SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM

TMLAI TRANSACTIONS ON
MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

VOLUME 5 NO 4, 2017
ISSN 2054-7390

Optimizing Hadoop for Small File Management

O. Achandair, M. Elmahouti, S. Khoulji, M.L. Kerkeb,
Information System Engineering Resarch Group
Abdelmalek Essaadi University, National School of Applied Sciences, Tetouan, Morocco.
o.achandair@gmail.com; mouradelmahouti@gmail.com; khouljisamira@gmail.com;
kerkebml@gmail.com

ABSTRACT

HDFS is one of the most used distributed file systems, that offer a high availability and scalability on low-cost hardware. HDFS is delivered as the storage component of Hadoop framework. Coupled with map reduce, which is the processing component, HDFS and MapReduce become the de facto platform for managing big data nowadays. However, HDFS was designed to handle specifically a huge number of large files, while when it comes to a large number of small files, Hadoop deployments may be not efficient. In this paper, we proposed a new strategy to manage small files. Our approach consists of two principal phases. The first phase is about consolidating more than only one client's small files input, and store the inputs continuously in the first allocated block, in a SequenceFile format, and so on into the next blocks. That way we avoid multiple block allocations for different streams, to reduce calls for available blocks and to reduce the metadata memory on the NameNode. This is because groups of small files packaged in a SequenceFile on the same block will require one entry instead of one for each small file. The second phase consists of analyzing attributes of stored small files to distribute them in such a way that the most called files will be referenced by an additional index as a MapFile format to reduce the read throughput during random access.

Keywords - Cloud Hadoop, HDFS, Small Files, SequenceFile, MapFile

1 Introduction

The exponential growth of data, generated continuously in different formats and from different sources, has lead traditional architecture and infrastructures to face many limitations. In the last decade, new technologies based on the cloud model, offered to many organizations the possibility to store and analyze their data in an efficient way and a timely manner, which help them uncover patterns, get insight and provide better services. Hadoop, is an open source framework that offers a distributed storage layer, HDFS [1], tightly coupled with a distributed processing engine,MapReduce [2]. Hadoop allowsthe partitioning of data and computation across clusters of thousands of machines, in such a way, that each machine compute its local or neighbor's data. Hadoop cluster offers high scalability simply by adding commodity servers, as each server can hold more data and process more tasks. The total computation capacity andstorage capacity can be expanded, in a transparent and an efficient manner. Hadoop guarantee a high fault tolerance and high availability. In fact, in the storage layer, HDFS replicate blocks between nodes, so whenever a machine of the cluster goes down, the data can still be accessed from others. Also, HDFS

DOI: 10.14738/tmlai.54.3214
Publication Date: 15th August, 2017
URL: http://dx.doi.org/10.14738/tmlai.54.3214

maintain the replication factor by replicating blocks to other available machines in case of failures. Moreover, in the processing layer, Hadoop can keep track of all the tasks, and restart them on other available machines whenever a host-failure occur during tasks processing. Hadoop is an Apache top-level project, built in modular approach, thatincludes multiple components and subprojects. The components of Hadoop ecosystem are classified as shown in “Fig. 1”

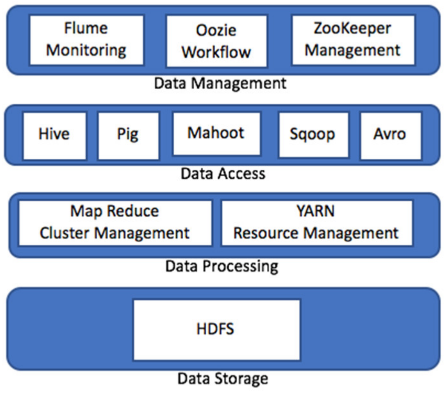

Figure. 1: Hadoop Data Layers Stack

The Data Storage layer, consist of HDFS as the main component that provide the physical access for read and write to the cluster. The Data Processing layer consists of MapReduce as the framework that enables users to write applications that can be processed in parallel according the MapReduce programming model, and YARN, as the resource management component. YARN keeps tracks of resources on the cluster and orchestrates all the assigned tasks. The Data Access layer, consist of the infrastructure that offer tools to manipulate and analyze data through scripting, modeling, querying. The Management layer, consists of the end user layer, it addresses data governance, integration and monitoring components.Hadoop has contributed hugely to handle and manage big data, itsstorage layer was designed to store and process large big files, which are in gigabytes or terabytes, but when it comes to a large number of small files, the performance of the cluster may decrease dramatically. In this research, we addressed the small file problem through an additional middleware called Small File Analyzer server (SFA). The SFA component interacts directly with the data storage and the data processing layers.

The rest of this paper is divided into the following, section 2 gives more details about the data access layer and the data processing layer. Section 3 lists the existing solutions in the related literature. Section 4 present the proposed approach. Section 5 is allocated for our experimental works and results. Finally, Section 5 for conclusion and expectation.

2 Backround

2.1 Hadoop Storage:

HDFS, The Hadoop distributed file system provides high reliability, scalability and fault tolerance. It's designed to be deployed on big clusters of commodity hardware. It’s based on a master-slave architecture, the NameNode as a master and the DataNodes as slaves. The NameNode is responsible for managing the file system namespace, it keeps tracks of files during creation, deletion, replication [3] and manages all the related metadata [4] in the server memory. The NameNode splits files into blocks and sends the writes requests to be performed locally by DataNodes. To ensure a fault-tolerance system, blocks replicas are pipelined across a list of DataNodes. This architecture as shown in "Fig.2", with only one single NameNode simplifies the HDFS model, but it can cause memory overhead and reduces file access efficiency when dealing with a high rate of small files.

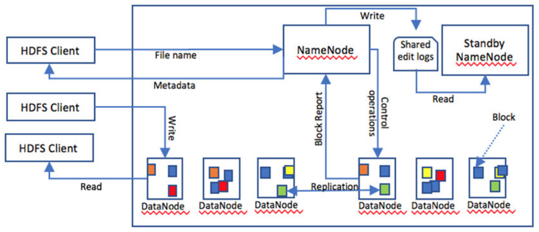

Figure. 2: HDFS Architecture

2.2 Hadoop Processing:

In the current version of Hadoop, Google re-architected the processing engine to be more suitable for most of big data applications needs. The major improvement of Hadoop was the introduction of a resource management module, called YARN, independently of the processing layer. This brought significant performance improvements, offered the ability to support additional processing models, and provided a more flexible execution engine. Because of its independency architecture, existing MapReduce applications can run on YARN infrastructure without any changes.

The MapReduce program execution on YARN can be described as follows:

(1) A user defines an application by submitting its configuration to the application manager
(2) The resource manager allocates a container for the application manager
(3) Resource manager submits the request to the concerned node manager
(4) The Node manager launches the application manager container
(5) The application manager gets updated continuously by the node manager nodes, it monitors the progress of tasks

When all the tasks are done, the application manager unregisters from the resource manager, like so, the container can be allocated again, See "Figure. 3".

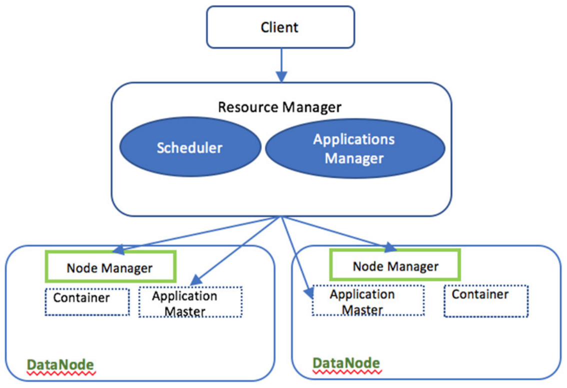

Figure. 3: YARN – Yet Another Resource Manager

To deal with the small file problem, numerous researchers have proposed different approaches. Some of those efforts have been adopted by Hadoop and are available for use natively, more precisely, Hadoop Archives (HAR Files) and SequenceFile.[5]

3 Small File Problem

In HDFS, a file is called small when its size is smaller than the size of the HDFS block. Each file or block is an object in the namespace, each object occupies around 150 bytes in the NameNode memory as the related metadata. Consider a file of 1Gb to be stored in HDFS, with a default block size of 64 Mb and a default replication factor of 3, then we will need 16x3 blocks on the DataNodes and 16x3x150=2400 bytes in the NameNode memory. Instead, if we consider 1000 files of 1Mb, and assume that each file will be stored independently on a block.Then the physical storage on DataNodes will remain the same as the 1Gb file, but we will need 600 000bytes in the NameNode memory. This is because there will be one entry per block which is 1000x3 (number of blocks x replication factor) and one entry per file which is 1000. Each block or file entry will take about 150bytes. As a result, for the same physical storage250 times additional memory space will be required, compared to the previous example. As a matter of fact, there is many fieldsthat produce tremendous numbers of small files continuouslysuch as analysis for multimedia data mining [6], astronomy [7], meteorology [8], signal recognition [9], climatology [10,11], energy, and E-learning [12] where numbers of small files are in the ranges of millions to billions. For instance, Facebook has stored more than 260billion images [13]. Inbiology, the human genome generates up to 30 million files averaging 190KB [14]. Massive numbers of small files can decrease dramatically the NameNode performance, as for each file access, the HDFS client needs to retrieve the metadata from the NameNode. Therefore, frequent calls for and frequent access to metadata reduce the latency during read and write throughput.

In terms of Hadoop processing, the time needed to process too many small files can be hundreds of times slower than processing one single large file that has the same total size. In fact, under a default configuration, Hadoop creates a mapper for each file. Like so, we will have a great number of mappers, that are costly resources.

4 Related Work

To deal with the small file problem, numerous researchers have proposed different approaches. Some of those efforts have been adopted by Hadoop and are available for use natively, more precisely, Hadoop Archives (HAR Files) and SequenceFile.

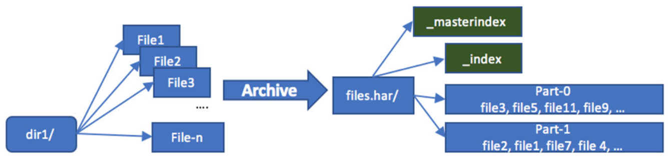

Figure. 3: HAR File Layout

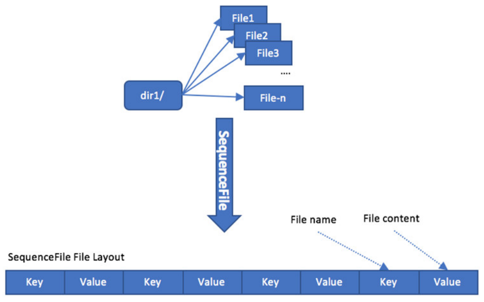

Figure. 4: SequenceFile File Layout

Hadoop Archive packs small files into a large file, so that we can access original files transparently, see "Fig. 3". This technique allows more storage efficiency, as only metadata of the archive is recorded in the namespaceof the NameNode, but it doesn't resolve other constraints in terms of reading performance.Also, the archive cannot be appended while adding more small files. The SequenceFile technique is to merge a group of small files ina flat file, as key-value pairs, while key is the related file metadata and value is the related content, see "Fig. 4".

Unlike the HAR files, the SequenceFile supports compression, and they are more suitable for MapReduce tasks as they are splittable [15], so mappers can operate on chunks independently.However, converting into a SequenceFile can be a time-consuming task, and it has a poor performance during random read access.

To improve the metadata management, G. Mackey et al. [16] merged small files into a single larger file, using the HAR through a MapReduce task. The small files are referenced with an added index layer (Master index, Index) delivered with the archive, to retain the separation and to keep the original structure of files.

C. Vorapongkitipunet al. [17] proposed an improved approach of the HAR technique, by introducing a single index instead of the two-level indexes. Their new indexing mechanism aims to improve the metadata management as well as the performance during file access without changing the implemented HDFS architecture.

Patel A et al. [18] proposed to combine files using the SequenceFile method. Their approach reduces memory consumption on the NameNode, but it didn't show how much the read and write performances are impacted.

Y. Zhang et al. [19] proposed merging related small files according to WebGIS application, which improved the storage effeciency and HDFS metadata management, however the results are limited by the scene.

D. Dev et al. [20] proposed a modification of the existing HAR. They used a hashing concept based on the sha256 as a key. This can improve the reliability and the scalability of the metadata management, also the reading access time is greatly reduced, but it takes more time to create the NHAR archives compared to the HAR mechanism.

P. Gohil et al. [21] proposed a scheme for combining small file, merging, prefetchingthe related small files which improves the storage and access efficiency of small files, but does not give an appropriate solution for independent small files.

5 The Proposed Approach For Small File Management

Currently, it's well known that small files can decrease dramatically the performance of Hadoop clusters. Previous solutions workaround this problem by packaging small filesin different formats. Those formats are saved transparently in HDFS as they can be divided into blocks with no specific constraints. Though the performance of MapReduce jobs can be greatly improved based on the way those small files are packaged, none of the adopted mechanisms take in consideration how to organize those small files during the merging phase. The core idea behind our approach is to store files when a client starts a stream that containssmall files, combined if relevant, with other client streams into a large file within the same block, and organize them later in an efficient way that we can prefetch the most probable called files first. This was achieved by using a Small File Analyzer, see “Fig. 5”.

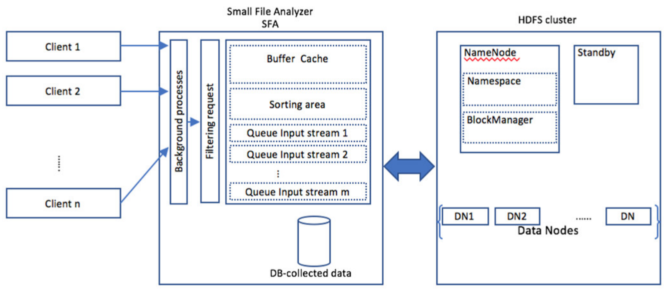

Figure. 5: SFA Architecture

The SFA operations consists of two phases, the first one consists of combining similar small files and store them on one block.

The second one consists of analyzing how small files are used, then put adequate groups in a MapFile. A MapFile is another format of packaged files offered by Hadoop, that consists of an indexed SequenceFile. See “Fig 6”.

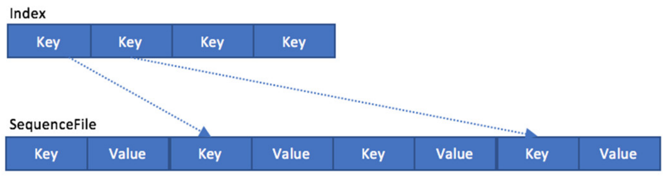

Figure. 6: MapFile File Layout

This second phase can be triggered or called manually as analyzing small files dependson how the cluster is using them, getting the most called small files depends on how many jobs are using them, also on the fact that the files are called together sequentially or extracted in a very few groups.

To improve the SFA analysis, we import records from the FSimageof the NameNode, which contain a complete state of the HDFS state, then we aggregate data to store it in a reference handled by the database of the SFA, see “Fig. 7”. All the clients’ jobs are forwarded from the SFA server first, in such a way, we keep track of more information to use in the SFA analysis.

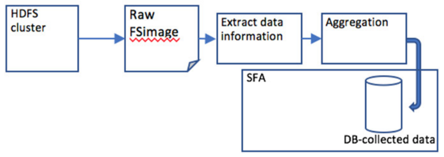

Figure. 7: MapFile File Layout

5.1 Presentation of The Two Phases Algorithms

Algorithm phase1: Filtering inputs & storing small files

```
---------------------------------------------------------------
Input: Client dataset
Output: SequenceFile of combined small on the same blockid
Step1: Get stream characteristics
Step2: Initialize combined queues
Step3: Request blockid from NameNode and lock it in the SFA blocklist.
Step4: Merge maximum small files into a SequenceFile to the locked blockid
Step5: Close current output stream
Free the block from the block list.
If the current block is full, NameNode will allocate a new block, and the full one is deleted from the SFA list.
---------------------------------------------------------------
```

To use the block capacity efficiently, SFA chains small file queues when different clients are requesting storage in HDFS at the same time. Based on the grouped small files in this step2, SFA will request a list of blocks. Each time a blockid is full, SFA deletes it from its blocklist. Like this SFA will write the next coming request to the first blockid in its list without requesting each time a new block from the NameNode.

Algorithm phase2: Analyzing Small Files

```
---------------------------------------------------------------
  Input:SFA inventory - state j
  Output: MapFile of Hot files - SFA inventory state j+1
  Step1: Download and Interpret FSimage using oiv interpreter
  Step2: Splitting FSimage rows and initialize SFA reference
```

Step3: Aggregate records

Step4: Get Top called sequence

Step5: Get Top called files per sequence

Step6: Define group of hot small files

Step7: Retrieve the hot files from original sequence and merge them in a MapFile

----

Once the hot files are listed, we can get the keys and values from their original SequenceFile, and create theMapFiles. SFA can schedule the migration in off-hours. This operation consists of three parts as follows:

```
// get the keys and values of the hot file list

SequenceFile.Reader reader = null;

Class<? extends Writable>keyClass= null;

Class<? extends Writable>valueClass= null;

try{reader= new SequenceFile.Reader(fs, sequenceFile, getConf());

keyClass= (Class<? extends Writable>) reader.getKeyClass();

valueClass= (Class<? extends Writable>) reader.getValueClass();

} catch (IOExceptionioe) {

MainUtils.exitWithStackTraceAndError(

"Failed to open SequenceFile to determine key/value classes: " + input, ioe);

} finally {

if (reader != null) {

try {

reader.close();

}}}


// move the SequenceFile to the new map file , rename it within the output location
```

```
try {

fs.rename(sequenceFile, mapData);

} catch (IOExceptionioe) {

MainUtils.exitWithStackTraceAndError(

"Failed to move SequenceFile to data file in MapFile directory: input=" + input + ", output="

+ output, ioe);

}

// create the index file for the MapFile

try {

MapFile.fix(fs, mapFile, keyClass, valueClass, false, getConf());

} catch (Exception e) {

MainUtils.exitWithStackTraceAndError("Failed to create MapFile index: " + output, e);

}

return 0; }
```

6 Performance Evalation

The proposed method in this paper is compared with the original HDFS about the usage of NameNode memory, and the performance of MapReduce jobs, during sequential and selective file access. We did a simulation on the Hadoop-2.4.0, our cluster consists of one NameNode 3.10 GHz clock speed, 8GB of RAM and a gigabit Ethernet NIC, and four DataNodes. All the nodes offer 500GB Hard Disk, and they are deployed on Ubuntu 14.04. The replication factor is kept as the default value 3 and the block size of HDFS is chosen as 64Mb. The experimental datasets are basically standard auto-generated.

Comparison of the NameNode Memory Usage

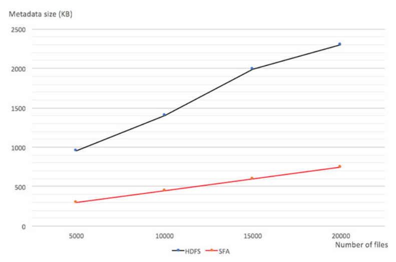

Figure. 8: NameNode memory usage

According to "Fig. 8", the NameNode memory usage of the original HDFS is biggest due to metadata entries for each small file and the inefficiency of block allocation. When we store small files through SFA, The NameNode memory consumption is too low due to file merging in Sequence File. But this doesn't show if our block allocation strategy is efficient or not. This will be tested in the next steps of this research, as we are introducing more factors to combine streams in the locked block.

6.1 Comparison of MapReduce jobs performance

In the following, we performed MapReduce jobs on four datasets. The results on HDFS refers to the measurementof time requiredof MapReduce job on SequenceFiles without retrieving the hot files. We ignored storing files without merging them in SequenceFile, as the low latency during MapReduce jobs has been already proved in the previous studies. In fact, this will lead to the creation of huge number of mappers and finally hanging the whole cluster. The results on SFA2 and SFA6 refer to themeasurementof the time required of MapReduce job after the second call and the sixth job iteration. Each measurement is performed after reorganizing small filesas suggested from the SFA

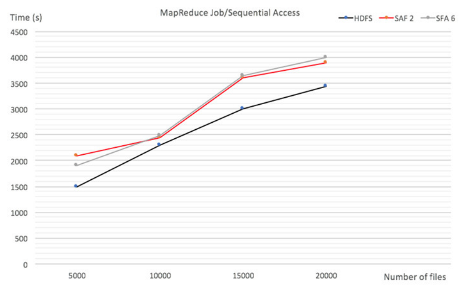

Figure. 9 MapReduce Performance/Sequential Access

According to "Fig. 9", even after many iteration of MapReduce jobs, reorganizing small files didn't improve the MapReduce performance, this is because of accessing indexes is not a necessity in such situations.

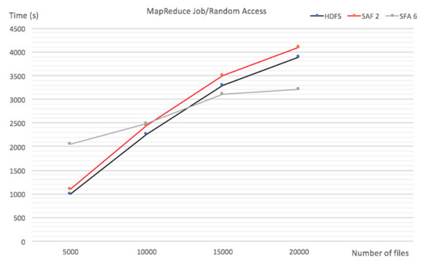

Figure. 10 MapReduce Performance on selective files

According to "Fig. 10", after six iteration, we observed that the performance of MapReduce job is slightly improved when the dataset gets bigger. The specific MapReduce jobperformed about 13minutes faster than the first execution. When the number of small files is too high, retrieving more adequate hot files to be grouped in a MapFile was a solution. Reading hot files in such a situationthrough an index improves the performance of that specific MapReduce job.

In the next step of our research, we will adjust the SFA by introducing more factors to combine related streams efficiently in the allocated blocks. Also, we will introduce the concept of cycles, as even if the

MapReduce jobs tends to call data independently, analyzing file calls during frequents periods of time can reveal new correlations.

7 Conclusion

Our approach provides a new allocation strategy for blocks when storing massive amounts of small files, it also addresses the aspect of analyzing the distribution of small files in SequenceFile format. This approach of classification of metadata based on number of calls can also be extended to include other factors such as owners, size, and age of datasets that are supported in the initial design of our SFA sever. Different formats are now supported in Hadoop to solve the small files problem, but there is a lack of standardization, as most of the solutions remain useful in specific environmentsbut not in others. Offering a system to analyze different aspects of the small files problem can help organizations to understand better the real factors that control the impact of their datasets.

REFERENCES

[1] Official Hadoop website, http://www.hadoop.apache.org.

[2] J. Dörre, S. Apel, and C. Lengauer, “Modeling and optimizing MapReduce programs,” Concurrency and Computation: Practice and Experience, vol. 27, no. 7, pp.1734-1766, 2015.

[3] D. T. Nukarapu, B. Tang, L. Wang, and S. Lu, “Data replication in data intensive scientific applications with performance guarantee,” IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 8, pp. 1299–1306, 2011.

[4] Y. Gao and S. Zheng, "A Metadata Access Strategy of Learning Resources Based on HDFS," in proceeding International Conference on Image Analysis and Signal Processing (IASP), pp. 620—622, 2011.

[5] T. White. Hadoop: The Definitive Guide, 4th Edition O’Reilly, 2015.

[6] B. White, T. Yeh, J. Lin, and L. Davis, “Web-scale computer vision using mapreduce for multimedia data mining,” in Proceedings of the Tenth International Workshop on Multimedia Data Mining. ACM, 2010, p. 9.

[7] K. Wiley, A. Connolly, J. Gardner, S. Krughoff, M. Balazinska, B. Howe, Y. Kwon, and Y. Bu, “Astronomy in the cloud: using mapreduce for image co-addition,” Astronomy, vol. 123, no. 901, pp. 366–380, 2011.

[8] W. Fang, V. Sheng, X. Wen, and W. Pan, “Meteorological data analysis using mapreduce,” The Scientific World Journal, vol. 2014, 2014.

[9] F. Wang and M. Liao, “A map-reduce based fast speaker recognition,” in Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on. IEEE, 2013, pp. 1–5.

[10] K. P. Ajay, K. C. Gouda, H. R. Nagesh, “A Study for Handelling of High-Performance Climate Data using Hadoop, Proceedings of the International Conference, pp: 197-202, April 2015.

[11] Q. Duffy, J. L. Schnase, J. H. Thompson, S. M. Freeman, and T. L. Clune, “Preliminary evaluation of mapreduce for high-performance climate data analysis,” 2012.

[12] C. Shen, W. Lu, J. Wu, and B. Wei, “A digital library architecture supporting massive small files and efficient replica maintenance,” in Proceedings of the 10th Annual Joint Conference on Digital Libraries (JCDL '10), pp. 391–392, June 2010

[13] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, “Data warehousing and analytics infrastructure at facebook,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010, pp. 1013–1020.

[14] J. K. Bonfield and R. Staden, “ZTR: A new format for DNA sequence trace data”, Bioinformatics, vol. 18, no. 1, (2002), pp. 3–10.

[15] J. Xie, S. Yin, et al. “Improving MapReduce performance through data placement in heterogeneous Hadoop clusters”, In 2010 IEEE International Symposium on Parallel & Distributed

[16] G. Mackey; S. Sehrish; J. Wang. Improving metadata management for small files in HDFS. IEEE International Conference on Cluster Computing and Workshops (CLUSTR). 2009. pp.1-4.

[17] C. Vorapongkitipun; N. Nupairoj. Improving performance of small-file accessing in Hadoop. IEEE International Conference on Computer Science and Software Engineering (JCSSE). 2014. pp.200-205.

[18] Patel A, Mehta M A. A novel approach for efficient handling of small files in HDFS, 2015 IEEE International Advance Computing Conference (IACC), pp. 1258-1262.

[19] Y. Zhang; D. Liu. Improving the Efficiency of Storing for Small Files in HDFS. International Conference on Computer Science & Service System (CSSS). 2012. pp.2239-2242

[20] D. Dev; R. Patgiri. HAR+: Archive and metadata distribution! Why not both?. IEEE International Conference on Computer Communication and Informatics (ICCCI). 2015. pp.1-6.

[21] P. Gohil; B. Panchal; J. S. Dhobi. A novel approach to improve the performance of Hadoop in handling of small files. International Conference on Electrical, Computer and Communication Technologies (ICECCT). 2015. pp.1-5.