iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** V **Month of publication:** May 2018

DOI: http://doi.org/10.22214/ijraset.2018.5314

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue V, May 2018- Available at www.ijraset.com

A Novel Architecture to Efficient utilization of Hadoop Distributed File Systems for Small Files

Vaishali[1] , Prem Sagar Sharma[2]
[1]*M. Tech Scholar, Dept. of CSE., BSAITM Faridabad, (HR), India*
[2]*Assistant Professor, Dept. of CSE, BSAITM Faridabad, (HR), India*

Abstract: Hadoop is known as an open source distributed computing platform and HDFS is defined as Hadoop Distributed File System having powerful data storage capacity therefore suitable for cloud storage system. HDFS was designed for streaming access on large software and it has low storage efficiency for massive small files. For this problem, the HDFS file storage process is improved and therefore files are judged before uploading to HDFS clusters. If the file is of small size then it is merged and its index information is stored in index file in the form of key-value pairs else it will directly go to HDFS Client. Also if all files are processed and no file left to be merged then the merged files go to the HDFS.[7]
Keywords: hadoop, hdfs, small files storage, file processing, file storing

I. INTRODUCTION

Hadoop is known as an open source distributed computing platforms. Its design is basically proposed for managing the big data. It changes the way that any organizations store, process and analyze data. The architecture of Hadoop is scalable, reliable and flexible. It allows data to store and analyze at very high speed. It provides services such as data processing, data access, data governance, security.[1]

HDFS is defined as Hadoop Distributed File Systems. As the name specifies it is distributed file-system that stores data on commodity machines which provides very high bandwidth across cluster. It is high fault tolerance and gives native support of large data sets as well as it stores data on commodity hardware.

[1] Basically it is specially designed file system for storing huge datasets with cluster of commodity hardware with streaming access patterns. Here streaming access patterns means that write once and read any number of times but content of file should not be changed. HDFS has powerful data storage capacity such that it is suitable for cloud storage systems.

HDFS was originally developed for large software and therefore it has low storage efficiency for large number of small files.

A. Hdfs Architecture

HDFS architecture (shown in fig-1) is suitable for distributed processing and storage as well as it provides file authentication and permissions. HDFS is comprised of NameNode, Secondary NameNode, Job Tracker, DataNode, Task Tracker.

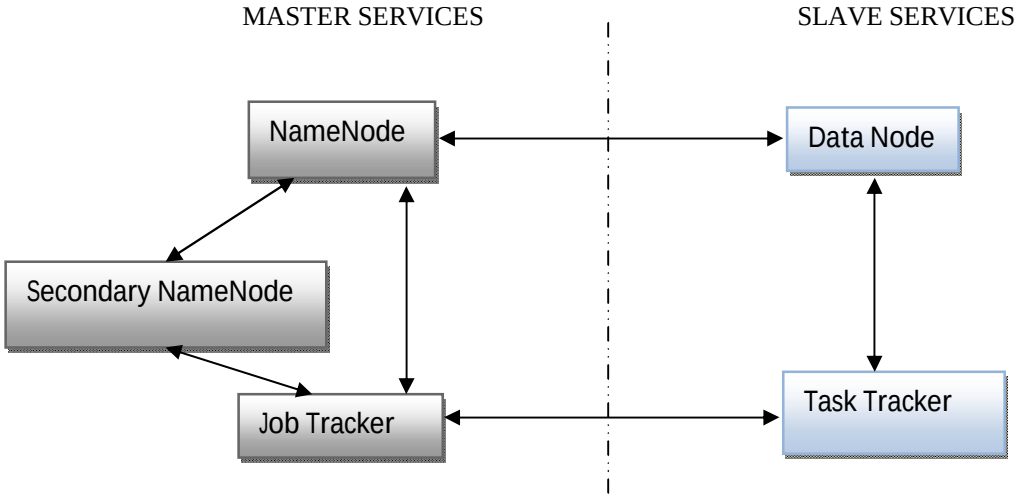


Fig-1: Architecture of HDFS

The communication between master services and slave services is done as it is shown in figure. As all master services can communicate with each other, similarly slave services can communicate with each other. NameNode can communicate with Data Node and vice-versa as well as Job Tracker can communicate with Task Tracker and vice-versa.

1) *NameNode:* NameNode seems like a manager. It creates and maintains the Metadata of the data. It tells client that store data in the available spaces. NameNode maintains three replications of the data including the original copy of data. If any replica is lost due to any failure, in that case, NameNode will create another replica of it.
2) *Secondary NameNode:* It is just a helper node for NameNode so that it helps in better functioning of NameNode. Its purpose is to have a checkpoint of file system Metadata present on NameNode. 'Checkpoint node' is another name of it.
3) *Job Tracker:* Job Tracker basically monitors everything and it knows very well that how many jobs are running. Job Tracker assigns tasks to Task Tracker.
4) *DataNode:* DataNode basically gives block reports and heartbeat to NameNode. If DataNode do not send heartbeat then in that case NameNode will assume that it is dead.
5) *Task Tracker:* Task Tracker is used to receive tasks from Job Tracker. All Task Tracker give some heartbeat back to the Job Tracker every three seconds to tell they are still alive. If they does not inform Job Tracker then in that case Job Tracker will assume that either they are working very slowly or may be dead.

II. LITERATURE SURVEY

Table-1: Summary of Various research papers

S.No	Paper Title	Author	Analysis	Findings
1	A Novel Approach to improve the performance of Hadoop in Handling of Small Files.[1]	Parth Gohil, Bakul Panchal, J.S. Dhobi	Drawback of HAR approach is removed by eliminating big files in archiving. Uses Indexing for sequential file created.	Improves the performance by ignoring the files whose size is larger than the block size of Hadoop.
2	An improved HDFS for small file	Liu changtong china,	Small file problem of original HDFS is eliminated by judging them before uploading to HDFS clusters. If the file is a small file, it is merged and the index information of the small file is stored in the index file with the form of key- value pairs	Access efficiency of NameNode is increased.
3	Dealing with small files problem in Hadoop Distributed File System	Sachin Bendea, Rajashree Shedbeg,	Comparative study of possible solutions for small file problem.	CombinedFileInput-Format provides best performance.
4	Hadoop Architecture and Applications	Kusum Munde, Musrat Jahan	Hadoop has designed to manage Big Data so that it changes the way the enterprises store, process and analyze the data. Hadoop provides services for large data sets such as data processing, data access, data governance, security.	Hadoop supports distributed storage and distributed computing. Data is stored in HDFS which uses block replicas so it is fault tolerant also.

5	An approach to solve a Small File problem in Hadoop by using Dynamic Merging and Indexing Scheme	Shubham Bhandari, Suraj Chougale, Deepak Pandit, Suraj Sawat	In this paper, instead of worn one NameNode for shop the metadata, NHAR uses manifold NameNodes so that NHAR lessen the freight of a sincere NameNode in symbol amount.	NHAR uses manifold NameNodes due to which the Load/NameNode is way fall.
6	Analysis of Hadoop over SAP software solutions	Veena V Deolankar, Nupoor Deshpande, Mandar Lokhande	In this paper, attempt has been made to prove that Hadoop can be used for large data processing as compared to SAP software solutions.	This paper provides the brief introduction of Hadoop and disadvantages of using SAP on large scale industries.
7	Google File System and Hadoop Distributed File System- An Analogy	Dr. A.P. Mittal, Dr. Vanita Jain, Tanuj Ahuja	With ever-increasing data, a reliable and easy to use storage solution has become a major concern for computing. Distributed File Systems tries to address this issue and provides means to efficiently store and process these huge datasets.	To effectively handle hard drive failures, power failures, router failures, network maintenance, bad memory, rack moves, misconfigurations, datacenter migrations across hundreds of thousands or millions of machines requires significantly better error monitoring, tooling and auto recovery.
8	Data Security in Hadoop Distributed File System	Sharifnawaj Y. Inamdar, Ajit H. Jadhav, Rohit B. Desai, Pravin S. Shinde, Indrajeet M. Ghadage, Amit A. Gaikwad.	In this, security of HDFS is implemented using encryption of file which is to be stored at HDFS so a real-time encryption algorithm is used. Encryption using AES results into growing of file size to double of original file and hence file upload time also increases so this technique removes this drawback.	Encryption/decryption, authentication and authorization are the techniques those much supportive to secure information at Hadoop Distributed File System. In future work, subject prompts produce Hadoop with a wide range of security techniques for securing information and additionally secure execution of job.

A. Limitation in Existing System

1) Decreasing the performance by accepting the small files and large files (i.e. whose size is smaller and larger than the block size of Hadoop respectively).

2) Access time for reading a file is greatly high.

3) Access efficiency is low of NameNode.

4) There is no automatic fault tolerance for NameNode failure.

5) As compared to GFS, hadoop cluster provides less error monitoring, tooling, and auto-recovery to effectively handle hard drive failures, power failures, router failures, network maintenance, bad memory, rack moves, misconfigurations, datacenter migrations across hundreds of thousands or millions of machines.
6) Hadoop not have any sort of security system so it uses techniques like encryption/decryption, authentication and authorization to secure information at Hadoop Distributed File System.

III. PROPOSED ARCHITECTURE FOR SMALL FILE STORAGE

In the improved architecture of HDFS (shown in fig-2), it basically has three layers: user layer, data processing layer and the storage layer based on HDFS.

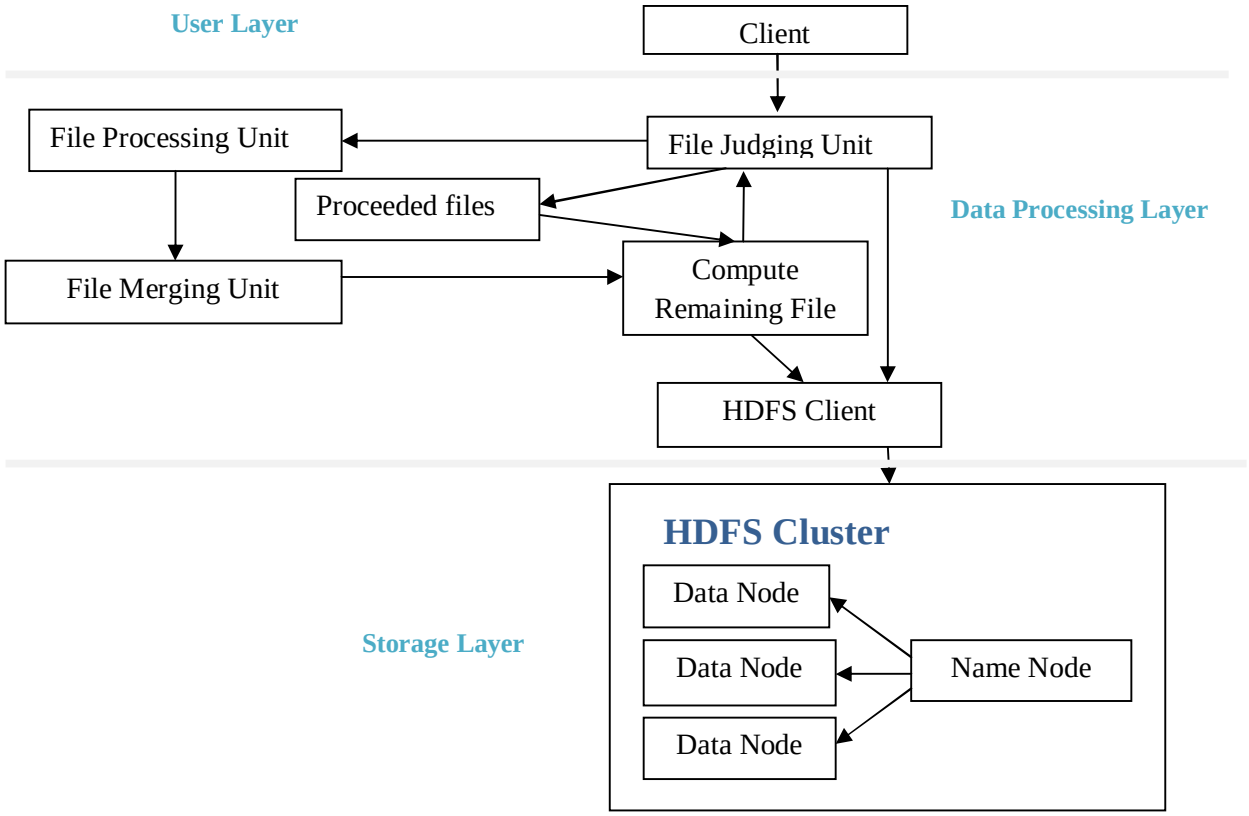


Fig-2: Proposed architecture of HDFS for small files

1) *User Layer:* This layer considered as the entry of the input of the whole store system which gives an interface to the user to upload file, browse file and download file
2) *Storage Layer:* This is the place where data resides and also it is the most critical layer of the whole store system. It have HDFS server which provides reliable and persistent storage capabilities.

A. *Processing Layer*

HDFS has the limited capability to support small file, this layer is designed basically for the same purpose. It mainly has four functional units:

1) File Judging Unit
2) File Processing Unit
3) File Merging Unit
4) HDFS Client.

B. *Index File for Files*

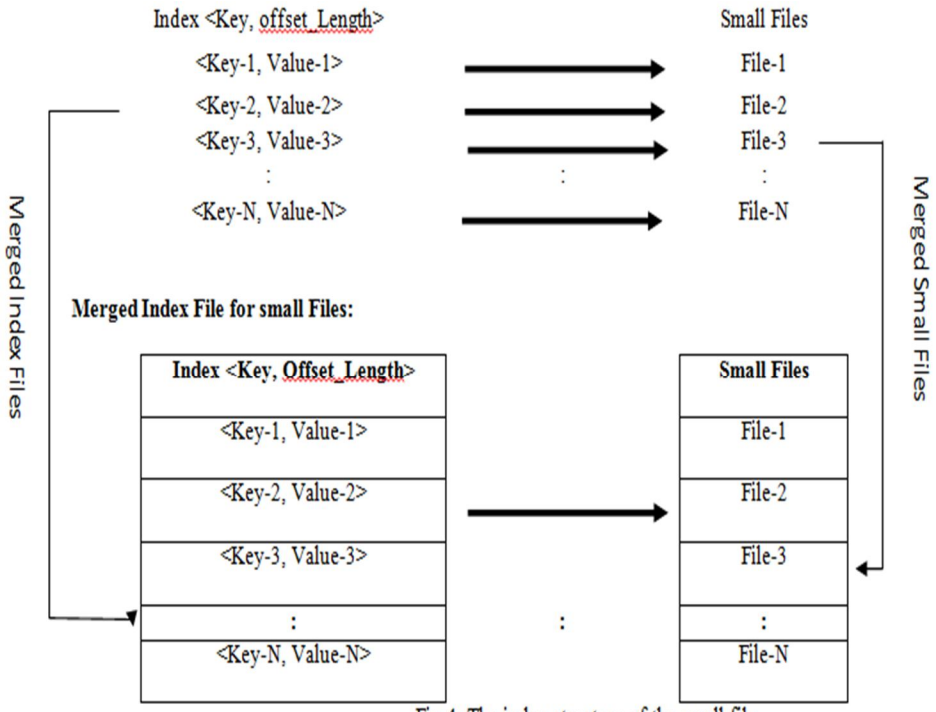


Fig-4: The index structure of the small file

1) *File Judging Unit:* Its main purpose is to determine the size of the file and checks if there is any need of merging process. It the file of user is of small size then there is need of merging process and that file is sent to the file processing unit. Otherwise, the file is directly sent to HDFS Client.[7]
2) *File Processing Unit:* The functions of file processing is to receive files from the file judging unit, it counts the size of the files, on the basis of order and size of small files it form an incremental offset from start and generate temporary index file (TempIndex). When the size and offset of small files is stored, it sends the small files and the corresponding temporary index sequentially to the file merging unit.[7]
3) *File Merging Unit:* The main function of this unit is merging the files. It merges the small files according to the order of small files. The temporary index files are merges generating a merged index file as shown fig-4. The actual content of small files is stored in the merged file. It records the merged index files of small files by <key, value> format. There is a unique value for retrieving small files which records the critical information of the small file and it is known as key. This records the offset of small file and length of small file. The end position of small file can be derived as "key_value + offset_length".
   When the small file is read then the key in merged index file is obtained according to the name of the small file then value is got by the key and resolved to obtain the offset and length of small file so the end position of small file can be derived. The small file in the merged file can be got by the start and end position of small file.
   There also exist a block which check out the proceeded files and hence update the counter by reporting to file judging unit that the files are processed and side by side it computes the remaining files so that in any case if number of files finishes then it directly sends them to the HDFS client otherwise files will go to file judging unit.
4) *HDFS Client:* HDFS write the received data in the stored layer. It establishes a connection between NameNode and DataNode with distributed file system instance. It notifies NameNode that which DataNode is used to write data block. The data writing operation is completed by calling the relevant document operating API provided by HDFS. If the file is of large size then the processing is same as writing otherwise the merged file and the corresponding merged index file are stored together on the same DataNode. The id of merged file and merged index file is same and the merged index file is protected by DataNode which is transparent for NameNode. When merging index file is successful then merged file is loaded into the memory. To speed up the read speed after first reading of small files, the merging index file is loaded into the cache. The merge block

index file is very small and the number of data blocks on the DataNode is limited as compared with entire cluster therefore these index files occupies very little memory of DataNode.[7] File processing flow in processing layer is shown in figure-3.

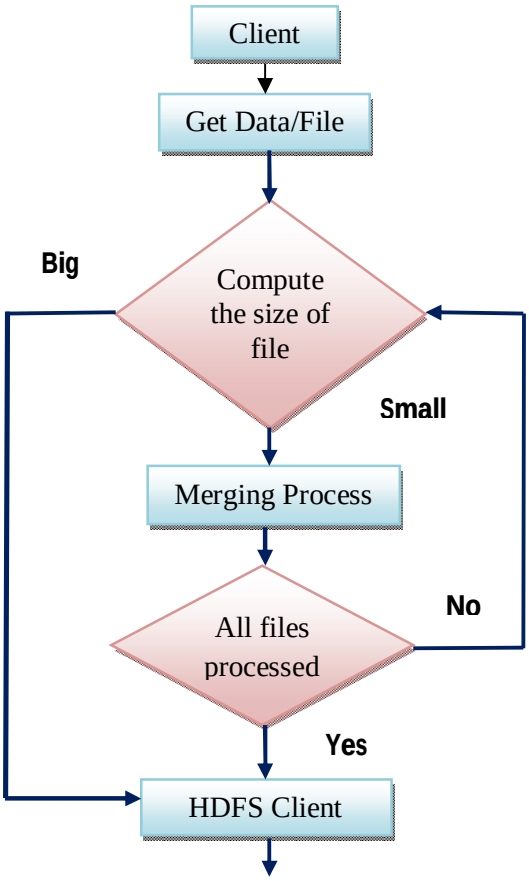


Fig-3: Processing flow in Data Processing Layer

The system takes the input from the user so that it gets data/file from the user. After get data/file from the user, compute the size of file. If the size of file is small then send this for merging process, if size of file is big then directly send data to the HDFS Client. Now, when small files merge by the merging process, so it will check for the condition that all the files are processed or not. If all files processed then it directly sends them to the HDFS Client else it will send them to the size computation unit of file and then again checks for the size of file and performs merges operation if needed.

PROCEDURE: Data Processing Layer
[For all Files of the client]

STEP-1: Read i^{th} file/doc of client

$size\_of\_file_i = fileJudging\_Unit(file_i)$

[File Processing Unit]

STEP 2: Check Condition for Merging

If ($size\_of\_file_i$ < threshold) // File is small
Then:
else if(allFileProcessed!=True)
Then
$merged\_block\_index = FileMergingUnit(file_i , size\_of\_file_i)$
data= merged_block_index;
return(data)

End Else if
End If
else
do
data=file$_i$;
STEP 3: [HDFS Client is to write the received data to the HDFS in the stored layer]
hdfs_Client (data);
End Else
STEP 4: Repeat Steps 1-3 while all files are not proceed.
STEP 5: Exit.

STEP-1: Read the document file of the client so that file goes to the File Judging Unit to check the size of file so that it can go for further processing to File Processing Data.
STEP-2: Now it check condition for merging
If threshold value is greater than size of file then it means that file is small. If all files are not processed then merged_block_index get the file and also the size of file and data is stored in merged_block_index.
Otherwise if size of file is greater than the threshold value then file of client will directly stored in the data.
STEP-3: Now HDFS Client is to write the received data to the HDFS in the stored layer that is hdfs_Client (data).
STEP-4: Repeat the step from 1-3 until all files are processed.
STEP-5: Exit.

C. Comparative Study and Analysis

Table 2 shows the comparative analysis of five methods to deal with small files problem in HDFS. The very first method HAR provides high scalability by reducing namespace usage and reading efficiency of files. There is a drastic change in reduce operation before and after archiving files which shows that there is increase in performance time.
With proposed architecture for efficient utilization of Hadoop Distributed File System writing and accessing performance of small files greatly increases and the average memory usage ratio of proposed architecture of HDFS is decreases as compared to original HDFS.

Table-2: Comparison & Analysis

Paper Name → / Parameters Used ↓	Reduction of data at Namenode in HDFS using Harballing Technique [41]	An improved small file processing method for HDFS[42]	Efficient Way for Handling Small Files using Extended HDFS[43]	Improving Performance of small-file Accessing in Hadoop[44]	An Innovative Strategy for Improved Processing of Small Files in Hadoop[45]
Method	Archive-Based	Index-Based	Index-Based	Archive-Based	InputFormat-Based
Positioning	Name Node	Data Node	Name Node	NameNode	Name Node
Memory Usage	Very Low	Low	Moderate	Slightly High	High
Reading Efficiency / Addressing Time	Moderate	Moderate	High	High	Very high
Performance	Moderate	Moderate	High	High	Very high
Overhead	Slight	High	Low	Low	Slight

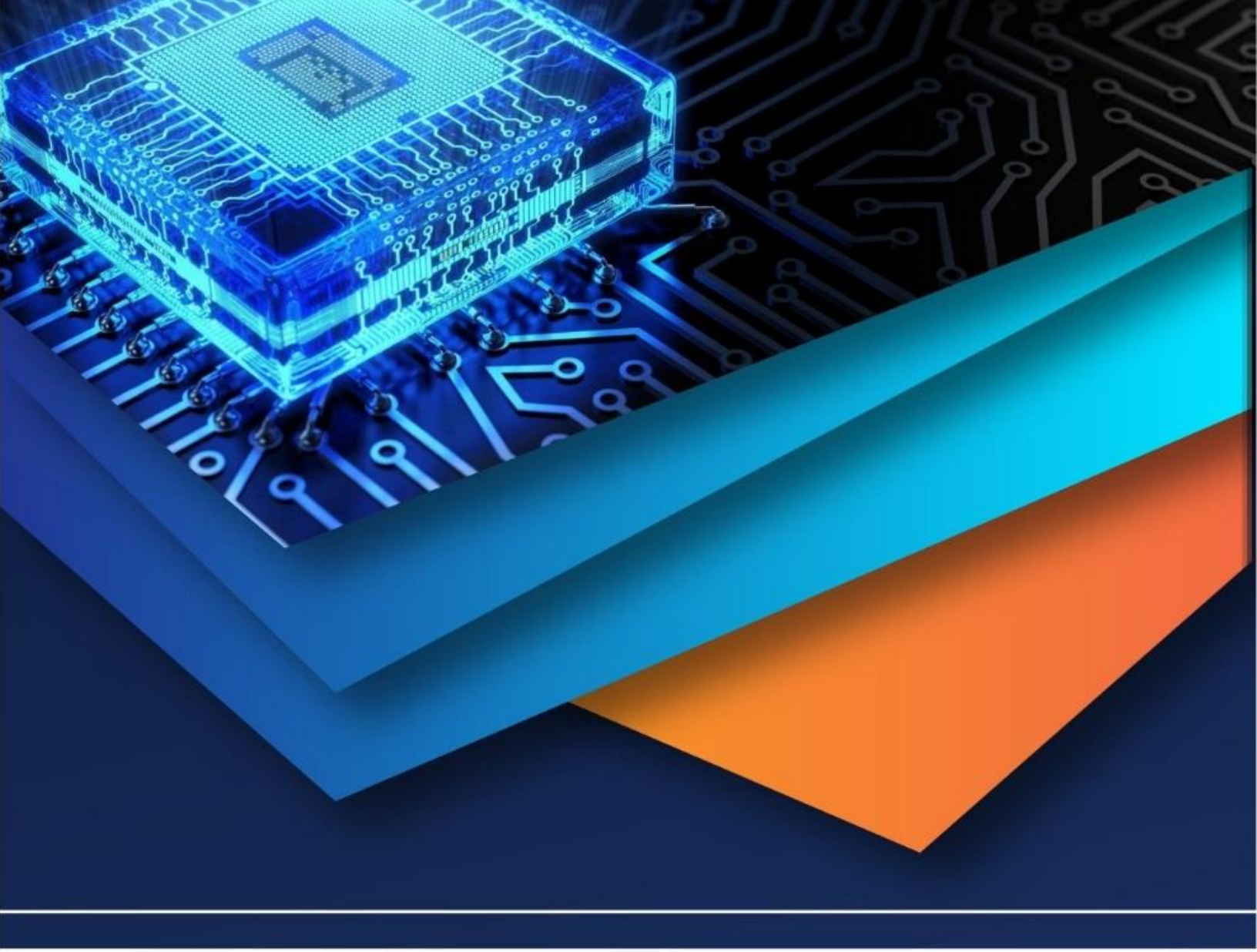
Proposed HDFS architecture allows for greater utilization of HDFS resources by providing more efficient metadata management for small files. Proposed Architecture only maintains the file metadata for each small file and not the block metadata. The block metadata is maintained by the NameNode for the single combined file alone and not for every single small file. This accounts for the reduced memory usage in the Proposed HDFS Architecture. It can improve the efficiency of accessing small files and reduces the metadata footprint in NameNode's main memory.

IV. CONCLUSION AND FUTURE WORK

This paper gives an insight of the architecture of storage of small files in Hadoop Distributed File Systems by providing efficient metadata management. In this paper the storage of file done on three layers of HDFS architecture. The HDFS file stored process is improved. It maintains the file metadata for each small file and not the block metadata. The block metadata is maintained by the NameNode for the single combined file alone and not for every single small file. This accounts for the reduced memory usage in the Proposed HDFS Architecture. It provides better access and storage efficiency of small files. In future work, the implementation of the above will be done.

REFERENCES

[1] Munde, Kusum; Jahan, Nusrat; "Hadoop Architecture and Applications", IJIRSET (International Journal of Innovative Research in Science, Engineering and Technology), Nov-2016, ISSN(Online): 2319-8753, ISSN(Print): 2347-6710, pp19090-19094.
[2] Bhandari, Shubham; Chougale, Suraj; Pandit, Deepak; Sawat, Suraj; "An approach to solve a Small File problem in Hadoop by using Dynamic Merging and Indexing Scheme", IJRITCC (International Journal on Recent and Innovation Trends in Computing and Communication), Nov-2016, ISSN: 2321-8169, pp227-230.
[3] Y. Inamdar, Sharifnawaj; H.Jadhav, Ajit; B.Desai, Rohit; S.Shinde, Pravin; M.Ghadage, Indrajeet; A. Gaikwad, Amit; "Data Security in Hadoop Distributed File System", IRJET (International Research Journal of Engineering and Technology), Apr-2016, e-ISSN: 2395-0056, p-ISSN: 2395-0072, pp939-944.
[4] Mittal, A.P.; Jain, Vanita; Ahuja, Tanuj; "Google File System and Hadoop Distributed File System-An Analogy", IJIACS (International Journal of Innovations & Advancement in Computer Science), March-2015, ISSN 2347-8616, pp626-636.
[5] Dhaulakhandi, Prachi, "A Study of Hadoop Ecosystem", IJRSM (International Journal of Research & Management), Aug-2016, ISSN: 2349-5197, pp9-12.
[6] V.Deolankar, Veena; Deshpande, Nupoor; Lokhande, Mandar; "Analysis of Hadoop over SAP Software Solutions", IJRSR (International Journal of Recent Scientific Research), Mar-2016, ISSN: 0976-3031, pp9212-9215.
[7] Changtong, Liu, "An improved HDFS for small file", ICACT, Feb-3, 2016, pp478-481.
[8] Gohil P, Panchal B, Dhobi J S., "A novel approach to improve the performance of Hadoop in handling of small files", Electrical, Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference on. IEEE, pp. 1-5.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY