Access efficiency of small sized files in Big Data using various Techniques on Hadoop Distributed File System platform

Neeta Alange[1†] and Anjali Mathur[2††]
neetaalange@gmail.com anjali_mathur@kluniversity.in
Research Scholar Associate Professor,
Department of Computer Science and Engineering,
Koneru Lakshmaiah Education Foundation,
Vaddeswaram, AP, India.

Abstract
In recent years Hadoop usage has been increasing day by day. The need of development of the technology and its specified outcomes are eagerly waiting across globe to adopt speedy access of data. Need of computers and its dependency is increasing day by day. Big data is exponentially growing as the entire world is working in online mode. Large amount of data has been produced which is very difficult to handle and process within a short time. In present situation industries are widely using the Hadoop framework to store, process and produce at the specified time with huge amount of data that has been put on the server. Processing of this huge amount of data having small files & its storage optimization is a big problem. HDFS, Sequence files, HAR, NHAR various techniques have been already proposed. In this paper we have discussed about various existing techniques which are developed for accessing and storing small files efficiently. Out of the various techniques we have specifically tried to implement the HDFS- HAR, NHAR techniques.

Key words:
HDFS, Flat Table Technique, Table Chain Technique, Small File Merging.

1 Introduction to HDFS

In today's world large amount of data is continuously emerging out which one has to maintain, store, process and analyze. Widely used cloud storage platform known as Hadoop which is open source framework, low cost, fault tolerance, and scalable system. It is basically used for primary data storage for all Hadoop applications. HDFS has master-slave architecture. It has two components: one name node and few data nodes. Name node holds metadata of the files. Name node comes into the picture frequently when client wants to access any data from the files. When we try to store multiple no. of small files, performance of HDFS is reduced. In current scenarios Healthcare centers, scientific fields, Education, Social Media and Industries produce large number of small files which are smaller than the HDFS block. Each file accommodates a separate position. It consumes large amount of space for storing the metadata in name node (150 bytes per file). Name node is called repeatedly to get the Information about files & its corresponding contents in data node. This entire transaction process (figure 1) makes the system slow. Various methodologies are proposed by researchers which are discussed under literature survey section.

Manuscript received July 5, 2021
Manuscript revised July 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.7.41

1.1 Hadoop Distributed File System (HDFS) Architecture

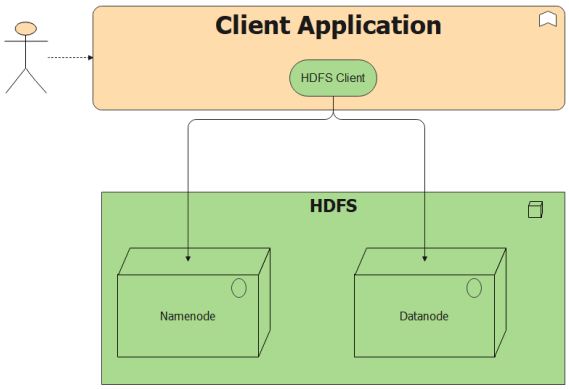

Figure 1: HDFS Architecture

1.2 Features of HDFS:

1. Data Replication
2. Fault Tolerance and Reliability
3. High Reliability
4. Scalability
5. High Throughput
6. Data locality

1.3 Drawbacks of HDFS:

1. **Small Files Problem:** It is not fit for small files. HDFS is deficient to support the random reading of files due to its high capacity design. Small files are smaller than the HDFS block size, default size is 128 MB. If we try to store this huge number of small files, HDFS cannot handle these lots of

small files. If there are many small files, the name node will be overloaded since it stores the metadata of HDFS [9].

2. **File Access Efficiency:** If the client wants to access or read the contents of the file which is stored in HDFS directly. Then following steps are followed by the client. Client sends a requests (along with file path) to the name node. The name node searches the file metadata which is located in its main memory. The name node responds to the client with its metadata. The metadata consists of the files block and its location which are stored in a data node to read the block contents. Client sends the request to the data node to read the contents. The data node returns the result containing the block content to the client [9].

3. **No Caching:** Hadoop is inefficient for caching. MapReduce cannot cache the intermediate data in memory for further requirement and this reduces the Hadoop performance [9].

4. **Network Transfers are more:** Complete Blocks are transferred for client requests [9].
5. **Slow Processing:** MapReduce is responsible for processing a large amount of data. It breaks the processing into 2 phases Map & Reduce. It is a time consuming to perform these tasks by increasing the latency which reduces processing speed [9].

2 Literature Survey:

2.1 HAR (Hadoop Archives)

HAR files are introduced for reducing the problem of multiple files which are putting pressure on the name nodes memory. A layered file system has been put on the HDFS. HAR files are created using Hadoop archive command. Reading through files in a HAR is not more efficient than reading through files in Hadoop. Each HAR file access requires two index files read as well as the data file to read, this makes the process slower[1][2].
HAR is implemented using the Flat Table Technique which is explained in next Section.

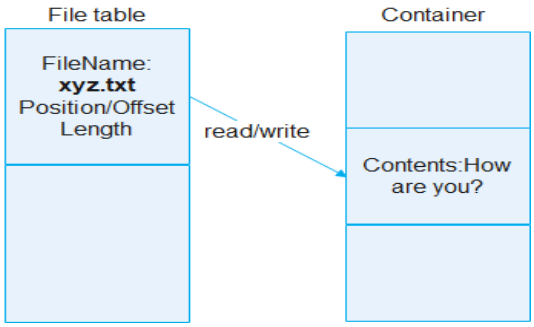

Figure 2: Flat Table Technique

2.1.1 Methodology:

In this method all file's metadata is stored in a single file, this file is called File Table. Another file is used to store actual contents of files, this file is called Container File. Both these files are stored on HDFS. File table contains metadata of a file such as filename, offset, and length. When a new file is created, its data is appended in container file and its offset in container file, length and name are added to file table. When a client requests a file, its metadata is retrieved from file table, which contains file's offset and length. Then from container file the block at offset and length is retrieved and sent back to client. As HDFS doesn't support updating file contents, but it allows to append contents to existing files. So, we can only add new files to file table but cannot update or delete existing files.

2.1.2 Algorithm:

```
globals:
indextable
containerfile
function add_file(filename, content)
location=append_content(containerfile, content)
add_file_entry(indexfile, filename, location,
len(content))
function get_file(filename)
 location, length = find_file(indexfile, filename)
data = read_content(containerfile, location, length)
return data
```

2.1.3 Characteristics:

1. It requires only one container to store all the files.
2. It follows Write Once methodology.

2.1.4 Advantages:

1. Flat File techniques reduce store wastage for small files.

2.1.5 Disadvantages:

1. Time required is more.
2. It follows Write Once Read Many methodology.

2.2 New HAR (NHAR)

To process the large amount of small sized files it requires more time for name nodes. The time required to access such kind of files needs to be reduced actually. The new system called NHAR [9] [11] is dependent on HAR of Apache Hadoop. NHAR is implemented by using table chain technique which is explained below.

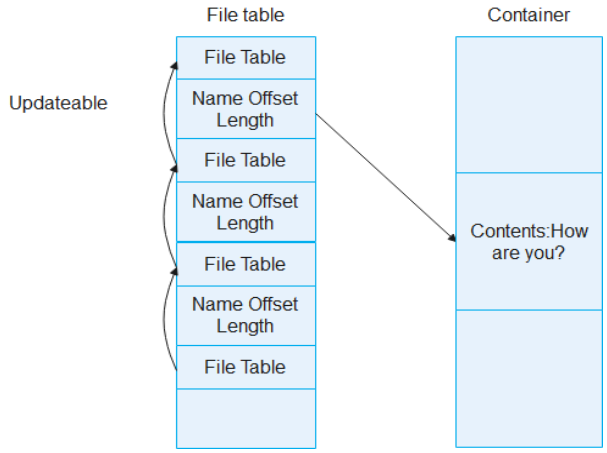

Figure 3: Table Chain Technique

2.2.1 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. Both these files are stored on HDFS. File table contains metadata of a file such as filename, offset, and length.

On startup the file table 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 continued until first file table is reached. While doing this is 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 is created to track new or updated files. In runtime all metadata create and update 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 table are appended to file table in HDFS. This creates a chain of file tables which makes it possible to update and delete files.

2.2.2 Algorithm:

```
globals:
 indexfile
 containerfile
 filetable
 newfiletable

function intialize()
 file_index_location=get_last_index(indexfile)
 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)
 location=append_content(containerfile, content)
 add_file_entry(newfiletable, filename, location,
len(content))

function get_file(filename)
 location, length = find_file(newfiletable, filetable,
filename)
 data = read_content(containerfile, location, length)
 return data

function close()
 last_index=get_last_index(indexfile)
 append_entry_table(indexfile, newfiletable)
 append_prev_index(last_index)
```

2.2.3 Characteristics:

1. It requires only one container to store all the files.
2. It requires a single file table to keep all the records of files. Here containers are responsible for storing the actual contents of the file.

2.2.4 Advantages:

1. File Table is Updatable.
2. Time required is less compared to Flat Table Technique.

2.2.5 Disadvantages:

1. Uses single container for all categories.
2. Uses more memory to store file table.

2.3 Spatiotemporal Small File Merging Strategy

In the existing work Lion Xiong et al. [1] have used usage pattern of small files for selecting candidate files for merging in a single container.

For usage pattern analysis they've used time stamp of file access, and grouped files with sequential time stamps and calculated support file each file group. The file group having large support value is stored in single container.

2.3.1 Advantages:

1. Files which are accessed sequentially are stored in same container.

2.3.2 Disadvantages:

1. This solution provides no caching, this will increase access time.
2. This process needs to be done periodically, which is a time expensive process.

3 Experimental Setup:

To implement these algorithms we have used the following platforms.

Experimental test is performed on a single node.

Table 1: 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	NetBeans 12.0
9	Dataset	Reuter's 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

4 Results:

Table 2: Memory & Time Requirement for Flat Table Technique

For 1000 Iterations	Flat Table	
Average File Size	Memory (in MB)	Time (in Sec.)
1K	16.2	56
5K	16.5	58
10K	17.1	64
50K	19.3	71
100K	27.4	97

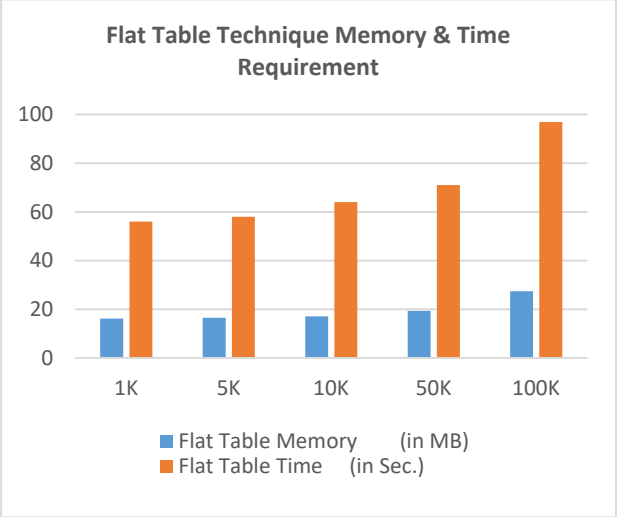

Figure 4: Flat Table Technique Memory & Time Requirement

Figure 4 shows that the memory & Time required for the Flat Table Technique.

Table 3: Memory & Time Requirement for Table Chain Technique

For 1000 Iterations	Table Chain	
Average File Size	Memory (in MB)	Time (in Sec.)
1K	19	45
5K	19.1	45
10K	21.2	48
50K	23.1	57
100K	35.8	70

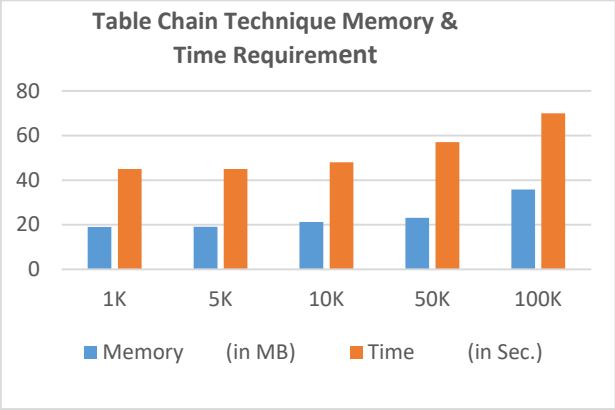

Figure 5: Table Chain Technique Memory & Time Requirement

Figure 5 shows that the memory & Time required for the Table Chain Technique.

Comparative Chart for the Flat Table & Table Chain Technique in terms of Memory & Time.

Table 4: Memory Requirement

For 1000 Iterations Average File Size	Flat Table Memory (in MB)	Table Chain Memory (in MB)
1K	16.2	19
5K	16.5	19.1
10K	17.1	21.2
50K	19.3	23.1
100K	27.4	35.8

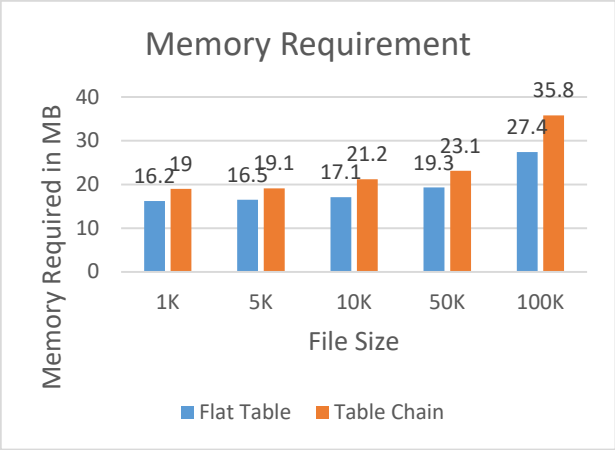

Figure 6: Memory Requirement for Flat Table & Table Chain Technique.

Table 5: Time Requirement

For 1000 Iterations Average File Size	Flat Table Access Time (in Sec.)	Table Chain Access Time (in Sec.)
1K	56	45
5K	58	45
10K	64	48
50K	71	57
100K	97	70

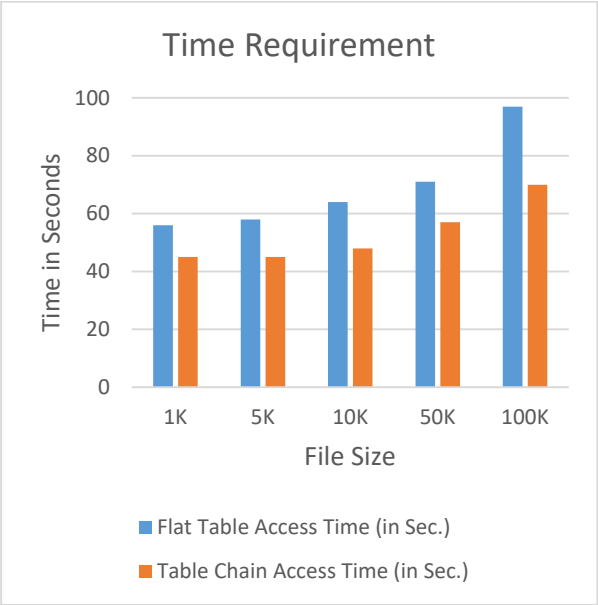

Figure 7: Time Requirement for Flat Table & Table Chain Technique.

Conclusion:

As per the experimental readings it is observed that Table Chain Technique requires more memory in comparison with Flat Table Technique. As well as the access time requirement is lesser in Table Chain Technique compared to Flat Table Technique. From this implementation the observation and result indicates that a new technique needs to be implemented to have the better access efficiency of small sized 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, "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

[3] D Sethia et al "Optimized MapFile Based Storage of Small Files in Hadoop" 17th IEEE /ACM International Symposium on Cluster, Cloud and Grid Computing, 2017, pp 906-912.

[4] Bing et al "A Novel Approach for Efficient Accessing of Small files in HDFS:TLB-MapFile" 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD).

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

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

[7] 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.

[8] Parth et al "A Novel Approach to Improve the Performance of Hadoop in Handling of small files" 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 5-7 March 2015

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

[10] https://data-flair.training/blogs/13-limitations-of-hadoop/

[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] 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.

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

[14] Online Reference https://aws.amazon.com/s3/

[15] Zhipeng et al "An Effective Merge Strategy Based Hierarchy For Improving Small File Problem on HDFS" IEEE Proceedings of CCIS2016, pp. 327-331.

[16] Chatuporn et al "Improving the Performance of Small-File Accessing in Hadoop" 11th International Joint Conference on Computer Science and Software Engineering (JCSSE, 2014), pp.200-205.

[17] 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

[18] Yonghua et al "SFS: A Massive small file processing middleware in Hadoop" IEICE, 18th Asia-Pacific Network Operations and Management Symposium (APNOMS) 2016.

[19] Kashmira et al , "Efficient Way for handling Small Files sing Extended HDFS" International Journal of Computer Science and Mobile Computing, Vol.3 Issue.6, June- 2014, pg. 785-789.

[20] Bo Dong et al "An Optimized Approach for Storing and Accessing Small Files on Cloud Storage". Journal of Network and Computer Applications 35 (2012) 1847–1862.

[21] Kun et al "MOSM: An Approach for Efficient String Massive Small Files on Hadoop" 2017 IEEE 2nd International Conference on Big Data Analysis(ICBDA), 2017

[22] Z. Gao et al "An effective merge strategy based hierarchy for improving small file problem on HDFS", in proceedings of 2016 4th IEEE International Conference on Cloud Computing and Intelligence Systems, CCIS 2016, 2016, pp. 327-331.

[23] Sachin et al "Dealing with small files problem in hadoop distributed file system", Procedia Computer Science Volume 79, 2016, Pages 1001-1012

[24] Tanvi et al "An extended HDFS with an AVATAR Node to handle both small files and to eliminate single point of failure" 2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI), 8-10 Oct. 2015

[25] Passent et al "HDFSX:Big Data Distributed File System with Small Files Support" 2016 12th International Computer Engineering Conference (ICENCO), 28-29 Dec. 2016, pp-131-135

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