, verifying the internal index file, returning the requested electronic pedigree, and prefetching. The details are described as following steps:

Step 1: Accepting the request of searching for specific electronic pedigree. The request contains some attributes such as pedigreeID and item serial number.

Step 2: querying the indexing table. According to the attribute in the request, the electronic pedigree information can be found in the corresponding indexing table.

Step 3: A) querying the cache. The cache checks whether the requested electronic pedigree exists. B) If it does, the requested electronic pedigree is read from the cache and the reading process has finished. Otherwise go to the next step.

Step 4: A) mapping to the merged file. B) Through querying the global mapping table and the global remapping table, the requested electronic pedigree can be mapped to the merged file.

Step 5: reading from HDFS. HDFS client sends a reading request of the merged file to HDFS. This step is the same as a normal reading process in HDFS.

Step 6: accessing the requested file. The internal index file of the merged file is split and used for the offset and the size of the small XML file. Then the requested electronic pedigree is split from the merged file, and is shown in the user interface layer.

Step 7: the prefetching mechanism is triggered and correlated files are stored into cache. The cache is updated.

E. *Prefetching*

In the current HDFS, there is no prefetching technology, which is a widely used storage optimization technique [14]. The prefetching technology can reduce response time for user to search for specific electronic pedigree by making use of electronic pedigrees' correlation and fetching data into cache before they are requested. Therefore, we use the prefetching mechanism to optimize the performance of the EPSS.

When an electronic pedigree is accessed by a user, there is great probability that its correlated electronic pedigrees will be searched for by other users of the same area in the next period. For example, Tom bought a fish at market in Shanghai and he accessed the corresponding electronic pedigree in the EPSS. It means that the fish of the same lot are sold at markets in Shanghai and some consumers who bought other fish of the same lot will probably use the EPSS to search for the fish's electronic pedigrees. What prefetching does is to add electronic pedigrees of these fish of the same lot into cache before consumers request. Therefore, the objects of prefetching are those correlated electronic pedigrees which are in the same merged file as the requested electronic pedigrees.

When a merged file is accessed from HDFS, the prefetching process begins. The internal index file is split from the merged file. Each small XML file's information is read from the internal index file, and it becomes a record in the cache. Then the merged file is stored in the web server which is also HDFS client.

Once an electronic pedigree is uploaded, merged into a big file and stored into HDFS, the electronic pedigree will not be changed. Thus, even though the electronic pedigree is fetched into the cache, it still keeps consistent. Meanwhile, the strategy chosen when small XML files are merged into big ones decides what correlated electronic pedigrees are in the same merged files, and influences the result of prefetching technology.

F. File remerging

Electronic pedigrees have characteristics that the frequency of accessing electronic pedigrees will turn down with time passing by. Especially, after the freshness date of goods ends, nobody will bother to ask about the electronic pedigree which is on behalf of these goods. The merged file in HDFS contains correlated electronic pedigrees, and goods of correlated electronic pedigrees usually have same freshness date and will not be searched for after a same time point. This assumption is common and easily understood. Therefore, after specific time, the merged file of electronic pedigrees can be merged again into bigger files, which we call file remerging.

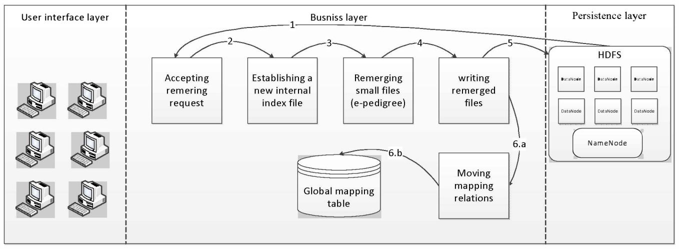


Figure 6. File remerging process in our approach

As is shown in Fig. 6, File remerging process mainly includes accessing merged files, establishing a new internal index file and merging merged files in turn. The detail is described as follows.

Step 1: accessing merged files. Once file remerging has been triggered, the requested merged files are read from HDFS.

Step 2: establishing a new internal index file. Old internal index files are split from the merged files and parsed to access electronic pedigrees' information of these merged files. The new internal index file records the name, the size and the offset of these electronic pedigrees.

Step 3: the new internal index file is written into the remerged file, and then all the small XML files in the old merged files are remerged in turn.

Step 4: storing the new one into HDFS. HDFS client sends a writing request of the remerged file to HDFS. This step is the same as a normal writing process in HDFS.

Step 5: removing the old ones from HDFS. HDFS client sends a deleting request of the merged files which internal small XML files have been remerged to HDFS. This step is the same as a normal deleting process in HDFS.

Step 6: moving the remerged electronic pedigrees' mapping relations. Since the electronic pedigrees have been remerged, global mapping table need to be updated. The records of remerged electronic pedigrees are removed from global mapping table and added to the global remapping table. The global remapping table has the same mapping function, except that there are records for those electronic pedigrees whose corresponding goods have been out of freshness date or nobody cares about it.

Since all the electronic pedigrees stored in HDFS should be available for a long time and we can't remove any electronic pedigree, the file remerging is the optimal approach to reduce the occupation of the NameNode further, in our opinion.

IV. EXPERIMENT AND EVALUATION

A. Experimental environment

The test platform is built on a cluster with 5 nodes which are based on 5 virtual machines. All the virtual machines are established in the same host. The host machine is of HP p7-1035n. The host machine has 4 Intel Core i5-2500S CPU @ 2.70GHz, 4 GB memory and 1 TB disk. The operation system is Windows 7 Enterprise.

The five nodes are utilized with 1 GB memory, 1 CPU and 20GB disk, respectively. The network adaptor is host-only model. The operation system is Ubuntu 11.10. Hadoop version is 1.0.2 and Java Version is 1.6.0. In these five nodes, one node acts as the NameNode and the others act as the DataNodes. The number of replications is set to 2 and HDFS block size is set to 64 MB during the tests.

B. Data set introduction

Our data set consists of 270,000 files, they belong to 27 directories, and each directory includes 10,000 files. The total size of these electronic pedigrees is 4.13GB. File sizes in these data sets range from several KB to less than 128 KB, which is much smaller than the fixed size of a block (64MB typically) in HDFS. And we can find that files between 1KB and 32KB account for 92.59% of total files. Fig. 7 shows the distribution of file sizes.

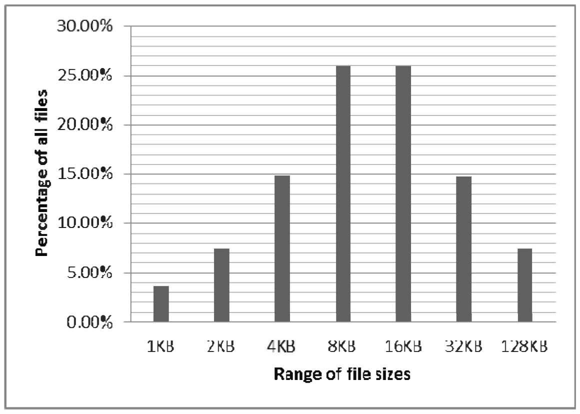
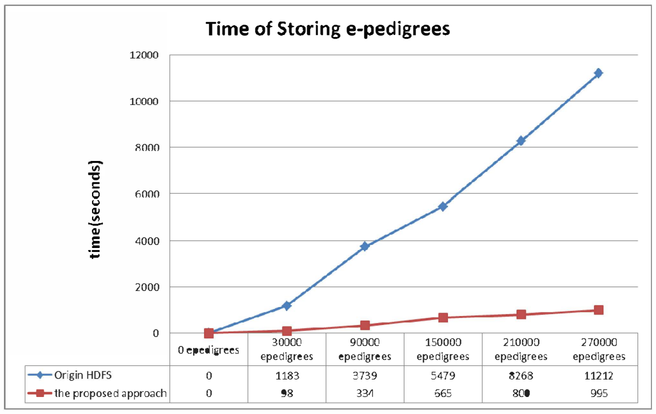


Figure 7. Size distribution of test electronic pedigrees

C. Experiment

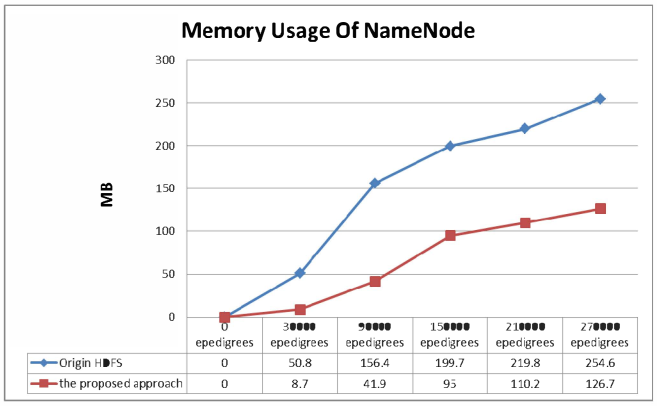
Testing content includes two main processes. The first one is file storing process; the second one is file accessing process.

Experiment of storing massive small files in the EPSS: The proposed approach is compared with origin HDFS for the storing process. For both the original HDFS and the proposed approach, the memory usage of NameNode and DataNodes and storing time are monitored when the system stores 30000, 90000, 150000, 210000 and 270000 electronic pedigrees respectively. The storing time comparison result is shown in Fig. 8. And the memory usage results are shown in Fig. 9 and Fig. 10.



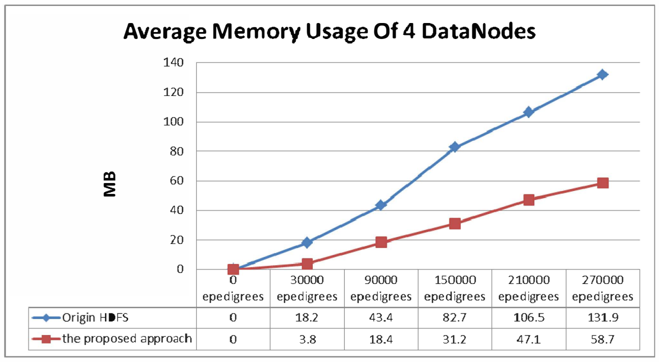
	0 epedigrees	30000 epedigrees	90000 epedigrees	150000 epedigrees	210000 epedigrees	270000 epedigrees
Origin HDFS	0	1183	3739	5479	8268	11212
the proposed approach	0	98	334	665	800	995

Figure 8. Comparison of storing time



	0 epedigrees	30000 epedigrees	90000 epedigrees	150000 epedigrees	210000 epedigrees	270000 epedigrees
Origin HDFS	0	50.8	156.4	199.7	219.8	254.6
the proposed approach	0	8.7	41.9	95	110.2	126.7

Figure 9. Comparison of memory usage of NameNode



	0 epedigrees	30000 epedigrees	90000 epedigrees	150000 epedigrees	210000 epedigrees	270000 epedigrees
Origin HDFS	0	18.2	43.4	82.7	106.5	131.9
the proposed approach	0	3.8	18.4	31.2	47.1	58.7

Figure 10. Comparison of average memory usage of 4 DataNodes

According to the experiment, the proposed approach has much better performance than origin HDFS in both storing electronic pedigrees and memory usage of the NameNode and the DataNodes. The efficiency of storing massive small files has increased and storing time is reduced by an average of 91%. Memory usage of the NameNode is reduced by an average of 50.1% and memory usage of the DataNodes is reduced by an average of 55.1%.

Experiment of accessing small files in the EPSS: The proposed approach is compared with origin HDFS for the accessing process. While preparing the experiment, three electronic pedigree access lists are established in order to simulate the proposed approach's performance in different access modes. In the first mode, 200 attributes corresponding to 200 distinct random electronic pedigrees are listed; in the second mode, 200 attributes corresponding to 200 electronic pedigrees are listed in which every 20 electronic pedigrees are correlated electronic pedigrees; in the third mode, 100 correlated electronic pedigrees and another 100 correlated electronic pedigrees are listed. For both the original HDFS and the proposed approach, reading time is monitored when the system reads these three electronic pedigree access lists respectively. The reading time comparison result is shown in Figure. 11. Access time of three lists are reduced by 88%, 92% and 99% respectively mainly due to file mapping and prefetching mechanism.

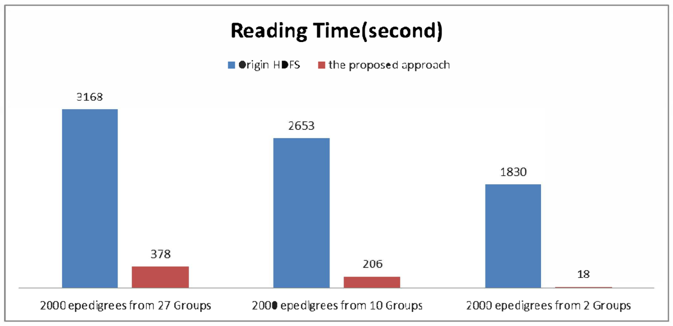


Figure 11. comparison of reading time

V. RELATED WORK

The small file problem of HDFS is concerned by both academia and industry. Preexisting researches on massive small file storage can be classified into two categories: general solutions and targeted solutions to meet particular systems. The former includes Harball [10], SequenceFile [8] and MapFile [9]. The later includes HDWebGIS [11] and BlueSky [12].

Hadoop Archives, which are also called Harballs, introduce another file layer on top of the HDFS. The technology creates a metadata entry for the index of the files it contains. This index can be considered as a meta-meta data layer for the data in the archives. SequenceFile provides a key-value pairs model for storing data. It uses the file name as the key and the file content as the value. SequenceFile allows compression too, unlike Hadoop Archives. MapFile is one kind technology based on the SequenceFile. It contains an index to help lookups by key easily.

HDWebGIS is an approach proposed to improve the I/O performance of massive small data of Geographic Information System storing on HDFS. It includes merging small files into a larger file and building a grouping schema specific to the WebGIS application. BlueSky is an approach proposed to optimize the efficiency of storing and accessing PPT files on HDFS. It includes merging correlated files into a larger file and a two-level prefetching system.

Nevertheless, all three general solutions try to solve small file problem of HDFS in the scenario that massive small files have been stored in HDFS, and they cannot provide an approach to merge small files into a big one dynamically. Moreover, targeted solutions are special and do not meet the requirements of the electronic pedigree system.

Compared with the above solutions, our approach in this paper differs in the following aspects.

1). The proposed approach is an electronic pedigree solution on HDFS, and the thought of it is also helpful for the files of XML type.

2). Dynamic grouping is designed in the proposed approach. It can avoid manual intervention and help gathering correlated files.

3). When merging files, file correlation and attribute serialization are involved. And four strategies can be chosen to meet different requirements of user searching for electronic pedigrees.

4). Global indexing tables and global mapping table are utilized to establish connection records among attributes, electronic pedigrees and merged files.

5). Prefetching technology and cache are designed to fetch electronic pedigrees into cache before they are requested to reduce accessing time.

6). For reducing the pressure of the NameNode, file remerging technology is added to our approach.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present a new approach to optimize the efficiency of storing and accessing small XML-formatted electronic pedigrees. Considering the characteristics of electronic pedigree, the proposed approach designs a global mapping table, a global indexing table, a file merging mechanism, a prefetching mechanism and a file remerging mechanism to optimize small file I/O performance of HDFS. The experiment results show that the proposed approach can efficiently reduce the time cost of storing and accessing small files and improve the memory usage performance of the NameNode and DataNodes.

Our future work includes three aspects: first, we will compare more optimization solutions in HDFS with our proposed method; second, we will apply our proposed method to more application scenarios, such as the storage of office files; third, electronic pedigrees usually contain many sensitive data, which could arouse a privacy problem when we deploy a central EPSS to store so many electronic pedigrees. Thus, we will investigate how to use other signature technologies and cryptography technologies to express the electronic pedigrees and support storing and retrieval operations.

ACKNOWLEDGMENT

This paper is supported by the 863 project (Grant NO: 2011AA100701), the project of Natural Science Foundation of Shanghai (Grant NO: 12ZR1402600), and the Opening Project of DNSLAB, China Internet Network Information Center, China. The corresponding author is Weili Han.

REFERENCES

[1] Hadoop official site, http://hadoop.apache.org/, 2012.

[2] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System", Mass Storage Systems and Technologies (MSST), 2010, pp. 1 - 10.

[3] J.Dean, S.Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Proc. of the 6th Symposium on Operating Systems Design and Implementation, San Francisco CA, Dec. 2004, Vol. 51.

[4] HDFS official wiki, http://hadoop.apache.org/hdfs/.

[5] T. White, Hadoop: The Definitive Guide. O'Reilly Media, Inc. 2009.

[6] W. Tantisiriroj, S. Patil, and G. Gibson, "Data-intensive file systems for internet services" Tech. Report CMU-PDL-08-114, Oct. 2008.

[7] Hadoop official site, http://wiki.apache.org/hadoop/ PoweredBy.

[8] Sequence File official wiki, http://wiki.apache.org/hadoop/ SequenceFile.

[9] Map file official javadoc, http://hadoop.apache.org/common/docs/ current/api/org/apache/hadoop/io/MapFile.html.

[10] Hadoop archive official javadoc, http://hadoop.apache.org/common/ docs/current/hadoop_archives.html.

[11] X. Liu, J. Han, Y. Zhong, Chengde Han, and Xubin He, "Implementing WebGIS on Hadoop: A Case Study of Improving Small File I/O Performance on HDFS," Proc. Of the 2009 IEEE Conf. on Cluster Computing, 2009, pp. 1-8.

[12] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, Y. Li, "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files", In: Proceedings on IEEE International Conference on Services Computing (SCC), 2010, pp. 65-72.

[13] T. White, The Small Files Problem, http://www.cloudera.com/ blog/2009/02/02/the-small-files-problem/.

[14] EPCglobal, Pedigree Ratified Standard, 2007.

[15] C. C. Tan, and Q. Li, "A Robust and Secure RFID-Based Pedigree System", Lecture Notes in Computer Science (Information and Communications Security), 2006, pp. 41-49.

[16] M. Harrison, and T.Inaba, "Improving the safety and security of the pharmaceutical supply chain", 2006.

[17] Wiki, http://en.wikipedia.org/wiki/Pedigree

[18] L. Atzori, A. Iera, G. Morabito. "The Internet of Things: a survey". Computer Networks, 2010.

[19] EPCglobal. Pedigree Ratified Standard, 2007, http://www.gs1.org/gsmp/kc/epcglobal/pedigree/pedigree_1_0-standard-20070105.pdf. Accessed April 2012.

[20] GS1. EPCIS - EPC Information Services Standard, 2011, http://www.gs1.org/gsmp/kc/epcglobal/epcis. Access April 2012.

[21] Y. Gu, T. Jing. "The IOT Research in Supply Chain Management of Fresh Agricultural Products." In Proceedings of the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 7382 – 7385, 2011.

[22] M. Harrison, T. Inaba. Improving the safety and security of the pharmaceutical supply chain. Networked RFID Systems and Lightweight Cryptography, III, 223-246, 2008.

[23] J. Muckstadt, D. Murray, J. Rappold, D. Collins. "Guidelines for collabo-rative supply chain system design and operation." Information Systems Frontiers, 4: 427–453, 2001.

[24] L. Zheng, H. Zhang, W. Han, X. Zhou, J. He, Z. Zhang, Y. Gu, J. Wang. "Technologies, applications, and governance in the Internet of Things". In: Internet of Things - Global Technological and Societal Trends. From Smart En-vironments and Spaces to Green ICT, River Publishers, 2011.

[25] W. Han, Y. Gu, W. Wang, Y. Zhang, Y. Yin , J. Wang, L. Zheng. "The Design of an Electronic Pedigree System for Food Safety", Information Systems Frontiers, 2012.