A Novel Approach for Improving Security and Storage Efficiency on HDFS

Yannan Ma, Yu Zhou, Yao Yu, Chenglei Peng, Ziqiang Wang, Sidan Du

Nanjing University, 163# Xianlin Avenue, Nanjing 210046, China

Abstract

Distributed file system for the storage of massive files have obvious advantages compared with the conventional file system. For instance, Hadoop Distributed File System (HDFS) implemented with commodity hardware has the advantages of low cost, high fault tolerance, scalability, etc. However, HDFS has the potential safety hazard due to the unencrypted data stored in Datanode, which may cause data leakage during the manipulation. In this paper, we propose a new architecture based on HDFS, combined with network coding and multi-node reading, to improve the security and storage efficiency of the distributed file system. Experiments have shown that the method proposed in this paper greatly improves the safety of files transfer and storage, while the speed of files reading has increased threefold compared to the original HDFS model.

Key words: HDFS; Network code; safety; high-efficiency;

1. Introduction

HDFS (Hadoop Distributed File System) developed by Apache Foundation, not only takes full advantages of the power of high-speed computing clusters and storage, but also demonstrates high performance in big data storage.

The main concern of network security is to ensure that irrelevant individuals who have no right to use, but attempt to obtain the remote service cannot read or modify the information which will be passed to other receivers. Most of the emergence of the network security problems are because malicious people try to intercept or modify the information which do not belong to him originally, in order to obtain some kind of benefit or harm others intentionally. It is obviously that to guarantee the network security not only needs to make the program without programming errors, but also to guard against hackers, stalkers, especially who have abundant time and money to brute force attack the system.

* Corresponding author. Tel.: +86 15077882201 ; fax: +0-000-000-0000.
E-mail address: steve.1990@126.com
Using encrypt files with public or private key to improve the security of HDFS is a general methods in the present. However, in order to ensure the security of system, the process of file compression and encryption are both completed by Namenode, which will increase the workload of Namenode and result in the low efficiency of the whole system. But once the application developers unveiled the public or private key generated by the algorithm, illegal users could still try out the real data by the method of brute force, to say the least. Therefore, how to guarantee the security of the system and improve the reading efficiency at the same time is a great challenge we have to face.

Based on the comprehensive consideration for data transmission efficiency and problems of security, we propose a new architecture based on HDFS, combined with network coding, in order to strengthen the security of file transfer and file storage. We encode the backups stored in Datanode and stored part of the encoded files in other backups to improve the security of file storage and adopt the idea of multi-node reading to improve the efficiency of file storage. When clients want to obtain files, data can be received from multiple nodes and the character blocks may be decoded through multiple backups at the same time. Experiments show that the method proposed in this paper greatly improved the safety of files transfer and storage, while the speed of files reading has increased threefold compared to the original HDFS model.

A brief overview of related work is given in Section II. We will introduce the framework for our system in Section III and describe the implement of the whole system in Section IV. The performance evaluate is shown in Section V. Our conclusions and limitation will be shown in Section VI.

2. Related Work

In order to increase storage efficiency, HDFS specially developed Hadoop Archive (HAR) to alleviate this problem. However, HAR occupies high memory usage and cannot update data, at the same time, the reason of restricting its further development is that it does not significantly improve the data transmission efficiency. Sequence file is faced serious challenges in the same way as HAR.

There are also methods to improve efficiency from system level. The basic idea for WebGIS is to merge massive files into a big one and to built index for each small file where hash index is used. Establishing secondary index directory or prefetching mechanism is also another way to improve reading efficiency. But both methods above need high requirement for the integrity of the file stored.

From above, various kinds of modified models for HDFS cannot both increase the efficiency for file reading, and enhance the security in the process of file transfer. In our model, we made some changes on the basis of the original HDFS model. We greatly improved the efficiency for file reading and avoided the possibility of files being stolen in the process of file transfer. At the same time, the system can guarantee that even if parts of saved files and backups in an acceptable range, it still can restore the original data completely in a decent efficiency.

3. Network Coding and Multi-node reading

Nowadays, users are not only increasingly focusing on the efficiency, but also pay close attention to the integrity and security requirements in the process of information transmission. Based on this, network coding technology emerges as the times require.
Network coding can be seen as a kind of information encryption. The system cannot cause information leakage or permanent damage and other adverse consequences even if parts or all of data in the information lose or are stolen in the process of transmission through the information channel. We have adopted the Random Linear Network Coding in this paper. This coding pattern has high probability of successful decoding with strong practicability. We chose \( n \) as coding coefficient first, then we can get \( n \) encoding factors \((a_1, a_2, a_3, \cdots, a_n)\) represented by binary and string coding matrix composed by \( n \) rows (the content of each row is assumed to be \((b_1, b_2, b_3, \cdots, b_n)\). As Fig.1(a) shown, the first line to the last line of character can be represented as \( a_1 \cdot b_1 + a_2 \cdot b_2 \) to \( a_n \cdot b_n + a_1 \cdot b_1 \).

If we choose encoding factor as \((p_1, p_2, p_3, \cdots, p_n)\), then the result of the second encoding can be represented as Eq.(1).

\[
\begin{pmatrix}
  p_1 \cdot (a_1 b_1 + a_2 b_2) + p_2 \cdot (a_2 b_2 + a_3 b_3) \\
  p_2 \cdot (a_2 b_2 + a_3 b_3) + p_3 \cdot (a_3 b_3 + a_4 b_4) \\
  \vdots \\
  p_{n-1} \cdot (a_{n-1} b_{n-1} + a_n b_n) + p_n \cdot (a_n b_n + a_1 b_1)
\end{pmatrix}
\]

In the process of file storage, we still regard “block” as a unit of storage, but the difference is that we treated each block as a matrix for Random Linear Network Coding and stored the block after coding in the location of the backup in the original HDFS model. In addition to this, we code the coding matrix twice again and store part of the coding matrix storing the second-encoding matrix and third-encoding matrix in the data node to improve reading efficiency. It is important to note that in order to guarantee the security of file storage, we will store part of data in the end of each encoded encoding factor in other backups, and the specific mapping for factors to factors will be stored in Namenode. The benefit is that we ensure that each backup does not contain a complete original data, even if the whole backup completely lose, illegal winner cannot restore any data effectively, because each encoding factor is incomplete. Unless the illegal winner obtains the overwhelming majority of a backup and also wins the mapping for each factor to factor from Namenode at the same time, otherwise, it is absolute that there does not exist the possibility of data being stolen.

When decoding the matrix, we read an entire line strings in code matrix randomly and put it in the decoding matrix. We elementary transformed the matrix when the characters in the matrix more than one line, rejecting the part of linear independence only. The character in the matrix is original data if and only if the current matrix is full rank matrix.

When reading, we can read data from the content which are stored in the position of the three backups at the same time and retain the part of linear independence only. It is worth mentioning that the coding coefficient of each line is saved at the end of the new matrix (in our model, the size of the valid data stored in the block is only 62MB while the size of the whole block is still 64MB, and the encoding factors are saved in the final space of 2MB).

Namenode was in charge of the part of coding while the work of decoding was done on the client side. The original data will not appear in the process of data transfer in the channel. Furthermore, system provides a significative function which is data recovery. Namenode will check Hash value of each node regularly, and then upload them to Namenode and compare to the original Hash value. There will be something wrong with that node if the two values are different. The content of that node will be recoded in order to ensure the correctness of the data according to the content of the adjacent nodes.

4. Framework of Our Approach

In this section, we introduced this improved file access methods in two aspects of writing and reading files. The schematic diagram was shown in Fig.1(b).

4.1. Write files

When client receive a request to write files,

1) Namenode will screen the files preliminary. File only less than 62Mb is considered as valid file. Due to the experimental environment needs, in this paper, all the test files were less than 62Mb, which are valid files. Size of files more than 62Mb will be operated in conventional filesystem.
II) Namenode is responsible for finding appropriate block to store files, and establishes the mapping for files to blocks. These mappings are regarded as part of metadata to store in the memory of Namenode. Compared to the original HDFS model, the difference is that, when the files size put in the block exceeds 62Mb, Namenode will stop to stock files into block, while the size of block Namenode selected is 64Mb. The remaining space will be used to store coding coefficients.

III) Random Linear Network Coding is on the filled block for the first time and the already encoded block’s location stored in Datanode is the location of the first backup of the original block.

IV) Let the already encoded block (stored in the first backup) Random Linear Network Code for the second time, and stored in the second backup which the original blocks backup stores.

V) Repeat step IV. The result of the third Random Linear Network Coding is stored in the location of backup3.

VI) In order to ensure that original data contained by any block is incomplete, parts of every factor in one block are interchangeable with other parts of every factors in other blocks. The mapping for factors to factors will be regarded as part of metadata to store in the memory of Namenode.

VII) Namenode modifies the mapping for files to blocks, and uses the new mapping for files to backup1, backup2, backup3 instead of the original mappings. Among them, nodes numbers, coding coefficients and length of coding string of the Random Linear Network Code in step III, IV, V are all depended on the experimental conditions. The specific values will be introduced in Section IV Simulation Results in detail.

4.2. Read files

When client receive a request to read files,

I) Client notifies Namenode which file user wants to read. After that, Namenode find the locations of three backups linking to the file to be read according to the mapping for files to three backups and mapping for factors to factors which stored as metadata in Namenode before.

II) Namenode decodes these three backups simultaneously and read fixed length characters from these three backups simultaneously, then takes these characters into a matrix. The length of the character to be read every time depends on the coding coefficients at the end of the backup (the last 2Mb in the block is specifically for storing coding coefficients). Every time the matrix receives a string of characters (except the first time), does a determinant of transformation, and abandons the linear dependence parts, only retains the linear independence parts. If and only if the remaining characters blocks constitutes a full rank matrix, Namenode stops reading characters, while the characters in this matrix is what the client wants.

5. Experiments and Performance Evaluation

In this section, we compare the efficiency of our approach, original HDFS model and Hadoop archives (HAR) when reading different size of files.

Our tests platform is built on a cloud experiment system with four servers under the public network which have Intel Core I7 4770k CPU of 3.5GHz, 32GB memory, 2TB SATA disk. The operating system is Ubuntu 12.04 LTS with kernel 3.2.0-23-generic. Hadoop version is 2.0.0 and java version is 1.6.0. One of the virtual machine is built as Namenode, while the other three are Datanodes. In our test, 2MB are considered to be a coding unit. The minimum coding string is 1KB in the coding matrix and we choose 8 as coding coefficient which means the size of 8KB files composed a coding block.

In order to pursue the mean bandwidth allocation, we set two thresholds and artificially limited the amount of data uploaded by a single node per unit time according to the bandwidth. If the amount of data uploads by one node more than the higher threshold within the allotted time, stop the node to upload data until the data size other nodes upload achieve the lower threshold.

Fig.2(a) shows the comparison of reading time for the original HDFS model and our approach when dealing with files in same size. We choose 3 nodes (backups) to upload files at the same time. From the figure we can see that the reading access time saves nearly 60% compared to the original HDFS while the saving time compared to Hadoop archives is nearly 50%. The reason for saving time slightly less than the theoretical value (67%) is that it takes some time for searching randomly linearly independent data and work out full rank matrix. Experiments show that the
model can shorten file reading time stably and the saving time does not appear large variance when the size of files increase gradually. In addition to this, even if any illegal users steal one or several blocks, it is impossible for him to restore the original data due to the data of each coding factors in each block is incomplete. If he wants to capture the data by brute force, he must obtain all the blocks and get the mapping for factors to factors from Namenode at the same time. On the other hand, for legitimate users, there is little possibility for stored files lost because the system will periodically check file storage situation according to the hash value. If a whole block is lost in the process of file transmission, the system can still restore the original data through the remaining blocks in a considerable time due to the data of each block are redundant.

![Fig. 2. Comparison of Three Models](image)

Fig. 2(b) shows the time the improved model spent while reading the same files with different number of coding nodes. It can be seen from the figure that the more nodes have been used, the higher reading efficiency we have. The reading time by six nodes will be a little more than half the reading time by three nodes because the distance between the location of the backups storage and the decoding matrix are different and it takes time for Namenode to inform each node to stop upload respectively after work out full rank matrix. If one of the six block missing, the file reading time will only about 10% more than reading files through 5 blocks. Because we have the mapping for factors to factors, even if there is only one block remaining, we can still restore most of the original data in extreme cases.

6. Limitation and Conclusion

In our approach, file storage time was extended by about 25%, because files needed to be encoded before storing. The file storage time extended slightly is acceptable compared to the shorten reading time.

In this system, we greatly improved the ability of security and storage efficiency of the system through special decoding mode and multi-nodes reading. Meanwhile, we can store part of the metadata in Datanode to reduce the memory usage of the system, so that the storage efficiency of the system may improve obviously through the decrease of Namenode’s workload because work of data encoding is done on the Namenode.

References