Storage Mechanism of Processing Magnanimity Small Files Applying HDFS Architecture

Wang Xin[1, a] and Ma Jun[2,b]

[1]Department of Computer Science and Technology, College of Geophysics and Information Engineering, China University of Petroleum-Beijing, Changping District, Beijing, China, 102249

[2]Department of Computer Science and Technology, College of Geophysics and Information Engineering, China University of Petroleum-Beijing, Changping District, Beijing, China, 102249

[a]xinwang@cup.edu.cn, [b]282759030@qq.com

Keywords: HDFS, I/O Efficiency, Disk Buffer.

Abstract. Pointing at the low efficiency problem of I/O operations of small files using HDFS application architecture, this article proposes a new method, that is, add disk buffer on DataNode to reduce addressing time of reading small files, and thus reduce reading time, and reduce the stress bringed by frequent NameNode access, so as to optimize the reading efficiency of small files. Experiment results show that the design programme is feasible, and adding disk buffer benefits for increasing efficiency of processing small files applying HDFS.

Introduction

Hadoop Distributed File System (HDFS for short), put out by Hadoop, is a distributed file system similar to GFS. It has the characteristics of high efficiency, high scalability and low price, and fits for storing massive amounts of internet data. Like the GFS, HDFS is designed for the problem of pocessing magnanimity index data in search business, which has the similar architecture and is often combined to a lager file, thereby increasing efficiency of batch process, meanwhile HDFS itself is optimized for processing big files. When practically used for some transaction except search, HDFS often faces the I/O efficiency problem of small files, especially for the transaction of mail attachment, software download and so on, in which data files are mainly small files, so the efficiency problem becomes more prominent.

Architecture of HDFS Application

Architecture of Traditional HDFS Cluster. Traditional HDFS cluster is made up with a NameNode and some DataNodes. As the main node, NameNode stores the metadata of the whole file system, and is charge of managing the data storage and data I/O in the cluster, and maintaining data node change and data safaty. Reading and writing operations of data from outter cluser must access the NameNode first, and then achieve DataNode list allocated by NameNode, and then go on with the real data operation with DataNode. Its advantages are that, the inner architecture of the cluser is very simple, only NameNode, DataNode and Switcher existing, distributed computation model such as MapReduce can make full use of the brevity to transfer data computation, so it can reach a high data throughput inner the cluster; meanwhile the brief architecture reduces the difficulty of maintenance work, and relatively improves the reliability of the data. But this architecture has many deficiencies[1-3] when used to other environment except MapReduce, for example, (1) reading and writing data from outer network needs to directly access the NameNode, so that frequent access could cause a lot of performance pressure on the NameNode; (2) the design of single switcher gathers the flow into a singler network segment between data transmission of inner and outer network and cluster management of HDFS, which causes frequent network busy and limits data transmission efficiency.

Efficiency Problem of Small Files' I/O in HDFS. The reason of HDFS's influence on small files' I/O efficiency problem includes 2 aspects[4]:

(1) Longer addressing time in the network.
(2) Small files' shorter transmission time in the network.

If lager propotion of time is used for addressing in the whole I/O process of a block of data, the efficiency of I/O will drop. Taking data reading in HDFS for example, after application program calls HDFS Client for reading data, HDFS Client will first request NameNode for the list of data blocks included in the file, and then read every data block according to the list one by one. So for small files, the proportion of addressing time in total data acquisition time increases significantly, and this leads to the decrease of the efficiency of data reading.

the Design of Application Architecture after Adding Buffer. According to its location, buffer data can be devided into 2 kinds, memory cache and disk buffer. Since HDFS holds a large number of data, it's obvious that lots of files can't be stored in memory, as a result, this article adopts a method of adding local disk buffer for HDFS to improve the efficiency of reading small files.

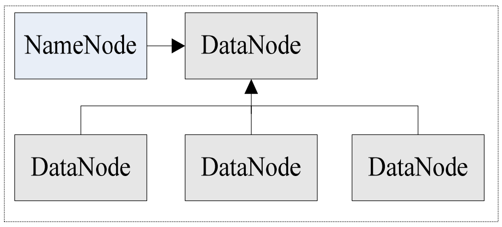

Fig.1 Data Reading Process on DataNode without Cache

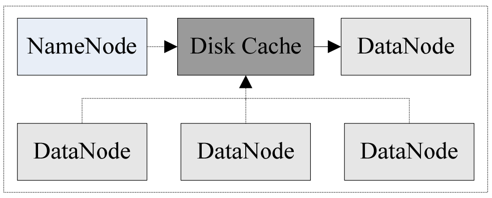

Fig.2 Data Reading Process on DataNode with Cache

Fig. 1 describes data reading process on DataNode in application architecture. DataNode firstly requests the position information of data block from NameNode. And then,it sends request of reading data to corresponding DataNode,and receives data. Fig. 2 describes the flow diagram of data reading after adding disk buffer. DataNode requests reading files to disk buffer; disk buffer checks whether the requested files exist in local buffer, if exist, it directly response the data, otherwise it calls HDFS Client to read the files from HDFS and stores to local buffer.

In this case, often accessed files are stored in local disk. So network addressing time is saved a lot, as is the inner bandwidth of HDFS. Accordingly, reading efficiency of small files is increased obviously.

Design and Implementation of Buffer

Small data files account for a large proportion in the internet, and seldom be modified once created, but frequently accessed files only take a low percentage. So efficiency problem of small files' I/O is inevitable in HDFS, but we can try to optimize it by adding buffer[5].

Replacement Strategies of Buffer. We can't put all files which need to read into the buffer because of the limited space, so the replacement strategies must be considered when the buffer fills up.There are two kinds of common replacement strategies: first access and weight calculation， we used the first access policy in the paper. First access stratage is that, the buffer manager program records the last access time of all files, when the buffer space is filled up, the file which has the earliest last access time will be removed from the buffer.

Percentage of Buffer Space in the Local Disk. To improve efficiency, the more local buffer space is, the better. But DataNode needs to leave more space for HDFS, so local buffer space and disk space for HDFS should be reasonablly allocated.

Assume that the whole space of all disks is S, the ratio of buffer is p, redundancy rate of data block is n, the ratio of often accessed files in all files is f, if all the files that often be accessed are stored in the buffer, we can can come up with an equation:

$$S \times p = S \times (1 - p) \div n \times f \tag{1}$$

That is:

$$p = (f \div n) \div (1 + f \div n) \tag{2}$$

So p has nothing to do with S, but only relates with f and n.

Size Limit of Buffer Files. When bigger file is swapped into buffer, it causes lots of small files to be swapped out of buffer. Meanwhile the bigger size the file is, the lower efficiency caused by file addressing time, so to limit buffer file size is necessary.

Assume that data block size is f(MB), average addressing time is t(ms), network transmission speed is s (MB/s), file need be stored in the buffer when the percentage of its addressing time in the whole reading time is more greater than or equal to r, we can catch up with the inequality:

$$t \div (f \div s + t) \geq r \tag{3}$$

That is:

$$f \leq t \times s \times (1 - r) \div r \tag{4}$$

Design of Buffer Program. Buffer program mainly includes the following 4 types:

BufferManager class: the server program of RMI, also the manager program of buffer. It manages buffer access from client, obtains the files to be buffered, swaps buffer and so on.

FileManager interface: interface to call methods provided by the server program.

FsClient class: the client program of RMI. It calls methods provided by the server program through the FileManager interface.

FileUtil class: the tool class shared by the server and client. It contains some public methods, such as getting file states and file data from HDFS.

Features of Buffer Program. Compared with ordinary buffer program, the disk buffer program for HDFS has the following features:

(1) Buffer program shoud exist out of HDFS, and run independently in the way of server, but not embeded in HDFS.

(2) Buffer program needs to read data multithreadingly from HDFS, and supports the client program multithreadingly reading data from buffer. This requires that, the buffer program must ensure thread safety by means of locking data shared by threads, and try its best to reduce influence on the stability of the whole program when single thread runs in error. Logs should be recorded so as to analyse the cause of the error.

(3) Buffer program should keep its high performance. Multi-thread operation must cost mor time and space on thread control, so performance should be improved by continually code optimization.

Analysis of Expriment Results

By tests of reading a large number of small files with buffer and without buffer, recording and comparing the time it took, we verified the effect of the buffer for improving the reading efficiency of small files in HDFS. The specific content of the experiment is the buffer efficiency test, which was divided into 6 times. According to the proportion fo buffer size in the whole size of files in HDFS, conditions were set with no buffer, 20% buffer, 40% buffer, 60% buffer, 80% buffer and 100% buffer, and every test read random files 2100 times in HDFS. Exepriment results are shown in Fig. 3.

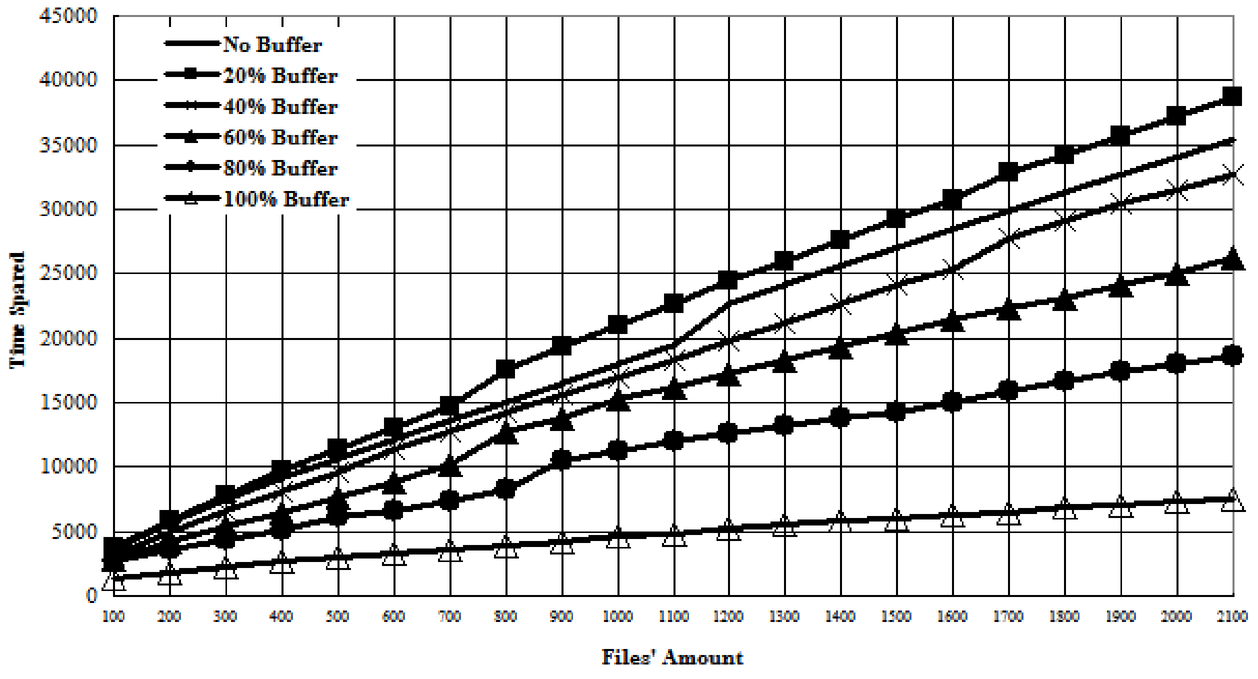

Fig.3 Line graph of experiment result

As can be seen from Fig. 3,

(1) With the increasement of buffer size, file reading time decreases significantly. It shows that buffer has good influence on optimizing the efficiency of reading small files, that is opitimization by buffer is feasible.

(2) With the decreasement of buffer size, reading time increases rapidly. In the case of 20% buffer, reading time has exceeded the time with no buffer.This is because, when buffer size reduces, the probability of existence in buffer of accessed files reduces, too. Buffer program need process the replacement of files frequently, and it obviously affects the buffer efficiency. Especially, when the buffer size is very small, the consumption of replacement work reduces the efficiency to worse degree than which with no buffer. Therefore, in actual application, it's considered that buffer size should be given as large as possible, and avoid the use of buffer when the space is not enough.

Summary

Experiment results show that disk buffer has better effect on efficiency optimization for reading small files, but it's just a temporary solution, making adjustment in HDFS internal should be better, which can make more thorough optimization for small files' I/O efficiency problem.

References

[1] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Yahoo! Press, June 5, 2009.

[2] Apache Hadoop. http://hadoop.apache.org/.

[3] Dhruba Borthakur, The Hadoop Distributed File System: Architecture and Design. Hadoop Docs.

[4] Luan Yajian, Huang Qunmin, Gong Gaochen, Zhao Tiezhu, Performance Optimization Study of Hadoop Platform.

[5] Chen Junjie, Zheng Weimin, A Dynamic Allocation Resolution Algorithm of the File Allocation Problem.