Er. Amrinder Singh[1], Dr. Gurjit Singh Bhathal[2]

[12]Department of Computer Science and Enginnering, Punjabi University, Patiala

---

Abstract – Big Data is becoming more popular and important part in processing larger amount of files for better analytics. This paper revolves around the area of handling small sized files in Hadoop and how it impacts the performance of the model. This also brings the solution for the problems related with handling of large number of small-sized files like memory problems of name-node server and reduced map performance in Hadoop. Proposed solution includes the sequence file architecture and is compared with the individual file type architecture using Gutenberg Small files dataset.

Keywords – NameNode, DataNode, HDFS, MapReduce, Hadoop

1. Introduction to HDFS

HDFS is an acronym for Hadoop Distributed File System. Apache Hadoop is open-source framework with software utilities. It is particularly used for processing of larger file sizes and storage in a cluster of distributed models. HDFS allows users to manipulate the data in a convenient way, which is stored in different cluster systems. The main benefit of HDFS is that it is highly reliable and tolerant to faults. Since it has a master-slave architecture, it has NameNode and DataNode.

Here, the master server is NameNode, which maintains the file system as well as file metadata. Also, it regulates file attributes such as permission of file, size, creation time and so on. NameNode always tracks the list of DataNodes, while handling the requests by the user. Besides this, it is responsible for operations, for instance, opening, closing or renaming the file and directories.

In the DataNode, data is being stored in the local system in the form of different blocks. DataNode is responsible for reporting about the list of blocks to the NameNode. It also handles read and write request of a user. Furthermore, it controls requests related to replication.

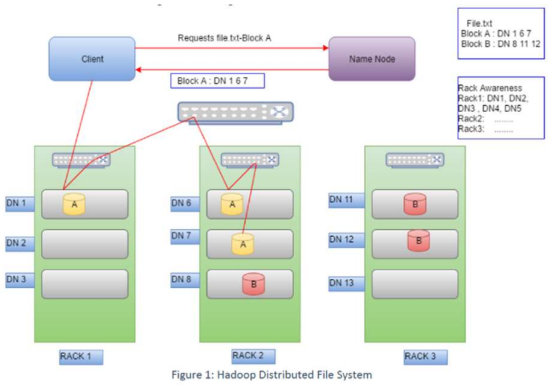

Figure 1.1 – HDFS

As shown in the aforementioned figure, whenever a request is generated by the user to access the file name file.txt, the request is sent to NameNode. NameNode holds all the information related to the blocks where file is placed in DataNode. In the given scenario, file.txt is present in Block A and B along with the redundant file block. Every Block gets duplicated or redundantly available in three different DataNodes in the form of distributed system. In this type of distributed system, the first copy of the file gets stored in rack number 1 and the remaining 2 copies of files in another rack. By doing so, tolerance of fault escalates during the event of updating /creation of file or failure of a system. Furthermore, because the data is present in the same file and can be accessible by DataNode, communication to and forth is much more convenient, thus avoiding the unnecessary bandwidth usage between the rack structure. It has been clearly seen in the given figure that NameNode holds the information of list of DataNodes in each and every rack.

1.2 About the Small Files

In the Hadoop File system, files that are lesser in block size than Hadoop Block size are considered as small files. It can be said that files lesser than 75% of Hadoop block size are to be considered as small files. By default, the block size of Hadoop is 64 megabytes, however, larger size blocks are in the range of 128 megabyte and 256 megabyte and so on.

1.2.1 What are the reasons behind small files??

Firstly, there are files which are small by default such as collection file images or text file data. Secondly, as the demand of data and information is rising, collection of files systems are in vogue, which can be accessed without any type of modifications where data is stored in such file systems. Lastly, when extended number of tasks are being performed unnecessarily, chunks of small data are generated.

1.3 Problems associated with Small Files

Hadoop is mainly designed to handle large data size files. Therefore, it does not perform well when we try to parse file having smaller file sizes. There are generally two problems which are linked with small files in HDFS:

1. **Memory Problem of NameNode server**: In Hadoop, every file is considered as object, which means each file and directory represent an individual object. The object is often approximately of 300 bytes of NameNode, in which 150 bytes are reserved for file name and its properties and the remaining 150 bytes are for DataNode information and block information. To illustrate this, 30million files having block size of 300 bytes of NameNode memory require approximately 720 Gigabytes of NameNode memory. As a result, NameNode needs 720Gb of data to be loaded from the disk, which will aggravate the performance and so as does the time. Other than this, since NameNode acts as Master node, it has to maintain the information regarding the block of

about the change in block information due to which consumption of network bandwidth will escalates. Lastly, if the number of blocks are more, NameNode will drain out the addressing capacity. Consequently, a large proportion of space will remain unutilized because of this.

2. **Reduced Map performance:**
   It is always a good practice to handle larger files having small quantity rather than handling larger quantity of small files as it requires less input/output tasks on a disk. Therefore, small files with larger quantity downgrades the performance of the Map. Another reason is that one block usually corelates with a file and the map jobs are segregated in such a way that every block assign with map process. So, if we have 10000 files with 10 Megabytes of data, each file will have a separate map task. Therefore, we will have 10000 map tasks along with map task configured in JVM. Instead of increasing the queue memory, we can process 800 files having 128 Megabytes of file size. By doing this, we will need to process 800 map tasks only and thus the performance will be improved. We have demonstrated the performance of different system having small files with larger in number and smaller number of large files. There are plethora of techniques to resolve the issue if Small Files. However, we will use sequence file technique and combine ot with CombineFileInputFormat technique to mitigate the number of map tasks.

2. Related Work

A study proposed by **Jilan et.al**.[1], depicts that in order to resolve the issue of small file, sorting of all the small sized files is required in a directory or folder and further compressing them into one larger one. This helps to shrink down the metadata present in NameNode. A single block can be created only if the total size of all the files is less or equivalent to the block's default size. In a case-scenario where file is distributed in more than one block, whole file data needs to be stored in the second block. The concept of global mapping is present that has the information of indexes for easy access of file system. Another similar type of work is proposed by **Dong et.al.** his software name is BlueSky, which is quite renowned courseware. Inn this, a directory like structure is created where information of similar files gets stored into single directory or folder along with the images having different screen-resolution. After that, all the files get amalgamated into one larger type of file. Also, the index is being created at the initial phase of the merged file. This method offers the technique of prefetching that fetches the files for further computation. There are two types of fetching have been discussed, which are as follows:

a. Local Index prefetching – In this Index file gets fetched and processed in the cache memory to avoid time consumption. It retrieves the data at much faster speed as there is no longer requirement to contact the NameNode.
b. Correlated File Prefetching- In this method, different corelated files get fetched beforehand in the cache as well.

In the [3] work, focus is put entirely on a particular area, in which each client has been given the certain permission to use uses the technique of har-baning which is a compression method of Hadoop API. It protects the job to be halted abruptly without stopping forcefully during the event of space drainage. By using this, number of tasks are set dynamically to gain optimization of data block size. The main aim is to merge small data files into larger one that could be utilized for map-reduced processing and thus reducing the space by the metadata at NameNode system. Also, it offers the concept of extended HDFS solution, where files are compressed and prefetched to dwindle the load at NameNode and thus improving the concept of file accessing. In our model, both the concepts of courseware software and weather data have been taken into account.

3. Problem Definition

The solution has been provided to solve the problem with Small Files in Hadoop Distributed File System. Also, it has been discussed what are the problem caused by Small Files.

3.1 About the Problems with Small Files

There are several problems associated and caused by Small files as it directly affects the performance of the NameNode address space. Firstly, it requires extra system resources to process the files. Furthermore, Small Files require extra computational time for Map-Reduce computation. This project particularly focused to provide the optimal solution to this problem of having Small Files and its computational cost. Other than this, our work will compare the evaluation of the NameNode memory issue through monitoring and utilizing the Small files before and after the issues being resolved. Also, the overall performance of the Map-Reduce will be accessed.

4. Proposed Solution Model

The solution involves collaborating the different small files in a particular directory to make a one large file for easy computation. We will further discuss how to handle Sequence File and how to build them to mitigate or eradicate the problem in HDFS. We have used Gutenberg datasets to evaluate the performance and test the efficacy of the model. Also, the performance factor of HDFS file system that is affected by different factors has also been discussed in the upcoming section.

4.1 Sequence File

A sequence of file is being generated to merge or collaborate the small files in a specific directory. The sequence file contains binary pairs of keys and values. The key is related to the filename, whereas the value relates to the content of the file. A sequence file can be segregated into multiple files and processed, which means sequence files can be pushed into multiple map tasks only if it is long enough. The sequence file contains the information of header along with one or more records. The format of header and record has been shown in figure below.

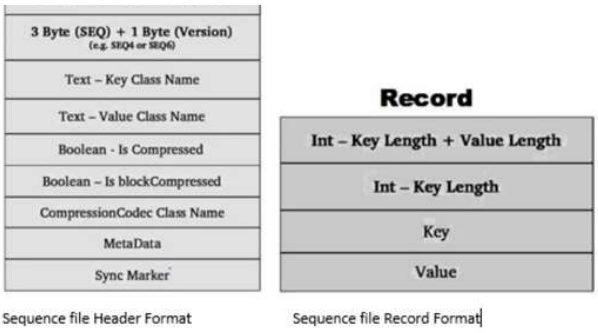

Figure 2 – Sequence File Header and Record Format

The SEQ behaves like an identifier for the sequence file, which has 3 bytes information along with the 1byte version number. There are 3 types of sequence file formats, which are as follows:

1. Uncompressed file format- In this, the pair of keys and values is stored without any alteration.
2. Record Compressed – In this, just the value is stored without alteration.
3. Block Compressed – Data is stored only when the threshold value is matched. If the threshold is reached to the value, pairs of key and value get compressed separately and sync marker is attached in between the blocks.

Here the flags are representing the format of the header. Record contains the length of key along with the record length which is followed by key length. The Sync Marker is usually attached after each 100 bytes.
The Sequence Files are the baseline of several other file types such as MapFile, SetFile and ArrayFile that have the potential to retrieve the key and value pairs. If random access of file is required, Map file can be used as it holds the information about the Index File to gather values based on key or Filename.

4.2 Implementation of HDFS

We will affiliate number of small file and form a sequence file in a specific directory. This can be done by making custom classes that are inherited from Hadoop API's which are as follows:

1. Custom Format Input Class through extension of FileInputFormat- Every content of file will be integrated as a single entity or record.
2. Custom Record Reader using RecordReader- Whole file will pe processed and a Byte of array will be generated by copying the data of a file.
3. Custom Mapper Class using extended Mapper - The name of the file will be stored as keys while the content of the file will be stored as values.
4. Sequence generation of File – This will create the sequence file as its output. Also, it will utilize the byte array and will club small file into a Sequence File.

By using CombineFileInputFormat API of Hadoop, we can achieve higher performance of the model as it helps us to task to generate the Sequence File.

5. Setup Experiment and its Evaluation

5.1 Multi-Node Hadoop Cluster Setup

Multiple-Node of Hadoop cluster contains both NameNode and DataNode is present. The NameNode and DataNode run on Mac OS X El Capitan along with 8GigaBytes of RAM. As far as the processer is concerned we have used Intel Dual-Core i5 1.6 GHz of speed. 3 values is set of replication factor and 64 Mb is the default block size. As discussed above, NameNode acts as 'Master' whereas 'Slave' is DataNode. Apart from this, Secured Shell or SSH connection has been established to communicate between the Master and Slave.

5.2 Experimental Results:

For the testing purpose, the datasets have been taken from Gutenberg, which is a collection of Electronic-books available free of cost. We have gathered the data of about 700 e-books from the website which is approximately 200 KB each. Also, we have tested and conducted an experiment to testify two following parameters:

1. Metadata size of NameNode
2. Performance of Map Reducer

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**

Figure 3 – Experiment Results in Table
One thing can be noted that metadata space requirement is quite less than several (696) individual small files. According to this, 150 Bytes is required for NameNode to hold the information regarding the files that are mapped with blocks. Other than this, additional 150 Bytes is needed for block allocation in DataNode. Consequently, it is vivid that large number of small sized files require additional resources that reduce the performance of the NameNode.

Performance of the Map Reducer:

We can calculate the performance of Map-Reduce is through word-count program on sequence file and every single file. The outcomes have been discussed though the following table:

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

Figure 4 – Performance with MapReduce
As per the aforementioned tabular data, it has been seen that Map operations for 696 Mapper operations are quite high, which means JVMs have been allocated each time it requires to compute tasks, file or block read like operations. Nonetheless, sequence file requires only single map operation. Moreover, the overall execution time is much more optimal and utilized properly for computing as compared with processing individual files.

CONCLUSION

Handling large number of small files is always a big problem in Hadoop which is addressed in this paper by proposing a solution that using sequence of files against the individual file

related with memory of the name-node server and provides reduced map performance. We have used Gutenberg dataset of small sized files and found that sequence of files based architecture provides better performance as compared with the individual files system with both metadata and map reduced functions. Time taken is reduced from 885msec to 37msec when sequence files method is compared with individual for mapreduce function and NameNode memory space occupied is also reduced from 208800 bytes to 300 bytes from individual to sequence file system.

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] Siddiqui, I.F., Qureshi, N.M.F., Chowdhry, B.S. *et al.* Pseudo-Cache-Based IoT Small Files Management Framework in HDFS Cluster. *Wireless Pers Commun* **113,** 1495–1522 (2020). https://doi.org/10.1007/s11277-020-07312-3
[8] Ahad M.A., Biswas R. (2019) Handling Small Size Files in Hadoop: Challenges, Opportunities, and Review. In: Nayak J., Abraham A., Krishna B., Chandra Sekhar G., Das A. (eds) Soft Computing in Data Analytics. Advances in Intelligent Systems and Computing, vol 758. Springer, Singapore. https://doi.org/10.1007/978-981-13-0514-6_62
[9] Gupta, L.: HDFS—Hadoop Distributed File System Architecture Tutorial (2015). http://howtodoinjava.com/big-data/hadoop/hdfs-hadoop-distributed-file-system-architecture-tutorial/
[10] Dong, B., Zheng, Q., Tian, F., Chao, K.-M., Ma, R., Anane, R.: An optimized approach for storing and accessing small files on cloud storage. J. Netw. Comput. Appli. **35**, 1847–1862 (2012)
[11] Vorapongkitipun, C., Nupairoj, N.: Improving performance of small-file accessing in hadoop. In: 2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 200–205. IEEE (2014)
[12] NCDC, " https://www.ncdc.noaa.gov/data-access"
[13] Gutenberg, " http://www.gutenberg.org/"