Optimizing the Storage of Massive Electronic Pedigrees in HDFS

Authors: Yin Zhang, Weili Han, Wei Wang, Chang Lei
Presented by: Yin Zhang
Oct 25th 2012

Electronic pedigree system

>>trustworthily tracking of the processes

>>Small-sized but huge volume of electronic pedigrees

Optimizating the storing and accessing of massive small XML files in HDFS

Abstract

- ✓reduce the metadata occupation at NameNode
- ✓improve the efficiency of accessing small XML files

- ✓Feasibility,
- ✓Effectiveness,
- ✓Efficiency.

Abstract

Reduce memory consumption of NameNodes by 50%

Improve performance of storing by 91%

Accelerate accessing by 88% in Hadoop

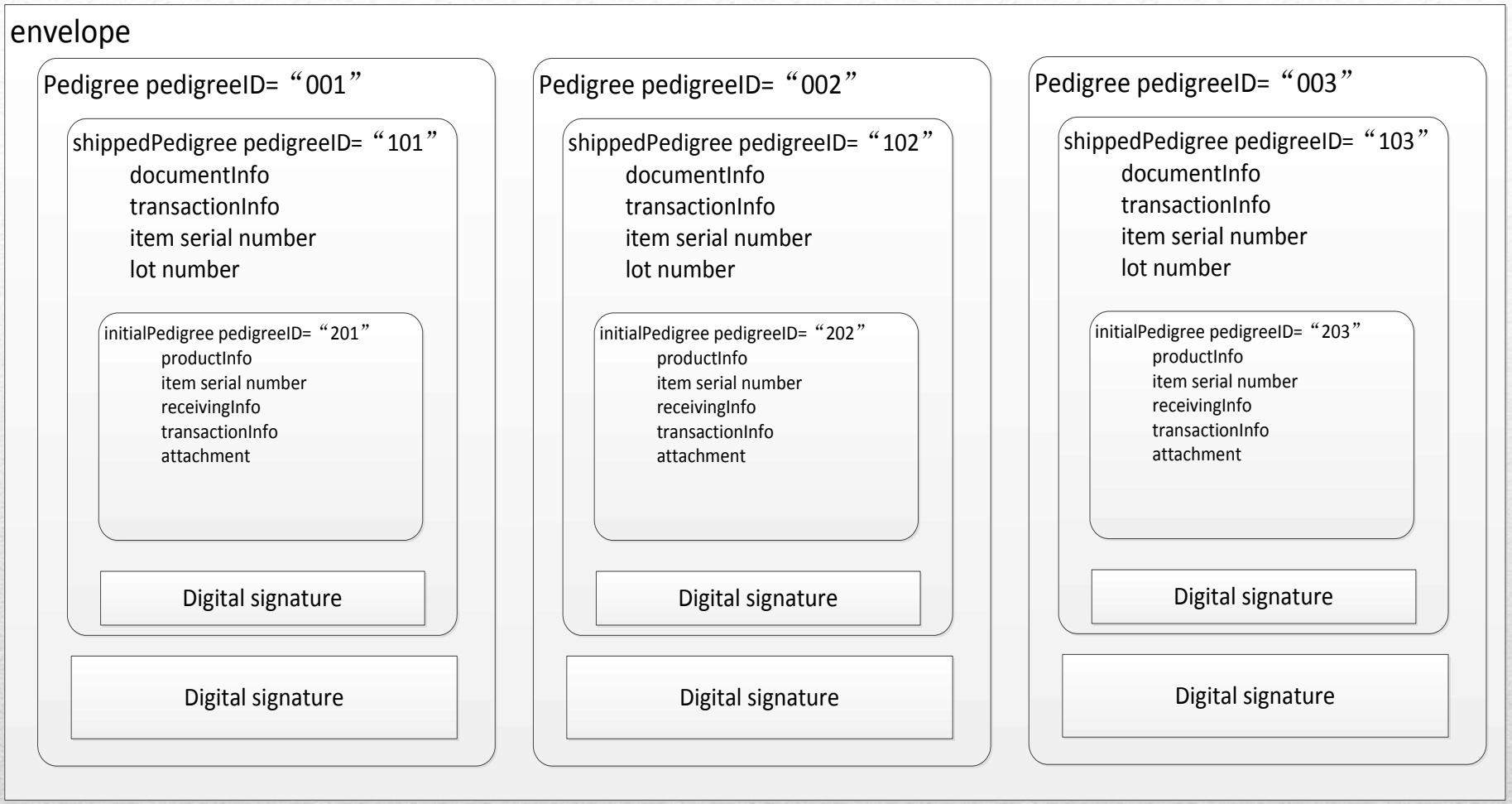
What is a pedigree?

Electronic pedigree

Background: electronic pedigree

- ✓Of a nested architecture
- ✓Tens of KB to hundreds of KB
- ✓Different types
- ✓Attribute ‘Lot number’
- ✓Attribute ‘Item serial number’
- ✓Attribute ‘pedigreeID’

Background: electronic pedigree



A simple sample

- ✓serialized attributes
- ✓One single-writer, multiple-reader Model
- ✓Correlated electronic pedigrees
- ✓freshness date of goods

Characteristics of electronic pedigree

Electronic Pedigree Storage Server

Manage massive electronic pedigrees

Access electronic pedigrees

Receive electronic pedigrees in two ways

Background: EPSS

Hadoop Distributed File System

- open-source software framework
- One single-writer, multiple-reader model

Performance bottlenecks for massive small files

Background: HDFS

① High cost for metadata management
② High memory cost for files
③ High time cost for contacting

Optimizing the Storage of Massive Electronic Pedigrees in HDFS

Disadvantages of HDFS in EPSS

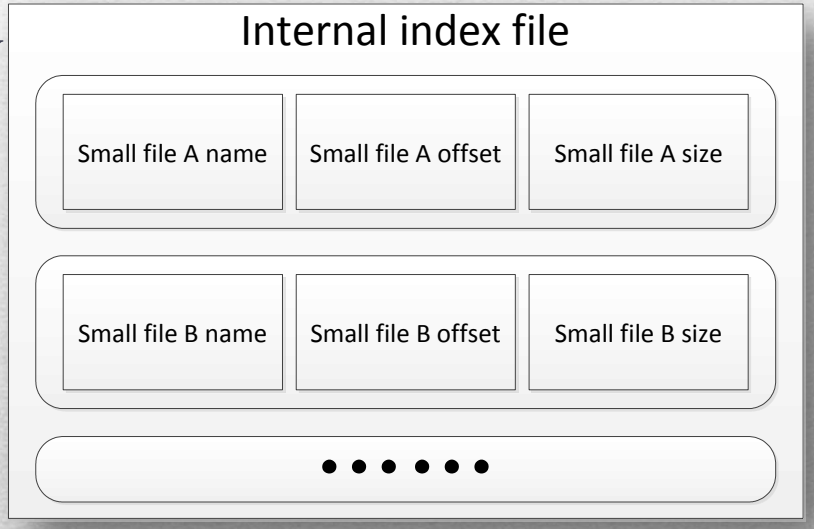
Merge correlated electronic pedigrees into a bigger file

>>decrease file size in HDFS

>>relieve NameNode's memory

Four strategies to merge

Internal index file



File merging

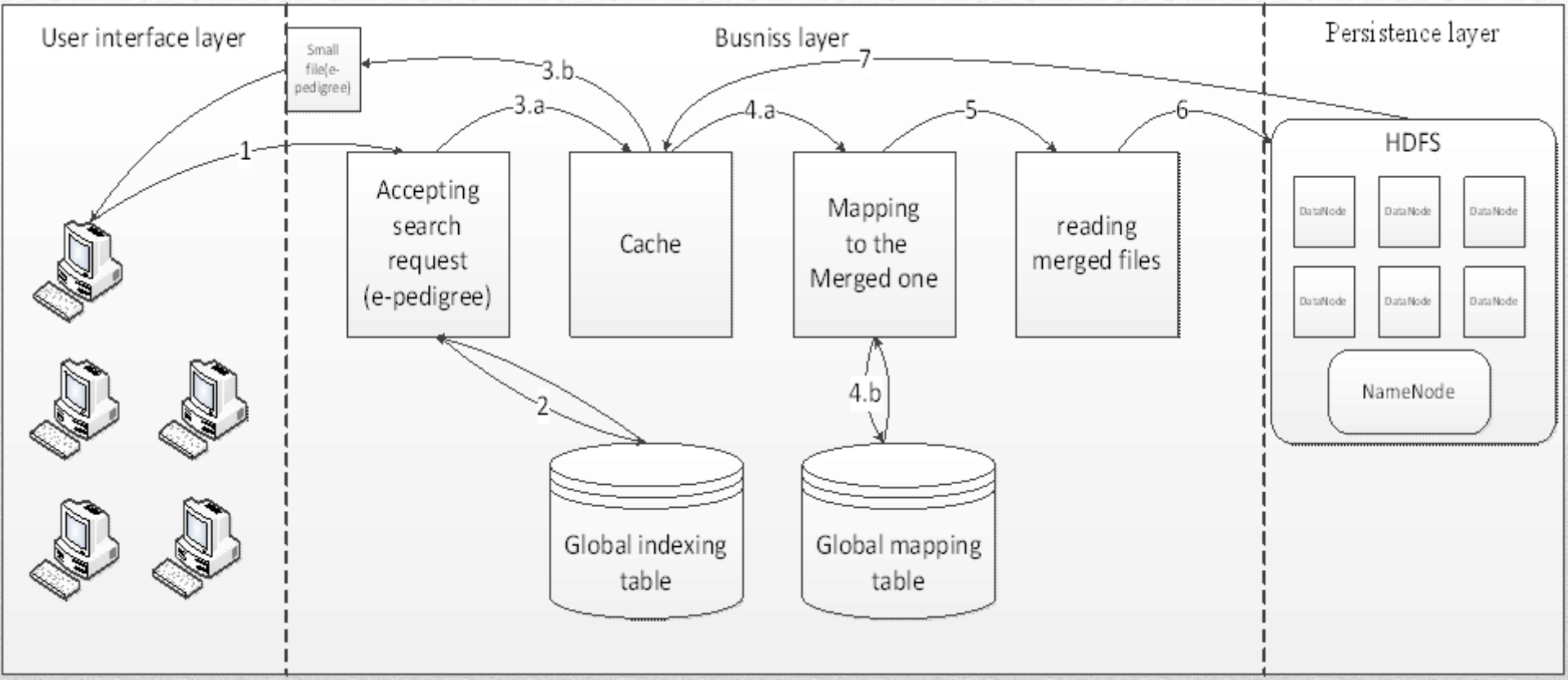
A global mapping table >> records between each small XML file and its merged file

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

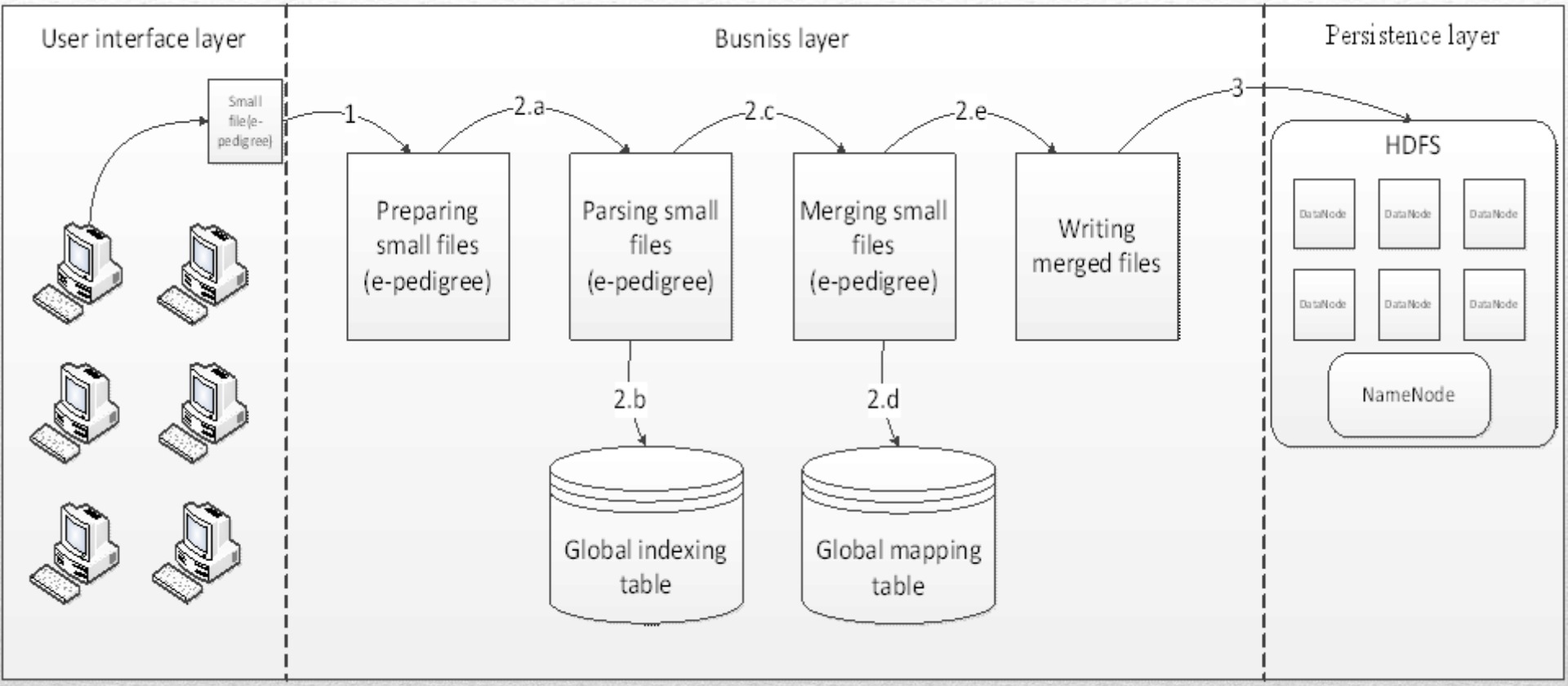
An indexing table >> records between attributes and small XML files

e.g. “pedigreeID=001”→“pedigree001”→“bigfile1”

File mapping



File reading

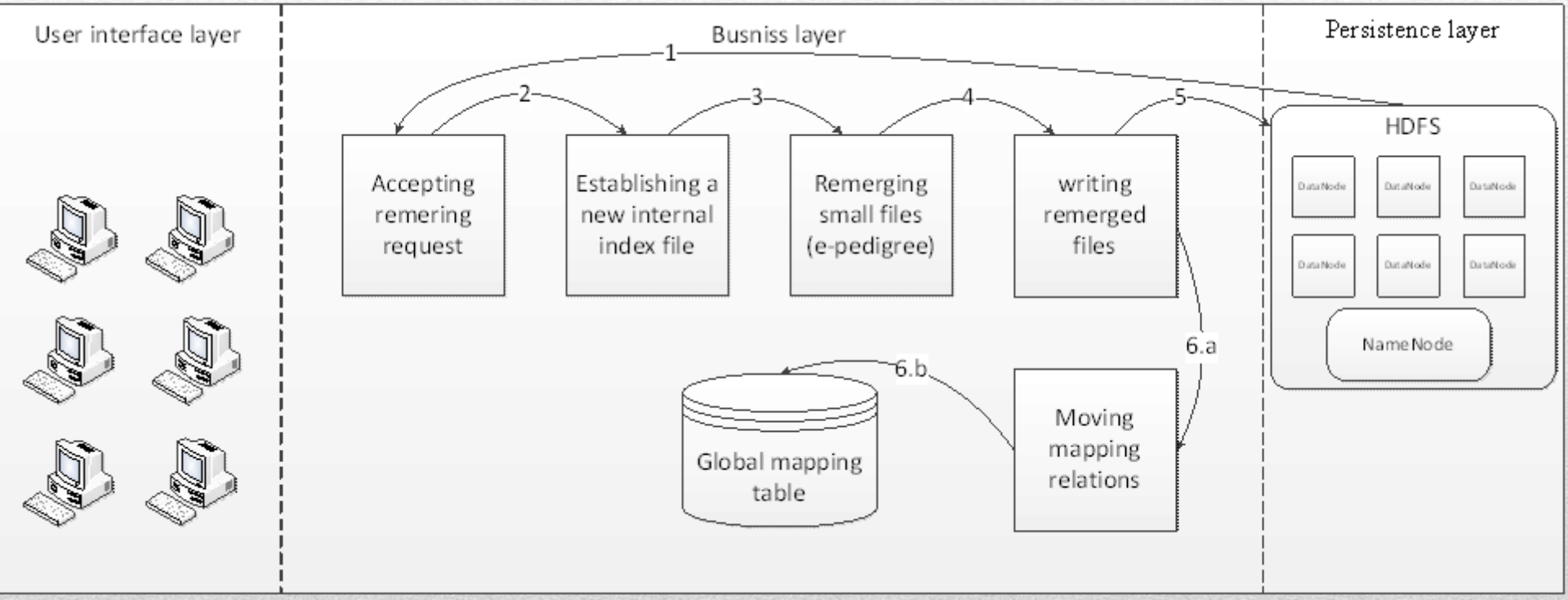


File writing

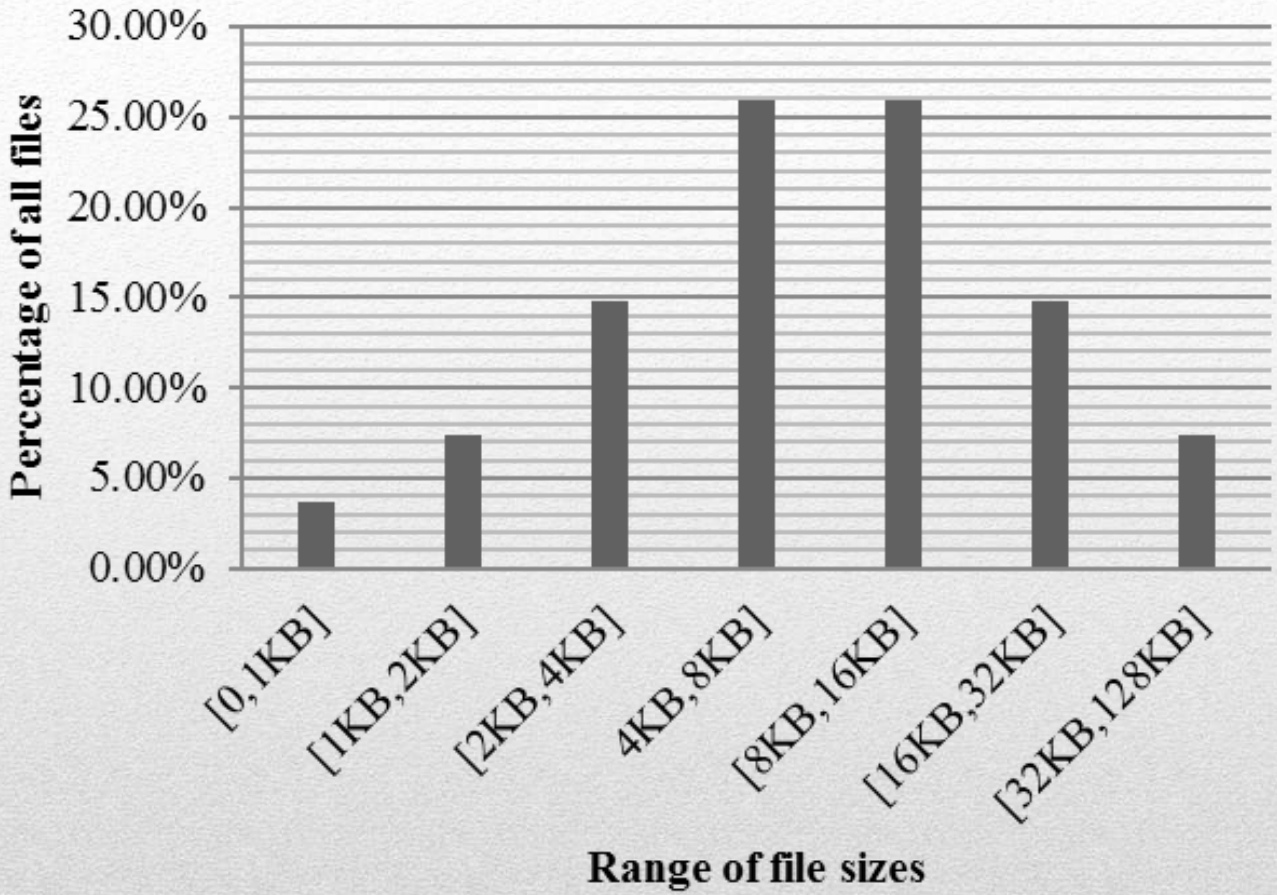
- ✓Reduce response time for user
- ✓Prefetch XML files in the same merged file
- ✓Consistent between cache and HDFS
- ✓Influenced by merging strategy

Prefetching

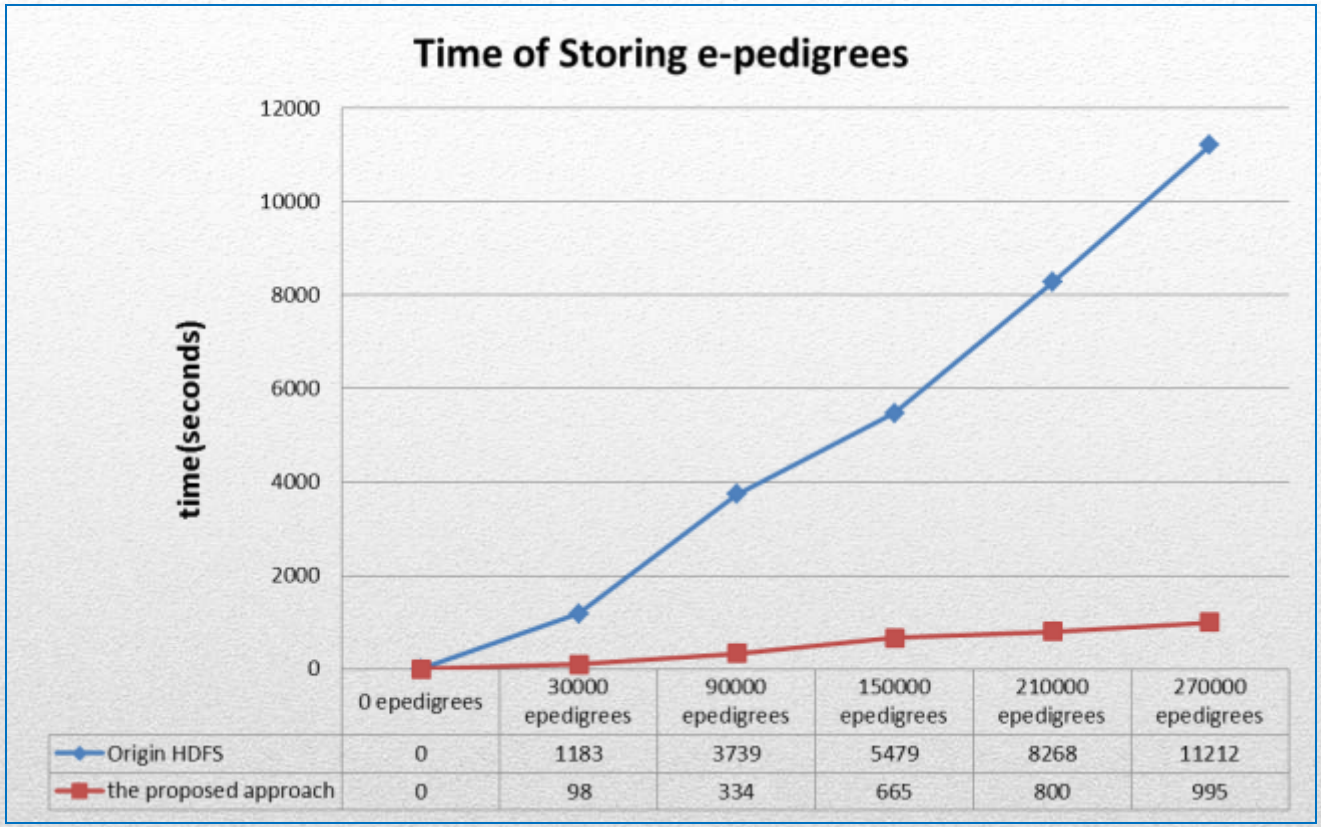
Frequency of accessing electronic pedigrees will turn down with time passing by, especially after the freshness date of goods ends



File remerging

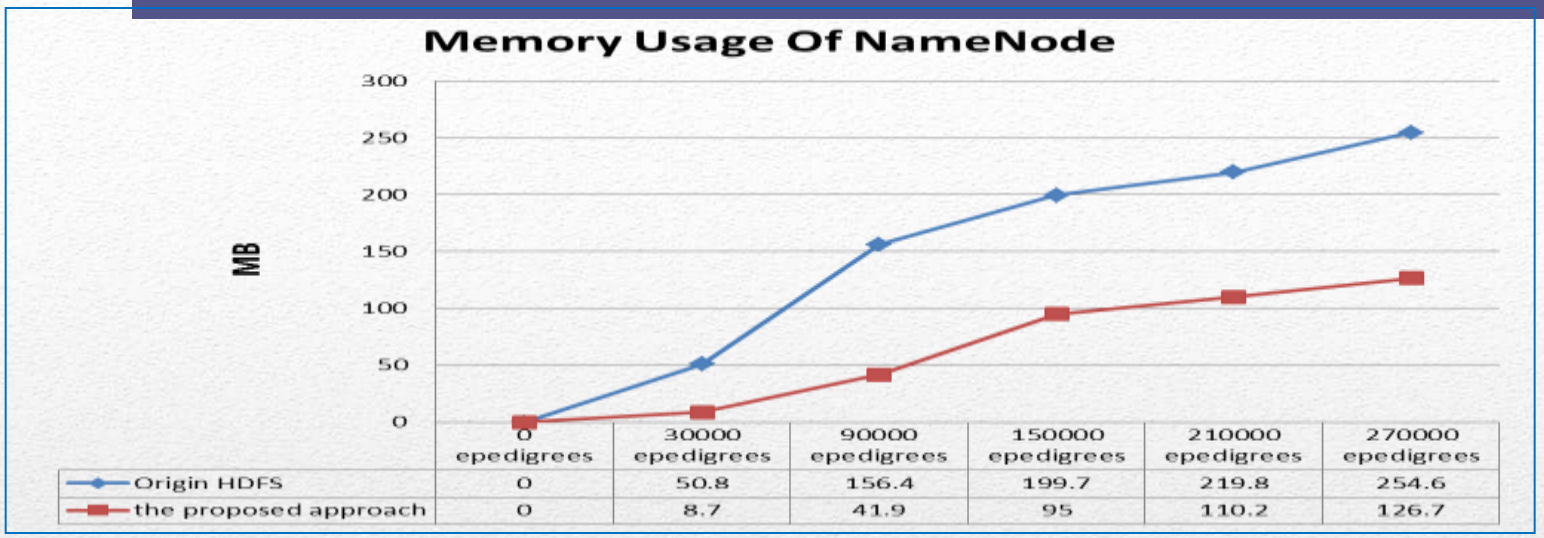

30.00%
25.00%
20.00%
15.00%
10.00%
5.00%
0.00%
Percentage of all files
[0,1KB]
[1KB,2KB]
[2KB,4KB]
4KB,8KB]
[8KB,16KB]
[16KB,32KB]
[32KB,128KB]
Range of file sizes

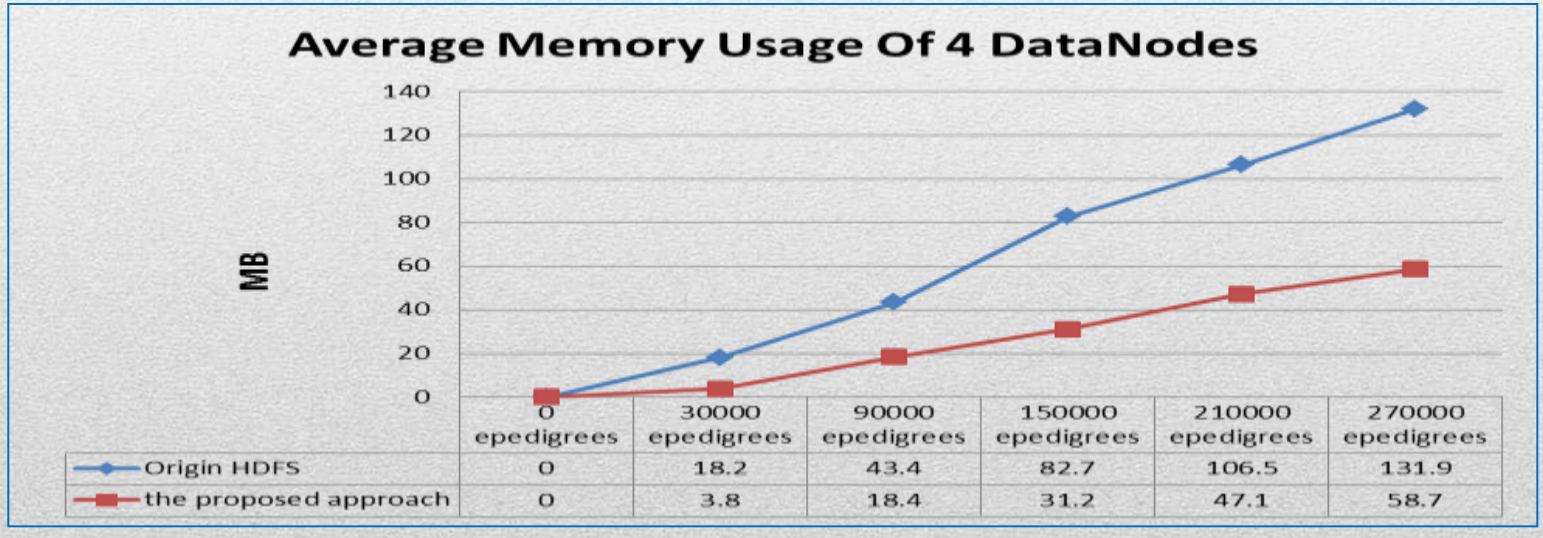
Experiment



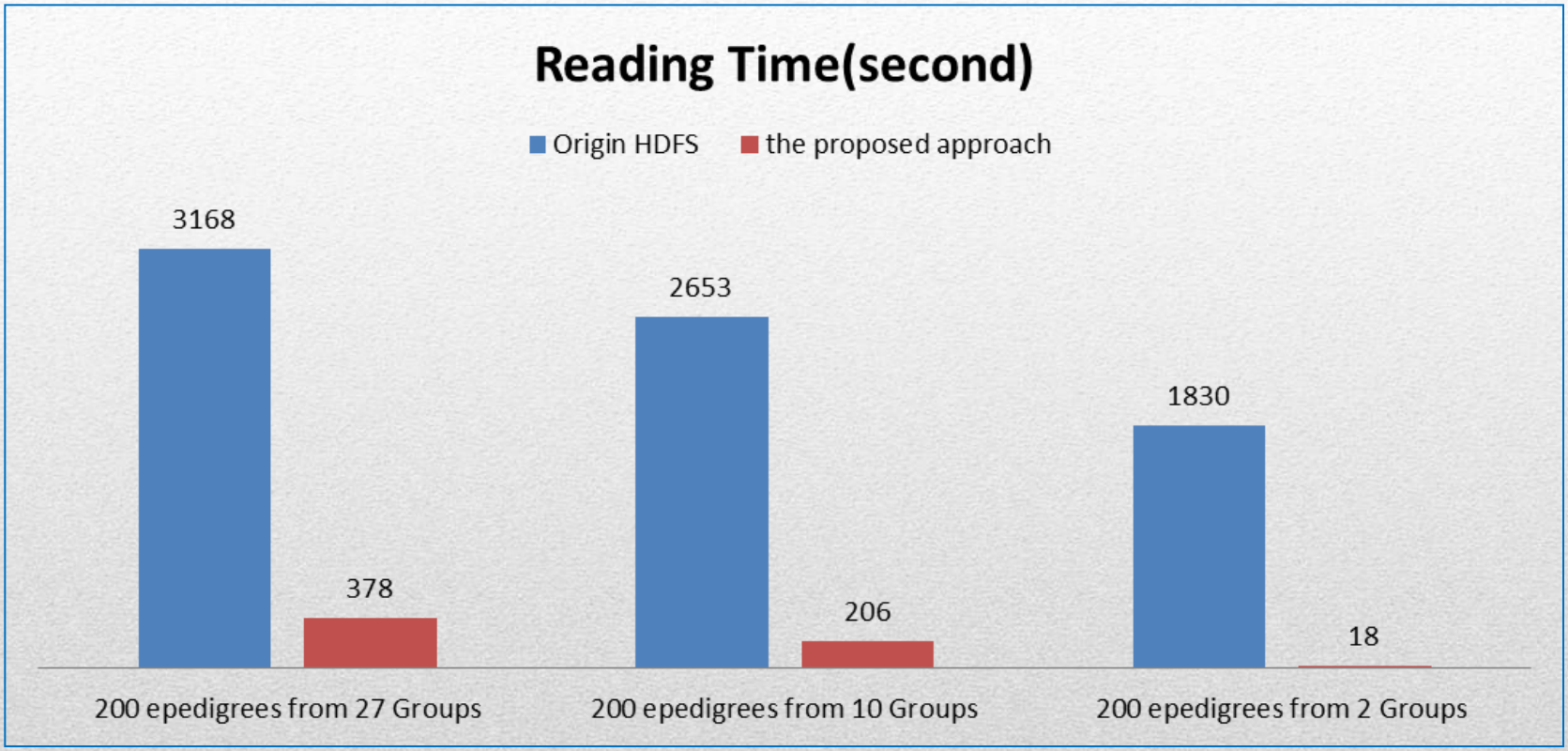
	0 epedigrees	30000 epedigrees	90000 epedigrees	150000 epedigrees	210000 epedigrees	270000 epedigrees
Origin HDFS	0	1183	3739	5479	8268	11212
the proposed approach	0	98	334	665	800	995

Evaluation


Memory Usage Of NameNode
MB
300
250
200
150
100
50
0
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


Average Memory Usage Of 4 DataNodes
MB
140
120
100
80
60
40
20
0
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

Evaluation


Reading Time(second)
Origin HDFS
the proposed approach
3168
378
2653
206
1830
18
200 epedigrees from 27 Groups
200 epedigrees from 10 Groups
200 epedigrees from 2 Groups

Evaluation

① Helpful for the files of XML type
② Dynamic grouping
③ File correlation and attribute serialization
④ Global indexing table and global mapping table
⑤ Prefetching technology
⑥ File remerging technology

Difference from related work

- ✓ Optimizing the Storage of Massive Electronic Pedigrees in HDFS
- ✓ Avoiding privacy problem of storing electronic pedigrees

Conclusion and future work

Q&A