IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-6 E-ISSN: 2347-2693

An Efficient Approach to Optimize the Performance of Massive Small Files in Hadoop MapReduce Framework

Guru Prasad M S[1*], Nagesh H R [2], Swathi Prabhu [3]

[1]Computer Science & Engineering, SDMIT, VTU-Belagavi, Ujire, India
[2] Computer Science & Engineering, MITE, VTU-Belagavi, Moodbidri, India
[3]Computer Science & Engineering, SMVITM, VTU-Belagavi, Udupi, India

[*]*Corresponding Author: guru0927@gmail.com, Tel.: +91-96869-30845*

Available online at: www.ijcseonline.org

Received: 25/May/2017, Revised: 02/Jun/2017, Accepted: 20/Jun/2017, Published: 30/Jun/2017

Abstract— The most popular open source distributed computing framework called Hadoop was designed by Doug Cutting and his team, which involves thousands of nodes to process and analyze huge amounts of data called Big Data. The major core components of Hadoop are HDFS (Hadoop Distributed File System) and MapReduce. This framework is the most popular and powerful for store, manage and process Big Data applications. But drawback with this tool related to stability and performance issues for small file applications in storage, manage and processing the data. Existing approaches deals with small files problem are Hadoop archives and SequenceFile. However, existing approaches doesn't give an optimized performance to solve small files problems on Hadoop. In order to improve the performance in storing, managing and processing small files on Hadoop, we proposed an approach for Hadoop MapReduce framework to handle the small files applications. Experimental result shows that proposed framework optimizes the performance of Hadoop in handling of massive small files as compared to existing approaches.

Keywords-Hadoop, Hadoop Distributed File System (HDFS), MapReduce, Hadoop Archives, Sequence File, Small Files.

I. INTRODUCTION

Big Data is a new phrase used to describe a massive volume of structured, unstructured and semi-structured data. In the year 2012, 2,72,000 Exabyte's of digital data were stored in the world. Today digital world data is exploded to 10,00,000 Exabyte's, International Digital Corporation (IDC) is estimated that digital world data will reach 35,00,000 Exabyte's by 2020 [1]. Traditional computing technologies are becoming inadequate to store, manage and process massive data sets [4]. Highly need of distributed computing framework to handle massive data sets. Today Hadoop is the most promising popular and powerful distributed computing framework. It is an Apache foundation framework based on implementation of MapReduce parallel programming model [2]. This model is an increasingly popular technology to process and analyze large massive data sets. It provides a reliable, scalable and robust computing framework. The major core components of Hadoop are HDFS and MapReduce. HDFS read large files as an input data, later divides the large file into data blocks (128 MB default) and stores the data blocks in the computing nodes. MapReduce is a programming model for application which processes data blocks in parallel [1-22].

The design of Hadoop is such that it provides a high performance to store and process large file applications. But Hadoop is not bound for large file applications because many applications generates small files few example are, weather sensors producing files which of size normally start from a few kilobytes to tens of megabytes [11]. In on-line education tutorials most file stored are power point and pdf whose average file size in between 5-10 megabytes. Social network servers like Facebook, Whatsapp, Instagram, etc., contains billions of images, each image size is less than 5 MB. Most videos size stored in You Tube are less than 50 MB. A small file is a file whose size is less than 50% of the HDFS block size which of size 128MB. But major issue here is Hadoop does not provide optimal performance for small file applications. Hadoop NameNode was designed to store metadata and data blocks information, each meta data occupies 150 bytes of memory [12-22]. When there are millions of small files, storing metadata and block information will impact on the allocated memory for

NameNode in RAM. HDFS stores the data blocks in computing nodes, too many small files will overhead the network traffic and consumes more time to store. MapReduce program was designed to process data blocks reside in computing nodes, large number of small files will create an overhead between MapReduce tasks and consumes more time to process. Some approaches such as Hadoop Archives [9], SequenceFile [10], were proposed on Hadoop to handle small files, but they did not give optimal performance to store, manage and process small file applications.

In this paper, we propose an optimized Hadoop MapReduce framework that will solve small files problems on Hadoop. It consists of two techniques they are File Manager and MapCombineReduce (MCR). File Manager provides four functions such as File Integrator, File Read, File Modify, File Delete. It solves memory pressure of NameNode and overhead of network traffic. MCR is a data processing technique proposed to process data blocks. It solves overhead between MapReduce tasks and improves the performance of data processing.

The rest of the paper is organized as follows. Section II is an illustration of related works about the proposed topic. Section III discuss the problem statements and existing approaches. Section IV proposes the proposed architecture. Section V presents comprehensive experiment results. Section VI concludes the paper.

II. RELATED WORK AND MOTIVATION

Fang Zhou, et al [3] described that Hadoop can easily store and analyze large files. However, it cannot provide good performance for small files on storage and process levels. To solve these problems authors proposed an SFMapReduce framework for small files. Authors presents two techniques, small file layout and customized MapReduce. The First technique solved the NameNode memory problem and second technique efficiently process data with SFlayout. Experimental results depict that SFMapReduce performance is 14.5 times better, compared to the original Hadoop and 20.8 times better compared to the HAR layout. In this paper authors solved the NameNode memory problem to a good level, but they didn't concentrate on network traffic between the computing nodes. The Authors haven't compared their work with the SequenceFile layout.

Xiaoyong Zhao, et al [4] illustrates that distributed storage required to store trillion GB of data. Hadoop is the robust framework to store and process large massive data. However, it is not suitable for handling massive small files. Authors presents a novel metadata-aware storage architecture on Hadoop for handling massive small files. This architecture includes merge module, first index module and metadata manage module. The proposed architecture on Hadoop decrease memory pressure on NameNode. In this paper authors concentrated only on metadata management on NameNode and they haven't concentrate to solve the small file processing issue. Experimental results didn't compare with any traditional approach and considered only MP3 files.

Kun Gao, et al [5] says that GIS technology generates massive small files, traditional technology cannot meet the demand of storage and processing of massive files. Today Hadoop is a leader to handle large massive files. But some factors impact on Hadoop to store and process massive small files. To improve Hadoop performance authors proposed Hilbert space filling curve to convert two dimensional data to one dimensional data. Authors concludes that the proposed method improves the efficiency of data retrieval. In this paper authors not compared their work with traditional methods. The proposed method solved only indexing problems on Hadoop, but the data processing problem remains unchanged.

Parth Gohil, et al [6] describes that Hadoop is a rising distributed computing framework to deal Big Data. Authors mentioned about massive small files problems i.e. Massive small files occupy more NameNode memory and MapReduce consumes more time to complete jobs. To solve performance issues, they proposed an approach. The proposed approach, merged all small files into a single large file. Experimental results illustrate that, proposed method reduces the memory pressure on NameNode and improves the MapReduce performance compared to traditional methods. In this paper author not mentioned or identified other parameter effect the Hadoop performance.

III. PROBLEM STATEMENT AND EXISTING APPROACHES

A. Problem Statement

Hadoop does not provide optimal performance for small files processing. Massive small files will impact on the following factors.

1. Impact on NameNode memory

HDFS divides a large file into data blocks. By default, it is 128 MB and stores the data blocks in the computing nodes. It consists of NameNode, which is the master node and Data Node, which is the computing nodes. NameNode was

designed to store each data block's metadata information. Metadata is an information of the data block like data block name, data block id, creator, created, permission, timestamp, etc. It occupies 150 bytes of the NameNode in RAM memory. If a file size less than the HDFS block size, even then HDFS treats it as one block. In this way as the number of small files stored in HDFS increases lots of metadata related to this blocks will be in NameNode RAM memory. For example, assume that 1,000,000 small files stored in HDFS then 143 MB of metadata is stored in NameNode memory. Massive small files will significantly impact on NameNode memory

2. Impact on MapReduce performance

HDFS will consider each small file as one block. For each block one Map task will be created. For massive small files large number of Map tasks will be created. It is an overhead for processing. This will bottleneck the overall MapReduce performance compared to large file processing.

B. Existing approaches

Existing approaches to handle small files in Hadoop are as follows:

1. Hadoop Archives

Hadoop Archives (HAR) [5] is an approach to alleviate the small files problem on Hadoop. It archives massive small files into HDFS blocks. The Hadoop Archive data format shown in figure 1. It has three layouts: a master index that stores hashes and offsets of the index which stores files information, a data that stores the actual data. The advantages of HAR are: It reduces the memory pressure on the NameNode and allows parallel access to archive files. The disadvantages of HAR are: HAR files are immutable, once an HAR file created not possible to add or remove file and HAR file used as input to MapReduce, but it difficult to read HAR files as compared to HDFS file.

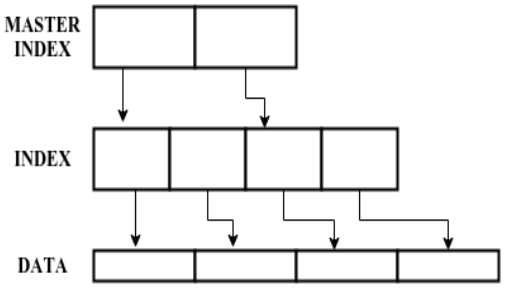


Fig 1. Hadoop Achieve Data Format

2. SequenceFile

SequenceFile [5]is an approach to solve the small files problems. It combines massive small files into a flat file. The SequenceFile data format shown in figure 2. It has key/value pairs, key is file name and value is file contents. It has three different formats: uncompressed key/value pairs, record compressed key/value pairs and block compressed key/value pairs. The advantages of SequenceFile are: It reduces memory pressure on the NameNode and overhead between MapReduce jobs. The disadvantages of SequenceFile are: If you have 1000 small files, then SequenceFile could contains 1000 keys, one key per file and if SequenceFile is compressed, MapReduce jobs consumes more time to process.

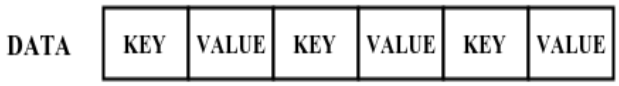


Fig 2. SequenceFile Data Format

IV. PROPOSED ARCHITECTURE

This section presents the techniques related to proposed architecture and implementation details. The proposed architecture is as shown in figure 3 which expands the baseline Hadoop MapReduce framework. It states that small files problem can be handled more effectively using two techniques.

- File Manager.
- MapCombineReduce

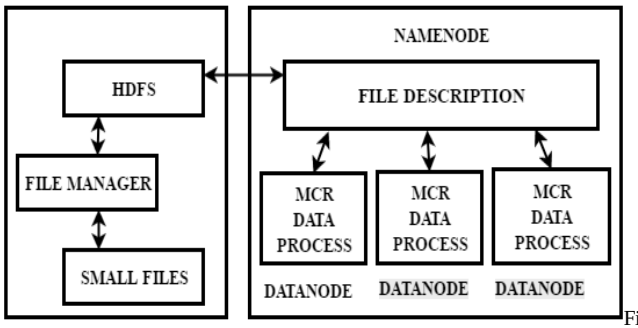


Fig 3. Proposed Architecture

A. File Manager

The proposed File Manger technique focuses effectively on managing metadata in NameNode. This technique is to solve memory pressure on NameNode, to optimize HDFS time to distribute files to computing nodes and to provide mutable property to HDFS files. File Manager provide four functions; they are as follows:

- File Integrator.
- File Read.
- File Modify.
- File Delete.

1. File Integrator

File Integrator is the first and most important function of File Manager. It is an innovative file layout designed to combine small files, whose size more than 80% of HDFS block size. The File Integrator data format shown in figure 5. It has two layouts an index and a data. Index includes information of file such as file id, file name, file size, offset and creation time. Data stores actual data of size 100MB. It efficiently manages metadata on NameNode, solves memory pressure of NameNode and optimize HDFS time to distribute files to computing nodes. Algorithm of File Integrator and flow chart (figure 4) is as follows:

Algorithm: File Integration

Step 1: Start the File Integration process

Step 2: Create a new file k++, where k=1.

Step 3: Read the contents of small file i, where i=1 to n (where n indicates total number of small files).

Step 4: if i value less than or equal to n then
goto step 5
else
goto step 8

Step 5: if (file size of i + file size of k) less than 100 MB then goto step 6
else goto step 2

Step 6: Write file i content to file k, increment i value.

Step 7: if file size of k greater than or equal to 100 MB then goto step 2
else goto step 3

Step 7: End of File Integration process

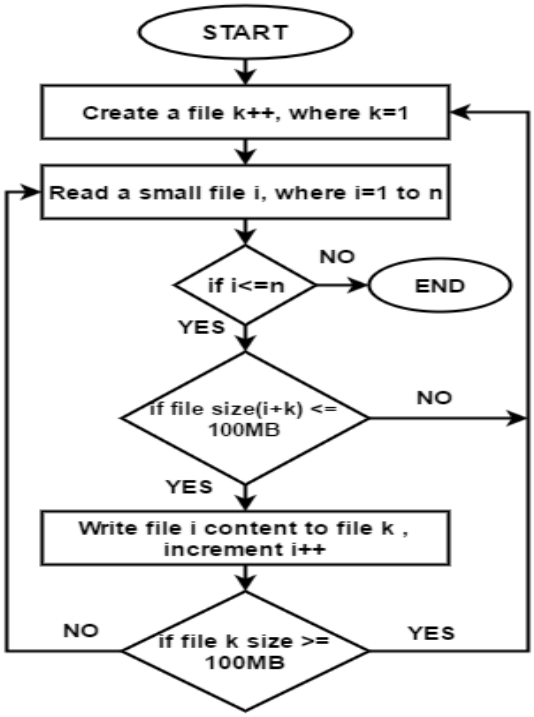


Fig 4. File Integrator Flow Chart

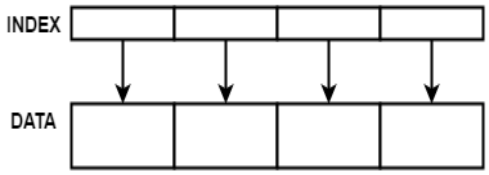


Fig 5. File Integration Data Format

2. File Read

File Read is the second function of the File Manager. Sometimes user need to read some files from HDFS, user have to run read program in the command line. Here, the user has to enter the file name in command line. File Read program will read file name and search it in File Integration data format. If file found, it will map file name with HDFS block id. Next, it will find the file content inside HDFS block and display the file content. Algorithm of File Read and flow chart (figure 6) is as follows:

File Read algorithm is as follows:

Step 1: Start the File Read process.
Step 2: Enter the file name to read.
Step 3: Search that file name in File Integration data format
If found goto step 4
Else goto step 7.
Step 4: Map file name with HDFS block id
Step 5: Find the file content inside the HDFS block.
Step 6: Display the file content.
Step 7: Display File name not found
Step 8: End of the File Read process.

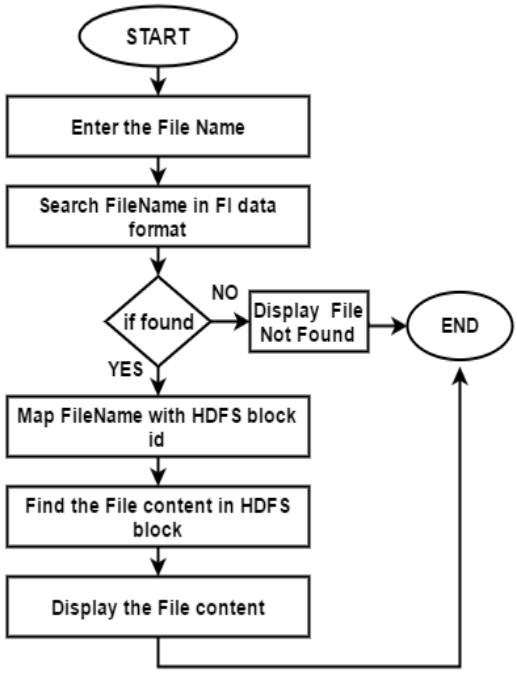


Fig 6. File Read Flow Chart

3 File Modify

File Modify is the third function of the File Manager. If user wishes to modify some files in HDFS, user have to run modify program in the command line. Here, the user has to enter the file name in command line. File Modify program will read file name and search it in File Integration data format. If file found, it will map file name with HDFS block id. Next, it will find the file content inside HDFS block and display the file content. Now, user can easily modify the file content and save the file back in HDFS. Algorithm of File Modify and flow chart (figure 7) is as follows:

Algorithm: File Modify

Step 1: Start the File Modify process.
Step 2: Enter the file name to modify.
Step 3: Search that file name in File Integration data format
If found goto step 4
Else
goto step 8.
Step 4: Map file name with HDFS block id
Step 5: Find the file content inside the HDFS block.
Step 6: Display the file content.
Step 7: User can modify and save the file.
Step 8: Display File name not found.
Step 9: End of the File Read process.

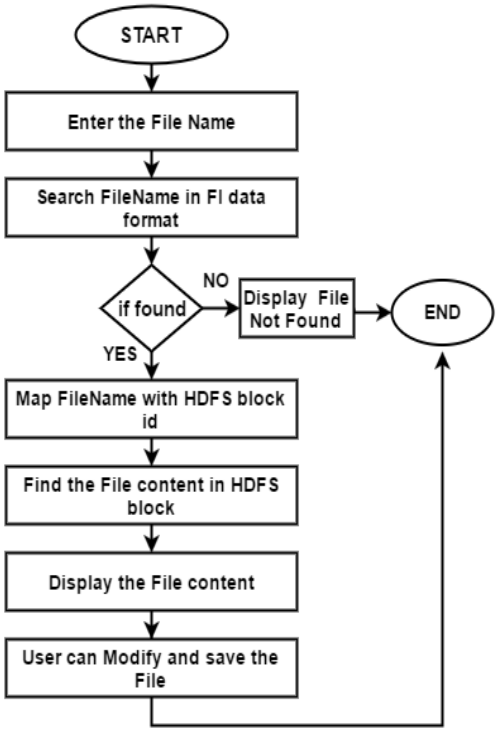


Fig 7. File Modify Flow Chart

4 File Delete

File Read is the fourth function of the File Manager. Sometimes user wants to delete some corrupt files in HDFS, user have to run delete program in the command line. Here, the user has to enter the file name in command line. File Delete program will read file name and search it in File Integration data format. If file found, it will map file name with HDFS block id. Next, it will find the file content inside HDFS block and display the file content. Now, user can easily delete the file. Algorithm of File Delete and flow chart (figure 8) is as follows:

Algorithm: File Delete

Step 1: Start the File Delete process.
Step 2: Enter the file name to delete.
Step 3: Search that file name in File Integration data format
If found goto step 4
Else goto step 8.
Step 4: Map file name with HDFS block id
Step 5: Find the file content inside the HDFS block.
Step 6: Display the file content.
Step 7: User can delete the file.
Step 8: Display File name not found.
Step 9: End of the File Read process.

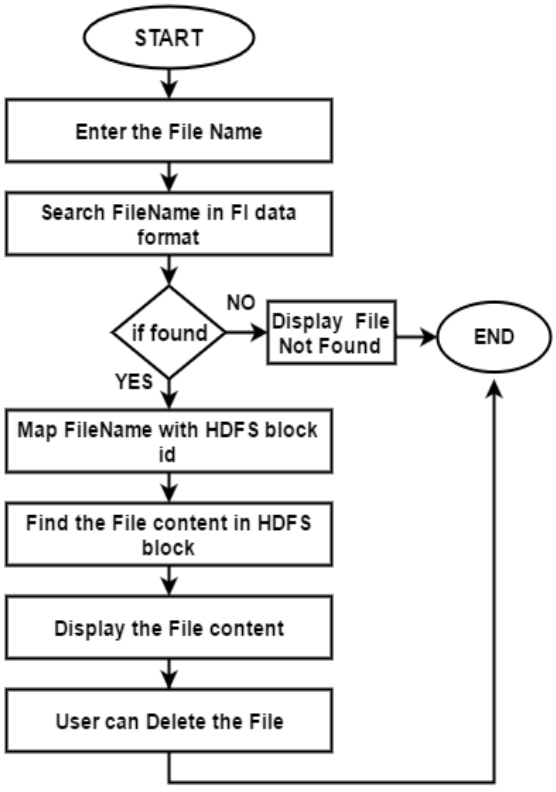


Fig 8. File Delete Flow Chart

V. RESULT AND DISCUSSION

The Performance of the Hadoop MapReduce framework with respect to the time taken to load data from local file system to the Hadoop Distributed File System, memory consumption of the NameNode and the data process time was initially benchmarked with the original Hadoop, Hadoop Archives (HAR), SequenceFile and then compared with results obtained using the Proposed Approach (Optimize Hadoop).

A. Experimental Environment

For the performance analysis, the test platform contains a Hadoop five node cluster with homogeneous hardware property, i.e. Each node in the cluster has a 3.8 GB RAM, Intel® Core i5 3470 CPU @3.20GHz * 4 processor. We setup cluster on Ubuntu 16.03 with Hadoop 1.7.2 stable release used the oracle jdk 1.8 and ssh configuration to manage Hadoop daemons. Our cluster setup is having 1 NameNode and 5 DataNodes for the purpose of an experiment. Configuration files such as mapred-site.xml, core-site.xml, hdfs-site.xml and yarn-site.xml are setup by default values with replication factor 2 and data block size 128 MB.

B. Workload Overview

The workload for the experiment contains a total of 3,900 files. The size of these files, range from ≥ 100 KB to ≥ 10 MB. The collective size of all files is 10GB which comprises image, document, pdf, video and presentation files. The distribution of file sizes is shown in figure 9.

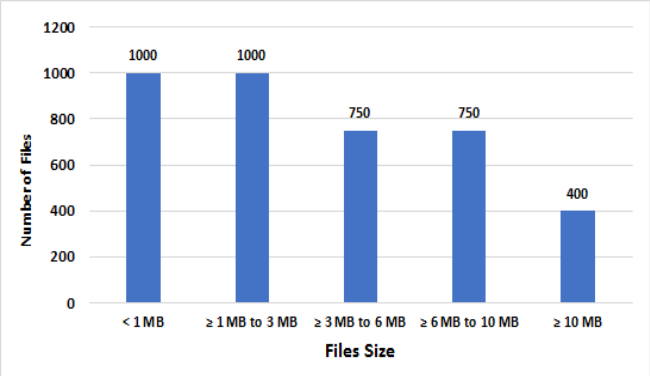


Fig 9. Distribution of File Sizes in Workload

C. Performance Measurement Parameter

The performance of the Hadoop cluster was measured on the following parameter.

a. Time taken to load data from local file system to HDFS.

b. Amount of memory consumed by the NameNode for storing metadata.

c. Time taken to process the data.

d. Overall Performance.

D. Data Loading.

To process small files, user need to load data from local disk to HDFS. HAR, SequenceFile and Proposed Approach converts small files to large file and load the data to HDFS. The formula to calculate data loading time is as follows:

$$T_{DL}=T_{CF}+T_{MF} \text{ ----------(1)}$$

Where,

T_{DL}= Total time taken for data loading.

T_{CF} = Time taken convert small files to large file.

T_{MF} = Time taken to move file to HDFS

Experiments are conducted to test the data loading time in the proposed approach, which is compared with the original Hadoop, SequenceFile and HAR. Table I shows the time taken by the original Hadoop, SequenceFile, HAR and Proposed Approach. Figure 10 depicts the chart of the time taken by the original Hadoop, SequenceFile, HAR and Proposed Approach.

The performance of data loading of the Proposed Approach is optimized than original Hadoop by 31.75%, SequenceFile by 46.63% and HAR by 38.99%. The obtained result clearly indicates that the data loading takes place faster in the Proposed Approach than original Hadoop, SequenceFile and HAR.

Table I. Time taken to load data from local disk to the HDFS

Technique	File Size in GB	Time Taken in seconds
Original Hadoop	10	463
SequenceFile	10	592
HAR	10	518
Proposed pproach	10	316

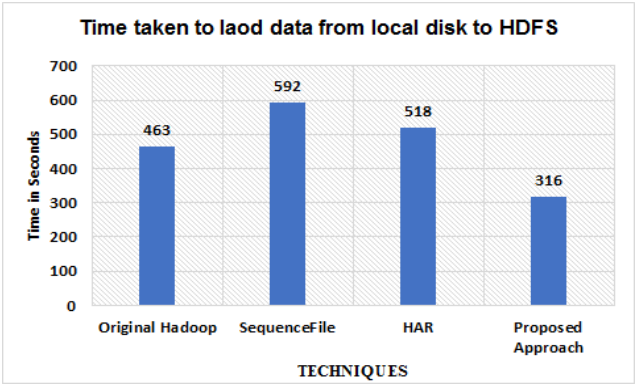


Fig 10. Time taken to load data from local disk to HDFS

E. Memory Measurement

The experiments are conducted to compare the NameNode memory consumption of the Proposed Approach with original Hadoop, SequenceFile and HAR. The computational formula to measure NameNode memory consumption is as follows:

$$M_{NN} = N_{DB} * M_{MD} + M_{IF} \text{ ----------(2)}$$

Where,

M_{NN} = Memory consumption of NameNode.

N_{DB} = Total Number of Data Blocks.

M_{MD} = Size of metadata information.

M_{IF} = Size of index file

A total of 3,900 small files was placed. In the original Hadoop each small file is represented as data block. The NameNode memory consume is 585,000 bytes of memory to store meta data information of 3,900 data blocks. HAR archives 3,900 small files into HDFS blocks. The NameNode memory consume is 355,000 bytes of memory to store meta data information of HAR file. The proposed approach consists of File Integrator; it integrates small files into a large file of 100 MB size. The proposed approach integrates 3,900 small files into 100 large files. The required NameNode memory is 15,000 bytes to store meta data information of 100 data blocks. Table II shows the NameNode memory consumed by the original Hadoop, SequenceFile, HAR and Proposed Approach. Figure 11 depicts the chart of the NameNode memory used by the original Hadoop, SequenceFile, HAR and Proposed Approach.

The NameNode memory consumption is minimized in Proposed Approach than original Hadoop by 80.35%, SequenceFile by 75.74 is % and HAR by 67.60%. The result obtained clearly indicates that the NameNode memory consumption of the Proposed Approach is lesser than original Hadoop, SequenceFile and HAR.

Table II. Memory usage of the NameNode

Technique	Memory usage in bytes
Original Hadoop	585,000
SequenceFile	474,000
HAR	355,000
Proposed Approach	115,000

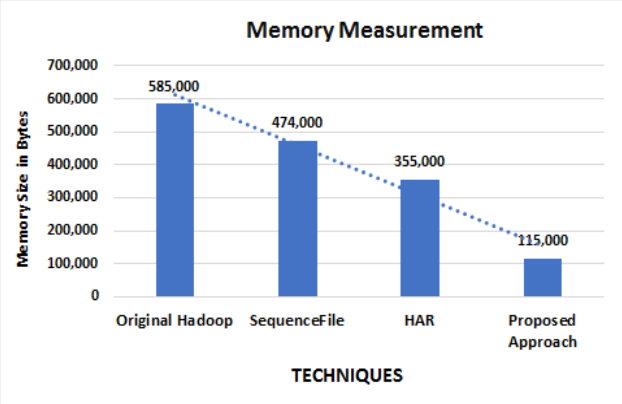


Fig 11. Memory consumption of the NameNode

F. Time taken to process files.

The experiments are conducted to compare the file processing time in the proposed approach with the original Hadoop, SequenceFile and HAR. Table III shows the time taken by the original Hadoop, SequenceFile, HAR and Proposed Approach. Figure 12 depicts the chart of the time taken by original Hadoop, SequenceFile, HAR and Proposed Approach.

The time required to processing data in Proposed Approach is optimized than original Hadoop by 79.36%, Sequence File by 68.93% and HAR by 45.39%. The result obtained clearly indicates that the performance with the proposed approach is better than original Hadoop, Sequence File and HAR

Table III. Time taken to process files

Technique	Time in Seconds
Original Hadoop	2,529
Sequence File	1680
HAR	956
Proposed Approach	522

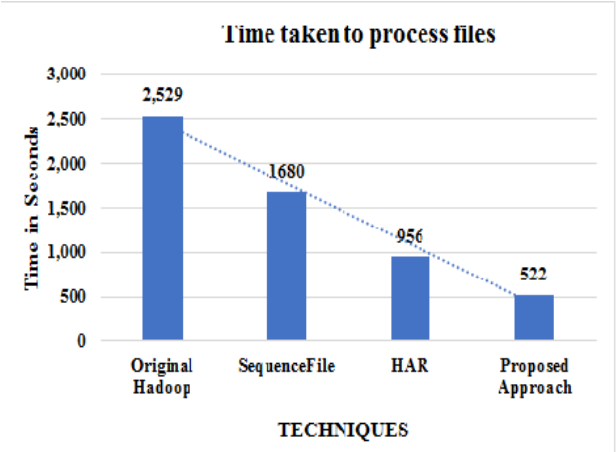


Fig 12. Time taken to process files

G. Overall Performance

The experiments are conducted to compare the total time taken to complete the job by the Proposed Approach with the original Hadoop, SequenceFile and HAR. The formula to calculate Overall Performance is as follows:

$$T_{CJ}=T_{DL}+T_{PF} \text{ ---------- (3)}$$

Where,
T_{CJ}= Total time to complete job.
T_{DL}=Time taken for data loading.
T_{PF}= Time taken to process files.

Table IV shows the time taken by the original Hadoop, SequenceFile, HAR and Proposed Approach. Figure 13 depicts the chart of the time taken by original Hadoop, SequenceFile, HAR and Proposed Approach. The Overall Performance of the proposed approach is optimized than original Hadoop by 71.20%, SequenceFile by 63.11% and HAR by 43.14%. The result obtained clearly indicates that Overall Performance of the proposed approach is better than original Hadoop, SequenceFile and HAR.

Table IV. Overall Performance

Technique	Time in Seconds
Original Hadoop	2992
SequenceFile	2272
HAR	1474
Proposed Approach	838

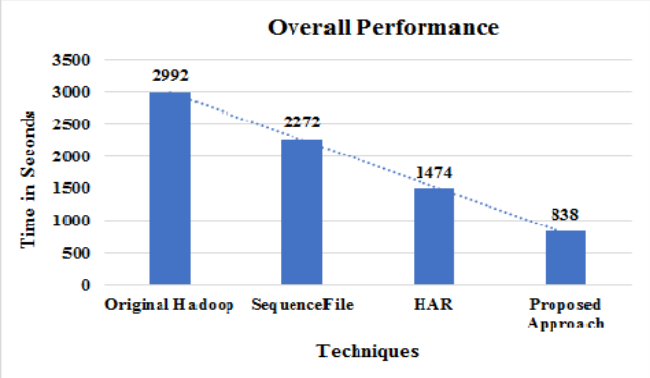


Fig 13. Overall Performance

VI CONCLUSION

Hadoop is a most popular, powerful and widely used open source distributed computing framework to handle large files. But drawback with this tool, it does not provide optimal performance for massive small files in storage, manage and processing levels. Existing approaches to solve small files problems on Hadoop are Hadoop Archives and SequenceFile. However, the performance of existing approaches is not optimal to solve small files issues on Hadoop. In order to achieve optimal performance, we proposed an optimized Hadoop MapReduce framework. In our Hadoop MapReduce framework, we propose two techniques, FileManager and MCR, which help to provide better storage, manage and processing services. Experimental result shows that the NameNode memory consumption of the Proposed Approach is smaller than original Hadoop by 79.36%, Sequence File by 68.93% and HAR by 45.39%. The Overall Performance of the proposed approach with respect to original Hadoop is increased by 71.20%, SequenceFile by 63.11% and HAR by 43.14%.

ACKNOWLEDGMENT

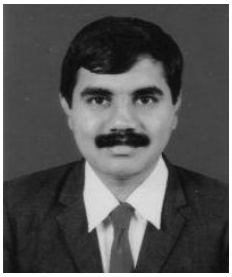
We would like to thank every member of the faculty at SDMIT, MITE and SMVITM for their guidance and support, which has helped us, complete this research project successfully

REFERENCES

[1] Sagiroglu S, Sinanc, D, “Big Data: A Review”, IEEE,2013, pp. 42-47.
[2] Mukhtaj Khan , Yong Jin, Maozhen Li, Yang Xiang, and Changjun Jiang “Hadoop Performance Modeling for Job Estimation and Resource Provisioning” IEEE transactions on parallel and distributed systems, vol. 27, no. 2, february 2016, pp 441-454
[3] Fang Zhou, Hai Pham , Jianhui Yue, Hao Zou ,Weikuan Yu. "SFMapReduce: An Optimized MapReduce Framework for Small Files." IEEE ,2015, pp. 23-32.
[4] Xiaoyong Zhao, Yang Yang, Li-li Sun, Han Huang. "Metadata-Aware Small Files Storage Architecture on Hadoop." Springer ,2012, pp. 136–143.
[5] KunGao, Xuemin Mao. "Research on Massive Tile Data Management based on Hadoop." IEEE ,2016, pp. 01-05.
[6] Parth Gohil, Bakul Panchal,1. S. Dhobi. "A Novel Approach to Improve the Performance of Hadoop in Handling of Small Files." IEEE ,2015, pp. 1-5.
[7] Tanvi Gupta, SS Handa. "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). Faridabad: IEEE, 2015. pp. 67-71.
[8] Aishwarya K, Arvind Ram A, Sreevatson M C, Chitra Babu, and Prabavathy B. "Efficient Prefetching Technique for Storage of Heterogeneous small files in Hadoop Distributed File System Federation." Fifth International Conference on Advanced Computing (ICoAC). IEEE, 2013. 523-530.
[9] Yanfei Guo et al “ iShuffle: Improving Hadoop Performance with Shuffle-on-Write” IEEE Transactions on Parallel and Distributed Systems, 2016, pp 1-12
[10] Guru Prasad M S, Nagesh H R and Swathi Prabhu “High Performance Computation of Big Data: Performance Optimization Approach towards a Parallel Frequent Item Set Mining Algorithm for Transaction Data based on Hadoop MapReduce Framework”, International Journal of Intelligent Systems and Applications,2017, pp75-84
[11] Guru Prasad M S, Raju K and Nagesh H R “Novel Approaches for Performance Optimization of Hadoop Multi Node Cluster Architecture”, Elsevier Publications, 2014, pp 391-399

[12] Katayoun Neshatpour, Maria Malik, Mohammad Ali Ghodrat, Avesta Sasan, and Houman Homayoun " Energy-Efficient Acceleration of Big Data Analytics Applications Using FPGAs" , IEEE International Conference on Big Data, 2015,pp115-123
[13] Ran Zheng, Qing Liu, Hai Jin. "Memory Data Management System for Rendering Applications." Second International Conference on Mathematics and Computers in Sciences and in Industry. IEEE, 2015. 302-308.
[14] Yang Zhang, Dan Liu. "Improving the Efficiency of Storing for Small Files in HDFS." International Conference on Computer Science and Service System. IEEE, 2012. 2239-2242.
[15] Yizhi Zhang, Heng Chen, Zhengdong Zhu, Xiaoshe Dong, Honglin Cui. "Small Files Storing and Computing Optimization." 11th International Conference on Natural Computation (ICNC). IEEE, 2015. 1269-1274.
[16] Bo Dong, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li, Ying Li. "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files." 2010 IEEE International Conference on Services Computing. IEEE, 2010. 65-72.
[17] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, Prabavathy B, Chitra Babu. "A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System." International Conference on Computer Communication and Informatics (ICCCI -2013). Coimbatore, INDIA: IEEE, 2013. 01-08.
[18] ChatupornVorapongkitipun, Natawut Nupairoj. "Improving Performance of Small-File Accessing in Hadoop." 11th International Joint Conference on Computer Science and Software Engineering (JCSSE). IEEE, 2014. 200-205.
[19] Neethu Mohandas, Sabu M. Thampi. "Improving Hadoop Performance in Handling Small Files." Springer (2011): 187-194.
[20] Grant Mackey, Saba Sehrish, Jun Wang. "Improving Metadata Management for Small Files in." IEEE, ,2009,pp.01- 04.
[21] J. W. Jiangling Yin, D. H. Jian Zhou, Tyler Lukasiewicz, and J. Zhang, "Opass: Analysis and optimization of parallel data access on distributed file systems," in IEEE International Parallel & Distributed Processing Symposium (IPDPS), IEEE,2015.
[22] R. Din, Prabadevi B.," Data Analyzing using Big Data (Hadoop) in Billing System ", International Journal of Computer Sciences and Engineering, volume-5,Issue-5,2017,pp 84-88.

Authors' Profiles



Dr. Nagesh H.R , Dean(Academic), Professor & Head, Department of Information Science & Engineering, A J Institute of Engineering & Technology, Mangalore, has got his M.Tech and Ph.D(Computer Engineering) from NITK Surathkal. He has published more than 50 research papers in National and International Conferences and journals. He has delivered more than 20 invited talks in topics like 'Component Based Software Development', 'Internet Security', 'Web Security', 'Web Engineering', 'Information Security' ,'Network Management', 'Promoting Global Cyber Security' ,'Security issues in Distributed Systems', 'Digital library and Information Search', 'Information Security Management' ,'Recent Trends in Information Technology' and 'Security issues in Cloud Computing'. He has also chaired many sessions in International and National level technical paper presentations. He has also published one chapter titled 'Proactive models for Mitigating Internet DoS/DDoS Attacks', in 'Selected Topics in Communication Networks and Distributed Systems', World Scientific, London, April 2010. He had also worked as Visiting faculty to NITK Surathkal and NITK-Science and Technology Entrepreneurs Park, Karnataka, Surathkal. Published two books titled 'Fundamentals of CMOS VLSI Design' for V Semester Electronics & Communication Engineering students of VTU: Pearson Education & 'VLSI Design' for V Semester Electronics & Communication Engineering students of JNTU: Pearson Education. Member of BOS for PG studies in Computer Science at Mangalore University and Manipal Institute of Technology for PG studies in Computer Science & Engineering. Worked as member of BOE and Exam coordinator in VTU Belgaum. Member of BOS in Computer Science & Engineering of VTU Belgaum for year 2013 to 2016.

Mr. Guru Prasad MS, Asst .professor, Dept of Computer Science & Engineering. Shri Dharmasthala Manjunatheshwara Institute of Technology, Ujire, Dakshinna Kannada. He got his M.Tech (Computer Engineering) from NMAMIT Nitte. He has published 7 research papers in International Conferences and Journals. He has delivered 10 invited talks on "Big Data Analytics". His interested area is BigData Analysis using Hadoop, Distributed computing and Cloud computing.

Ms. Swathi Prabhu, Asst. Professor , Dept. of Computer Science & Engineering, Shri Madhwa Vadiraja Institute of Technology & Management, Bantakal, Udupi. She got her M.Tech (Computer Engineering) from NMAMIT Nitte. She has published 4 research papers in National and International Conferences and journals. Her interested area is BigData Analysis using Hadoop, Distributed computing, parallel computing.

