International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 07 | July -2017 www.irjet.net p-ISSN: 2395-0072

An Improved De-Duplication Technique for Small Files in Hadoop

Ishita Vaidya[1], Prof. Rajender Nath[2]

[1]M.Tech Student , Department of Computer Science and Applications , Kurukshetra University, Kurukshetra
[2] Professor, Department of Computer Science and Applications , Kurukshetra University, Kurukshetra

---

Abstract: *HDFS works as one of the core component of the Hadoop ecosystem, as it stores the large data sets in a master slave architecture using the commodity hardware. To store files in HDFS, many de-duplication techniques are given but the existing techniques do not merge the unique small files together to improve the Name node storage efficiency. To address the problem of small files an improved de-duplication technique is proposed in this paper that eliminates the files redundancy by using hash index, it allows to merge the unique small files only. The proposed technique is experimentally found better than the original HDFS technique in writing time and the overall storage efficiency.*

Keywords: **De-Duplication, Hadoop, Hadoop distributed file system, Small File Problem, Small File Storage.**

I. INTRODUCTION

Hadoop, an open source software framework is an important tool to manage and store big data using the commodity hardware. The large companies like Facebook, Netflix, Yahoo and Amazon uses Hadoop to manage the unstructured large data sets [1]. Hadoop uses two main components, the Map reduce framework and the Hadoop distributed file system as the basic tool for the big data analytics. The map reduce framework use the mappers and reducers to organize and process the data in the multiple computing nodes. The map-reduce and the HDFS work on the same data nodes. The distributed file system of Hadoop works as the database of the large files and stores the data in the data nodes with the metadata in the name node. HDFS works on the commodity hardware and thus to improve the fault tolerance of the system, the data is stored with 3 replicas on the same name node and another name node. This makes the HDFS as Fault Tolerant, scalable and low cost system to store the large data sets.

Small File Problem in HDFS : HDFS file system is designed to work on large datasets. Small files on the other hand imposes a heavy burden on the Hadoop distributed file system. The files less than the default block size are considered as the small files in HDFS [1]. The small files do not decrease the storage efficiency of the data node which can be explained as if the size of small file is 4MB and the data block size is 64MB , the 4MB file takes only the required space in the data block and the rest is free for the further files to be stored. So the small file problem in HDFS can be explained in the following manner.

(i) The meta data of the files are stored in the name node and when large number of small files are stored in the data node, the meta data of each small file takes a huge amount of space in the name node, which decreases the storage efficiency of the name node.

(ii) The map file needs a lot of seeks and hops to map the large number of small files instead of mapping small number of large files which actually concludes for a heavy traffic in the Hadoop distributed file system.

De-Duplication: Data De-duplication is a productive approach to avoid the redundant data in the big data technologies and thus reduce the network traffic by avoiding duplicate data over transmissions. The de-duplication strategy is used by using different hash functions such as MD5, SHA1, SHA256, SHA512, RipeMD 160, Tiger128, Tiger160, Whirlpool etc.

The workflow of the de-duplication process depends upon the type of data as the data is divided into chunks or the files are directly used to create the hash values and check for the duplicate data. The de- duplication phenomena had been used with Hadoop Distributed File System earlier to reduce the duplicate entries in the data nodes which can be described through some literature survey in the section followed.

In this paper the combination of the file merging and the de-duplication technique is used to increase the efficiency of the Hadoop distributed file system. The basic approach is to remove the duplicate entries of the same files and index the map of the duplicate files to the duplicate index of the proposed system. A file merging technique is also used to merge the small files together on the basis of the incoming files and then a merged file index is added using the start position and the end position of the small files to access the small files from the merged file.

The rest of the paper is organized as: Section 2 discusses the work that has been carried out in small files and de-duplication, section 3 describes the proposed technique with its architecture, section 4 describes the experiments and results carried out through the technique and finally section 5 concludes the paper with the future scope of this method.

II. RELATED WORK

An optimized approach was developed in 2012 by B. Dong et al. [2] to store and access the small files on Hadoop distributed file system. In this approach first of all, the cut off points between the large and the small files was given and then a file merging and prefetching scheme was introduced to handle the small files in HDFS. The files were merged together on the basis of their correlation phenomena where the files that belonged to a large logical structure were merged together and then a file prefecthing and caching mechanism was also added to the merged files.

In 2013 Chandrasekar et al. [3] introduced Extended Hadoop distributed system that merged the small files on the client side such that the client specified the name of the small files to be merged. The phenomena that the files were not split into two blocks was also considered. In this phenomena, if the file size was larger than the remaining data node size , then the file was sent to the next data node instead of splitting the file into two parts. In this approach the user specified the name of the small file to fetch the file from the merged file. A remote procedure call was used to read a file from the Data Node and a prefetching and caching mechanism was also used. This approach improved the storage efficiency of the Name Node and access efficiency of Data Node.

In 2014 Guru Prasad et al.[4] extended the basic Hadoop model and named it as the optimized approach which consist of the merge model. In this approach the files were directly merged from the output of the Map Reduce phase by combining them into large files. The large files were then divided into blocks according to block size. The data blocks were then again processed by Map Reduce. The approach decreased the time taken to move the file into HDFS and increased the Name Node storage efficiency as compared to original HDFS.

An Improved HDFS was proposed in 2016 by L. Changtong [5] that added a processing layer in the basic Hadoop architecture such that the files that were considered as the small files were merged in the file merging unit with the Temporary Index. The Temporary Index was created using an incremental offset with the start position considering the length of the small files. The small files and the temporary index were merged together using an append writing operation. The Improved HDFS approach thus decreased the Name node memory usage and the access time per 1 MB than that of HAR and original HDFS but the writing time was bit more than HAR and quiet less than original HDFS.

Yonghuo et al.[6] presented a small file processing middleware that used the concept of temporal continuity to merge the small files in the merged file. In this phenomena the a time unit was set such that the files that came within that period was considered to be merged together. An index structure was introduced with the merged files that used the trie-tree approach to retrieve the small files. In addition to the file merging and indexing structure , a caching structure was also added to improve the access time of the approach. The caching and prefetching structure actually fetched the correlated files in order to increase the retrieval speed.

A De Du system [7] was developed in 2013 that used HDFS storage system with HBase employed in order to implement the indexing scheme for the duplicate files. In this approach the new incoming file was considered as the source data and was saved inside the storage system and the hash value was stored in the HBase index of the system. In the system, whenever a duplicate file entered , the system did not upload the file and instead update the user with a link to the same file that was already existing in the system. The system increased the writing efficiency and removed the overall redundancy thus improving the storage efficiency. Although this approach improved the storage efficiency but unique small files were still uploaded to HDFS, that decreased the name node storage efficiency which was major drawback for the approach.

A dynamic de-duplication decision making system was introduced in 2014 [8], which used the de-duplication strategy to improve the utilization space in the data centres that used the Hadoop distributed File System as the storage systems. The proposed system deleted the useless duplicates in order to improve the storage efficiency of the data centres . This approach used a 2-tier storage architecture that is the pre filter and the post filter. The pr filter worked on the file level while the post filter worked on the block level. The pre filter worked on the client side as it checked the file before uploading , If the file already existed in the system , the redundant file was not uploaded. In the post filter of the proposed approach the redundant data blocks were removed when the already existing data was updated again thus creating the redundant blocks. The proposed approach increased the overall capacity of the system thus improving the utilization space of the data centres for big data management.

A bucket based data de-duplication technique was introduced in 2016 [9] in which a fixed size chunks were made from data and then these chunks of data were sent to the MD5 hash map further where the hash values of the fixed size chunks were created. The hash values of these chunks were compared with data from the map reduce algorithm such that the duplicate entries were not stored in the Hadoop Distributed File System and the proposed approach approved a high reduction in data size of storage system with less amount of time consumed for hashing and maintaining the chunk size.

III. PROPOSED DE-DUPLICATION TECHNIQUE

To overcome the issue of small file problem in HDFS as mentioned above, an improved technique of file merging and data de-duplication is combined together. In this approach a pre-processing unit is added to the original HDFS structure which checks for small files, duplicate entries and then merge the unique small file. The pre-processing unit further contains the file judging unit, the de-duplication unit and the file merging unit which are discussed below as three subparts of our proposed approach:

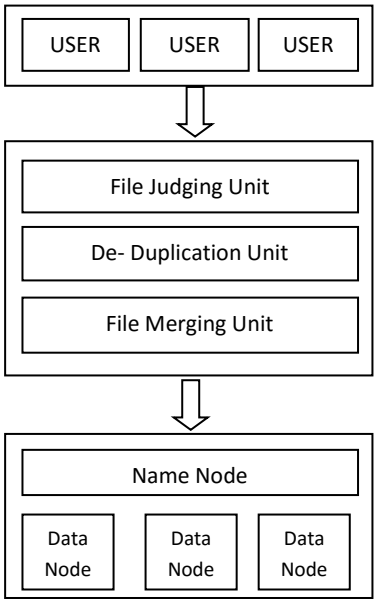

Figure 1. Architecture of The De-Duplication Technique

The File Judging unit: This unit judges the incoming file weather it is small or large. The files with size same as that of block size takes roughly same amount of memory on the name node and thus can be considered as large files and the file size less than block size is considered as small files. So the small files are send for further processing into the hash value calculator in the de-duplication unit and the large files are directly send to the HDFS storage system.

The De-Duplication Unit: It calculates the hash values for the incoming small files using MD5 hash function and then compares the new hash value with already existing hash value. If the hash value already exists, a map is created with a link to the already existing file and the count is incremented by 1. If the hash does not exist then the new file is added to the file merging unit and the new hash value is send to the duplicate index in the data node where the files are to be saved after merging. The working steps of the de-duplication unit are:

Pseudo Code for De-duplication:

```
//Input: Small Files
//Output: Merged file with duplicate index and
//merged index
Step 1: Read the small Files
Step 2: Calculate Hash value
Step 3: If (calculated hash ! = existing hash)
        { Store Hash in duplicate index &&
          Send the file to merging unit &&
          Store the path in merged index}
   Else If (calculated index == existing index)
          { Map the file from merged index &&
            Link the file map with hash value}
Step 4:Send the files to the file merging unit.
```

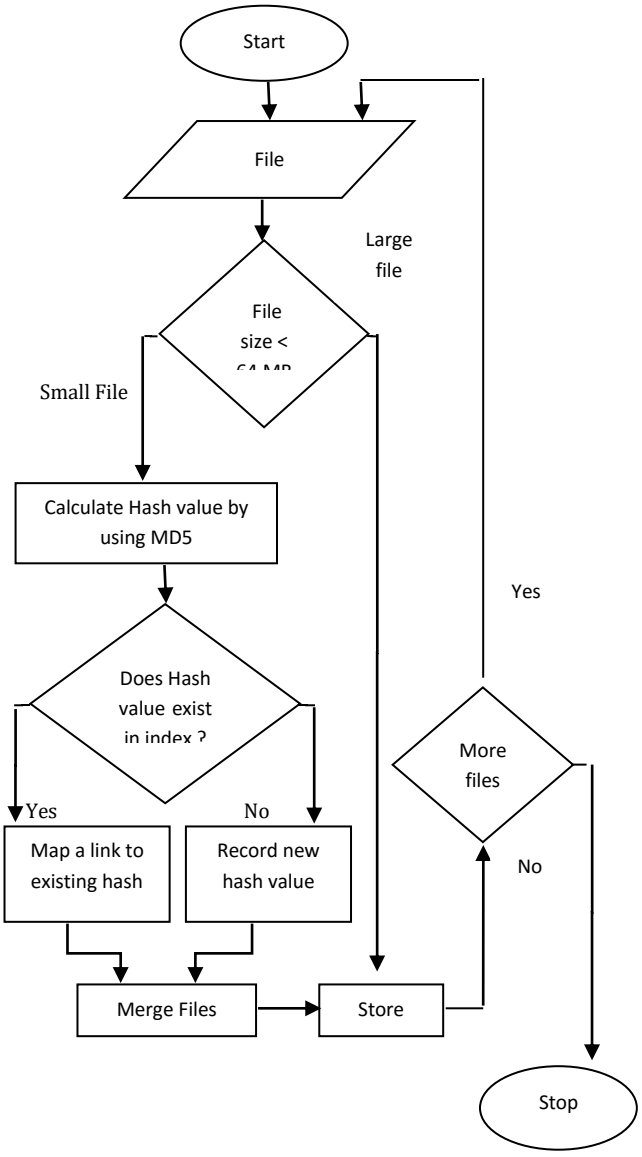

Figure 3 . Flowchart for the De-Duplication Technique

The flowchart in figure 3 explains the processing structure of the de-duplication unit .

The File Merging Unit: The file merging unit merges the small files together by saving the files in the incoming order with the start and the end position. The files in this unit when reaches the block size, then the files are merged together and a merged as well as duplicate index are added to the data node with the merged file.

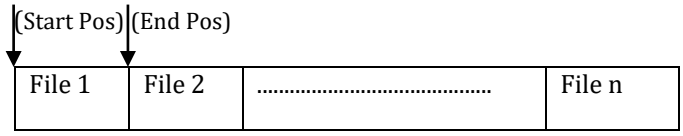

Figure 2 . Merged Index for the merged Small Files

The merged index in Figure 2 merged the small files using the start position and the end position of the small files. The end position of one file is considered as the start position of the next file thus storing the n number of files in the same format.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

The proposed technique was implemented in Java language. The dataset is generated by a random file creator and the number of files included are 500, 1000, 2000, 4000 and 6000 with 30%, 36%, 68%, 48% and 57% duplicate files respectively. The size of small files are 1KB - 10KB and large files are also simulated. The writing time and storage efficiency of Data Node in the proposed approach is compared with that of original HDFS . The index storage of Name node and the combined index storage of the proposed approach is also compared with that of the default HDFS architecture.

The parameters used to analyze the performance of the proposed approach are: a) Writing Time of the Files, b) Data Node storage efficiency, c) Name Node storage efficiency, d) Combined Meta Data Size.

a) Writing Time of Files:

Writing time is the time required to upload the small files into the HDFS system.

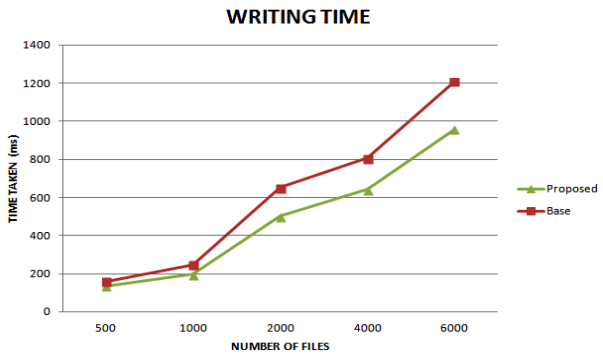

Figure 4. Writing Time of Files

Figure 4 shows the comparative graph between the original HDFS and the proposed approach. As the duplicate files are not written again, the writing time of the files is decreased and the efficiency of the proposed approach is found better than the HDFS considerably.

b) Storage Efficiency:

Storage efficiency of Data Node is referred to as the combined or the merged file stored in the data node. The overall storage efficiency is improved using the de-duplication technique in the proposed approach.

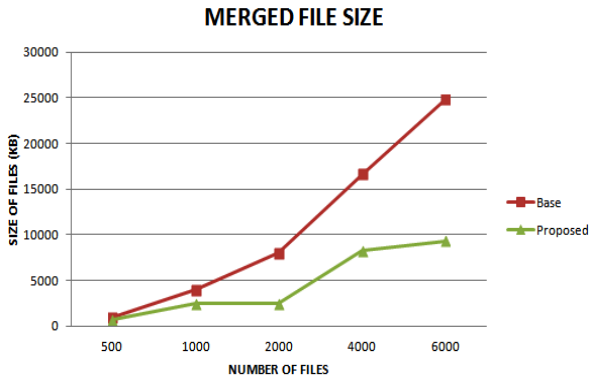

Figure 5. Storage Efficiency of Data Node

The graph in Figure 5 shows the comparative analysis of the storage efficiency of the data node for original HDFS and the proposed approach. As the number of files increases, the storage space for combined file also increases linearly but the storage space for combined file in the proposed approach is less as compared to original HDFS due to the de-duplication phenomena.

c) Name Node Storage Efficiency

The Name Node storage efficiency is evaluated from the meta data stored in the Name Node for the combined file. The Name Node storage efficiency is increased with the merging of small files and avoiding meta data for redundant files.

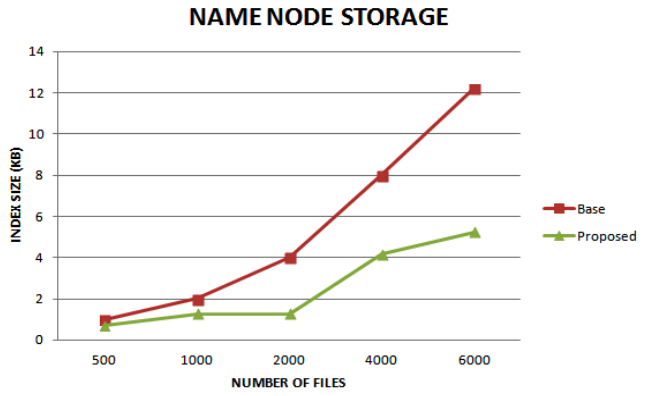

Figure 6. Name Node Storage Efficiency

The graph in the Figure 6 gives an analysis on the Name Node storage efficiency between the two approaches such that in original HDFS the Name Node storage is on a linear growth that makes the Name Node as a bottleneck in the original HDFS architecture.

d) Combined Meta Data Size

The combined meta data is the overall index size as two more indexes are added in the proposed approach. A duplicate index is added to the proposed approach with already existing Name Node meta data storage and the combined file size. The overall index size of the proposed approach is still less than that of the original HDFS.

Merged Index Size: The merged index size is the size of the index of the merged small files. The merged index with the de-duplication concept takes less amount of space as compared to the one without the de-duplication phenomena.

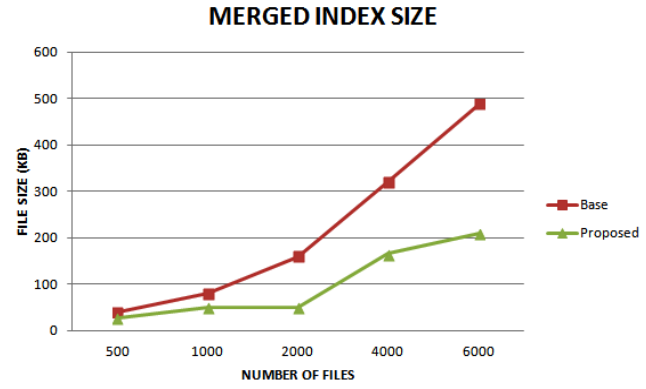

Figure 7: Merged Index Size

The graph in Figure 7 explains the difference between the merged files with de-duplication and the merged files without de-duplication phenomena.

Duplicate Index Size: Duplicate Index is the hash values stored to check the redundant data and thus remove the data duplicacy in the file storage system.

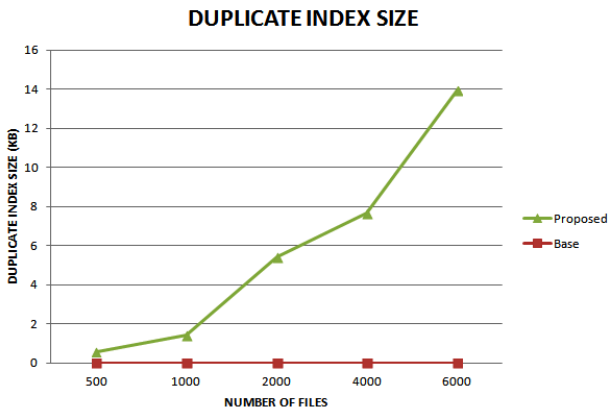

Figure 8 : Duplicate Index Size

The graph in the figure 8 only shows the increase in the duplicate file index that is the number of hash value with an increase in the number of files.

Name Node Storage: The name node storage thus defined as the amount of meta data stored in the name node. The Name Node storage in the proposed approach is comparatively very low.

The graph Figure 6 shows the comparison between the Name Node index storage in the original HDFS and the de-duplicate technique.

Combined Index Size: The combined index is the sum of all the index and the Name Node storage. The combined index of the proposed approach take less amount of space as compared to original HDFS.

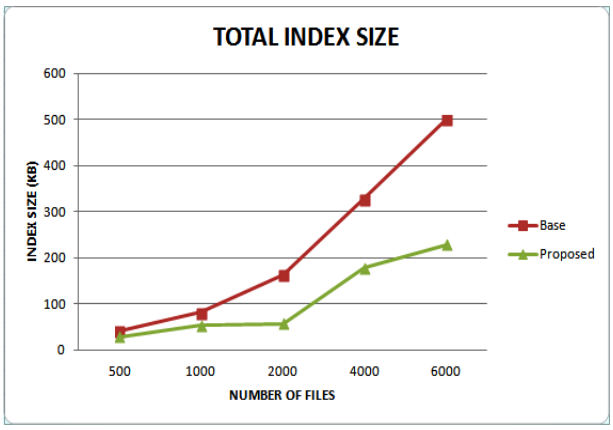

Figure 9: Combined Index Size

Figure 9 analyzes the overall combined index of the approaches and thus satisfies that the proposed approach takes less amount of space as compared to original HDFS.

V. CONCLUSION

This paper has proposed an improved de-duplication technique by adding a pre-processing unit, which checks for small files duplicate entries and then merges unique files only. The proposed technique has been implemented in Java and evaluated against the original Hadoop technique. Experimentally, It has been found that the writing time, the data node storage efficiency, the Name Node storage efficiency and the combined meta data index efficiency have improved considerably as compared to Hadoop technique.

VI. REFRENCES

[1] S. Bende and R. Shedge, "Dealing with Small Files Problem in Hadoop Distributed File System," in Elsevier Procedia 7th International Conference on Communication, Computing and Virtualization, Maharshtra, 2016, pp. 1001-1012.

[2] B.Dong et al., "An optimized approach for storing and accessing small files in cloud storage," Journal of Network and Computer Applications , Elsevier, vol. 35, pp. 1847-1862, July 2012.

[3] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, Prabhavaty B, and C Babu, "A Novel Indexing Scheme foe Efficient Handling of Small Files in Hadoop Distributed File System," in Proceedings of IEEE International Conference on Computer Communication and Informatics (ICCCI-2013), Coimbatore, 2013, pp. 1-8.

[4] G. Prasad M S, Nagesh H R, and Deepthi M, "Improving the performance of processing for small fies in Hadoop:A Case study of Weather Data Analytics," International Journal of Computer Science and Information Technologies, vol. 5, no. 5, pp. 6436-6439, 2014.

[5] L. Changtong, "An Improved HDFS for Small Files," in International Conference in Advance Communication and Technology, Wuhan , China, 2016, pp. 474-477.

[6] Y.Huo, Z.Whang, XX.Zhang, W.Li, and Z.Cheng, "SFS:A Massive small file processing middleware in Hadoop," in The 18th Asia-Pacific Network Operations and Management Symposium(APNOMS)2016, China, 206, pp. 1-4.

[7] Z Sun, J Shen, and J Young, "A novel approach to data deduplication over the engineering-oriented cloud systems," University of Wollongong Research Online, pp. 45-57, 2013.

[8] R-S Chang, C-S Liao, K-Z Fan, and C-M Wu, "Dynamic Deduplication Decision in a Hadoop Distributed File System," International Journal of Distributed Sensor Networks, pp. 1-14, April 2014.

[9] N Kumar, R. Rawat, and S. C. Jain, "Bucket Based Data Deduplication Technique," in 5th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), Noida, 2016, pp. 267-271.