Optimization of Small Sized File Access Efficiency in Hadoop Distributed File System by Integrating Virtual File System Layer

Neeta Alange[1]
Research Scholar
Department of Computer Science and Engineering
Koneru Lakshmaiah Education Foundation
KL Deemed to be University, Vaddeswaram, AP, India

Anjali Mathur[2]
Associate Professor
Department of Computer Science and Engineering
Koneru Lakshmaiah Education Foundation
KL Deemed to be University, Vaddeswaram, AP, India

***Abstract*—Storage for large datasets, handling data in different formats and data getting generated with high speed are the major highlights of the Hadoop because of which the Hadoop got invented. Hadoop is the solution for the big data problems as discussed above. In order to give the improved solution (in terms of access efficiency and time) for small sized files, this solution is proposed. A novel approach called VFS-HDFS architecture is designed in which the focus is on optimization of small sized files access problems with significant development compared with the existing solutions i.e. HDFS sequence files, HAR, NHAR. In the proposed work a Virtual file system layer has been added as a wrapper over the top of existing HDFS architecture. However, the research work is carried out without altering the existing HFDS architecture. In this paper drawbacks of existing techniques i.e. Flat File Technique and Table Chain Technique which are implemented in HDFS HAR, NHAR, sequence file is overcome by using Bucket Chain Technique. The files to merge in a single bucket are selected using ensemble classifier which is a combination of different classifiers. Combination of multiple classifiers gives the better accurate results. Using this proposed system, better results are obtained compared with the existing system in terms of access efficiency of small sized files in HDFS.**

***Keywords*—HDFS; Small sizes files; virtual file system; bucket chain; ensemble classifiers; text classification**

I. Introduction

This Hadoop Distributed File System (HDFS) is a core component of Hadoop which works at storage layer of Hadoop. Advantages of HDFS includes varied data sources, availability, scalable, cost effective, low network traffic, ease of use, performance, high throughput, compatibility, fault-tolerant, open source and multiple language support etc. Apart from these multiple advantages, HDFS have limitations too such as handling small files, slow processing speed, support for batch processing, no real time data processing, no Delta Iteration, latency, not easy to use, security etc. Storing and processing of large number of small sized files in HDFS is a major problem. Hadoop Archives provides an efficient way to handle with small files [20].

Hadoop working is best with respect to big data files; small sized files are handled inefficiently in HDFS. NameNode carries the metadata information in memory for the files which are stored in HDFS. Consider a file which is stored in HDFS having size as 1 GB and the NameNode is responsible for storing the information such as filename, offset, length etc. which are split into 1000 fragments and stored all 1000 small files in HDFS. It is mandatory for the NameNode to store metadata information of 1000 small files in memory. This is not efficient way; first it takes up a lot of memory and second soon NameNode will become a bottleneck as it is trying to manage a lot of data [30].

Due to the outbreak of a COVID-19 pandemic, the entire world is now working in online mode. As a result, a massive volume of data has been generated, making it extremely complex to store, analyze and handle. The term big data generally used for this. The generated large volume of data which is in the form of structured or unstructured format. There is an immense need of analyzing and classifying this data in terms of size. HDFS gives the solution for this, which is responsible for handling large files in GBs or TBs in size of unlimited storage. HDFS works closely with the small number of large files. It is inefficient for handling large number of small files. Large number of small sized files consume more memory on the NameNode of the HDFS. Access mechanism of small sized files concept is a major problem in HDFS[8],[9],[10].

1) Paper organization: The rest of the paper is organized as follows: Section II gives the related surveys present in the existing literature. Section III provides the discussion on existing solutions on the given problem statement. Section IV describes the proposed work. Section V focusses on experimental set up and results produced. Section VI finally concludes the article.

II. Literature Review

Xiong et al [1] employed a small file usage pattern to identify candidate files for merging into a single container. Identified gap in this article as and when the HDFS user request pattern changes there is a chance of cache missing.

Zhipeng et. al [4] discussed about merge strategy based hierarchy for improving the small file problem in HDFS, Bing et.al [5] created a novel approach HDFS:TLB MapFile for efficient accessing of small files, Sachin et al [6] discussed about different techniques of dealing with a small files problem. Authors discussed about Hadoop Archive i.e. achieving technique which binds number of small files into HDFS blocks. Another technique improved HDFS model is index-based i.e. index is built for each small file to reduce the waste caused by them which reduces the burden on the NameNode. EHDFS gives an improved indexing mechanism and prefetching of index information. Lion Jude Tchaye-Kondi et al. [13] encouraged using hash functions to create a perfect file. To get the metadata of a specific file, this approach removes the requirement of parsing the index file. Xun Cai et al. [14] improved the access and storage efficiency of small files. Yanfeng Lyu et al. [15] proposed an efficient merging method that substantially reduces the access time for small files by using caching and prefetching methods. X. Fu et al. [16] proposed a block replica placement technique for effectively processing small files where files are merged as per the pre-determined parameters. Qi Mu et al. [17] advised a method for dealing with small files that is both efficient and effective. T. Wang et al. [18] suggested a method based on the behavior of small files access to build association between the small files. In this article, the concept focusses exclusively on file size. The gap identified in this, file contents have not been checked. If there is a mismatch in the file sizes, then cache will be missed.

III. Existing Methods

A. HAR (Hadoop Archives)

HAR files were created to reduce the problem among several files putting a pressure on the memory of name nodes. On the HDFS, a layered system has been installed. The Hadoop archive command is used to create HAR files. It is not more efficient to read through files in a HAR than it is to read across files in Hadoop. Each HAR file access needs the reading of two index files in addition to the data file, which slows down the process [2],[7],[19],[20]. The flat table technique is used to implement the HAR.

Limitations of HAR files:

- Hadoop archive file once created is not updatable i.e. to adding or removing of the file is not possible.
- This archive file will contain a replica of all the original files when '.har' is created and it will take more space as the original files.

B. NHAR (New HAR)

For NameNode, processing a large number of small files takes more time. In fact, the amount of time it takes to access such data should be minimized. NHAR [2],[7],[11],[12] is a novel solution that relies on Apache Hadoop's HAR. The table chain technique is used to implement NHAR.

C. Spatiotemporal Small File Merging Strategy

In previous work, Lion Xiong et al. [1] employed a small file usage pattern to identify candidate files for merging into a single container. They employed file access time stamps to analyze usage patterns, then grouped files with sequential time stamps and generated support files for each file group. A single container is used to store a file group with a high support value.

IV. Proposed System

In this paper, the proposed technique is implemented by creating a wrapper over existing HDFS architecture without altering the HDFS architecture[5]. The Wrapper contains the Virtual file table which maintains the records of every file category wise like offset, length etc. Using large number of small files with different sizes analyzed the experimental results. Ultimately the time required for accessing files in HDFS is improved.

A. Bucket Chain Technique

Working methodology of the proposed system is shown in Fig. 1. The root file table contains the list of files category wise, in which the information is gained about the existence of file category. The Bucket or child per category file table contains the metadata of the per category container which holds the metadata as Name, Offset and length of the category. Every child per category file table gives the metadata of the per category container file where the actual contents of the file can receive. Then it can directly access the contents of the actual file as that of reading or getting a file which is working with HDFS.

The advantages of the technique are if a user tries to access the same file repeatedly, the cache only gives the information quickly about the file location reducing the multiple searching and reading overheads.

B. Methodology

In this method all file's metadata is stored in a single file, this file is called File Table. This file contains linked list of tables, forming a table chain. Another file is used to store actual contents of files; this file is called Container File. One container file is used for each category. All these files are stored on HDFS. File table contains metadata of a file such as filename, offset, and length.

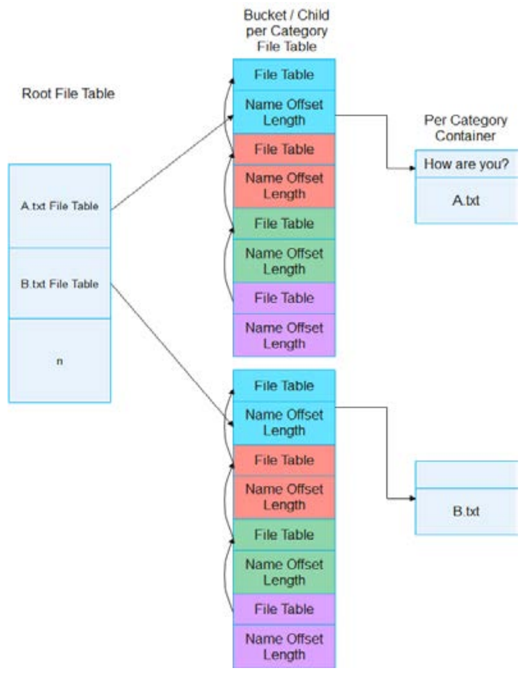

Fig. 1. Bucket Chain Technique for Proposed System (VFS-HDFS).

Before storing every file, it is classified using ensemble classifier to get its category. That file is then stored in container of that category and corresponding file table is also updated.

On startup, the file table for each category is loaded in memory. First the latest file table is located in file table and its contents are added to in-memory file table, then previous file table is located and the process continues until first file table is reached. While doing this, a file's metadata is already found in in-memory file table, then that entry is discarded (as it has become old due to updating or deletion of file).

In every run, a new in-memory file table for each category is created to track new or updated files in that category. At runtime all metadata is created and updated operations are performed on this in-memory file table. Actual contents of the files are appended in container file. On shutdown, the contents of in-memory file tables are appended to file table in HDFS. This creates a chain of buckets, each bucket stores list of file tables which makes it possible to update and delete files.

For reclaiming space used by deleted or updated files, the technique of pruning used. In this technique new copies of file tables are creating while skipping the deleted or updated entries. This will reduce space required for file tables and containers.

Caching is used to reduce time required for reading files in already retrieved blocks. Each cache entry stores tuple of category, file, position, length and its complete block content. This will reduce time at the cost of increased memory requirement.

C. *Algorithm: Bucket Chain*

```
globals:
    filenametable
    indexfilemap
    containerfilemap
    filetablemap
    newfiletablemap
    classification_model
    categorylist
function intialize()
    filenamemap=load_filenamemap()
    for cat ∈ categorylist
        indexfile=indexfilemap[cat]
        containerfile=containerfilemap[cat]
        filetable=filetablemap[cat]
    file_index_location=get_last_index(indexfil
    while file_index_location != NULL
    indices=read_index(file_index_location)
            filetable = filetable ∪ (indices - (filetable ∩ indices))
    file_index_location=get_prev_index(file_index_location)
function add_file(filename, content)
    cat = classify(classification_model, content)
    location=append_content(containerfilemap[cat], content)
    add_file_entry(newfiletablemap[cat], filename, location, len(content))
    add_filenametable(filenametable, filename, cat)
function get_file(filename)
    cat = get_category(filenametable, filename)
    location, length = find_file(newfiletablemap[cat], filetable[cat], filename)
    data = read_content(containerfilemap[cat], location, length)
    return data
function close()
    for cat ∈ categorylist
        last_index=get_last_index(indexfilemap[cat])
        append_entry_table(indexfilemap[cat], newfiletable[cat])
        append_prev_index(last_index)
    update_filenametable(filenametable)
```

D. *Advantages of Bucket Chain Technique*

1) It has separate category wise containers.
2) It contains cache memory.
3) Pruning is applied to remove unused files in containers to reduce the memory wastage.
4) Optimal File Table Size.
5) Access time efficiency improved.

E. *Text Classification*

It is used to classify the text documents automatically into one or multiple defined categories (Fig. 2). Category of news article from Reuters-21578 Text Categorization dataset [21] is taken into consideration for text classification Text classification consists of main components [23].

1) Dataset Preparation.
2) Feature Engineering.
3) Model Training.
4) Improve Performance of text classifier.

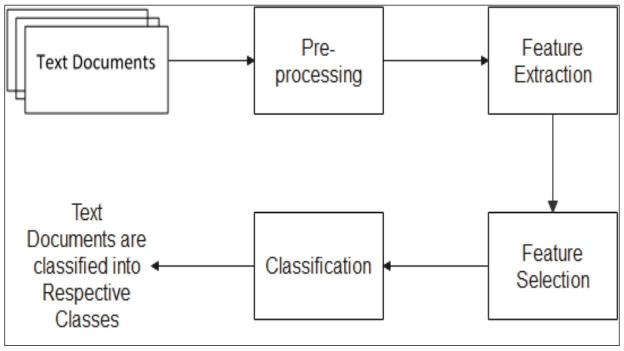

Fig. 2. Text Classification Task.

F. Datasets Used

Experimentation is done on the Reut2-000.sgm & Reut2-002.sgm files. of Reuters dataset.

Table I gives information about the datasets used.

TABLE I. DESCRIPTION OF DATASET

Name of Dataset	Data Types	Default Task	Attribute Type	No. of Instances	No. of Classes	Year
Reuters-21578 Text Categorization Collection	Text	Classification	Categorical	1000	70	1997

G. TF-IDF

The technique stands for Term Frequency- Inverse Document Frequency which is used to quantify words from documents; a weight has been assigned to each word which states the significance of each word in the document and its collection. TF-IDF score signifies the relative importance of a keyword or a term in the document and the entire corpus. It is calculated as the logarithm of the number of the documents in the corpus divided by the number of documents where the definite term appears [23].

Formula: idf(t) = log (N/df)

The tf-idf metric is calculated by considering the total number of documents, dividing it by the number of documents that which contain a word and calculating the logarithm.

tfidf(t) = tf(t) * idf(t)

H. Classifiers used for Experimentation:

1) J48 classifiers: It deals with issues such as numeric attributes, missing values, pruning, predicting error rates, decision tree induction complexity etc[3].

2) Random forest: It determines the outcome based on decision tree predictions. It estimates an average of the output of various trees [24].

3) Naive bayes classifiers: It is capable of dealing with both discrete and continuous data. It can handle a large number of predictions and data sets [25].

4) Ensemble classifiers: For classification purpose the ensemble learning method is used. Fig. 3 describes the architecture of ensemble classifier. These ensembles combine multiple hypotheses to form a better hypothesis. Ensemble learning supports to improve machine learning results as compared to a single model by combining several models. Basic idea is to learn a set of classifiers and to allow them for vote. Combination of Random Forest, Naïve Bayes and J48 classifiers are used for experimentation [26].

Advantages: Motivation behind to use ensemble classifier is to improve the predictive accuracy.

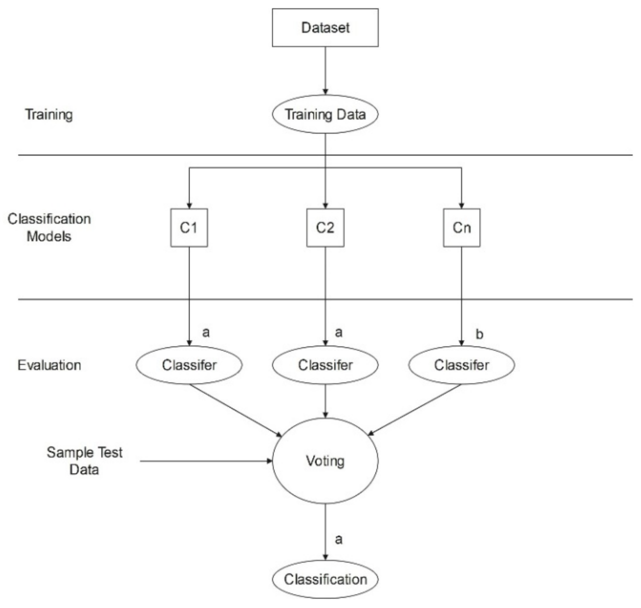

Fig. 3. Architecture of Ensemble Classifier.

V. EXPERIMENTAL SETUP, RESULTS AND ANALYSIS

The proposed system is implemented on top of existing HDFS architecture. The proposed algorithm is experimented on Fedora 32.0 Operating System, Hadoop 3.2.0 and Java 1.8.0 Version. It contains 16 GB of RAM and 500 GB of Hard Disk with i5-5500U CPU @ 2.20GHz processor. Reuters-21578 Text Categorization Collection dataset is used to test the proposed system.

Table II shows the experimental set up used to execute the proposed work.

Table III shows how much memory and time is required to access small sized files in bucket chain algorithm.

TABLE II. EXPERIMENTAL SETUP

Sr. No.	Parameters	Description
1	No. of Nodes	Single Node (Acts as both Master & Slave)
2	Node Configuration	Intel(R) Core(TM) i5-5500U CPU @ 2.20GHz
3	RAM	16 GB
4	Hard Disk	500GB
5	Operating System	Fedora 32.0
6	Execution Platform	JDK 1.8.0
7	Hadoop Version	3.2.0
8	Development Tool	Net Beans 12.0
9	Dataset	Reuters containing TEXT files
10	Number of Files considered	2138
11	File Size Range	Average From 1 KB to 100 KB
12	No. of Iterations	1000
13	No. of Classes	70

TABLE III. MEMORY AND TIME REQUIREMENT FOR PROPOSED SYSTEM (VFS-HDFS I.E. BUCKET CHAIN TECHNIQUE)

Memory & Time Requirement for Proposed Technique (VFS-HDFS i.e. Bucket Chain Algorithm) for 1000 Iterations		
Average File Size	**Memory (in MB)**	**Time (in Seconds)**
1K	30.7	39
5K	31	39
10K	35.2	41
50K	40.5	46
100K	55.2	55

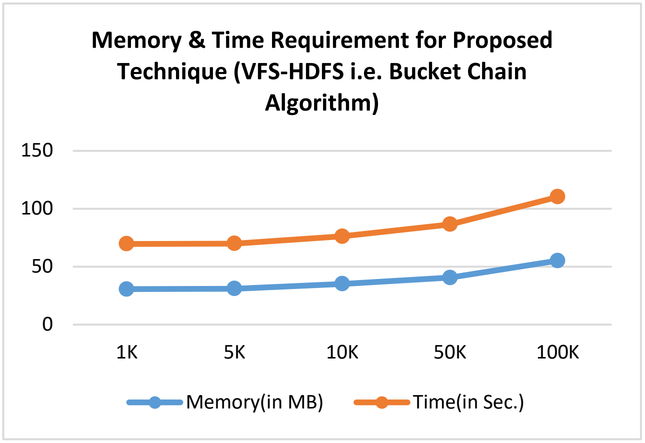

Fig. 4. Performance with respect to Time and Memory for Proposed Technique (VFS-HDFS, using Bucket Chain Algorithm).

The above graph in Fig. 4 indicates the performance of the proposed technique in terms of time and memory.

Table IV shows the comparative chart of Experimental results of existing technique and proposed Technique VFS-HDFS using Bucket chain algorithm. where the memory requirement is increased due to the cache.

The graph in Fig. 5 shows the comparative chart of experimental results of existing and proposed technique VFS-HDFS using Bucket chain algorithm.

Table V shows the comparative chart of Experimental results of existing technique and proposed technique-VFS-HDFS using Bucket chain algorithm; where small files access time efficiency is increased.

TABLE IV. COMPARATIVE CHART FOR MEMORY REQUIREMENT FOR EXISTING (HAR & NHAR) AND PROPOSED TECHNIQUE (VFS-HDFS USING BUCKET CHAIN ALGORITHM)

Memory Requirement for 1000 Iterations			
Average File Size	**HAR (in MB)**	**NHAR (in MB)**	**VFS-HDFS (in MB)**
1K	16.2	19	30.7
5K	16.5	19.1	31
10K	17.1	21.2	35.2
50K	19.3	23.1	40.5
100K	27.4	35.8	55.2

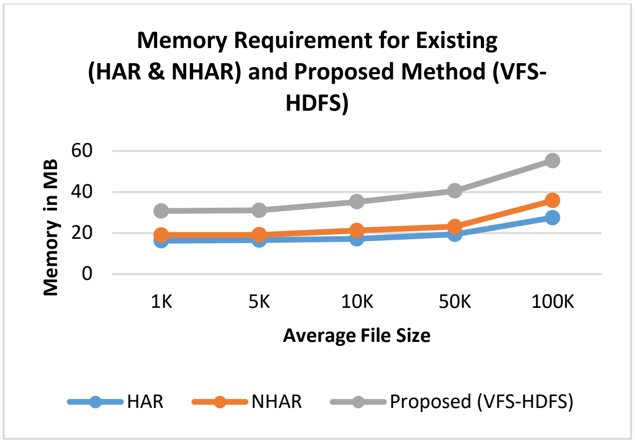

Fig. 5. Memory Requirement Existing and Proposed Technique VFS-HDFS using Bucket Chain Algorithm.

TABLE V. COMPARATIVE CHART FOR TIME REQUIREMENT FOR EXISTING AND PROPOSED TECHNIQUES USING BUCKET CHAIN ALGORITHM

Time Requirement for 1000 Iterations			
Average File Size	**HAR (in Seconds)**	**NHAR (in Seconds)**	**VFS-HDFS (in Seconds)**
1K	56	45	39
5K	58	45	39
10K	64	48	41
50K	71	57	46
100K	97	70	55

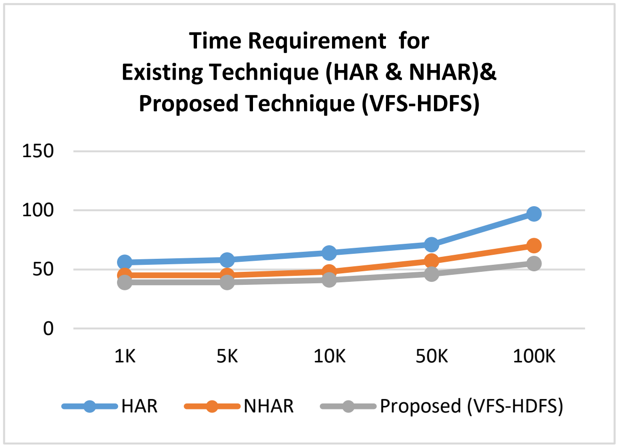

Fig. 6. Time Requirement for Existing and Proposed Technique VFS-HDFS using Bucket Chain Algorithm.

Fig. 6. shows the comparative chart of Experimental results of existing technique and proposed Technique-VFS-HDFS in terms of time using Bucket chain algorithm.

Classifier Metrics:

Table VI gives the information about different classifiers used and the corresponding experimental results.

Confusion Matrix for a single class:

Confusion matrix is created for class “earn” from reut2-002.sgm file.

TABLE VI. DIFFERENT CLASSIFIER METRICS ALONG WITH ENSEMBLES

Parameters	Naïve Bayes	J48	Random Forest	Ensemble (Naïve Bayes+J48+Random Forest)
Total Instances	962	962	962	962
Correctly Classifies Instances	684	738	899	922
Incorrectly Classified Instances	278	224	63	40
Accuracy	**71%**	**77%**	**93%**	**96%**

TABLE VII. CONFUSION MATRIX

Total Instances N=962	Actually Positive	Actually Negative
Predicted Positive	131	40
Predicted Negative	0	791

Table VII shows the confusion matrix [22] of the instances of the earn class.

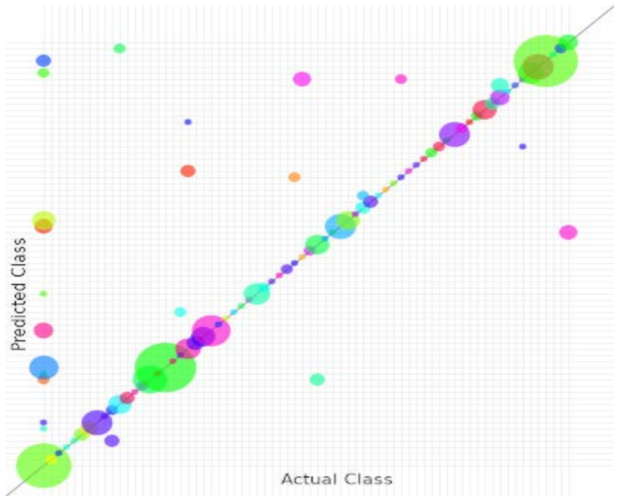

Fig. 7. Accuracy of Classification and Predictions.

The above graph in Fig. 7 shows the bubble chart of accuracy of classification and predictions of total 70 classes. X-axis indicates the actual class. Y-axis indicates the predicted class. Instances on 45-degree line are correctly classified instances. Incorrectly identified instances are randomly shown in random positions. Size of bubble indicates number of instances in that position.

VI. CONCLUSION

In this paper, a novel approach called *VFS-HDFS* architecture is proposed in which exclusive focus is on optimization of the access efficiency of small sized files in HDFS with a significant improvements compared with the existing techniques i.e. flat table (HAR) and table chain (NHAR) technique. A new algorithm is proposed called bucket chain algorithm and the entire work is executed using this algorithm where memory requirement is increased to store the files in cache and time required to access the files from the cache is reduced. The proposed work uses the ensemble classifiers to classify the data sets. It helps to group the similar files in same container. Comparison is made between the experimental results of existing approach-i.e. flat table and table chain techniques and the proposed technique-bucket chain technique. In the proposed system it is observed that time required to access the files is reduced at the cost of memory required to store the files.

REFERENCES

[1] Lian Xiong et al. “A Small File Merging Strategy for Spatiotemporal Data in Smart Health”, IEEEAccess Special Section on Advanced Information Sensing and Learning Technologies for Data-Centric Smart Health Applications, Volume 7, 2019.

[2] Neeta Alange, Anjali Mathur, “Access efficiency of small sized files in Big data using various techniques on Hadoop Distributed File System Platform”, International Journal of Computer Science and Network Security Volume.21, No.7, July 2021.

[3] N. Saravanan et. al “Performance and Classification Evaluation of J48 Algorithm and Kendall’s Based J48 Algorithm (KNJ48)” International Journal of Computational Intelligence and Informatics, Vol.7:No.4, March 2018.

[4] Zhipeng et al “An Effective Merge Strategy Based Hierarchy for Improving Small File Problem on HDFS” IEEE Proceedings of CCIS 2016, pp. 327-331.

[5] Alam et al. "Hadoop Architecture and its issues." International Conference on Computational Science and Computational Intelligence (CSCI), 2014 Vol. 2. IEEE, 2014.

[6] Sachin et al “Dealing with small files problem in hadoop distributed file system”, Procedia Computer Science Volume 79, 2016.

[7] Ankita et al “A Novel Approach for Efficient Handling of Small Files in HDFS”, IEEE International Advance Computing Conference (IACC, 2015), pp.1258-1262.

[8] Nivedita et. al “Optimization of Hadoop Small File Storage using Priority Model”, 2nd IEEE International Conference On Recent Trends in Electronics Information & Communication Technology (RTEICT), pp. 1785-1789, May 2017.

[9] Awais et al “Performance Efficiency in Hadoop for Storing and Accessing Small Files” 7th International Conference on Innovative Computing Technology (INTECH 2017), pp.211-216.

[10] Neeta Alange, Anjali Mathur, “Small Sized File Storage Problems in Hadoop Distributed File System”, 2nd International conference on Smart Systems and Inventive Technology (ICSSIT 2019) vol. pp. 1198-1202, November 2019 proceedings published in IEEE Digital Xplore.

[11] Shubham et. al “An approach to solve a Small File problem in Hadoop by using Dynamic Merging and Indexing Scheme”, International Journal on Recent and Innovation Trends in Computing and Communication [IJRITCC], November 2016, Volume: 4, Issue:11.

[12] Priyanka et al "An Innovative Strategy for Improved Processing of Small Files in Hadoop", International Journal of Application or Innovation in Engineering & Management (IJAIEM), Volume 3, Issue 7, July 2014, pp. 278-280, ISSN 2319 – 4847.

[13] J. Tchaye-Kondi, Y Zhai et al. “Hadoop Perfect File: A fast access container for small files with direct in disc metadata access”, 2019, arXiv:1903.05838.

[14] X. Cai, C. Chen et al. “An optimization strategy of massive small files storage based on HDFS”, in Proc. JIAET, 2018, PP. 225-230.

[15] Y. Lyu, X. Fan, and K. Liu, ``An optimized strategy for small _les storing and accessing in HDFS," in Proc. IEEE Int. Conf. CSE, IEEE Int. Conf. EUC, Jul. 2017, pp. 611_614.

[16] X. Fu,W. Liu, Y. Cang, X. Gong, and S. Deng, ``Optimized data replication for small _les in cloud storage systems," Math. Problems Eng., vol. 2016, pp. 1_8, Dec. 2016.

[17] Q. Mu,Y. Jia, and B. Luo, ``The optimization scheme research of small _les storage based on HDFS," in Proc. 8th Int. Symp. Comput. Intell. Design, Dec. 2015, pp. 431_434.

[18] T. Wang, S. Yao, Z. Xu, L. Xiong, X. Gu, and X. Yang, ``An effective strategy for improving small _le problem in distributed _le system," in Proc. 2nd Int. Conf. Inf. Sci. Control Eng., Apr. 2015, pp. 122_126.

[19] Online Reference Apache Hadoop, http://hadoop.apache.org/.

[20] Online Reference https://blog.cloudera.com/blog/2009/02/the-small-files-problem/.

[21] Dua, D. and Graff, C. "UCI Machine Learning Repository Dataset" [http://archieve.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2019.

[22] Online Reference https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/.

[23] Online Reference https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/.

[24] Online Reference https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/ random forest classifier.

[25] Wikipedia contributors. Naive Bayes classifier. In Wikipedia, the Free Encyclopedia. Retrieved 14:24, September 30, 2021, from https://en.wikipedia.org/w/index.php?title=Naive_Bayes_classifier&oldid=1039393803.

[26] Online Reference https://subscription.packtpub.com/book/data/9781789955750/7/ch07lvl1sec45/bagging-building-an-ensemble-of-classifiers-from-bootstrap-samples.