Small Files in HDFS

Shiva Nalla, Sai Sravanthi Nudurupati, Tejaswini Gutti

1. Background

1.1 Hadoop Distributed File System (HDFS)

Hadoop is an apache open source framework used for storage and processing large data across distributed clusters using simple models. HDFS is designed for easy manipulation of data stored across different clusters. HDFS is highly fault tolerant.

HDFS follows master slave architecture. It consists of mainly two nodes:

1. NameNode
    a. NameNode acts as master server
    b. It is responsible for maintaining file metadata and file system action
    c. Maintains file name, file properties like creation time, permission, size of file, blocks that the file is spread and corresponding DataNodes
    d. NameNode is rack aware i.e. it keeps track of list of DataNodes in the rack
    e. Manages the client request
    f. Capable of executing operations like closing, opening and renaming of files and directories
2. DataNode
    a. Stores data in its local system in the form of blocks
    b. Reports back to NameNode periodically with lists of blocks they are holding
    c. Services read/write requests as per client request
    d. Handles all the replication requests

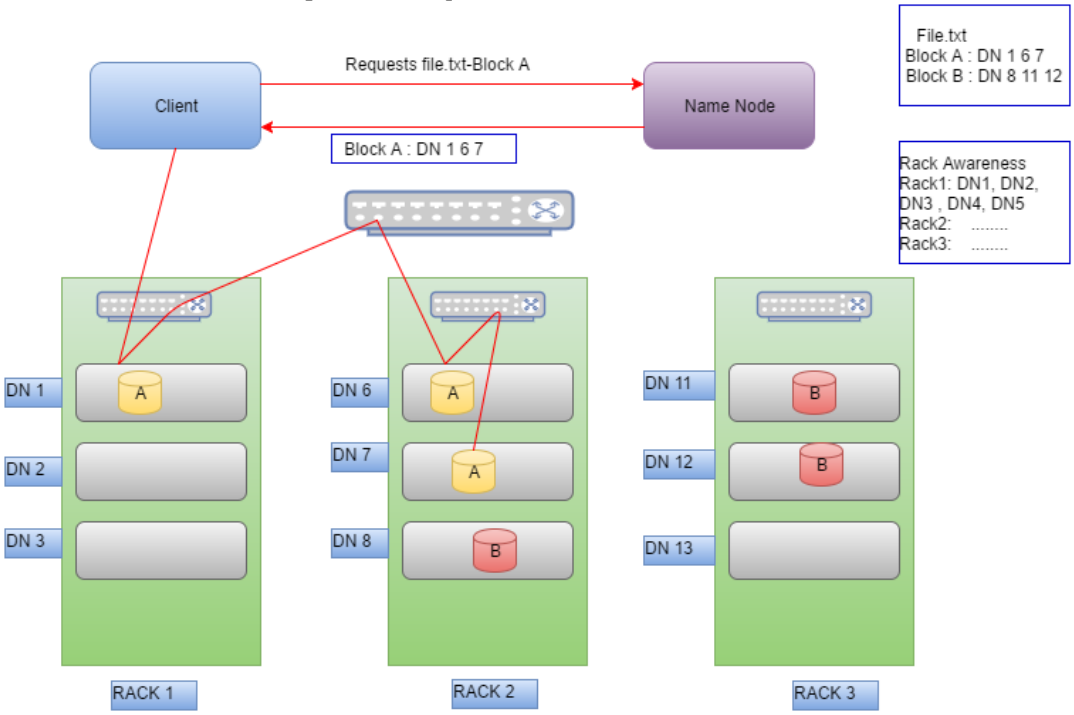

Figure 1: Hadoop Distributed File System

In HDFS, when client requests for a file say file.txt, the request is redirected to the NameNode (Figure 1 depicts the HDFS architecture). NameNode has the information of which blocks and DataNodes holds the file. In this case file.txt is spread across Block A and Block B and replication factor being 3, each block is replicated three times and distributed in three different DataNodes. The block is distributed in such a way that one copy is placed in say rack1, other two copies are stored in same rack say rack2 but different from first copy. In this way fault tolerance is achieved in case of rack failure and also that when writes are performed, network bandwidth consumed will be less because if DataNodes are in same rack, then it can communicate through internal switch avoiding unnecessary communication through main switch between the racks. We can see in the Figure 1 that NameNode also maintains the list of DataNodes in the each rack.

1.2 What are Small Files?

Any file which is significantly smaller than Hadoop block size can be named as a small file. We can define that any file less than 75% of Hadoop block size can be categorized as a small file. Default Hadoop block size is 64MB but its trending towards larger block size and can define the block size to be 128MB, 256MB etc.

Reason for Small Files:

- Few files are inherently small by nature, for example, large corpus of images
- In organizations, nowadays with increasing demand towards data availability and information production, data is being pushed into the file system as and when it's being generated without any modification
- Small chunks of data is produced when more number of reducer tasks are used than necessary

1.3 Problems with Small Files

Hadoop is developed for dealing with large data. It performs poorly when it comes to small files. Below are the two main issues with small files in HDFS:

1. NameNode Memory problem

Everything in Hadoop is represented as an object i.e. each file and each directory is represented as an object. Each file/object occupies approximately 300 bytes of NameNode memory where 150 bytes is for file name and file properties and another 150 bytes is for block and DataNode information. For example, let's say we have 30 million files each occupying a block and each block requires 300 bytes of NameNode memory. So we need a total of 720 GB of NameNode memory.

Let's discuss the impact of having huge consumption of NameNode memory by small files.

- When NameNode loads or restarts, 720 GB of data must be read from disk. I/O operation is really expensive in terms of time
- NameNode being the master node, it should maintain all the information about which blocks consist which DataNodes. So DataNode periodically updates the NameNode about the block information. If there are more number of blocks, it results in high consumption of network bandwidth
- NameNode will run out of addressing capacity if there are more number of blocks (more number of small files). With more number of blocks, more information must be maintained in NameNode. Hence, in spite of lots of space availability in HDFS, it still cannot be utilized due to this reason.

2. Map Reduce Performance

It is always better to deal with small number of large files than to with large number of small files because large of small files will cause large number of random disk I/O. It will degrade the performance of Map Reduce operation. Another reason for bad performance is because of the way map reduce works. Usually at least one block corresponds to a file and map jobs are divided in a way that each block will be assigned with a map process.

If there are 10000 files each of 10MB data then each file will be having a map task. So in total we have 10000 map tasks and each map task is configured with its own JVM. If a Hadoop clusters has 20 nodes, and each node is capable of processing 5 to 20 concurrent mappers; with total of 100 slots, the queue will be very huge. Instead if we have the same data in the form of 800 files each of 128 MB, we would require only 800 map tasks. There will be a huge difference in performance. We have run a small program to demonstrate the difference in performance between large number of small files and small number of large files. Results will be discussed in Implementation section.

There are number of techniques to address Small Files problem. We have used sequence file technique and also tried to combine it with CombineFileInputFormat technique to reduce the number of map tasks used in generation of sequence file.

2. Related Work

In the research work published by Jilan et.al. [1], the work aims to overcome the problems with small files by sorting all the small files in a directory by size and merging them to form one bigger file to reduce the metadata maintained in the namenode. If the total size of all files is less than or equal to the default block size, a single block is created; otherwise multiple blocks are created; in such cases, if a file is happens to be distributed across 2 blocks, the entire file is pushed into the second block. A global mapping is maintained that maintains associated indexes for accessing files quickly. A similar approach can be seen in the work done by Dong et.al. on a courseware software called as BlueSky, a popular online courseware. The courseware maintains a directory structure where all the files belonging to a particular courseware goes into one directory and a number of images of different resolutions for the presentation are also stored while uploading it (PPT) file into the file system. All the image files and related courseware presentations are treated as correlated files and are merged to form a larger file. An index file is also created and placed at the beginning of the merged file. It proposes the technique of prefetching, which is fetching related files in anticipation of those files being accessed in the near future; two types of Prefetching are discussed: Local Index prefetching in which Index file is fetched and maintained in the cache to enable faster retrievals without contacting the namenode and Correlated File Prefetching, in which several correlated files are also pre-fetched into the cache.

The work done in [3] focuses on a specific problem area where every client is assigned a quota in the filesystem for both the space to be utilized and number of files. It proposes the technique of harballing that utilizes compression strategy of Hadoop API and also designs a solution that does in-job/ dynamic archival of directories and files to gracefully provide space for map-reduce operations when running out of space so that the running job can be completed successfully without halting abruptly. [4] discusses about Hadoop achieve files and Sequence files and proposes a similar solution as [3] where in the number of reduce tasks are dynamically set to an optimum number based on the input data block size. [5] works on a technique similar to [1] and works on eliminating small files which are more apparent in weather data. The solution is based on merging multiple small weather data files into a larger file that can be used in future map-reduce processing, thus reducing the space needed by the meta-data at the namenode and improving

the performance of map-reduce computations. [6] proposes an Extended HDFS solution in which the work done is very similar to the work done in [2] where files are merged and prefetching strategy is employed to reduce the burden on NameNode and also provides an improved file accessing strategy. Our work done in this project is more closer to the approach discussed in [4], however, the technique discussed can also be used to build an efficient solution to handle specific cases like the courseware software in [2] and the weather data in [5].

3. Problem Definition

Provide an implementation of a solution for overcoming the Small Files Problem in Hadoop Distributed File System and provide an evaluation of the solution against the problem areas caused by Small Files.

3.1 Problem Description

Small Files can cause a huge bottleneck to the NameNode address space that supports the DataNodes and also consume more system resources and time to perform a Map-Reduce computation. The project work shall aim at implementing a solution to overcome the Small Files problem. Also, the work shall provide a comparative evaluation of the NameNode memory space utilized before and after the Small Files issue is addressed. Also, the performance of the Map-Reduce computations shall be analyzed before and after the Small Files issue is addressed.

4. Proposed Solution

The solution approach consists of merging all the small files in a folder to create a larger Sequence File. We will briefly discuss about Sequence Files and discuss how we can build a Sequence File to resolve the Small Files problem in HDFS. The implemented solution is further tested and evaluated using datasets from Project Gutenberg. Various parameters that affect the performance of HDFS file system under the presence of large number of Small Files are discussed in the next section on Experimental Setup and Evaluation.

4.1 Sequence File

The idea is to merge all the small files in a folder into a sequence file. Sequence file consists of binary key and value pairs. The Key corresponds to the filename and value corresponds to file content. Sequence file can be split and processed i.e. sequence file can be given as input to multiple map tasks if it is long. Sequence file consists of a header followed by one or more records. The header and record format is depicted in Figure2.

SEQ acts as an identifier for sequence file which requires 3 bytes followed by version number of one byte. There are three formats of sequence file

1. Uncompressed format - The key and value pair is stored without change
2. Record compressed format - Only the value is compressed and stored
3. Block compressed format - Data is not written until a threshold is reached. When that threshold is reached, key and value pairs are compressed individually placed and sync marker is inserted in between blocks.

There are flags to represent the format used in the header. Record has the total record length i.e. key length plus record length followed by key length, key and value to distinguish between the keys and values. Sync marker is placed after every 100 bytes depending on the format used.

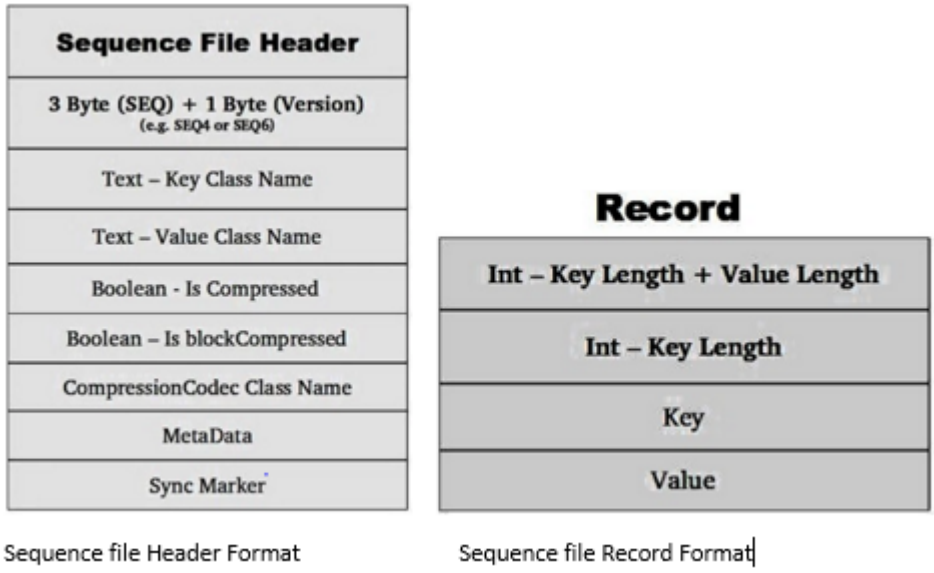

Figure 2: Sequence File Header and Record Format

It should be noted that Sequence Files are the base data structure for many other types of files including MapFile, SetFile, ArrayFile that have additional capabilities to retrieve keys and values, in addition to the benefits of a SequenceFile. Whenever random access is needed, one can always use a MapFile which maintains an additional Index File to retrieve values based on a key or filename.

4.2 Implementation

Multiple Small Files in a folder are merged into a sequence file by creating custom classes extended from Hadoop API as follows:

- Custom Input Format class by extending FileInputFormat
  - Each file content is processed as a single record
- Custom Record Reader class by extending RecordReader
  - Full file is processed as a record
  - Data of file is copied into a byte array and stored
- Custom Mapper class by extending Mapper
  - File names are stored as keys; File contents are stored as values
- Sequence File Generation
  - A single reducer produces the output as Sequence file
  - Makes use of byte arrays generated and merges small files to a Sequence File

This solution can be optimized further possibly by using CombineFileInputFormat API of Hadoop that helps in reducing the number of Mapper operations during the Map-Reduce job carried out to generate a Sequence file.

5. Experimental Setup and Evaluation

5.1 Hadoop Multi-Node Cluster Experimental Setup

A Hadoop multi-node cluster consisting of a NameNode and a DataNode is setup. Both the NameNode and DataNode are running on Mac OS X El Capitan with 8 GB RAM and 1.6 GHz Intel Dual-Core i5 Processor. The replication factor is set to 3. Default block size is used which is 64 MB. The NameNode is named "master" and the DataNode is named "slave". SSH is setup between the master and the slave machines.

5.2 Experimental Results:

The datasets for testing the solution were collected from the official website of Project Gutenberg which is a digital library of free e-books. 696 e-books were collected and the average size of each file was around 200 KB. Experiments were conducted to mainly to test and study 2 important parameters:

a) Name Node Metadata Size
b) Map Reduce Performance

NameNode Memory:

	Individual Small Files	**Sequence File**
Number of Files	696	696 files merged into a sequence file
Average size of files	Approximately 200 KB per file	148 MB
Number of blocks occupied	696 blocks of 64 MB	3 blocks of 64 MB
NameNode Memory Space Occupied	(150 bytes + 150 bytes) * 696 = **208800 bytes**	150 bytes + 150 bytes = **300 bytes**

It can be easily seen that the space occupied by the meta-data for a Sequence File is far less than the space occupied by the 696 individual Small Files. As discussed earlier, the NameNode needs 150 bytes to maintain file information mapped with its blocks and another 150 bytes to maintain the block allocation mapped to datanodes. Hence, having a huge number of small files is definitely going to have a performance hit on the NameNode's capability to support datanodes as is evident from the above table of results.

MapReduce Performance:

The performance of Map-Reduce operations is tested by running the Word-Count program on both of them, individual files and the Sequence File. The results are tabulated in the table below:

	Individual Small Files	**Sequence File**
Number of files	696	1
Number of Map Operations	696	1
Time Taken (in sec)	885	37

From the above table, we can see that the number of map operations required by the job is 696 mapper tasks which is a huge burden on the map-reduce job to consistently allocate jvms for map operations and perform file, block read operations. On the other hand, the Sequence file needed a single map operation. Also, the overall time taken to compute the Map-Reduce operation was far less in Sequence files when compared to the time taken to process Individual Small Files.

6. References

[1] An Improved Small File Processing Method for HDFS : Jilan Chen, Dan Wang, Lihua Fu, Wenbing Zhao @ JDCTA 2012
[2] A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files : Bo Dong, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li, Ying Li @ IEEE 2010
[3] Improving Metadata Management for Small Files in HDFS : Grant Mackey, Saba Sehrish, Jun Wang @ IEEE 2009
[4] Improving Hadoop Performance in Handling Small Files : Neethu Mohandas and Sabu M. Thampi @ Springer 2011
[5] Improving the Performance of Processing for Small Files in Hadoop: A Case Study of Weather Data Analytics : Guru Prasad M S, Nagesh H R , Deepthi M @ IJCSIT Vol5 2014
[6] Managing Small Size Files through Indexing in Extended Hadoop File System : K.P.Jayakar, Y.B.Gurav @ IJARCSMS 2014
[7] https://www.ncdc.noaa.gov/data-access
[8] http://www.gutenberg.org/