*International Journal of Engineering & Technology*, 7 (2.6) (2018) 221-226

International Journal of Engineering & Technology

*Website: www.sciencepubco.com/index.php/IJET*

***Research Paper***

# solution for the future: small file management by optimizing Hadoop

**O. Achandair[1]*, S. Bourekkadi[2],M. Elmahouti[1],S. Khoulji[1], M.L. Kerkeb[1]**

*[1]Information System Engineering Resarch Group*
*AbdelmalekEssaadi University, National School of Applied Sciences, Tetouan, Morocco.*
*[2]Laboratory of Research in Management Sciences of Organization, IbnTofail University,*
*National School of Commerce and Management,Kenitra, Morocco*
**Corresponding author E-mail: o.achandair@gmail.com,*

**Abstract**

Hadoop Distributed File System (HDFS) is designed to reliably store very large files across machines in a large cluster. It is one of the most used distributed file systems and offer a high availability and scalability on low-cost hardware. All Hadoopframework have HDFS as their storage component. Coupled with map reduce, which is the processing component, HDFS and Map Reduce (a processing component) have become the standard platforms for any management of big data in these days. HDFS however, in terms of design has the ability to handle huge numbers of large files, but when it comes to its deployments to handle large amounts of small files it might not be very effective. This paper puts forward a new strategy of managing small files. The approach will consists of two principal phases. The first phase will deal with the consolidating of aaclients input files, storing it continuously in a particular allocated block, that is a SequenceFile format, and so on into the next blocks. In this way we avoid the use of multiple block allocations for different streams, this reduces calls for available blocks and also reduces the metadata memory on the NameNode. Note the reason for this is that groups of small files packaged in a SequenceFile on the same block require one entry instead of one of each small file. The second phase will involve analyzing the attributes of stored small files so they can be distributed them in 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

Traditional architecture and infrastructure in recent times face a lot of limitations because of the rapid growth of data coming in continuously in different formats and from different sources. Over time, new cloud based technologies have come which permit organisations to store and analyze their data in the most efficient and timely manner, uncover patterns and provide better services. By adding commodity servers, Hadoop clusters offer high scalability because each server can holdgreat amounts of data and hence efficiently process more data. This therefore leads to expansion in the storage and computation capacities[1][2].

With Hadoop you can be sure of high fault tolereance and high availability. The storage layers, HDFS replicate blocks between nodes, so even if a cluster in a machine go down, the data can always still be accessed by others. HDFS can also replicate blocks to other available machines in case of any failure. More so, in the processing layer, Hadoop can keep record of all tasks, and even restart them on another machine in case of a host-failure during task processing.

Hadoop is an Apache top-level project, built in modular approach, that includes multiple components and subprojects. The components of Hadoop ecosystem are classified as shown in "Fig. 1"

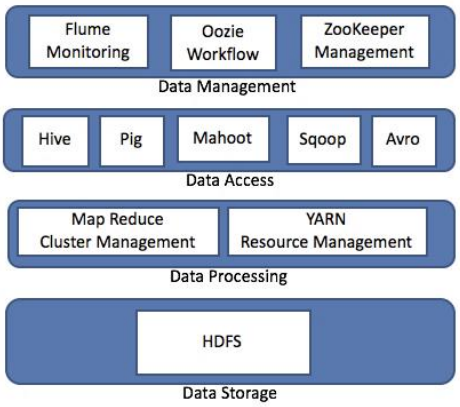



**Figure 1**: Hadoop Data Layers Stack

The layer labeled Data Storage, consist of HDFS which is the main component and provides the physical access for read and write to the cluster. The next layer Data Processing consists of the MapReduce which is the framework that always users to write different applications that can be processed in parallel according the MapReduce programming model, and YARN, as the resource management component. YARN is the area that takes record of resources on the cluster and builds all the assigned tasks. The Data Access layer is the layer that consist of all the infrastructure that offer tools to manipulate and analyze data through scripting, mod-

**Copyright © 2018Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.**

eling, querying. The Management layer, consists of the end user layer, this layer addresses issues of data governance, integration and all components of monitoring. Hadoop helps greatly in the management of huge data, the storage layer is particularly designed to process and store large big files, which are in gigabytes or terabytes, but when it comes to a large number of small files, its performance may decrease dramatically. This paper seeks to address the small file problem through the technical addition of a middleware known as Small File Analyzer Server (SFA). SFA component will interact directly with the data storage and the data processing layers.

This paper is divided into different sections, each one addressing a particular issue;

section 1 will focus on giving details about the data access layer and the data processing layer

Section 2 will build a list of existing solutions in related literature.

Section 3 set forth a proposed approach.

Section 4 presents an allocation for experimental works and results.

Finally, Section 5 for conclusion and expectation.

## 2. Back Ground

### A. Hadoop Storage

Like it has already been mentioned, the Hadoop distributed file system provides high reliability, scalability and fault tolerance. Its design always for it to deploy big clusters of commodity hardware, which is based on a master-slave architecture. With the NameNodeas master and the DataNodes as slave. The responsibility of the NameNode is management of file system namespace, that is it takes charge of tracking files during creation, deletion, replication and manages all the related metadata in the server memory. The NameNode also works to split files into blocks so that the right requests can be sent to the DataNodes be performed locally. 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[3][4].

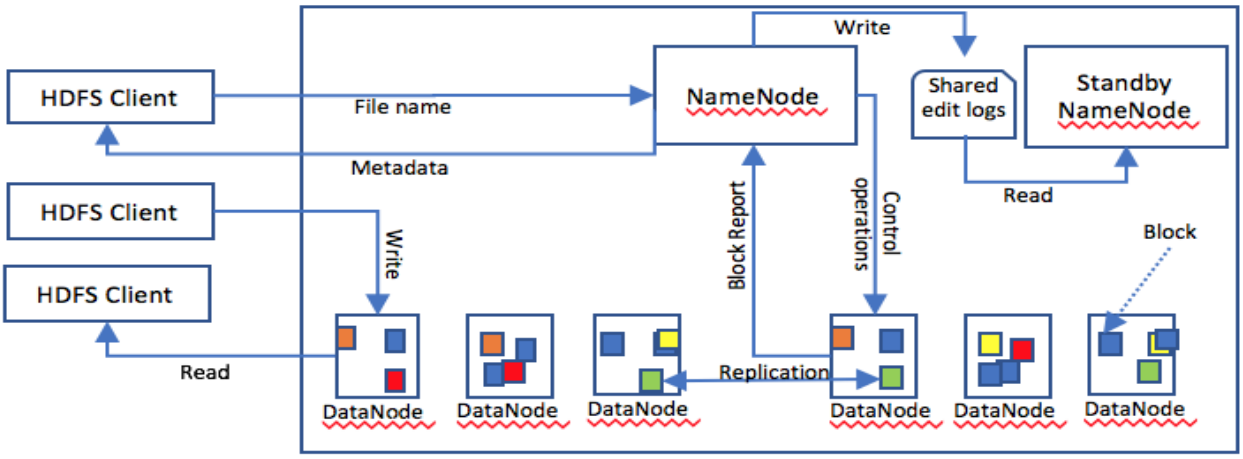



**Figure 2:** HDFS Architecture

### B. Hadoop Processing:

The current version of Hadoop, Google did some work on the processing engine so it would be more suitable for most of big data applications needs. The major adjustments made on the Hadoop was the introduction of a resource management module, called YARN, independent of the processing layer. This significantly improvement performance, and offered it the ability to support additional processing models, and also provided a more flexible execution engine. Because of its independent architecture nature, existing MapReduce applications can smoothly 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 performed, the application manager can just unregister from the resource manager, like so, the container can be allocated again, See "Fig. 3".

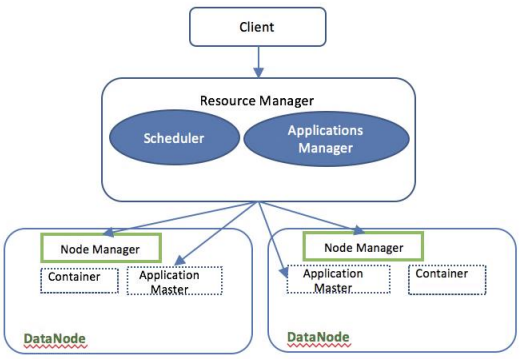



**Figure 3:** YARN – Yet Another Resource Manager

Numerous researchers have proposed different approaches to deal with the small file problem. 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

There are files in the HDFS which are smaller than the size of the HDFS block. Each file or block is an object in the namespace and occupies around 150 bytes in the NameNode memory as the related metadata. Consider if a file of 1Gb was to be stored in HDFS, with a default block size of 64 Mb and a default replication factor of 3, this means we would need 16x3 blocks on the DataNodes and 16x3x150=2400 bytes in the NameNode memory. Now if we

consider 1000 files of 1Mb, and assume that each file will be stored independently on a block. This shows that 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[6][7][8]. As a result, for the same physical storage 250 times additional memory space will be required, compared to the previous example. There are many fields that produce tremendous numbers of small files continuously such as that for the analysis of multimedia data mining, astronomy, meteorology, signal recognition, climatology, energy, and E-learning where numbers of small files are in the ranges of millions to billions[9][10][11]. A fine example is Facebook, which has stored more than 260billion images. The human genome generates up to 30 million files averaging 190KB. 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[12][13].

Considering Hadoop processing, the time required to process a huge number of small files can be a hundred times slower than processing one single large file that has the same total size. Under a default configuration, Hadoop creates a mapper for each file[14].

## 4. Related Work

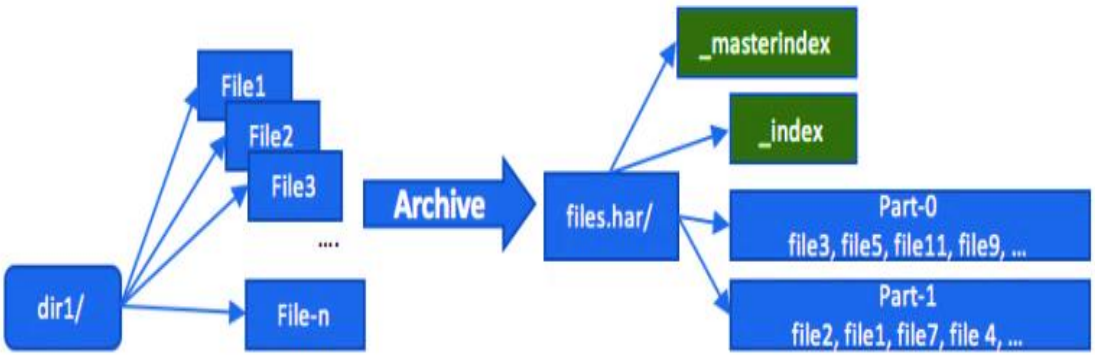



**Figure 4:** HAR File Layout

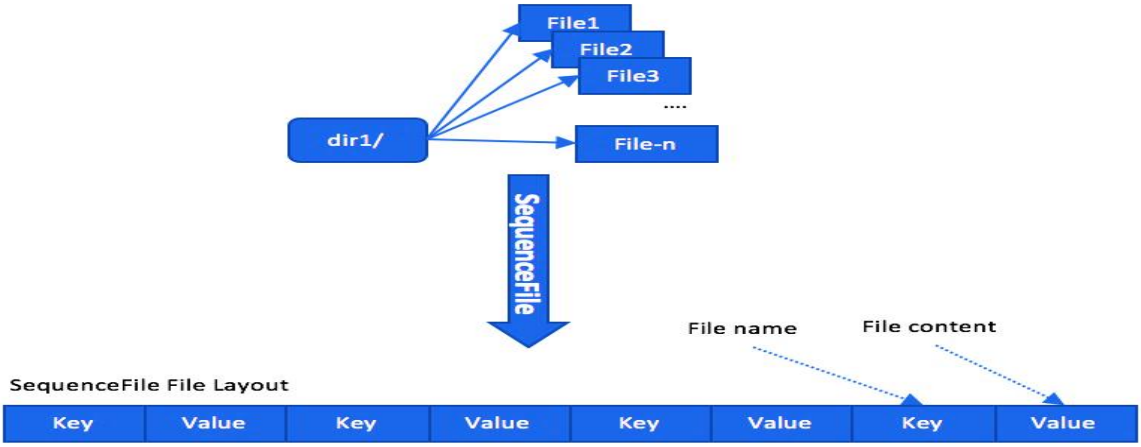



**Figure 5:** Sequence File Layout

Fig. 3 represents a Hadoop Archive that packs small files into a large file, so that it can access original files transparently. This technique ultimately allows for more and better storage efficiency, as only metadata of the archive is recorded in the namespace of the NameNode, but it doesn't resolve other constraints in terms of reading performance. This archive also cannot be appended while adding more small files. The technique used by the SequenceFile is to merge a group of small files in a 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.

C. Vorapongkitipun , al. 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 , al. 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. [16][17][18]

Y. Zhang et al. 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 , al. 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 themetadata management, also the reading access time is greatly reduced, but it takes more time to create the NHAR archives compared to the HAR mechanism[19][20].

P. Gohil , al. proposed a scheme for combining small file, merging, prefetching the related small files which improves the storage

and access efficiency of small files, but does not give an appropriate solution for independent small files[21].

## 5. The proposed approach for small file management

Currently, small files dramatically decrease the performance of Hadoop clusters. Previous solutions worked around the problem by trying to package small files in different formats. The formats are saved transparently in HDFS as they could 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 principal idea behind this approach is to store files when a client starts a stream that contains small files, 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 can be achieved using a Small File Analyzer, see "Fig. 5".

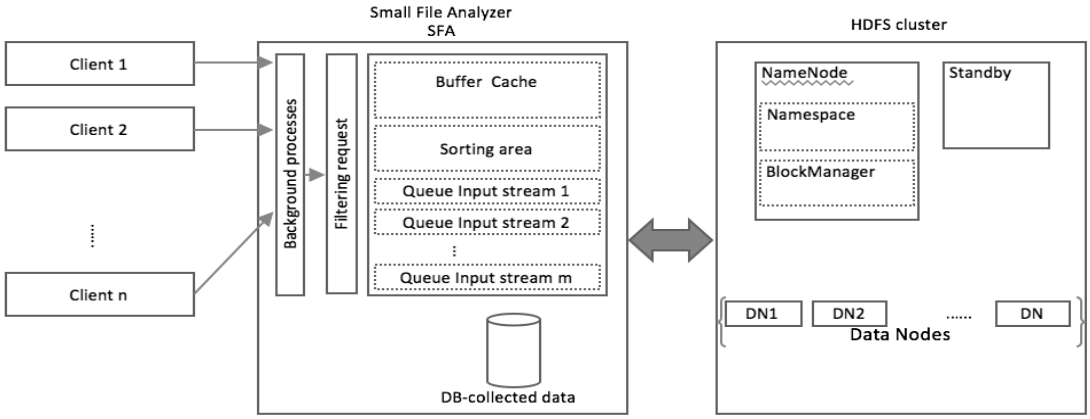



**Figure 6:** SFA Architecture1

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".

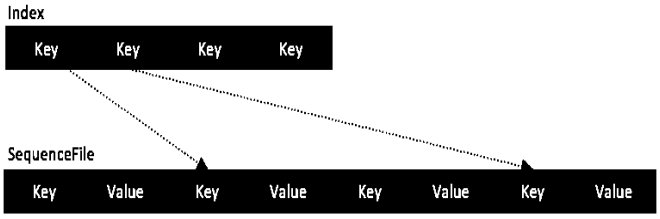



**Figure 7:** Map File Layout

Phase 2 can be triggered or called manually as analyzing small files depends on 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.

in order to improve the SFA analysis, we need to import records from the FS image of the NameNode, this contains a complete state of the HDFS files, 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.

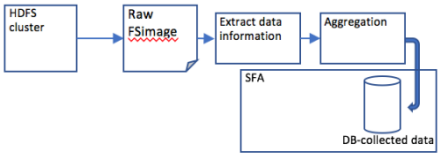



**Figure 8:** Map File Layout

*C.* 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.

---

Based on the grouped small files in this step2, SFA will request a list of blocks. Each time a block id is full, SFA deletes it from its blocklist. Like this SFA will write the next coming request to the first block id 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 the MapFiles. 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<?extendsWritable>keyClass= null;
Class<?extendsWritable>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 Evaluation

A simulation has been done 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..

*D.* Comparison of the NameNode Memory Usage

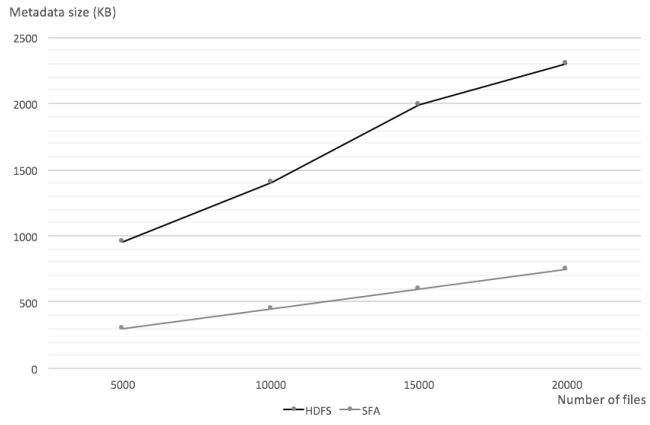



**Fig. 8:** Name Node memory usage

From "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. This doesn't however show if our block allocation strategy is efficient or not.

## 7. Comparison of MapReduce jobs performance

We performed MapReduce jobs on four datasets. The results on HDFS referred to the measurement of time requirement for 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 the measurement of the time required of MapReduce job after the second call and the sixth job iteration. Each measurement is performed after reorganizing small files as suggested from the SFA

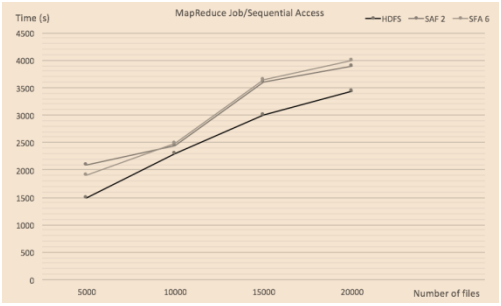



**Figure 9**: MapReduce Performance/Sequential Access

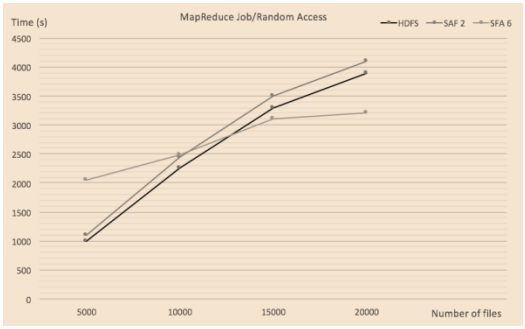



**Figure 10:** MapReduce Performance on selective files

From "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.

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 job performed 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 situation through an index improves the performance of that specific MapReduce job.

Probably in our next research we will adjust the SFA by introducing more factors to combine related streams efficiently in the allocated blocks. We could also 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.

## 8. Conclusion and Future Works

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 environments but 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.

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. the future work will be more precise, in order to help researchers to use tachnology to obtain positive results.

## 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] D. 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.