An Efficient Approach for Merging Small Files in HDFS Storage and Accessing Small Files by Using Name Node Method in Big Data

J. Sirisha[1], K. Swathi[2], P. Rama Koteswara Rao[3]

[1]Sr. Assistant Professor, Dept. of CSE, PVP Siddhartha Institute of Technology, sirisha.j@pvpsiddhartha.ac.in
[2]Professor, Dept. of CSE, NRI Institute of Technology, kswathi@nriit.edu.in
[3]Professor, Dept. of ECE, NRI Institute of Technology, prkr74@gmail.com

Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 28 April 2021

Abstract

Big Data is one of the most requested techniques in the modern world of software development. In Big Data, the treatment of distributed files is performed by the open-source software framework called Hadoop on the product hardware cluster. For the storage of Big Data, the Framework is considered the most powerful. The HDFS Name Node component is used to store all sorts of files, folders, and blocks or metadata. The HDFS is specially designed to handle large files, but this framework will not properly handle a large number of small files. Proposed systems introduce that how the Name Node memory overheads the data storage reduces the storage of the huge number of small-size files in the HDFS. This approach will be very helpful in understanding the memory consumption and workload in the Name Node reduces the distributed file system called Hadoop.

Keywords: Bigdata, Cluster, HDFS, Name Node, Merge File.

1 Introduction

Currently, cloud computing has become the most important computing system in web design computing and growing around the globe. Distributed storage is the fundamental piece of cloud computing as it gives the information storage access of enormous informational indexes for end clients at whatever point and any place needed in the conveyed record system [1]. Replication of information is an integral part of distributed storage to the accessibility of information, limiting the inactivity of access to information, and balancing the burden on a few workers simultaneously. Subsequent implementation of the framework expanded considerably. Hadoop is open-source programming for the gigantic informational indexes to encourage ability, supervise, study, and access utility in frames transmitted through a huge number of frames. The Hadoop instrument began with Google, Facebook, Twitter, and so forth to store and process a huge amount of information. Hadoop consists mainly of the following two sections: Map Reduce and Hadoop Distributed File System (HDFS) [2].

HDFS is a conveyed recording frame that deals with the preparation of the document through a gigantic number of machine inappropriate frames with the least equipment needed for the calculation. HDFS mainly supports composing a single reading of many kinds of responsibility on access to information streaming and Big Data collections. It provides the information replication block of information for saving against equipment disappointments. HDFS is expert/slave engineering and primarily incorporates the following three parts: NameNode, DataNode, and Clients [3]. HDFS contains a unique NameNode which, for the most part, is responsible for the supervision of the namespace for registration frames with deference block substitution system and information node. It stores the metadata or document namespace in Dynamic Random Access Memory (DRAM) for quick access and keeps the duplicate of the record Framework namespace (FsImage) on the plate for information recuperation.

Any adjustment or updating to the record framework namespace are put away in the EDitLog and opportune converge with the FsImage so that put away duplicate of the document framework namespace consistently be state-of-the-art. NameNode contains record metadata; file directories, document content squares which incorporate update time, file length, block size, property, replication, and access data. NameNode contains record metadata; file folders, document content squares that incorporate update time, file length, block size, ownership, replication, and access data. DataNodes store the information squares document and are responsible for serving reading writing to clients. The information nodes perform the procedure on the squares based on the direction given by the NameNode [4]. HDFS primarily offers replication for deficiency lenient and saves against hub disappointment. HDFS for the most part concedes from another record framework in the accompanying cases: 1) Fault-open minded 2) Low expense equipment 3) Increased Throughput MapReduce is another component that bolsters the equal producing and preparing enormous informational collections. It deals primarily with similarity, business planning and deals accordingly with disappointments. Guiding the work in MapReduce outline measures the key-estimate pair to produce intermediate value key pairs. Reducing work in MapReduce outline coordinates transient qualities related to a similarly moderate key [5].

2. Related Works

As HDFS employees grow up, the HDFS traps are also found. Among these, there is the horrific demonstration when the HDFS processes practically nothing and as regularly as conceivable coordinated reports. HDFS is expected to plan a colossal data transfer instead of moving countless reports [6]. Shvhko of Yahoo! depicts the organized arrangement and usage of HDFS. Understood that assessment of applications would typically make gigantic recordings was wrong and that new applications for HDFS should store incalculable reports more discrete. Depicted as there is only a solitary NameNode in Hadoop which keeps all the metadata in essential memory, a gigantic number of little records produce an immense impact on metadata execution of HDFS [7][8]. Investigation on little record issue of HDFS has pulled in the colossal thought and it is acknowledged that three issues ought to be gotten comfortable a more appropriate way. The essential question is the distinctive verification of 'how little will be near anything'. Reports of less than 16MB are few files, no control or hobby was required of the same. The problem that follows is collecting small documents. Small documents in two kinds: (i) reports which are pieces of a huge authentic disk (ii) archives which are practically nothing.

The third problem concerns the responses to the minor archiving problem. Action plans are divided into two categories: overall plans and exceptional plans [9]. Hadoop Archive (HAR), SequenceFile, and MapFile are typical general responses for smoothing small folders for Hadoop used by various reviewers. HAR is an archive chronicle desk that packages various small records into huge HDFS blocks so that the main reports can be obtained life and successfully equivalent. It contains Metadata (in the form of _index and _masterindex reports) and data. Partial records contain the substance of the reports that are essential for the records [10]. A SequenceFile file is a level record and gives a coherent data configuration to resemble key sets. It uses the name of the report as the key and records the substance as the value, and supports compaction and decompression at the cradle or square. Small records can be put into a single action plan report that is prepared using MapReduce chipping to the progress archive.

A MapFile is a kind of SequenceFile orchestrated with a recording to question the movement by key. It comprises two reports, a data folder, and a more modest archive. All organized key sets are accounted for in the data record and the key area information is processed in the list archive. MapFile does not search for the report in its entirety by searching for a specific key. The reviewers grouped the page reports into one extraordinary document and created a folder for each book to store the book pages for the e-libraries. No plan was in place to improve passage efficiency. Analyzed a phenomenal game plan which solidified the little archives into a colossal one and manufactured a hash list for every single record which stores the little data of Geographic Information System on HDFS. The connection record meeting and some variations in the data were taken into consideration.

3. Small File in HDFS

HDFS is a distributed file system that manages file processing among a large number of machines in distributed systems for computing with minimum hardware requirements. The HDFS capacity is progressively reduced when it begins to manage the storage and access functionality of the number of messages of small files. A massive number of small files is now present in current systems like climatology, energy, astronomy, business, biology, e-Learning, and e-Library. Therefore, the critical problem in HDFS is storage management, replication, and access to small files. Handling a massive amount of small files and storage is the heavy burden that is created at NameNode. The access efficiency of small files is degraded and the memory usage of the name node is increased. Metadata handling performance is also influenced by the number of small files in HDFS. For small files, HDFS appears as a bottleneck for processing metadata services.

A. NameNode: This component is primarily used to store and manage the metadata of data files that are stored in the DataNode in the HDFS. It is also used to maintain the relationship between the NameNode and the DataNode and it is like maintaining the heartbeat. It also maintains access to and control over the data file storage. To replicate, NameNode retains the metadata. NameNode replication factor is 3 by default. For each object, the 150-byte memory size is used by NameNode to store metadata.

B. DataNode: It is used to process the HDFS cluster that has freely available data blocks to store a large number of data files. In the HDFS cluster, the replication factor may be stored by the DataNode in the different data blocks available. The NameNode component is supported by the DataNode to maintain the link between them. HDFS is specifically designed for handling large files, but the framework does not properly handle a large number of small files. So to solve the storage problem of the huge number of small-size files with metadata creation and management, we proposed an approach for each small size of files in the NameNode.

3.1 Proposed Approaches to Handle a Large Number of Small Size Files

In HDFS, the distributed file system called Hadoop is used to solve the issue of managing a large number of small files. Our proposed approach helps improve the efficiency of accessing a large number of small files and the storage capacity. The primary objective of this approach is to reduce the operating overhead associated with creating metadata in NameNode and to back up metadata by reducing memory consumption. The architecture structure proposed distributed file system called Hadoop architecture as follows. This framework is made up of the following elements:

3.1.1 HDFS client

This component is used for playback and writing of two types of files, Big Size files and a large number of small files. If the size of the file is greater than 128 MB then it is called a Big data and if it is lesser than 128 MB, it is called a small size file.

File Writing Process: This process includes identifying the file type, identifying the file size, and merging the files. To determine the file type, the file type identifier unit is used. To identify the file size, the file size identifier unit is used. If the size of the file is found smaller, the file merging unit is used to merge the small size files into the DataMerge files, and for that Index, Merge file is created.

File Reading Process: This process is used for reading files available in HDFS. As a first step, the file read unit checks that the file size is large or small.

3.2 Optimize Storage of Small Files with NameNode in HDFS

To reduce overhead and memory consumption in the HDFS NameName, the optimized strategy is used to store small files. The policy we used in this proposal is divided into two sections called File Processing Unit and File Fusion Unit.

Processing of files: In this unit, the entry is taken from the user. This unit judges the file size that we populate as input as either small file size or large file size. If the size of the file is found greater than 128 MB, then it is considered a Big data and if it is smaller than 128 MB, it is a Small-size file. The large file size is stored directly in the HDFS while the small files are sent to the file fusion unit.

File Fusion: In this unit, receives small files that are sent from the File Processing Unit. After receiving the small files, the files are merged in the DataMerge files. Now a merge index file is created. After completing the merge process, the DataMerge files and the merge index file are routed to the HDFS for storage as illustrated in Figure 1.

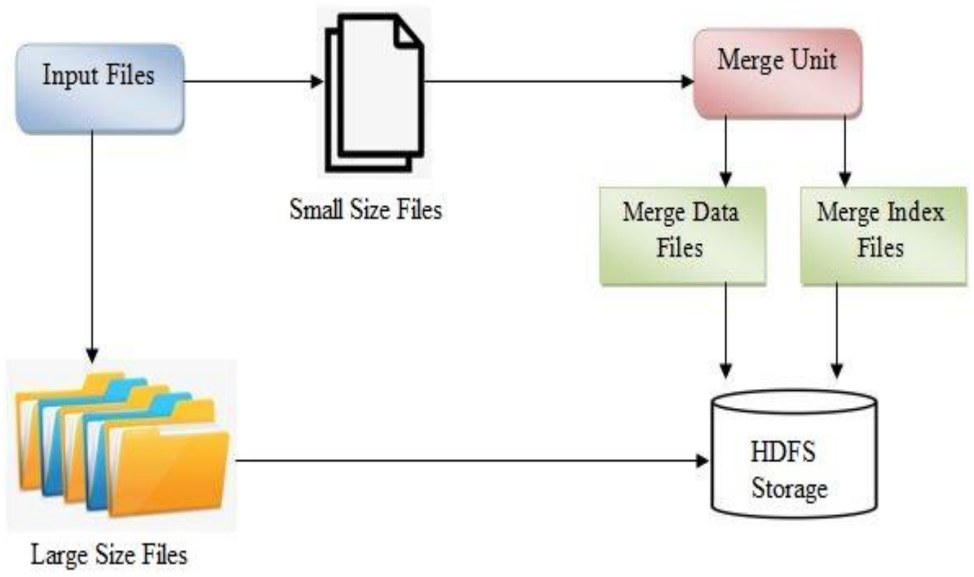

Figure 1: Method for Proposed Merge system using NameNode HDFS Storage

The Small File Fusion Algorithm Fusion calculations have the idea of classifying a considerable number of small records into huge documents. It reduces the length of metadata storage for a large number of small documents. Access to information control and inertia can now be improved and access time saved where necessary. Here the recordings are grouped into four kinds they are: Serious reading, Focused composing, Intensive reading and composing, and Lean reading and composing. The idea behind all this is to group all the little documents into four kinds and to bring together the documents that have a place with explicit classifications and divide the documents into squares of information. At this stage, the convergence of the third and second classes into the fourth classification and after mixing the records of the first class into these squares. If a few squares are not in all cases consolidate all records has a place with the first classification. The basic thought is that few records are converged in huge documents and designated storage between block times.

Pseudocode of Small File Merging Algorithm

```
BlockSize_hdfs = configured (block size of storage)
max_Size = BlockSize_hdfs * th-Value /*threshold value analyse*/
Bytes_merge = 0
File_merge = Array_List ()
for (File: directory) {
Length_file = length.File ()
if (Length_file <= max_Size) {
Bytes_merge = Bytes_merge + Length_file
 File_merge.add(File)
 }
}
```

4 Results and Discussion

In HDFS, each file, directory, and box are counted as a single object. For each object, metadata is created into memory by NameNode. To store metadata, 150 bytes are occupied by NameNode in memory. The data files provided by the users are divided into different data blocks of 128 MB by the HDFS and the namespaces for each block are stored by the NameNode shown in Table 1. The storage calculation is resolved by three storage usage settings in the form of 10K, 20K, and 30K counts of small files are shown in Figure 2. Using the HDFS approach with the proposed approach called Novel Hadoop Distributed File System (NHDFS), the file is stored in MB size by the node. CPU and energy consumption is less in the proposed system when compared with the existing system are shown in Figures 3 and 4.

Table 1 Memory used by HDFS and Proposed HDFS

File	Small Files	File Size	Memory used by HDFS	Memory used by Proposed HDFS
1.	20000	1GB	1.53MB	0.00115 MB
2.	30000	1GB	2.46MB	0.00116 MB
3.	40000	1GB	3.85MB	0.00118 MB

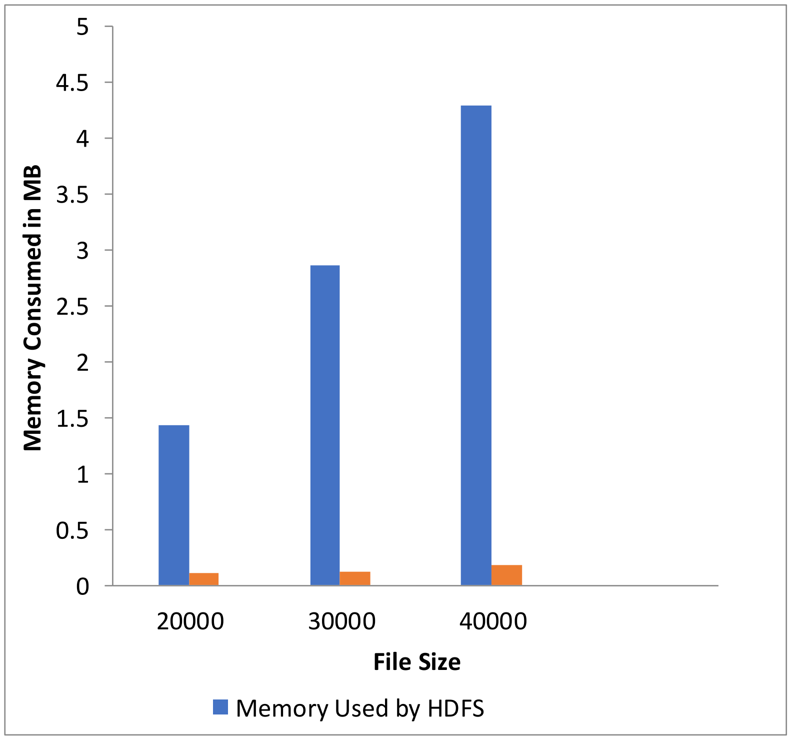

Figure 2: Memory Consumption by HDFS and Proposed HDFS

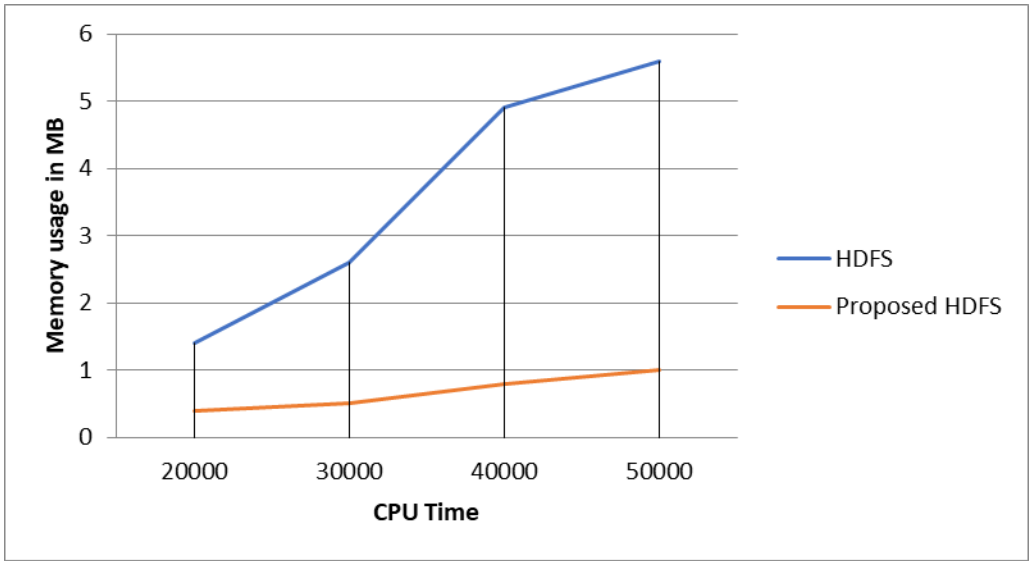

Figure 3. CPU Time Vs Memory consumption

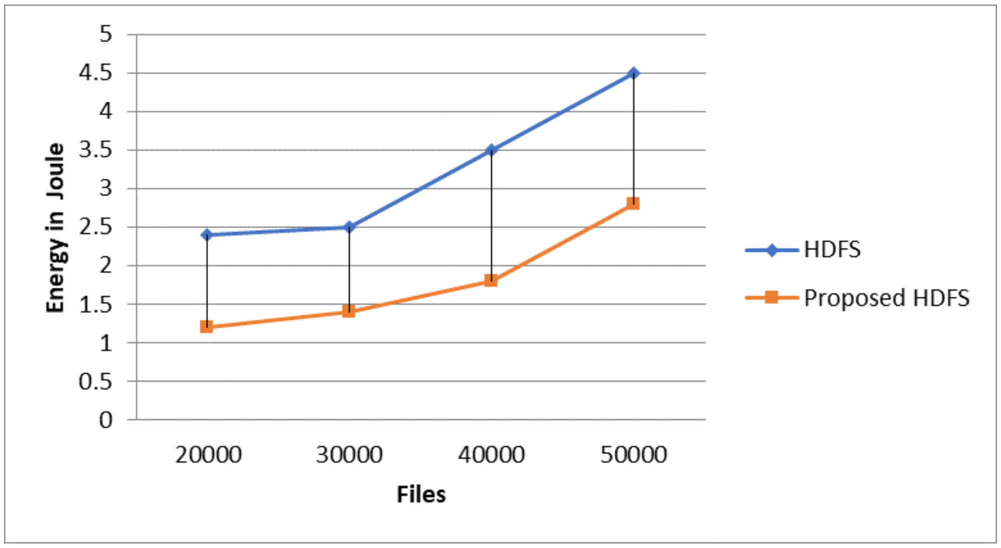

Figure 4. Energy Consumption

5. Conclusion

The Hadoop distributed system running in a cluster environment is an open-source framework. To work with Big Data files, HDFS is used due to the way it manages to store Big Data files effectively. However, HDFS does not efficiently manage the storage of small files in very large quantities. It also creates a problem of managing the Big Data and the creation of Big Data for every single small size files which because of the main reasons for the lack of consumption in memory. So the processing and storage of small files in great numbers using Hadoop become more problematic. On Hadoop, the new and optimizing HDFS framework is featured in this article. This NOHDFS frame is specially designed to provide the solution to store a large number of small files and also the HDFS memory consumption is optimized.

References:

1. Zeebaree, S. R., Shukur, H. M., Haji, L. M., Zebari, R. R., Jacksi, K., & Abas, S. M. (2020). Characteristics and analysis of hadoop distributed systems. Technology Reports of Kansai University, 62(4), 1555-1564.
2. Cui, Y., Kara, S., & Chan, K. C. (2020). Manufacturing big data ecosystem: A systematic literature review. Robotics and computer-integrated Manufacturing, 62, 101861.
3. Scolati, R., Fronza, I., El Ioini, N., Samir, A., & Pahl, C. (2019). A Containerized Big Data Streaming Architecture for Edge Cloud Computing on Clustered Single-board Devices. In Closer (pp. 68-80).
4. Chandrasekar, S., Dakshinamurthy, R., Seshakumar, P. G., Prabavathy, B., & Babu, C. (2013, January). A novel indexing scheme for efficient handling of small files in hadoop distributed file system. In 2013 International Conference on Computer Communication and Informatics (pp. 1-8). IEEE.
5. Richter, S., Quiané-Ruiz, J. A., Schuh, S., & Dittrich, J. (2014). Towards zero-overhead static and adaptive indexing in Hadoop. The VLDB journal, 23(3), 469-494.
6. Jiang, L., Li, B., and Song, M. 2010. The optimization of hdfs based on small files. In 3rd IEEE International Conference on Broadband Network and Multimedia Technology, IC-BNMT. 912-915.
7. Mackey, G., Sehrish, S., and Wang, J. Aug 31- Sep 4, 2009. Improving metadata management for small files in HDFS. In proceedings of IEEE International Conference on Cluster Computing and Workshops. New Orleans, USA. 1-4.
8. Min, L., and Yokata, H. 2010. Comapring hadoop and fat-btree based access method for small file I/o applications. Web-age information management, Lecture notes in computer science. Vol. 6184, Springer. 182-193.

9. Shen, C., Lu, W., Wu, J., and Wei, B. 2010. A digital library architecture supporting massive small files and efficient replica maintenance. In Proceedings of the 10th annual joint conference on digital libraries. ACM. 391-2.
10. Liu, X., Han, J., Zhong, Y., Han, C., and He, X. 2009. Implementing webgis on hadoop: a case study of improving small file i/o performance on HDFS. In IEEE international conference on cluster computing and workshops, CLUSTER'09. 1-8.