Small Files Access Efficiency in Hadoop Distributed File System a Case Study performed on British Library .rtf files

Neeta Alange (neetaalange@gmail.com)
Koneru Lakshmaiah Education Foundation, KL Deemed To Be University
P. Vidya Sagar
Koneru Lakshmaiah Education Foundation, KL Deemed To Be University

Research Article

Keywords: HDFS, Small files, Bucket Chain Technique, British Library

Posted Date: January 12th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2453995/v1

Abstract

In today’s world storing a large amount of data, large datasets, handling data in various forms is a challenging task. Data is getting produced rapidly with major small sized files. Hadoop is the solution for the big data problem except few limitations. This method is suggested to provide a better one for small file sizes in terms of storage, access effectiveness, and time. In contrast to the current methods, such as HDFS sequence files, HAR, and NHAR, a revolutionary strategy called VFS-HDFS architecture is created with the goal of optimizing small-sized files access problems. The existing HDFS architecture has been wrapped with a virtual file system layer in the proposed development. However, the research is done without changing the HFDS architecture. Using this proposed system, better results are obtained in terms of access efficiency of small sized files in HDFS. A case study is performed on the British Library datasets on .txt and .rtf files. The proposed system can be used to enhance the library if the catalogue is categorized as per their category in a container reducing the storage, improving the access efficiency at the cost of memory.

1. Introduction

The HDFS of Hadoop works at storage level which enables high amount access to application data, while Hadoop is an open source framework that aids in the storage, processing, and analysis of huge volumes of data. HDFS is a module in hadoop which handles large data sets. The main mechanism for storing data in Hadoop applications is called HDFS. Application data access with high throughput is made possible. It offers a mechanism to manage huge amounts of organized and unstructured data and is a component of big data. HDFS is made for processing massive amounts of data when scalability, flexibility, and performance are essential. Hadoop is based on the idea of keeping a small number of very large files and utilizes a master/slave architecture to store data in HDFS. While the NameNode, the master node in HDFS, only maintains the metadata of HDFS—the directory tree of all files in the file system—while the DataNode, a slave node, stores the actual data as instructed by the NameNode. NameNode does not store the actual data or the dataset. A single NameNode, a master server that manages the file system namespace and regulates client access to files, makes up an HDFS cluster. A number of DataNodes, typically one per node in the cluster, are also present and manage storage relevant to the nodes that they run on. In the data center, all of the nodes are typically arranged in the same physical rack. Then, the data is divided into distinct blocks and distributed among the numerous data nodes for storage. HDFS has two main components – data blocks and nodes storing those data blocks. Hadoop is lightning fast because of data locality – move computation to data rather than moving the data, as it is easier and make processing lightning fast. The Same algorithm is available for all the nodes in the cluster to process on chunks of data stored in them. HDFS differs from other file systems, Small data blocks are the norm for file systems. (Around 512 bytes), however HDFS's block sizes are higher (Around 64 MB). In conventional file systems, many discs seek for larger files, whereas in HDFS, data is read sequentially after each seek [1].

2. Hdfs File Storage

Files storage in HDFS:

Files are divided into blocks by DFS, and each block is kept on a DataNode. The NameNode, the cluster's master node, is connected to numerous DataNodes. These data blocks are replicated and dispersed throughout the cluster by the master node.

Block Storage in HDFS:

Large files are to be continuously stored by HDFS across numerous machines in a large cluster. Every file is stored as a series of blocks, with the exception of the last block in each file being the same size. DataNodes' blocks are copied for fault tolerance. Block size and replication factor can be customized for each file.

Storage of 1GB file in HDFS:

File Size: 1GB =1024 MB

Configured block size: 128 MB

Total blocks needed: 1024 MB/128 MB = 8 Blocks.

1 DataNode will contain 8 blocks to store 1 GB file.

If the Replication Factor is set to 3.

1 GB file is -> 8 Blocks * 3(RF) =24 Blocks.

Total Memory on different DataNodes: 128 MB * 24 Blocks = 3072 MB.

Table 1. HDFS Block and Memory Requirement with Replication Factor

File Size in MB	Configured Single Block Size in MB	Blocks Required to Store specified file	Replication Factor	Total Blocks Required with Replica	Memory Required to store blocks in MB
(1)	(2)	(3) = (1) / (2)	4	5	6
1024	128	8	3	24	3072

Table 1 shows the HDFS Block and memory requirement with replication factor with 1GB file.

Metadata:

Metadata are facts about other facts. It is kept in NameNode, where information about the DataNodes data, including the data's storage location and replicas, is kept. NameNode uses 150 bytes for file metadata storage and 150 bytes for block metadata storage for each replica.

What are Small Files?

A file that is considerably smaller than the HDFS block size is referred to as a small file (default 64MB). Storing of small files and the problem is that HDFS can’t handle lots of files.

3. British Library: An Introduction

The most complete research collection in the world is available at the British Library, the country's national library. It offers information services to the scientific, business, academic, and research communities. Artifacts from every period of written civilization are included in the collection's estimated 170 million + objects. The library maintains the country's collection of printed and digital publications and adds about three million new items each year. Inspiring exhibitions at the library contextualize these holdings and share their narratives with the public [2].

4. Proposed System

Figure 1. Shows the architecture of proposed system. In this proposed architecture just a virtual file system layer integrated as a wrapper at the top of the Original HDFS without altering the HDFS architecture. Client application interact with the system through VFS-HDFS API. This virtual file system layer consists of the Virtual File Table. File table is maintained per container (as per category). This file table per container maintains information such as File name, Offset, Length and Category of all the files stored on VFS. File table is a linked list of buckets. Each bucket refers to previous bucket. File entries in newly added buckets get precedence over entries from previous bucket. This will allow to modify file contents of already stored files.

Need of Classifiers?

Files containing similar contents have a high probability that those files are used by same client. For grouping similar files (as per their category) in a same container different text classifiers are used. For this experiment Naïve Bayes, Random Forest, J48 and Ensemble classifiers are used.

Motivation to use Ensemble Classification?

No single classifier is found to be sufficient for classifying text contents, therefore combination of multiple classifiers have been used.

Bucket Chain Methodology:

In this system, a single file called File Table stores all of the file's metadata. The tables in this file are connected together to form a table chain. Another file, known as a container file, is used to store the actual contents of other files. Each category has a single container file. File table and bucket file table both contain filename, offset, and length as well as other file metadata.

The category to which each file belongs is determined using an ensemble classifier as the file is being stored. The associated file table is likewise updated, and the file is then saved in a container for the category.

Each category's file table is loaded into memory at launch. The most recent file table is found first, and its contents are added to the in-memory file table. The next file table is then found, and the process is repeated until the first file table is reached. When doing this, if the metadata for a file is already present in the in-memory file table, that entry is removed (as it has become old due to updating or deletion of file).

Each iteration generates a fresh in-memory file table for each category to keep track of newly added or modified files. On this in-memory file table, all metadata creation and update operations are carried out in real time. The container file contains the actual contents of the files. The contents of the in-memory file table are added to the HDFS file table upon termination. This establishes a chain of buckets, each of which has a list of file tables that may be updated and deleted.

Pruning is a method for recovering space that has been occupied by updated or deleted files. This method creates fresh copies of file tables while omitting the changed or removed items. As a result, file tables and containers will take up less space.

By using caching, reading files from previously received blocks takes less time. Every cache entry has a tuple including the category, file, location, length, and the entire block content. This will speed up the process but require more RAM [6]

5. Experiments And Results

Table 2
Data with Standard Deviation

Data with Standard Deviation					
(from SQL Server STDDEV function) included					
File Type	No. of Files	Average File Size	Minimum File Size	Maximum File Size	Standard Deviation
doc	13348	272343	0	42811904	674660.289
docm	1	601859	601859	601859	N/A
docx	211	507997	6308	46311671	3466513.39
epub	510027	14610136	0	4167084940	54934421.6
htm	1	15449	15449	15449	N/A
html	154375	57658	0	3610704	105079.177
lit	1	288171	288171	288171	N/A
mobi	3	16651147	1227878	28513943	13987139.4
msg	36	99271	26624	1056256	205015.358
pdf	19312072	1715613	0	3820163267	8907656.73
ppt	5	623718	30208	2112000	855216.223
pub	3	3318272	1403904	7147008	3315782.64
rtf	1733	401837	827	6596844	400088.727
Txt	112521	6592	0	1297185	11162.5078
xls	8225	95370	10283	6989824	324501.987
xlsx	27	46569	12819	840704	159359.731
Total	20112589	2018563	0	4167084940	12525185.1

Table 2 shows the data with standard deviation which is collected from the SQL server STDDEV function included and this information for this case study is received from the research group of British Library [2].

The proposed system is tested using .txt and .rtf files. For the experimentation purpose the parameters used are File Type, No. of Files, Average File Size, and Minimum & Maximum File Size of the .txt and .rtf file format of British Library shown in Table 3 below.

Table 3
Test Case Data from British Library datasets for specified files (Bytes)

File Type	No. of Files	Average File Size	Minimum File Size	Maximum File Size
.txt	112,521	6,592	0	1,297,185
.rtf	1,733	401,837	827	6,596,844

Table 3 provides the test case data from British library datasets for .txt and .rtf files including minimum, maximum and average file size [2].

Table 4
Metadata per block along with Replication Factor (Bytes)

Metadata Per Block	Metadata Per File	Replication Factor
150	32	3

Table 4 shows the metadata considered per file along with the replication factor. Metadata per block is considered as 150 Bytes for the experimentation. Default value of Replication Factor is 3. Metadata per file is considered as 32 Bytes which includes (Filename = 20 Bytes, length = 4 Bytes, offset = 8 Bytes) shown in above Table 4.

Table 5
Test Case data from British Library Datasets for specified files with Average File Size, No. of Files and total size (Bytes)

File Type	Average File Size	No. of Files	Total Size
txt	6,592	112,521	741738432
rtf	401,837	1,733	696383521

Table 5 gives the information about File Type, Average File Size, No. of Files and the total size of the .txt and .rtf files.

Metadata/file is calculated as per the formula given:

Metadata/File = No. of files*Metadata/file for container*Replication Factor

Metadata/Container is calculated as per the formula given:

Metadata for Container = (Metadata/Block * Total Size / (128 * 1024 *1024) * Replication Factor)

Total Metadata is calculated as per the formula given:

Total Metadata = Metadata Per for container Files + Metadata for Container

Table 6
Total metadata required for existing system and proposed system

File Type	HDFS Existing System		Proposed System		
	Metadata Per Replica in Bytes	Total Metadata in Bytes	Metadata Per for container Files in Bytes	Metadata for Container in Bytes	Total Metadata in Bytes
.txt	16878150	50634450	10802016	2487	10804503
.rtf	259950	779850	166368	2335	168703

Table 6 gives the results of existing HDFS and proposed system for .txt and .rtf files respectively.

For the Existing HDFS System, the Metadata per replica required is 16.88 MB and the total metadata required to store the files is 50.63 MB. While in the proposed system as it uses the per category container files the metadata required to store the container files is 10.80 MB and metadata required to for storing the container files is 0.00249 MB (2487 Bytes) which results the total metadata required to store the files is 10.80 MB. The results for .rtf files type is also shown in Table 6.

Table 7
Total metadata required for existing system and proposed system.

Parameters	Existing HDFS	Proposed System
Time in Seconds	1015	480
Memory in MB	15.41	52.35

Table 7 shows the comparative table for Time and Memory required for accessing and storing the files with respect to existing and proposed system.

Figure 3 shows the comparative chart for existing versus proposed system in terms of time and memory.

Table 8
Comparative chart for existing versus proposed system

Sr. No.	Parameters Used	Flat Table (Existing Technique)	Table Chain (Existing Technique)	Bucket Chaining (Proposed Technique)
1	Method	Flat Chaining	Table Chaining	Bucket Chaining
2	Access Time (Lesser is Better)	Very High	High	Low
3	File Indexing Overhead	Slight (Very Less)	Moderate	Increased (High)
4	Memory Footprint	Low	Low	High
5	Read/write Efficiency	Low	Low	High
6	Storage Efficiency	Very Low	Low	High
7	Scalability	Very Low	Moderate	High
8	Data Mutability	No	Yes	Yes

Table 8 describes the comparative chart of the existing techniques- flat and table chain techniques and the proposed system-bucket chain technology [4] [5].

6. Conclusion

Experimentation is done on Existing System and the Proposed System, results are compared from the experiment and it is found that storage required to store the files as per their category is condensed and the metadata required to store the per container file is increased. The memory requirement is increased while storage is reduced and the access time required to access per container file is improved. If the proposed system is implemented on British library dataset there is a chance of reducing the storage and improving time needed to access documents at the cost of increased memory requirement. In the future the system will be useful for the effective implementation of e-Library in education, Health Care centers etc. to give the efficient output in terms of access time and the storage of the files.

Declarations

Ethics declarations:

Authors Contribution:

Each author contributed equally in each part.

Conflict of interest

The authors declare that they have no conflicts of interest.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publications

The authors claim that none of the material in the paper has been published or is under consideration for publication elsewhere.

Funding

Author declared that no funding was received for this Research and Publication.

Acknowledgment

The authors thank for providing characterization supports to complete this research work.

References

1. Online Reference Apache Hadoop:, http://hadoop.apache.or/
2. https:// andE-mail Reference
3. Lian, X., 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)
4. Neeta Alange, A., Mathur: “Small Sized File Storage Problems in Hadoop Distributed File System” 2nd International Conference on Smart Systems and Inventive Technology (ICSSIT 2019) IEEE Xplore Part Number: CFP19P17-ART; ISBN: 978-1-7281-2119-2
5. Neeta Alange, A., 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, (2021)
6. Neeta Alange, A., Mathur: “Optimization of Small Sized File Access Efficiency in Hadoop Distributed File System by Integrating Virtual File System Layer”,International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 13, No. 6, (2022)
7. 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, (2018)
8. Zhipeng, et al.: pp. 327–331. (2016)

9. Alam, et al.: "Hadoop Architecture and its issues." International Conference on Computational Science and Computational Intelligence (CSCI), 2. IEEE, 2014. (2014) Vol
10. Sachin, et al.: “Dealing with small files problem in hadoop distributed file system”, Procedia Computer Science Volume 79, Ankita “A Novel Approach for Efficient Handling of Small Files in HDFS”, IEEE International Advance Computing Conference (IACC, 2015), pp.1258–1262. (2016)
11. Bullarao Domathoti, C., Madala, C.S.R., Berhanu, A.A.: Yamarthi Narasimha Rao, "Simulation Analysis of 4G/5G OFDM Systems by Optimal Wavelets with BPSK Modulator", Journal of Sensors, vol. Article ID 8070428, 13 pages, 2022. (2022). https://doi.org/10.1155/2022/8070428
12. Anuradha, T., Lakshmi Surekha, T., Nuthakki, P., Domathoti, B., Ghorai, G., Shami, F.A.: "Graph Theory Algorithms of Hamiltonian Cycle from Quasi-Spanning Tree and Domination Based on Vizing Conjecture", Journal of Mathematics, vol. 2022, Article ID 1618498, 7 pages, (2022). https://doi.org/10.1155/2022/1618498
13. Saba, T., Rehman, A., Haseeb, K., et al.: Cloud-edge load balancing distributed protocol for IoE services using swarm intelligence. Cluster Comput. (2023). https://doi.org/10.1007/s10586-022-03916-5
14. Braik, M.: Hybrid enhanced whale optimization algorithm for contrast and detail enhancement of color images. Cluster Comput. (2022). https://doi.org/10.1007/s10586-022-03920-9
15. Jia, Z., Fan, X., Wang, H.: Retraction Note: Multimedia and multi-feature cluster fusion model based on saliency for mobile network applications. Cluster Comput. (2022). https://doi.org/10.1007/s10586-022-03945-0
16. Oyelade, O.N., Ezugwu, A.E.-S., Mohamed, T.I.A., Abualigah, L.: "Ebola Optimization Search Algorithm: A New Nature-Inspired Metaheuristic Optimization Algorithm," in IEEE Access, vol. 10, pp. 16150–16177, doi: (2022). 10.1109/ACCESS.2022.3147821
17. Jeffrey, O., Agushaka, A.E., Ezugwu: Laith Abualigah, warf Mongoose Optimization Algorithm, Computer Methods in Applied Mechanics and Engineering,Volume 391,2022,14570,ISSN 0045-7825, https://doi.org/10.1016/j.cma.2022.114570
18. Laith Abualigah, M.A., Elaziz, P., Sumari, Z.W., Geem, A.H.: Gandomi,Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer. Expert Syst. Appl. Volume 191. **116158**, 0957–4174 (2022). https://doi.org/10.1016/j.eswa.2021.116158

Figures

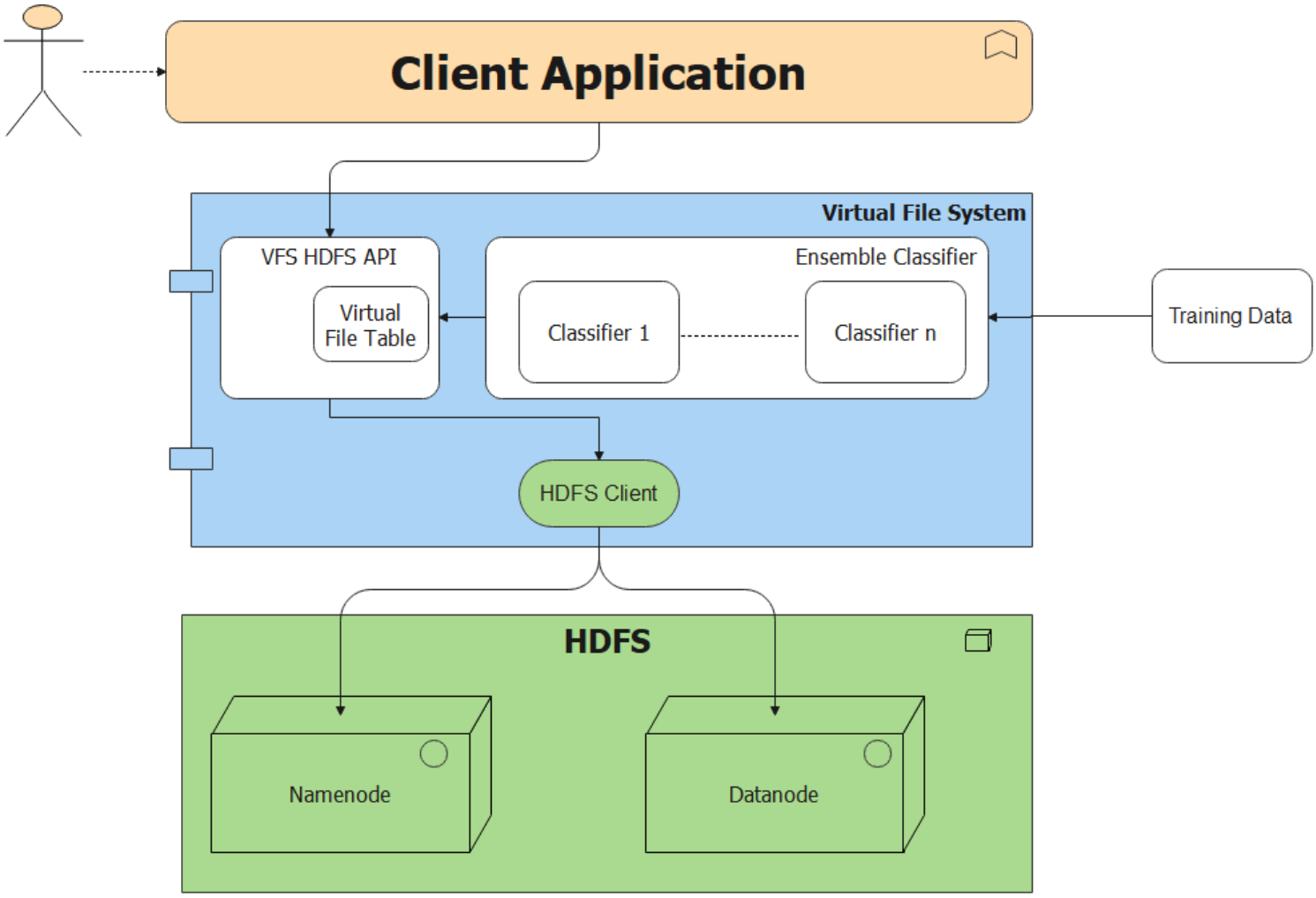

Figure 1

Proposed System Architecture [6]

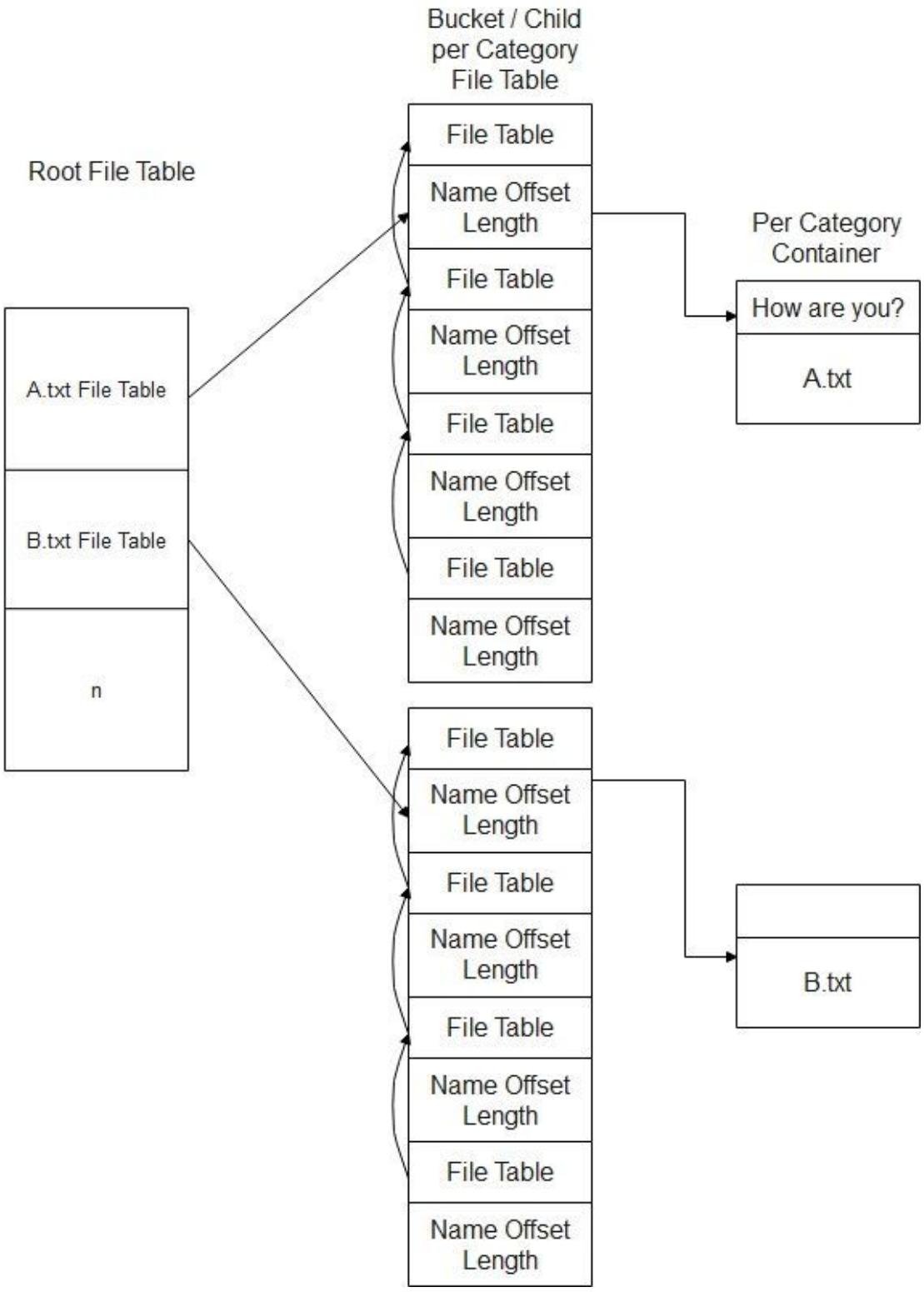

Figure 2

shows the bucket chain technique which is used in the proposed system.

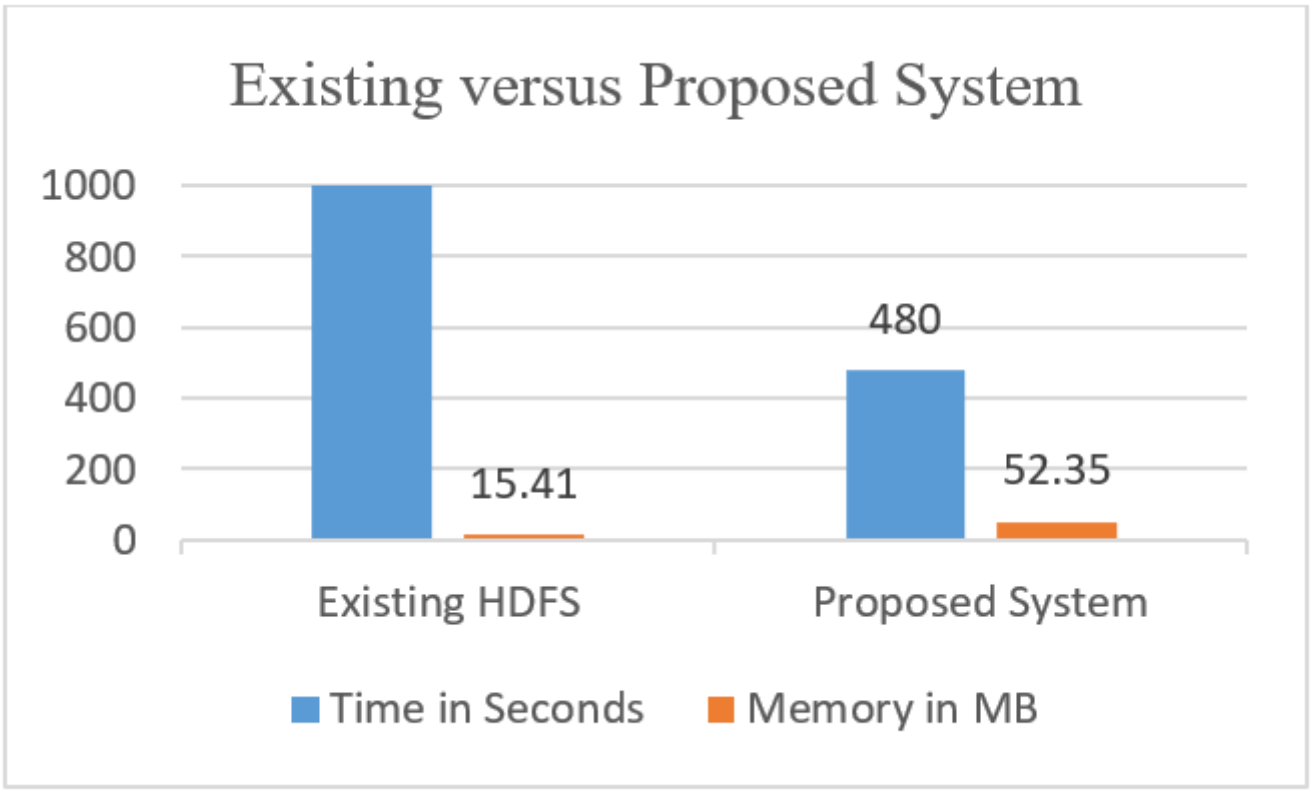

Figure 3

Existing versus Proposed System chart