Small file problem in Hadoop and solution to improve access and storage performance

Neseeba p.b C[#1],Dr. M Sharmila kumari [*2]
M.Tech Student, *Professor
Department Of Computer Science & Engineering, P. A. College of Engineering, Mangalore, 574153, India

Abstract— HDFS gives the programmer unlimited storage and is the only reason behind turning to Hadoop. But when it comes to storing lot of small files there is a big problem. HDFS is capable of handling large files which are GB or TB in size. Hadoop works better with a small number of large files and not with large number of small files. Large
Number of small files take up lots of memory on the Name node. Each small file generates a map task and hence there are too many such map task with insufficient input. Storing and transforming small size file in HDFS creates an overhead to map reduce program which greatly affects the performance of Name node.
Large amount of high speed data presents new kind of challenges that traditional database system has limits to resolve. Every organization had expertise to manage structured data but the world had already changed to unstructured data. All of these needed to now bring to a single platform and build a uniform system and Big Data fulfil this need. Big Data does not mean lots of data. It is actually a concept providing an opportunity to find new insight in to existing data and guidelines to capture and analysis future data1.Apache Hadoop is freely available java based software framework. It offers a powerful distributed platform to store and manage Big Data. It runs applications on large clusters of commodity hardware. It processes thousands offer a bytes of data on thousands of the nodes. The major advantage of Hadoop framework is that it provides reliability and high availability and sequence files are used for feature extraction.

.***Keywords***— HDFS, File merging, New har, combined input format, sequence input format

I INTRODUCTION

Merchants selling products on the Web often ask their customers to review the products that they have purchased and the associated services [1]. As e-commerce is becoming more and more popular, the number of customer reviews that a product secures increases rapidly. There will be hundreds or thousands comments for a trendy product in the web. This makes it difficult for a potential customer to read all of them to make a decision on whether to purchase the product or not. It also makes it difficult for the manufacturer of the product to keep track and to manage customer opinions [1].So we want to get the summary of the reviews. We can use clustering algorithms which is a part of data mining. There are mainly two types of clustering algorithms: Partitioning and Hierarchical [2].This work is using incremental clustering algorithm to discover the top-k clusters including different groups of reviews about one product. Initially we are using batch short text summarization algorithm to form th e clusters of comments. After that for each newly coming comment we have to include it in an existing matching cluster. For that we are using incremental clustering algorithm. After that we are performing sentimental analysis to classify it as either positive comment or negative comment.
The rest of the paper is organized as follows. Section II describes the problem. Section III explains the related works. Section IV describes representation of comments as term vectors. Section V briefs the clustering definitions. Section VI illustrates the methodology. Section VII detailed study of present case. Section VIII briefs the work and concludes. Section IX describes the future work.

II PROBLEM DESCRIPTION

In the present world of cloud computing, a huge amount of data is being generated and this data needs to be stored,
processed and analyzed. HDFS [1] serves as primary storage system of Hadoop and has a master -slave architecture. HDFS is a common representative for Cloud service file system running on clusters and has been widely used to support Cloud applications. Having on-demand availability to computation, high scalability and large storage capacity, cloud computing [2] is the ideal solution for big data analytics and processing. Deploying Hadoop on Cloud eliminates the operational challenges of running Hadoop. HDFS is suitable for the analysis of large datasets of applications such as machine learning, data mining, etc. But, HDFS ignores the problem associated with access and storage of small files. In applications like meteorology, energy, e-Library, messaging apps, e-Business, and e-Learning [3], high-performance computing files and healthcare records, data generated is in huge numbers, but ,of very small sizes, typically ranging from 10KB to 10MB. Datasets of digitized clinical data and genomic data records that helps researchers in performing data analytics on healthcare data are all stored in the form of small files. Presently there are no available standard and generic file systems which are specifically designed for storing, processing and analyzing small files. Face book has also faced a

similar problem; it has about 15 billion photos [4] of its users growing at a rate of 220 million/week. For storing photos, Face book has developed Haystack a distributed system to serve their specific needs. All currently available local file system, distributed file system, and the object-based storage systems such as Ceph,GFS, Google File, EXT4 are developed and designed for large files. Their performance degrades significantly in the case of processing and storing small files .Moreover, among the distributed system at the helm presently, Hadoop is prime big data analytics platform in the present data world. It is the leading platform regarding performance, reliability, scalability. Thus, there are various reasons contributing to Hadoopsmall file problem [5]. Firstly, the Name Node memory foot prints a big issue. Every block, file, and directory in Hadoop is represented as an object in the memory on Name Node. As a rule, each small file requires 150 bytes of memory for storing its metadata. By reducing the number of small files on Hadoop cluster, we can reduce the Name Node memory consumption .Also, access time and network IO time are reduced as the number of requests to Name Node reduces with less number of small files. Secondly, the presence of a large number of small files will deteriorate the performance of Map Reduce processing as a huge number of small files results in a large volume of random disk IO. For Map Reduce performance, often disk IO is one of the prime limiting factors. Reading the data from, one large merged sequential file will always take less time as compared to reading the same size of data through multiple random reads. If we are able to store data in less number of blocks, the performance impact due to disk IO will be significantly mitigated. Practically, the generation of small files has become increasingly common. To solve the small file problem, we mainly merge small files and store the large file after merging. But, presently available solutions have not considered the size distribution of files and internal fragmentation caused by merging of files into consideration. In this paper, we firstly analyze, compare and contrast various solutions available for solving Hadoop small files problem, including Hadoop ownsolutions and other solutions which target various aspects of small files problem. In this paper, we propose a novel algorithm named as, OMSS (OptimizedMapFile based Storage of Small files) to improve storage efficiency and processing time for small files in Hadoop. The OMSS technique reduces memory wastage done due to internal fragmentation by using Worst fit technique for merging several files into a Map File [10]. We have implemented and compared the results on a Hadoop setup on cloud and find that our proposed algorithm performs better than basic MapFiletechnique.Our solution can be applied to various big data applications such as in healthcare industry, where huge number of Electronic health records (EHR) is stored in healthcare information systems. Also, there are millions of users of various messaging apps, wherein text messages constitute major chunk of data and this data is generally backed up every few minutes. It can be used in big data analytics and gaining insights in huge E-Commerce data. The rest of this paper is organized as follows. Section IIvdiscusses background on Hadoop and Problems associated with small files; Section III describes related work; Section vI describes our proposed solution, OMSS; Section V contains experimental results and Section VI concludes the paper

III. BACKGROUND

In this section, we discuss architecture of Hadoop DistributedFilesystem (HDFS) and then the problems associated with processing, storing and accessing small files.

A. HDFS

HDFS [1] is the primary storage system of Hadoop and has a master/slave architecture. It consists of Name Node/s and Data Nodes as architectural components. HDFS is one of the file systems of Hadoop written in Java which divides a large file into small blocks and distributes them to various nodes called Data Nodes. Redundant copies of each block are maintained in HDFS to avoid loss of data in case of failures. Name Node maintains a database containing a mapping from logical files to physical blocks in the Data Nodes. Block replication is also managed by the Name Node. The Name Node handles the management of the file system namespace, metadata, and requests from clients to access data. The Data Nodes provide block storage and serve IO requests from clients. They also create, delete, and replicate data blocks up on getting command from the Name Node.

B.PROBLEMS WITH SMALL FILES

The HDFS is originally designed for storing and processing a files. We can consider a small file as any file that considerably smaller than the Hadoop block, i.e., 64 MB.However, the small file problem is not just caused by the small size of files, if a large number of files in Hadoop cluster are marginally greater than an increment of the block size, the user will encounter the same challenges as small files. In present-day scenario with the help of cloud computing ,everybody wants data to be available in real time. Due to real time syncing of data, Hadoop ingestion is made to run frequently, which in turn leads to the creation of small files in HDFS. HDFS has problems in processing and handling small files as mentioned below:

1) Name Nodes High memory consumption*:* Metadata in stored in main memory of Name Node. In general, the metadata of a file consumes about 150 bytes of main memory. With say, 24 million files in HDFS, Name Node will require 3.5GB of main memory for storing metadata. Therefore, an application consisting of huge number of small files will eventually consume all memory spaces of Name Node. Hadoop framework represents every block, file, directory as an objectin the memory. So, If we are able to reduce the number of small files on Hadoop cluster, we can also reduce network impact and startup time along with the Name Node memory footprint.

2) The Map Reduce performance problem: Having a large number of small files degrades the performance of Map Reduce[6] processing. The reason for performance degradation is that a huge number of small files means a significant amount of random disk IO, which is often one of the prime limiting factors in Map Reduce performance. Reading the same data in single sequential read will generally take less time than reading via multiple random reads. We can reduce this performance penalty due to random disk IO if we can store our data on few large blocks.

3) High storing time*:* Storage time of small files in HDFS is quite high. For example [7], to store 550,000 small files into HDFS system, it requires around 7.7 hours. The files in this case vary in size from 1KB to 10KB. On the other hand, if we attempt to store these files using a local file system, it takes considerably less time. Thus showing unfeasibility of storing time.

4) Name Node performance degradation*:* HDFS takes a significant amount of time for managing metadata as it requires nodes to cooperate. When HDFS client receives a request for accessing a file it needs to retrieve metadata of the file from Name node for each such request for file access. For small files ,the major overheads are metadata management and disk seeks. While data transfer takes very short time. HDFS client has to contact Name Node frequently and as a result performance of Name Node is impacted, when there are huge number of small files in system.

II.I RELATED WORKS

In recent times, various organizations using Hadoop as well as research community have given a lot of attention towards small file problem .Some of the general solutions for solving small file problem are:

HAR : Hadoop Archive (HAR) [8] archives small files in to files using Hadoop archive method. Hadoop archive method uses Map Reduce to merge small files into fewer large HDFSfiles. It contains metadata files and data files. The metadata file contains a index and master index. Index file has the907 name and location of files that are part of an archive.

HAR is effective in reducing memory consumption of Name Node. The data file consists of files which are archived together. But there are many drawbacks of HAR, each access to HAR file requires 2 reads from the index file and one read from the data file. And also, new files cannot be appended to HAR ,they are immutable once created. We need to re-create HAR for inserting new files

.Sequence File : Sequence File [9] is a data structure which stores the data in the form of binary key-value pairs. Also, it acts as a container for small files. Here key is file name and value is file contents. Compression and decompression are supported both at record level and block level. Sequence File also has several disadvantages. For a particular key, It does not have mechanism for update and delete operation; it only supports append method; and secondly, this approach ha slow access efficiency as it takes quite a long time to make a Sequence file. If a user needs to look up for a particular key ,the complete sequence file is required to be read .Map File:

A Map File [10] is a sorted Sequence File. It comprises of two files, one index file and another one is data file. It maintains index file to store key location information to allow lookup of data by key. The key-value pairs are sorted by key and stored as records in the data file. Map File facilitates to look up for key without needing to read full file .Drawbacks of Map File are similar to Sequence File. They cannot provide flexible APIs for applications as only append method is supported for a particular key. Moreover, while storing files correlations between files are not considered inall three methods namely, HAR, Sequence File, and Map File .Various authors have also proposed application specific and several domain specific solutions for small file problems.

L.Xuhui Liu, et al. [7] proposed an approach where the main idea was to combine small files into large ones to reduce the file number and build an index for each file. Additionally, they provided some features such as grouping neighboring files and reserving several latest version of data considering the characteristics of Web GIS access patterns. Their technique was similar to Misfile. They did not focus on size distribution and processing time of small files.

G. Mackey et al.[11] proposed a scheme based on HAR file layout to solve the small files problem. Their primary focuswas on the scheduling of jobs on Hadoop cluster using quota policy. The HAR file layout itself has several disadvantages as mentioned above in this paper. They did not introduce any good solutions for reducing access time, processing time and memory consumption at major nodes of Hadoop. Dong et al. [12] presented a scheme in that, firstly, it merges all correlated small files of a PPT courseware in to a larger file to reduce the memory footprint on NameNode.Secondly, to reduce the access time of small files, it uses a two-level perfecting mechanism. The authors in their subsequent research[13], introduced an approach which focused on improving the storage capabilities and reducing access time of small files on HDFS.

File merging used by them is similar to other techniques. They classified files as structurally-related files, logically-related files, and independent files. And a pre fetching scheme was applied for structurally-related small files, while file grouping and pre fetching scheme were used for managing logically Related small files. Both the approaches proposed by authors were very similar to Map File technique. In their work, they considered correlation among small files and tried to identify the relationship among various small files.

But, here also size distribution of files was not considered while merging small files. Their prime focus was on the needs of their real education system.

To further improve the storing capability and accessing time of the small files on HDFS, Chandrasekhar S et al.[14] proposed a solution based on the works of Dong et al.,namely Extended Hadoop Distributed File System (EHDFS).Here, in this scheme, correlated files are identified by the client and combined into a single large file to minimize total count of files. An indexing mechanism was created to access the individual files from the corresponding combined file. Further, index prefetching was also provided to improve Io performance and minimize the load on Name Node. In this paper, we propose anOptimizedMap File based algorithm for Storage of Small files, considering the size distribution of files. The concept of considering size distribution offices for merging small files was proposed by Hue He et al. The solution presented by Hue
He et al. [15] was a Tetris merge algorithm which was based on the balance of data block. The
merging strategy was similar to filling of gaps in Tetris. In the proposed scheme, based on the volume of small files, the small files are evenly distributed into bigger files. Taking the size distribution of small files into consideration, we have proposed an OMSS algorithm. In our proposed algorithm, we have modified the merging process of Map File in order improve storage and processing efficiency. We have used worst fit strategy to merge files into Map File. Moreover, using OMSS, we have been able to reduce processing time and memory burden on the primary nodes of HDFS. Hence using OMSSwe can resolve the small files problem significantly and benefit with various applications such as healthcare and education which host large, small sized data by providing fewer memory requirements and faster processing time.

IV. PROPOSED WORK

In this paper, we have proposed a modified Map File technique which attempts to reduce memory wastage while merging small files into large Map Files. To overcome problems in handling small files, we merge small files into a larger file and store the merged file on HDFS. While merging small files, variation in the size distribution of files is not taken into consideration. We propose a new algorithm OMSS (OptimizedMapFile based Storage of Small files) which takes into account size distribution of files present in a file set. It merges the small files into larger files based on Worst fit strategy that helps in reducing internal fragmentation in data blocks, which in turn leads to fewer data blocks consumed for the same number of small files. Less number of data blocks means fewer memory overheads at major nodes of Hadoop cluster and increased efficiency of data processing.

In our proposed algorithm we have reduced internal fragmentation in Map File using Worst-Fit strategy. In the worst-fit
strategy, process is placed in the biggest block of unallocated memory available by memory manager. As we have mentioned earlier small file problem is not just due to very small files, but it is also created by files that are smaller than block size but due to less space available in the data block, they spillover to next data block of Data Node and create the small file problem. As mainly, all the merging algorithms are based on the cut-off size of the merged file, while merging many small files, if the size of the large file being created reaches the cut-off point, the files are merged into a bigger file called as dequence file which are used for feature extraction.

Inthis process if a file crosses cut off size, it is removed from that large file, and internal fragmentation occurs in the merged file. As shown in fig. 2, while merging 3 files i.e. 1,2,3 total sizes of these 3 files crosses the cut-off size pre decided by emerging algorithm, then file 3 is removed from the file being merged. And thus leaving behind memory unutilized. Hence its imperative to take into account size distribution of files and merging files accordingly; By this approach we can optimizememory requirements of major nodes of Hadoop cluster.

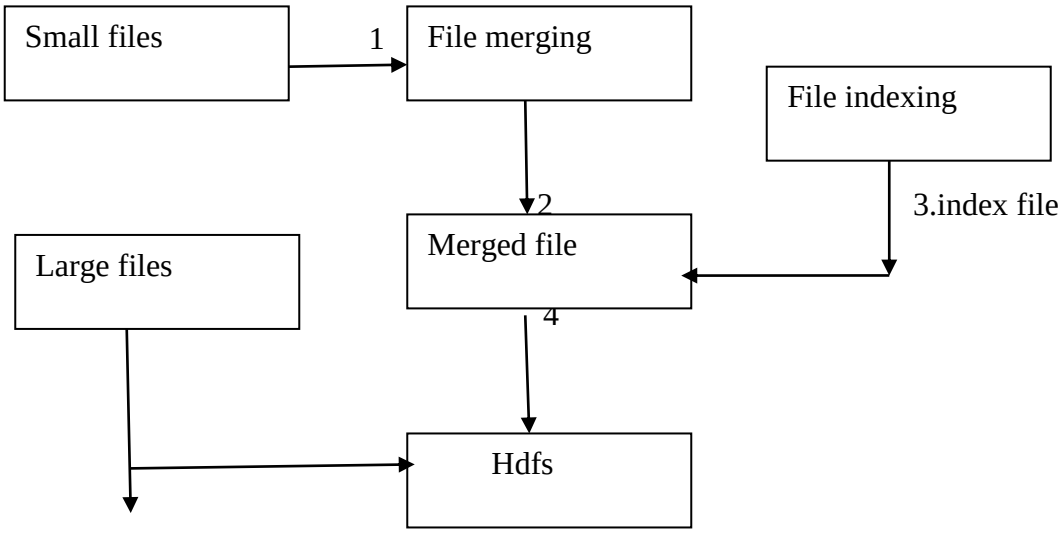


Fig. 1. Schematic Diagram of Proposed solution

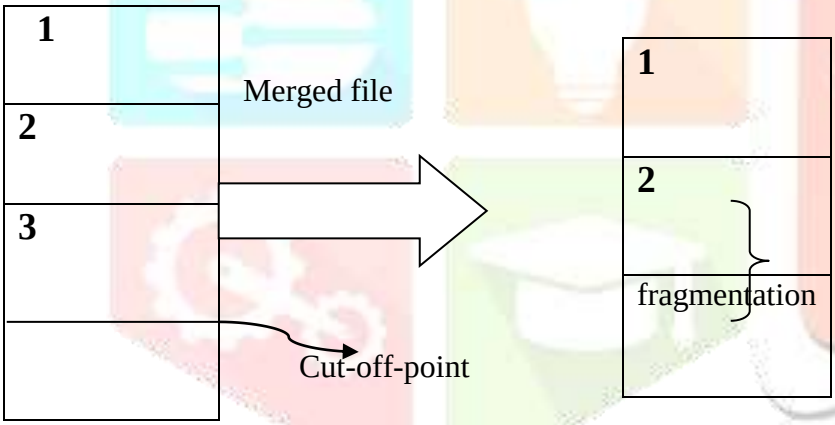


Fig. 2. Problem of Internal Fragmentation

The File Set contains the set of files to be processed into Hadoop Cluster .As shown in fig. 1, we first segregate large files Present in the File Set from small files. Then the large files are directly put into Hadoop Cluster for processing. Using the worst fit strategy of file merging, as proposed in our algorithm, we sort the small files in the File Set into various merging queues. Small files are put into the queue with the maximum merge limit. Files are placed into the merging queue until the queue size become equal to merge criteria. After the files are sorted into various queues, the files are merged and converted in to Map Files, one per queue. Map File is a file containing sorted key-value pairs. Here key is file name and value is file contents. Map File also maintains an index file to facilitate faster access of small files. These Map Files are then put in to Hadoop Cluster for processing. Our proposed File merging strategy for Map Files consists of two parts

(1) File filtering based on the size of small files,

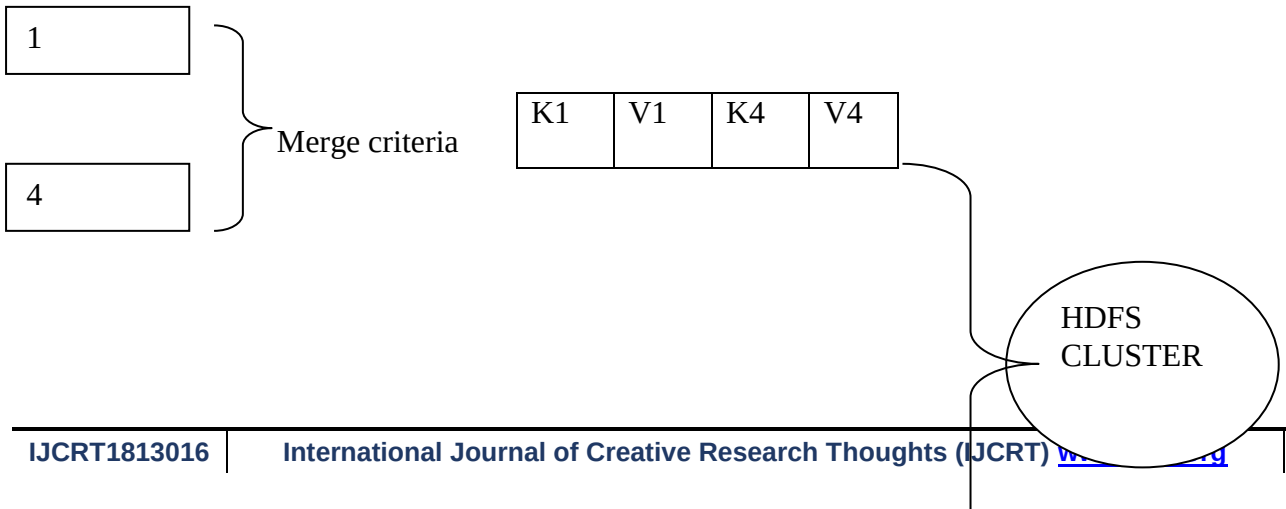
(2) File merging using worst-fit strategy.

1) File Filtering Criteria*:* For optimizing performance ofHDFS for handling small files, it is crucial to decide the cutoff point between small files and large files. Many applications have a huge number of files, which are considerably small in size .For example, in the field of climatology [16], some applications contains 450,000 files with an average size of 61MB.In Sloan Digital Sky Survey, there were around 20 million image files [17] with an average size even less than 1MB.Although Dong etal. discussed the cut-off point between large and small files, they just presented the experimental analytical results.

Algorithm 1 File Merging Algorithm

1: Initialize File set(Fs), Merge Criteria(Ms), Merge-
Queue(Qm)
2: for each file Fi in Fs do
3: if (Fi belongs to Fs and Fi is not a small file) then
4: Go to Step 3
5: else
6: Go to Step 7
7: Merging of the files using worst fit strategy where merge limit is the maximum limit. Select the queue with the max merge limit i.e. max free size
8: Insert the file into selected queue and calculate the queue size(Qs) and merge limit as:
Qs = Qs + file size(Fi)
merge limit = Ms - Qs
9: if (Qs==Ms) then
10: Make a sequence file of the files in the queue
11: else
12: Got o Step 2
13: Put all the sequence files generated and large files present in file set into HDFS for processing.

2) File Merging: We can understand file merging process as shown in fig. 3, there are five files namely 1,2,3,4,5 in the file list. Out of which File 3 represents a large file, while all other are small files. Here, sizes of file 1,2,3,4,5 are taken6,7,200,4,2 units respectively. And let's consider threshold limit to be 10 units. Our file merging algorithm puts the file into various merging queues based on worst fit strategy. File1 and 4 are placed into the 1st queue, while 2 and 5th into the second one. The files in the queue are then converted into Map File (key, value pair). The Map Files generated and the large file are then put into HDFS cluster for processing. As described in algorithm 1, We first initialize the set of all input files(Fs), the criteria for merging the files (Ms) and the merging queue (Qm).
Next, we iterate for every input file of our set, if the file is a large file, we put it into the Hadoop Distributed File system for its indexing, processing ,and storage in the cluster. Else, if the file is indeed a small file hen we merge it into the Merge Queue, using the worst fit strategy, that is, we put it into the queue having the maximum possible empty space. After merging the file into the merge queue, its queue size is incremented by the file size, and the merging limit is decremented by the queue size. Whenever ,for a queue, the queue size becomes equal to the merge size limit, the queue is converted into a sequence file. And these quince files are put in the Hadoop DFS for processing. This algorithm makes use of the worst fit strategy, to reduce internal fragmentation in storing of files. As the hole (empty space) which is left after inserting a file into a queue is the maximum, hence the chances of accommodating any new file
into the left over space are maximum. Thereby wastage of space is minimized.



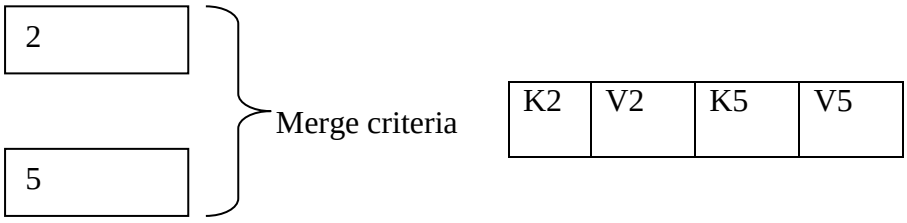


Fig. 3. Schematic Diagram of File Merging Strategy

CONCLUSION

In proposed method we combine many number of small files and store them as large file after merging the available merging algorithm does not consider the size distribution of files as well as internal fragmentation. In our algorithm we combine presently available solution with Hadoop native solution and various solutions which resolves various aspects of small file problem. Present solution reduces memory wastage by reducing internal fragmentation by using worst fit strategy to combine small files in to large file. This solution can be used big data application such as in health care industry where Large number of electronics records are generated and stored. and also used in various message apps which generates major chunks of textual data. Finally it uses map reduce program to extract data from social website and all this data will be stored in small files and converted to single large file called as sequence file. these files are used for any kind of feature extraction.

REFERENCES

[1] Hdfs architecture guide. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs design.html

[2] P. Mell and T. Grance, "A nist definition of cloud computing. nationalinstitute of standards and technology. nist sp 800-145," 2009.

[3] B. Dong, Q. Zheng, M. Qiao, J. Shu, and J. Yang, "Bluesky cloudframework: an e-learning framework embracing cloud computing," in *IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 577–582.

[4] M. Marcon, B. Viswanath, M. Cha, and K. P. Gummadi, "Sharingsocial content from home: a measurement-driven feasibility study,"in *Proceedings of the 21st international workshop on Network andoperating systems support for digital audio and video*. ACM, 2011, pp. 45–50.

[5] T. White. The small files problem. [Online]. Available: http://blog.cloudera.com/blog/2009/02/the-small-files-problem/

[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113,2008.

[7] X. Liu, J. Han, Y. Zhong, C. Han, and X. He, "Implementing webgison Hadoop: A case study of improving small file i/o performance on hdfs," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–8.

[8] Hadoop archive guide. [Online]. Available: https://hadoop.apache.org/ docs/r1.2.1/hadoop archives.html

[9] Sequence file. [Online]. Available: https://wiki.apache.org/hadoop/SequenceFile

[10] Mapfile. [Online]. Available: https://hadoop.apache.org/docs/r2.6.0/api/ org/apache/hadoop/io/MapFile.html

[11] G. Mackey, S. Sehrish, and J. Wang, "Improving metadata management for small files in hdfs," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–4.

[12] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, and Y. Li, "A novelapproach to improving the efficiency of storing and accessing small fileson hadoop: a case study by powerpoint files," in *Services Computing(SCC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 65–72.

[13] B. Dong, Q. Zheng, F. Tian, K.-M. Chao, R. Ma, and R. Anane,An optimized approach for storing and accessing small files on cloudstorage," *Journal of Network and Computer Applications*, vol. 35, no. 6,pp. 1847–1862, 2012.

[14] S. Chandrasekar, R. Dakshinamurthy, P. Seshakumar, B. Prabavathy, andC. Babu, "A novel indexing scheme for efficient handling of small filesin hadoop distributed file system," in *Computer Communication andInformatics (ICCCI), 2013 International Conference on*. IEEE, 2013,