ISSN 2277-3061

A Seismic Data Processing System based on Fast Distributed File System

Jun Li[1], Changsen Pan[2], Menghan Lu[3]
[1,2,3]Department of Automation,CAS Key Laboratory of Technology in Geo-spatial Information Processing and Application System, University of Science and Technology of China, Hefei, China
[1]Ljun@ustc.edu.cn
[2]chspan@mail.ustc.edu.cn
[3]menghan@mail.ustc.edu.cn

ABSTRACT

Big data has attracted an increasingly number of attentions with the advent of the cloud era, and in the field of seismic exploration, the amount of data created by seismic exploration has also experienced an incredible growth in order to satisfy the social needs. In this case, it is necessary to build a highly-effective system of data storage and process. In our paper, we aim at the properties of the seismic data and the requirement to the performance of IO, and establish a distributed file system with the goal of processing seismic data based on the Fast Distributed File System (Fast DFS), then test our system through a series of operations such as file write and read, and the results show that our file system is very proper and effective when processing seismic data.

Keywords

Big Data, Seismic Exploration, SEGY File Format, Fast Distributed File System

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol.14, No.5
www.ijctonline.com, editorijctonline@gmail.com

1. INTRODUCTION

When coming into the year of 2013, the term big data[1] has been mentioned much more times than ever before following after terms the Internet of Things, cloud computing[2]. People use this term to describe and define the large amount of data created during this era of information explosion, and name some related technological development and innovation. The earliest quote of term big data can date back to the open project Nutch of the Apache Org[3]. At that time, big data[4] is described as the large amount of data which is used to update the network searching engine, and batch processing and analyzing at the same time. With the release of Google's MapReduce[6] and Google File System (GFS)[5], big data can be not only used to describe the large amount of data, but also used to cover the rate of data processing. Nowadays in this period of huge amount of data, the range of the big data's application has also become increasingly wider, covering fields such as IT, astronomy, geology, biology, military, medical, E-commerce. Take IT as an example, there are about 6 billion searching requests per day to handle in Baidu, reaching dozens of petabyte; there are tens of millions of transactions per day in Taobao, exceeding more than 20 terabyte per day; the records of surfing the Internet for users in China Unicom has also reached 10 terabyte per day. With the development of information society, the significance of big data has also become increasingly prominent[2].

In our paper, we focus on processing seismic data. Seismic exploration is an important method to survey oil, natural gas and solid resources before drilling, and is also widely used in other aspects such as exploration of coal field and geology engineering, region geology research and crustal study[7]. As the increasingly demand for oil and gas in modern society, the technology of seismic exploration also develops faster than ever before , thus resulting in huge amount of data. In this case, how to store and process these data has become a headache in many related areas.

Besides, distributed file systems appear as early as 1970s, and gain a considerable progress in recent years[8]. There are lots of mature file systems such as Google File System (GFS), Hadoop DFS (HDFS)[10], FastDFS[11] ,Lustre and so on. In our work, we carry out our research on Fast DFS, and create a distributed file system aiming at processing seismic data in order to achieve the goal of highly-effective storage and big data read performance.

The remainder of this paper is organized as follows: Section 2 describes the format of seismic data and our changes according to this kind of format; Section 3 details the architecture of Fast DFS and its working theory, and then we add our work into this file system to create a new file system in order to satisfy the needs of seismic data; Section 4 tests our new file system with a series of operations and compares the performance with other file systems; Section 5 concludes our work.

2. INTRODUCTION TO SEISMIC DATA

2.1 Features and requirements of seismic data

Seismic exploration exploits the difference between elasticity and density of subsurface medium, through observation and analysis of the earth response to seismic waves stimulated artificially, and then speculates the nature and forms of underground rock[7]. Usually stimulating seismic waves by shots is the first step, when transferring into underground and encountering the interface of rocks with different medium nature, the seismic waves will be reflected and refracted and finally received by detectors. The received signals of seismic waves have to do with source characteristics, the location of detection points and the nature and structure of underground rocks where seismic waves pass through, and the nature and forms of underground rocks can be speculated by processing and analyzing these seismic waves. It can be referred by the theory of seismic exploration that the data collected can be small in block but large in quantity, that is, the size of data collected by every shot and every detector can be very small, and also exists the feature of write once and read many, but the quantity of data that is read each time can be very large, in that case, this particular feature should be taken into account.

Besides, the data collected should satisfy the needs of different kind of customers, thus resulting in the different kind of data forms that are obtained each time. From the theory we can see that users can obtain data according to the number of shots and detectors, such as CSG(common shot gather), CMG(common mid-point gather), CRG(common receiver gather) and so on, Figure 1 shows some of the common ways to obtain data from multiple receivers.

ISSN 2277-3061

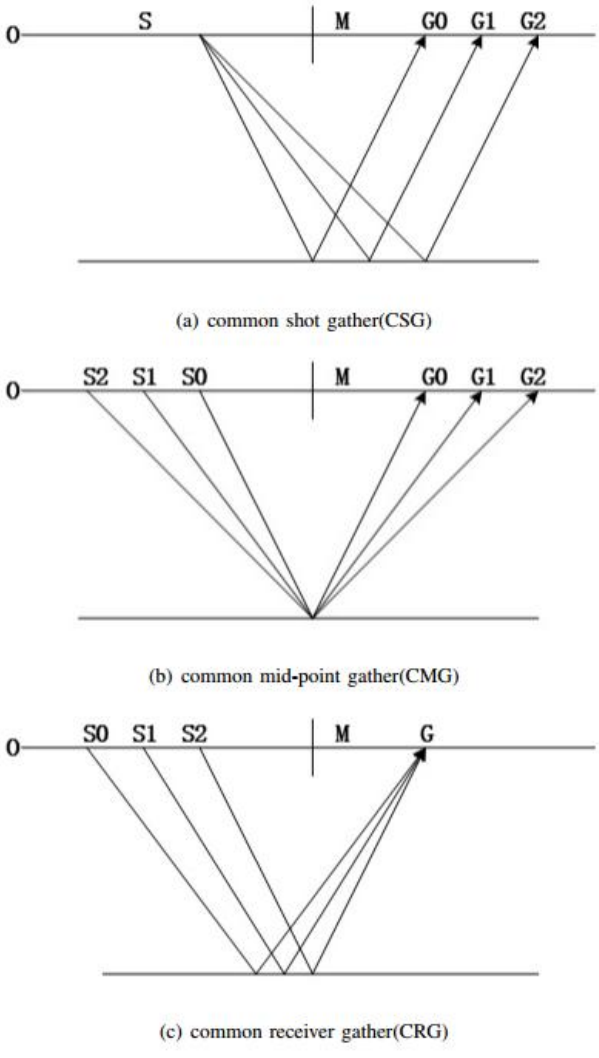

Fig.1. Some common ways to obtain data from receivers

In the paper, since seismic data can have some specific features and the requirements of many users, we should take these elements into account when establishing our file system.

2.2 Seismic data format SEG-Y

SEG-Y format[9] is the most important data format in seismic exploration, so it is helpful for us to process seismic data to learn something about SEG-Y format. The figure below shows the construction of SEG-Y file format.

Fig.2. Some common ways to obtain data from receivers

From the figure 2 we can see that there are three components in standard SEG-Y format: the first one is EBCDIC file header, 3200 bytes which consists of 40 cards (80 characters each row and 40 rows), it is used to store some description information of seismic data; the second one is binary file header, 400 bytes, it is used to store some key information of the SEG-Y file, including SEG-Y data format, sampling points, sampling interval, measurement unit and so on, these information are generally fixed in some location in binary file header; the third one is the actual seismic trace, each of which consist of 240 bytes of trace header information and data. Trace header data are generally included shot number, trace number, sampling points, coordinates of seismic trace, but some key parameters' locations (locations of shot number and trace number in trace header) are not fixed.

ISSN 2277-3061

3. DISTRIBUTED FILE SYSTEM Fast DFS

3.1 A brief introduction to Fast DFS

Fast DFS[11] is a open-source distributed file system which is similar to GFS, it is accomplished by pure C language, supporting many UNIX operation systems such as Linux, FreeBSD, AIX and so on, it can only be accessed by proprietary API. Fast DFS is a distributed file system that is suited to Internet, which can manage files including file storage, file synchronization, file access (file upload, file download), and take many mechanism into account such as redundancy, load balance and linear expansion. Besides, it also focuses on highly-available, highly-efficiency and solve the problems such as high capacity and load balance.

3.2 Fast DFS architecture

Compared to existing popular distributed file systems such as GFS and HDFS, Fast DFS has some unique features in architecture and designation, which is reflected in lightweight, grouping and peer-to-peer structure. The figure below shows the architecture of Fast DFS[13]

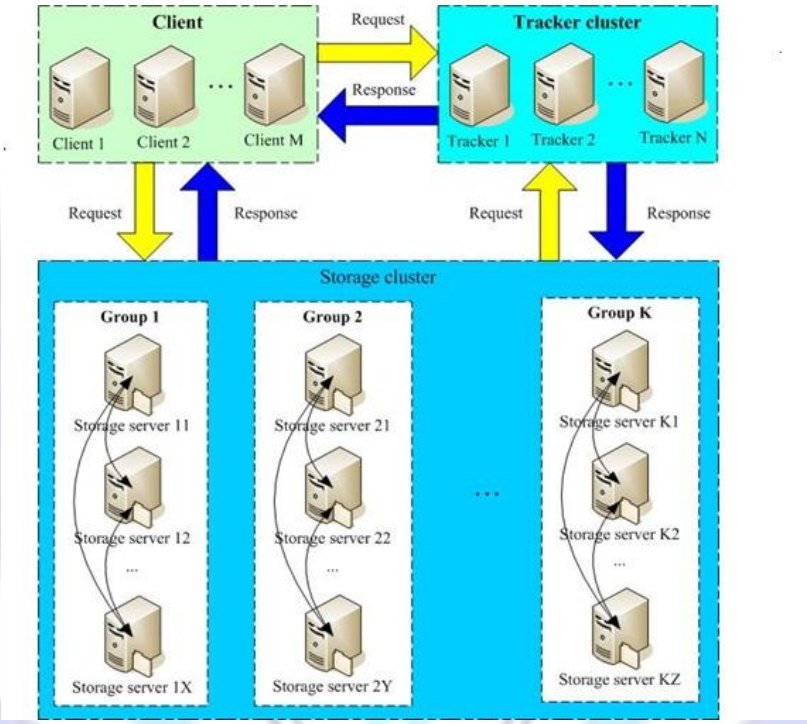

Fig.3. Fast DFS Architecture

From the figure 3 we can see the features of Fast DFS architecture. 1 lightweight: there are only two roles in the whole Fast DFS----Tracker Server and Storage Server. Tracker Server is a central node, it is responsible for load balance and schedule; Storage Server uses OS's file system to storage files. When a client uploads a file, the file's ID is not assigned by the client but created by Storage Server and then returned to the client. There are group names, file's relative path and filename in the file ID, Storage Server can locate the file according to the file ID. In that case, it is not necessary for Fast DFS to store the file index information. 2 grouping: Fast DFS uses packet storage pattern, there are one or more groups in one cluster, and one or more storage servers in one group, and the relationship between many storage servers in the same group is redundant by each other, that is, the files of the same storage servers are all the same. File upload, download and delete can be operated on any one of the storage servers, which is very flexible and of high controllability. For example, when uploading files, client can directly assign which groups to upload. When there are much pressure on storage servers of one group, new storage servers can be added into this group to increase the ability to serve (vertical expansion); when there are not many capacities in the system, new groups can be added into the system to increase the ability to serve (horizontal expansion). 3 peer-to-peer structure: there can be many Tracker Servers in Fast DFS, and there doesn't exist single point failure. The relationship between tracker servers and storage servers in one group are all equal. However, in traditional Master-Slave structure, Master is the single point and it can only support write operation. If Master doesn't work, then one of the Slave should become Master, in that case, the logic that should be accomplished will become very complicated. On the other hand, compared with Master-Slave structure, in peer-to-peer structure, all the nodes' status are the same, every node can be Master, so single point of failure[10][14] will not exist.

3.3 Fast DFS working principle

When Fast DFS is working, generally clients and Storage Server have a connection to Tracker Server, Storage Server reports to the Tracker Server of its state information, including remaining disk space, file synchronization situation and the number of file upload and download[13]. Storage servers in different groups will not communicate with each other while

storage servers in the same group will connect with each other and synchronize files at the same time. However, when a new storage server is added, one of these storage servers will synchronize all the existing data (including source data and backup data) to this new server.

4. DESIGN OF SEISMIC DATA FILE OPERATION SYSTEM BASED ON FAST DFS

4.1 Optimization of seismic data format

We have already learned about that the format of seismic data is SEG-Y from section 2, that is, header data of 3600 bytes added with the following data body and there also exists 240 bytes of header data in every data body structure. Header data defines some related information about seismic exploration, since this paper only focuses on the storage of seismic data and its highly-efficient performance, so we ignore most of the information stored in SEG-Y format and only take into account some related information such as shot number, begin detector number and final detector number. Since these information stand for some location information of the stored data, and have a close connection with the following operations (download parameters, index), so it is very appropriate to only consider these as key information; on the other hand, owing to the insignificance of other information, we only set these values to 0 in our work.

In SEG-Y format, data are generally stored together with header data, clients have to read this unrelated header data when get useful data, it is very time-consuming when there exists too many seismic traces, thus resulting in low-efficiency. So we do some optimization based on SEG-Y format, and separate header data and data body, then add index information between them. In that case, we can read data according to the index information and completely avoid reading header data, and the efficiency can be greatly improved.

We test our optimized data format, and take the simplest array structure as index, and compared with the original SEG-Y format. In the experiment, we assume that all the data are one-dimensional (in a straight line), 1000 shots multiple 2000 traces' data, 3600 bytes of header data, 240 bytes of trace header and 30KB in each trace. We get the results as follows:

Table 1. Time results between original and optimized data format

Common-shot data extraction tests			Common-detector data extraction tests		
Coordinates	SEG-Y format(s)	Optimized format(s)	Coordinates	SEG-Y format(s)	Optimized format(s)
（0,0）	0.224142	0.187487	（8,0）	11.62846	10.854803
（10,0）	0.238612	0.197799	（200,0）	11.60726	10.806037
（100,0）	0.215936	0.18243	（900,0）	11.70377	10.671067
（500,0）	0.277212	0.197988	（1200,0）	11.68548	10.64334
（998,0）	0.203376	0.185533	（2000,0）	11.60835	10.717666

From the charts we can see that the time consumed in the optimized data format is relatively shorter. In that case, the results can be more prominent when the amount of data is larger than before.

4.2 Distributed file system focuses on seismic data

1) Advantages of Fast DFS on seismic data process

Fast DFS is a dedicated file system that has the advantages of lightweight, supporting high-concurrent access, load balance and scalable, while other file systems such as GFS, HDFS, Luster are all universal Distributed file systems. The highlight of Fast DFS is that it can support small files' storage with very high efficiency. As we claimed before, one of the seismic data's features is that it is small in size but large in quantity[12], that is, the size of data can be very small (only dozens of KB in each shot and each trace), but the amount of data will be read each time (maybe hundreds of MB or several GB). This feature can be very well suited to the advantages of Fast DFS, so it can achieve very high efficiency when designed by Fast DFS. Meanwhile, the code of open-source Fast DFS is written by C language, which is high-efficient and easy to revise.

2) Architecture of seismic data file system based on Fast DFS

Inspired by the architecture of Fast DFS, the architecture we have adopted is similar to it. The figure below is the basic cluster architecture.

ISSN 2277-3061

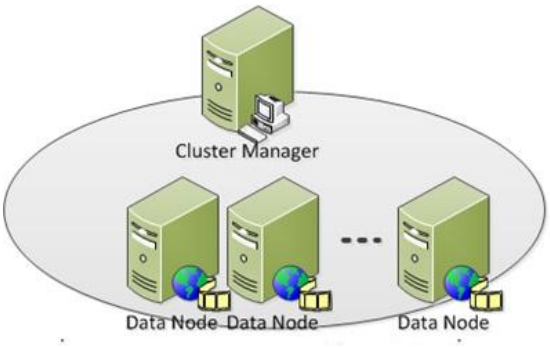

Fig.4. Basic cluster architecture

In the figure 4 , there are two major components: Cluster Manager (CM) and Data Node (DN). CM is responsible for managing cluster, managing storage, accessing and delivering tasks, level-one data index and load balance; DN is responsible for storing data, reading data, level-two data index and data aggregation and computation. They are similar to Fast DFS and our system is based on these two structures.

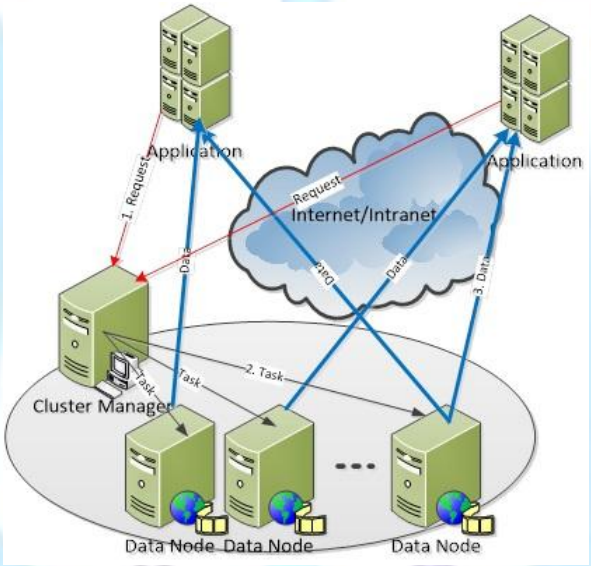

Fig.5. The system's working principle

The figure 5 shows how this system works: first APP send data requirement to CM, CM decomposes the tasks and assigns them to many DNs, then APP directly interact with these DNs, DNs read data and return them to APP.

When client uploads file to DN, the data that will be uploaded should include shot number, begin trace number, end trace number and size of data, for the convenience of DN to establish level-two index. CM queries which DN can store the new file, and in the process of querying, CM should make sure that data of the same shot number but different blocks should be delivered in different groups in order to achieve the goal of load balance. When upload finishes, DN reports to CM that new file has been created, the content of the report should include file's shot number, begin and end trace number, CM will establish level-one index according to these information. Meanwhile, DNs in the same group should synchronize the new file with the same information for the target DN to build level-two index. When client downloads file from DN, the command of download operation should also include shot number, begin and end trace number. DN will query the related data through level-two index based on these information and then return to the client to finish download operation. We take Trie structure to form level-one index, Trie is a tree-like structure, a mutation of hash tree. Its efficiency is higher than hash since it reduces many unnecessary comparisons. We also take RB tree to form level-two index, it is a highly balanced binary search tree, it can guarantee the time needed when the circumstance is at the worst level. When downloading files, due to the variety of needs of customers, the system may download file in blocks according to the shot number, begin and end trace number that is required, so we add multithread to make the procedure much faster than ever before.

3) Communication protocols

There must be communication problems in order to achieve the whole distributed file system. In our work, these problems can exist between CM and DN, DN and DN. The communication protocols in Fast DFS are concentrated in some important files, we should add or revise some of the commands in them based on the needs of actual seismic data, and finally establish our file system.

According to the description that has been discussed before, we add or revise the following commands:

1. DN to CM

   Tracker_Proto_Cmd_Storage_Report_New_File:

   DN reports to CM that new files have been created: when Client successfully uploads files to DN, DN should report to CM of the information about the new files' properties, such as shot numbers, trace numbers, group numbers, so that CM can establish level-one index.

2. Client to CM

   Tracker_Proto_Cmd_Service_Query_Store:

   CM queries which DN to store data: since we should take load balance into account, so data in the same shot but in different blocks should be stored in different groups.

   Tracker_Proto_Cmd_Service_Query_Fetch:

   CM queries which DN has the data to download: we should query this according to some information such as shot number, begin and end trace number.

3. DN and DN

   Storage_Proto_Cmd_Sync_Create_File:

   Synchronize files between DNs when uploading files finishes: when synchronizing, some important information are needed so that target DN can build level-two index.

4. Client to DN

   Storage_Proto_Cmd_Upload_File:

   Upload files: when uploading data, shot number, range of the trace of data are needed so that DN can build level-two index.

   Storage_Proto_Cmd_Download_File:

   Download files: when downloading data, shot number, range of the trace of data are also needed, in that case, DN can query level-two index and return the required data. On the other hand, the data downloaded may not only be the whole file, but some trace data as well.

4.3 Read and Write working principle

In seismic data, the most commonly used operations are write data and read data[7]. To facilitate identification, we define the form of the filename as ``shotNumber_beginRecvNumber_endRecvNumber''. In our Distributed file system (we call BDSS for short), the level-one index will be established in the tracker and the level-two index will be in the storage.

1) File write

When file system starts, we can operate file write, and data have been created as the form of our optimized structure (header data and data body are separated rather than stored together like SEG-Y), and we should also specify the shot number, begin trace number, end trace number of data. Write command is assigned as: command <config filename> <local filename> <shot_number> <begin_recv_number> <end_recv_number>. From the form of our command we can see that since we have signified shot number, begin and end trace number when writing files, clients can write their important data to the file system more conveniently, thus more flexible and humane.

The figure 6 shows how BDSS write the file. first, the client requires to the tracker server about which storage server can write the file; second, tracker server accepts the requirement and return the related IP address and port number of the storage server that is available; third, the client use these information to establish connect with this server and write the file, and then the storage server should report information about the written file to the tracker server, the tracker server will add it to level-one index; finally, the storage server will return the file ID to the client when it report the new file successfully, and the file operation finishes.

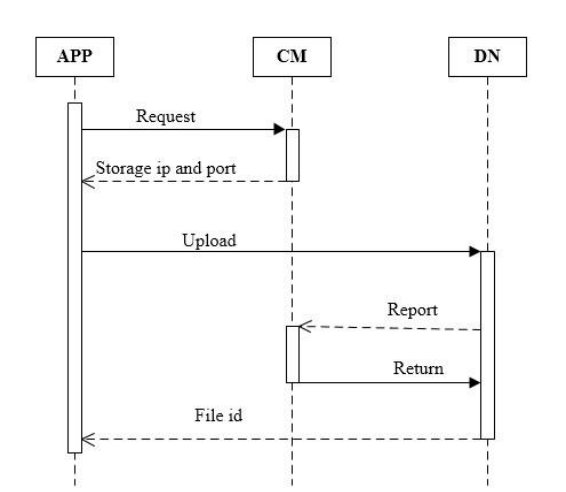

Fig.6. Write operation

2) File read

Similar with file write, reading files should also specify the file's shot number, begin and end trace number, while DNs query related data through level-two index according to these information to performance reading. Meanwhile, to improve efficiency, we limit the range of trace number, that is, we can read 100 traces of data at most (this value can be assigned randomly, but should take everything into account), however, the client's requirements can be random, so when the range of traces that are required is too large, it is necessary to separate data, and take multithread methods to improve the efficiency. reading command is assigned as: command <config filename> <filename>, the form of the filename is signified as “data_shotNumber_beginRecvNumber_endRecvNumber”.

The figure 7 shows how BDSS read the file. first, the client requires to the tracker server about which the storage server can read the file with a require parameter file ID number; second, the tracker server queries which storage server has this file, and return the related IP address and port number to the client; finally, the client uses these information to have a connection with this storage, and then gets the file.

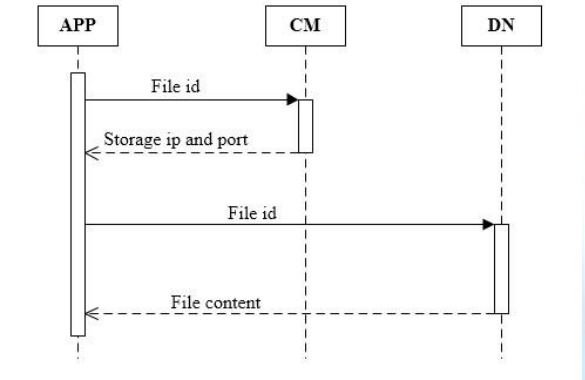

Fig.7. Read operation

As we have fully considered the advantages of the system architecture and different clients’ requirements. Besides, similar to write operation, clients can assign the location information of files that are to be read, thus more flexible, high-efficient and humane.

4.4 Compatison of the effciency

As for the evaluation for random reading, we test this read performance among BDSS, the original Fast DFS and Hadoop DFS based on the same cluster.

1) Cluster configurations

The specific configurations for our cluster are as shown in table 2:

Table 2. Cluster contigurations

IP address	Nodes	Memory	CPU	Operating Systems	Ethernet	HDs
10.1.0.100	CM	4G	E5410 @ 2.33GHz 2(8)	Centos6.4	Gigabit	WD1600BEKT 160G
10.1.0.5	DN	4G	E5506 @ 2.13GHz 1(4)	Centos6.4	Gigabit	ST3300656SS 300G
10.1.0.7	DN	4G	E5506 @ 2.13GHz 2(8)	Centos6.4	Gigabit	MBA3147RC 160G
10.1.0.9	DN	4G	E5520 @ 2.27GHz 2(16)	Centos6.4	Gigabit	ST3300656SS 300G
10.1.0.11	DN	4G	E5506 @ 2.13GHz 1(4)	Centos6.4	Gigabit	ST3300656SS 300G
10.1.0.13	DN	4G	E5506 @ 2.13GHz 1(4)	Centos6.4	Gigabit	ST3300656SS 300G
10.1.0.15	DN	4G	E5504 @ 2.00GHz 2(8)	Centos6.4	Gigabit	ST3300656SS 300G

2) Performance evaluation and results analysis

The version for our Fast DFS is v3.06 while the Hadoop DFS is v1.02. We use 11 different groups of data to perform our research, the specific data are as shown in table 3:

Table 3. Specific data configurations

File size	32k	64k	128k	256k	512k	1m	2m	4m	8m	16m	32m
File Number	100000	50000	50000	50000	20000	20000	20000	10000	5000	2000	1000
Read Number	8192	8192	8192	8192	4096	4096	4096	2048	2048	1024	512

The method for our research is to write all the data listed in the table above to every distributed file system in order to make sure that each system has the same amount of data, and then reading the data from these systems randomly to the client`s memory. For example, there are four clients to read the files of 32KB, after the files have been written, each client will read 8192 files randomly out of 100000 files. Next, we test the performance for simultaneously random reading for one client, two clients, three clients and four clients respectively. We continue our testing by repeating the above process with file size ranging from 32KB to 32MB. Each testing will be repeated three times, and then record their average value. Finally, we get the figures, as shown in figure 8:

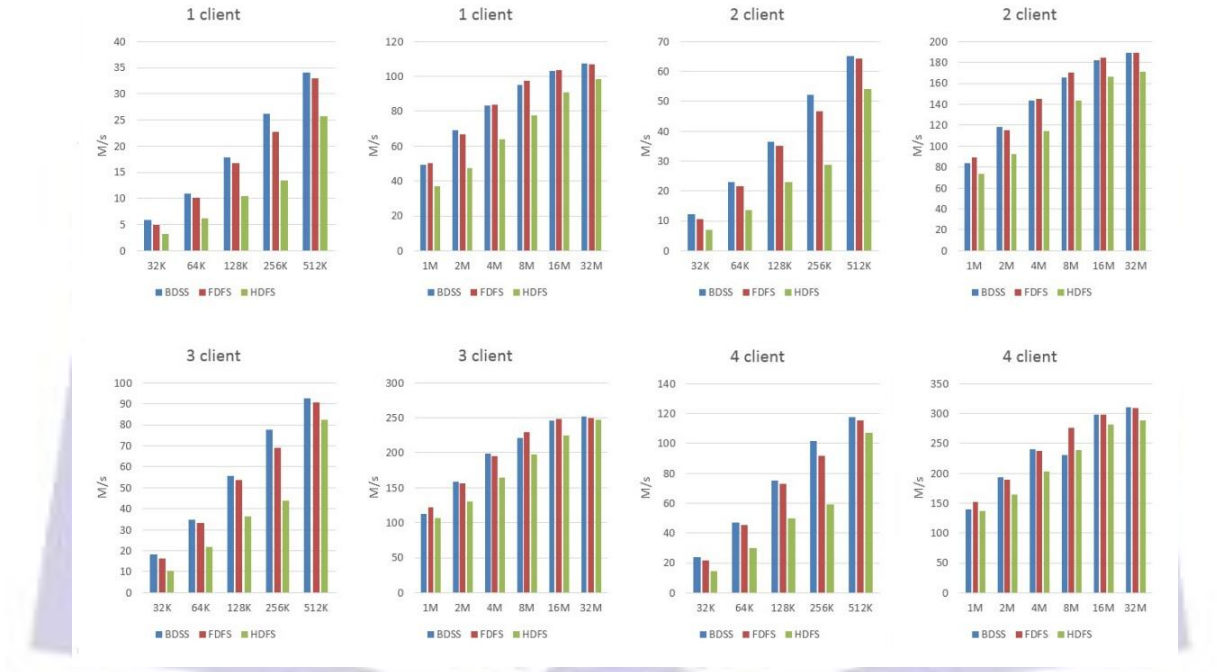

Fig.8. Performances comparison among three systems with 1,2,3,4 clients

From the four figures we can see that, the reading performance of the three distributed file systems has a positive relationship with the file size until it has reached to 16MB due to the gigabit ethernet and the own limitations of the hard disk, in that case, the overall speed cannot improve and the stability becomes worse either.

Basically, the performance of FDFS and BDSS is better than HDFS, especially when the file size is small. For example, the speed for BDSS, FDFS and HDFS is 5.94MB/s, 4.96MB/s and 3.22MB/s when reading the file of 32KB, the speed of BDSS is twice as fast as HDFS. The reason is that FDFS and BDSS is a light-weight distributed file system and is very good at handling small files. But as for HDFS, numerous number of small files will cost large quantity of memory usage in CM node, thus resulting in worse performance.

We can also see that, compared to FDFS, the performance of BDSS has also improved, especially when the file size is small, which can up to 19.8%, owing to the two-level index structures we have added in it. While in FDFS, it is necessary for clients to know the information returned when writing files, and then analyze this information to get the specific DN. What is more, the overall system should also check this filename. Our system has already ignored all this trivial operations, thus improving the whole performance.

There still remains one thing we should focus on: the number of client. When the client number increases, it is obvious that the performance of reading has also increased greatly. Through all the results presented by the clients, we can find that

the performance of the whole system has not been fully developed when there is only one client. In this case, we should use more clients to improve the overall performance.

5. CONCLUSION

This paper focuses on properties of seismic data, through breaking the head data and data body apart and store them separately, the overall reading performance has been improved. Besides, according to the `small in size, large in quantity[12]' property for seismic data, we have developed a new distributed file system aiming at seismic exploration data based on the fast distributed file system. We add our own details to improve the overall performance as well as avoiding some drawbacks existed in the original system. The experimental results show that our system can perform better than the original fast distributed file system and Hadoop file system.

Our future work will focus on further improve our file system. Since there still exists some performance issues even though the overall speed has been improved, we should make a step further to concentrate on small file problems and some other implementation details, such as file merging, file mapping, prefetching methods and so on.

ACKNOWLEDGMENTS

The work was supported by National Natural Science Foundation of China (Grant No:61331017).

REFERENCES

[1] Patel A B, Birla M, Nair U. Addressing big data problem using Hadoop and Map Reduce [C]//Engineering (NUiCONE), 2012 Nirma University International Conference on. IEEE, 2012: 1-5.

[2] Rajaraman A, Ullman J D. Mining of massive datasets [M]. Cambridge University Press, 2012.

[3] https://nutch.apache.org/

[4] Sagiroglu S, Sinanc D. Big data: A review[C]//Collaboration Technologies and Systems (CTS), 2013 International Conference on. IEEE, 2013: 42-47.

[5] Ghemawat S, Gobioff H, Leung S T. The Google file system [C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.

[6] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[7] Claerbout J F. Fundamentals of geophysical data processing [J]. 1985.

[8] Howard J H, Kazar M L, Menees S G, et al. Scale and performance in a distributed file system [J]. ACM Transactions on Computer Systems (TOCS), 1988, 6(1): 51-81.

[9] Zhou H W. Practical Seismic Data Analysis[M]. Cambridge University Press, 2014.

[10] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system [C]//Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010: 1-10.

[11] http://code.google.com/p/fastdfs/

[12] Jia B, Wlodarczyk T W, Rong C. Performance considerations of data acquisition in hadoop system [C]//Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, 2010: 545-549.

[13] http://bbs.chinaunix.net/forum-240-1.html

[14] Borthakur D, Gray J, Sarma J S, et al. Apache Hadoop goes realtime at Facebook [C]//Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011: 1071-1080.