Impact of Small Files on Hadoop Performance: Literature Survey and Open Points

Tharwat EL-SAYED, M. Badawy, and Ayman EL-SAYED

Computer Science & Eng. Dept., Faculty of Electronic Eng.,
Menoufia University, Menouf 32952, Egypt.
tharwat_uss89@hotmail.com, mohamed.badawi@el-eng.menofia.edu.eg
ayman.elsayed@el-eng.menofia.edu.eg

(Received: 3 Jan. 2018 – Accepted: 10 Apr. 2018)

ABSTRACT

Hadoop is an open-source framework written by java and used for big data processing. It consists of two main components: Hadoop Distributed File System (HDFS) and MapReduce. HDFS is used to store data while MapReduce is used to distribute and process an application tasks in a distributed processing form. Recently, several researchers employ Hadoop for processing big data. The results indicate that Hadoop performs well with Large Files (files larger than Data Node block size). Nevertheless, Hadoop performance decreases with small files that are less than its block size. This is because, small files consume the memory of both the DataNode and the NameNode, and increases the execution time of the applications (i.e. decreases MapReduce performance). In this paper, the problem of the small files in Hadoop is defined and the existing approaches to solve this problem are classified and discussed. In addition, some open points that must be considered when thinking of a better approach to improve the Hadoop performance when processing the small files.

Keywords: Hadoop, Big Data, Small Files, Hive, HBase, Amazon EMR, S3DistCp.

1. INTRODUCTION

Hadoop is an open-source software framework and is developed to run in a distributed environment for parallel computing and storage. It is known that Google File System (GFS) was the atom of Hadoop Distributed File System (HDFS). The HDFS consists mainly of several DataNodes which are used for storing the data and one NameNode which is used for organizing client access to the DataNodes and manages the metadata of the stored files, Figure 1 represents the HDFS architecture [1-6].

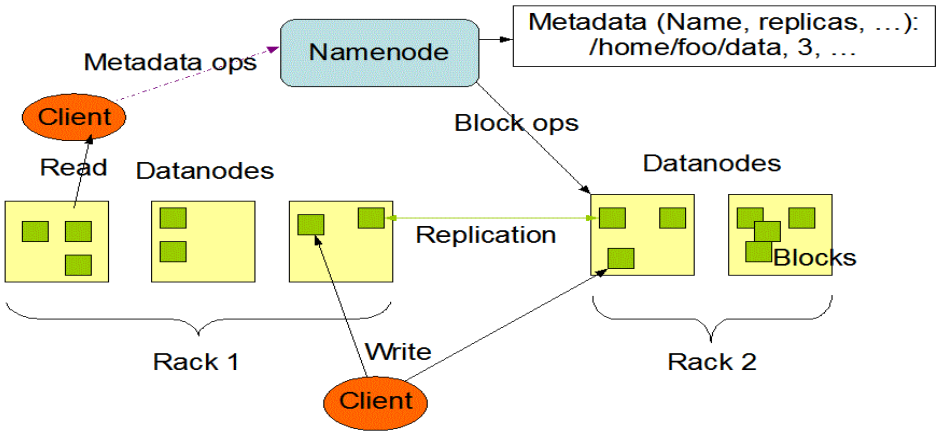

Figure 1: The HDFS Architecture

Hadoop was developed for dealing with large files and enabling streaming data access patterns [7]. These large files should be in the range of DataNode block size (128 Mbyte) or its doubles. Files that are less than the DataNode block size results in a huge burden on Hadoop performance, like high memory consumption and increasing CPU processing time [8, 9]. So, there is a big challenge in storing large number of small files in the HDFS and processing them. Solving the problem of the small file is more complex. The reasons why Hadoop faces the problem of small files: the management of the NameNode memory and the MapReduce performance [10].

To evaluate the performance of Hadoop there are some evaluation metrics that are shown in [11] as follow: DataNode Memory: the number of blocks consumed by each file reflects the optimum use of the DataNode memory. Files of size smaller than the default block size will waste a lot of DataNode memory. NameNode Memory: defines the size of the metadata stored in HDFS. The capacity of namespace of the system is limited by the physical memory. Files, directories, and blocks are named objects in HDFS and each of which requires 150 bytes. CPU time spent: defines the time taken by the MapReduce application. The less CPU processing time the highest Hadoop performance. This paper presents a literature survey of the impact of small files on Hadoop performance and discusses the existed solutions to overcome this problem.

This paper is organized as follow: In section 2, we provide a taxonomy of small files in Hadoop solutions and the advantage and disadvantage for each approach for small files problem. In section 3, we discuss the open points. Finally, we conclude this survey in section 4.

3. The Existed Solutions for the Small Files problem in Hadoop

This section presents a literature survey a comparative study of the following approaches: Hadoop Archive Files, Federated NameNodes, Change the ingestion process/interval, Batch file consolidation, Sequence files, HBase, and S3DistCp.

3.1 Hadoop Archive Files

Principles: For solving the small files problem, the Hadoop Archive approach (HAR) was done by Hadoop itself. It is obvious from the name that an archive file (.har) should be created [12], the main idea in the HAR approach is to put the small files in an archive file. Converting a large set of small files into one (.har) file results in a good enhancement on the NameNode meta-data. Figure 2 shows the HAR architecture [13].

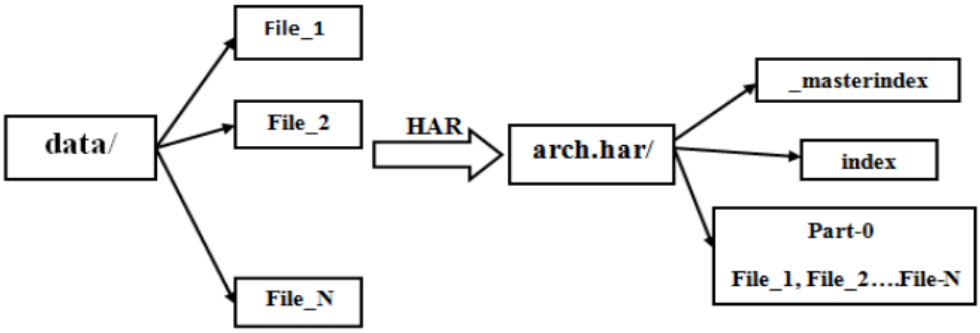

Figure 2: HAR Architecture

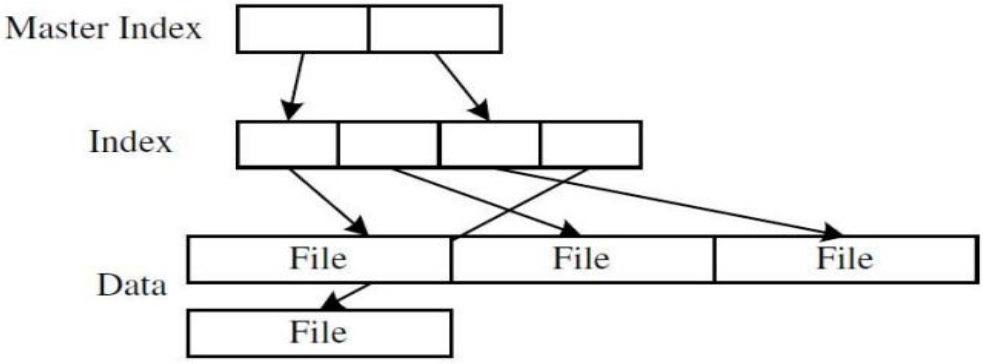

Figure 3: Reading Files from HAR

MapReduce programs can access the part files in parallel. To access a file from part-0 two index files should be used. Figure3 shows how the reading process done in HAR. Two indexing steps is an overhead to the NameNode, as two different index files should be stored in NameNode. **Discussion:** HAR files used to overcome the problem of the NameNode, but the processing performance may worsen. HAR Files should be used when data stored for archival purposes [14]. If the small files are part of our normal processing flow, another approach should be used.

3.2 Federated NameNodes

Principles: in large cluster, there are thousands of DataNodes and a single NameNode that holds the all the object metadata, this strategy is not a good solution to make a single NameNode holds all the metadata, Federated NameNodes is a good solution to overcome this problem by allowing to have multiple NameNodes in the cluster [15, 16]. If the cluster houses multiple tenants and applications, each NameNode will store the metadata for a definite tenant. Figure 4 shows the architecture for federated NameNode approach.

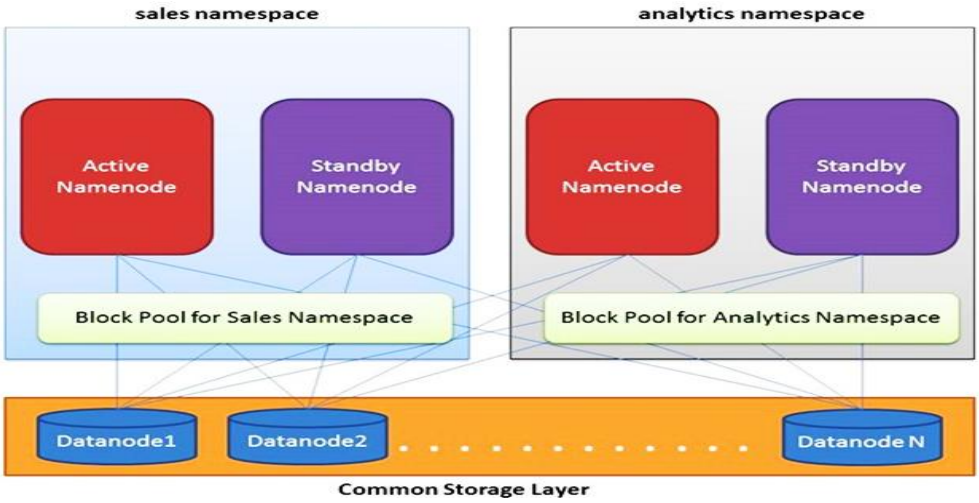

Figure 4: Federated NameNode Architecture

Discussion: Federated NameNodes make isolation for the object metadata; each NameNode will store only the tenant metadata.

3.3 Change the Ingestion Process/Interval

Principles: To get rid of the small files problem easily, is to avoid the small files to be stored inside the HDFS, this process can be done by changing the data ingestion interval process inside the Hadoop, changing the source system

to produce large files instead of the large number of the small files, or converting the small files to large ones by concatenating them before storing in the HDFS. If we are ingesting 10 MB of data per hour, and the ingestion is modified to be once a day, we will produce a 1x240MB large file instead of 24x10MB small files [17-18]. **Discussion:** we may not have control over the source system creating the files or business needs require that we ingest data at interval frequencies such that small files are unavoidable [2]. If small files are unavoidable and data ingestion intervals should be short, then another solution should be reached.

3.4 Batch File Consolidation

Principles: in this approach, the problem of the small files is solved by bagging these small files in large files, the small files will be stored in a definite folder and a MapReduce job will run periodically for rewriting the small files into larger files. This solution can be done by using only 2 lines of Pig, both a load statement and a store statement [19-20], as following:

A = load '/data/inputDirectory' using PigStorage();

Store A into '/data/outputDirectory' using PigStorage();

If we have 1000 small files in a definite folder and 5 reduces are specified for the MapReduce job, the 1000 small files will be merged into 5 larger files. This will reduce the DataNode memory consumption; the metadata stored in the NameNode, and will in turn enhance the Hadoop performance. These MapReduce jobs require cluster resources during execution, and are scheduled to run during off hours. However, they should be run frequently enough so the small files impact over the Hadoop performance does not become too extreme. **Discussion:** Batch file consolidation does not index or save the original file names. So, if the names of original files are important when processing, then batch file consolidation will not be the perfect solution for solving the problem of the small files.

3.5 Sequence Files

Principles: The Sequence files approach solves the problem of saving the names of the small files [21-22], each sequence file will contains two components: (1) The key which stores the file name, (2) The value which stores the file content. Table 1 shows how the small files would be stored in a sequence file:

Table 1: Storing the small files in a sequence file.

Key	Value	Key	Value	Key	Value
file1.txt	file1 contents	file2.txt	file2 contents	fileN.txt	fileN contents

If we have 2,000,000 small files stored in the HDFS, then the created sequence file will contain 2,000,000 keys, one key per file. Sequence files support the option of block compression, and these Sequence files are split able, this means that the MapReduce jobs will launch each map task for each block (128MB) rather than each map task for each small file. This reduction in the number of all the map tasks results in a large reduction in the MapReduce operations execution time and in turn enhancing the Hadoop performance. The Sequence Files approach is a perfect solution when the name of the input files needed to be saved. In addition, if a large number of the small files should be stored at the same time in Hadoop, then multiple sequence files should be created in parallel to aggregate this large number of the small files. **Discussion:** Hadoop files are immutable and cannot be appended to, so we need to ingest a large number of the small files at a time in order to make the sequence file works well. In addition, Hive software does not work well with sequence files structure. For creating one sequence file a large time will be consumed, so the sequence file is not practical in case of existing a plenty of large files.

3.6 HBase

Principles: HBase tables can be used for restoring a large number of HDFS small files as individual records. HBase approach is the best solution, if the data access pattern is based on a well-defined or random-access lookups. Several advantages result in using HBase as, high-velocity data inserts, real time processing and individual record lookups. HBase approach performance is not well in case of full data scan, so, for queries tend toward full table scans, HBase is an inefficient solution [23, 24]. **Discussion:** if the data access pattern tends toward full table/file scans, then the HBase approach may not be optimal. Balancing HBase with other cluster processes requires advanced system administration. Additionally, HBase performance depends largely on data access patterns and this point should be considered before we choose HBase for solving the small file problem.

3.7 S3DistCp

Principles: This solution is only used by users of Amazon EMR. Amazon EMR clusters are short lived, storing the data in Amazon S3. S3DistCp is a tool provided by Amazon used for distributed copying of data from S3 to HDFS. The utility enables to concatenate files together by using of group by and target size options. This is useful when we have large number of the small files stored in S3 that should be processed by using Amazon EMR. S3DistCp has two benefits; concatenating several small files and making the data appear faster than normal HDFS storage. The results show that the performance improvement using this mechanism is 15x [25-26]. **Discussion:** processing a large number of the small files still results in launching more map tasks than necessary decreasing performance. S3DistCp works the batch file consolidation approach.

4. Discussion and open points

The DataNode block size is set to 128 Mbyte and there is a trend to double it, because of this trend, small files problem becomes unavoidable and solving

this problem is a good achievement. Several approaches have been produced for solving the NameNode memory consumption problem and the MapReduce performance problem like Hadoop Archive Files, Federated NameNodes, Change the ingestion process/interval, Batch file consolidation, Sequence files, and HBase and S3DistCp. Sometimes choosing between these approaches becomes elusory but can be evaluated according to some metrics like the number of DataNode blocks consumed to store each file, NameNode memory consumed and the time taken by the CPU to perform operations on the data(MapReduce performance). Sequence Files approach uses (key/value) pair technique for solving the small files problem in Hadoop, this approach has several advantages but it is still suffering from some drawbacks like (1) It is slow to convert existing data into Sequence Files. (2) It is not suitable in case of large number of large files stored in the HDFS. An enhancement must occur on the Sequence Files approach to overcome its previous drawbacks. We are going to enhance the sequence files strategy in order to improve the Hadoop performance with small files as efficient as possible. In addition, there are some points that should be considered when we talk about the recent approaches, as follows:

4.1 NameNode Consumed Memory

An observed enhancement over NameNode Memory used space can be reached when we use approaches like HAR approach, Change the Ingestion Process/Interval approach, Batch File Consolidation approach, Sequence Files approach, HBase approach and S3DistCp approach, but when we use Federated NameNodes approach it makes isolation for the object metadata and each NameNode will store only the tenant metadata, so, Memory enhanced by isolation not by reducing the used space.

4.2 Processing time

MapReduce applications processing time will be enhanced when we use approaches like Federated NameNodes approach in case of homogenous systems, Change the Ingestion Process/Interval approach, Batch File Consolidation approach, Sequence Files approach, HBase approach in case of real time processing and individual record lookups and S3DistCp approach, but MapReduce applications processing time will be increased when we use

approaches like HAR approach, Federated NameNodes approach in case of heterogeneous systems and HBase approach in case of full data scan.

4.3 Hadoop performance

The performance of Hadoop will be enhanced when we use approaches like Federated NameNodes approach, Change the Ingestion Process/Interval approach, Batch File Consolidation approach, Sequence Files approach, HBase approach in case of real time processing and individual record lookups and S3DistCp approach, but when we use approaches like HAR approach and HBase approach in case of full data scan results in a bad Hadoop performance.

5 Conclusion

Hadoop framework was developed for storing and processing big data. However, the size of the files stored in the HDFS impacts on the Hadoop performance. For large files, (the file size close to or double the DataNode block size), Hadoop runs well. Nevertheless, for small files (file size significantly smaller than the block size), Hadoop performance degrades. This is because, storing and processing small files inside the HDFS results in a high overhead in the system. Briefly, the default DataNode block size is 128 MB and each file will be stored in a separate block so the DataNode storage will be consumed for storing small files. In addition, each file will be defined as a name object in HDFS and each of which requires 150 bytes so a large numbers of the small files stored in the system metadata occupies a large portion of the namespace. The time consumed by the MapReduce operations will increase. Therefore, the performance of the Hadoop will go down when large number of small files are stored in the HDFS. In this paper, several open points, which should be taken into consideration to improve Hadoop performance with small files, are discussed.

References

[1] Youssef M. ESSA, Gamal ATTIYA and Ayman EL-SAYED, "**Mobile Agent based New Framework for Improving Big Data Analysis**", Proceeding of the 2013 IEEE International Conference on Cloud Computing and Big Data (CloudCom-Asia 2013), Fuzhou, China, December 16-19, 2013.

[2] White, Tom, " **Hadoop: The definitive guide**", O'Reilly Media, Inc., 2012.

[3] Wang, Feng, et al. "**Hadoop high availability through metadata replication**" Proceedings of the first international workshop on Cloud data management. ACM, 2009.

[4] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung, "**The Google file system**", ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

[5] Mall, Nupur N., and Sheetal Rana, "**Overview of Big Data and Hadoop**", Imperial Journal of Interdisciplinary Research 2.5, 2016.

[6] Manjunath, R., R. K. Channabasava, and S. Balaji, "**A Big Data MapReduce Hadoop distribution architecture for processing input splits to solve the small data problem**", Applied and Theoretical Computing and Communication Technology (iCATccT), 2016 2nd International Conference on. IEEE, 2016.

[7] Mir, Mansoor Ahmad, and Jawed Ahmed, "**An Optimal Solution for small file problem in Hadoop**", International Journal of Advanced Research in Computer Science 8.5, 2017.

[8] Zheng, Tong, Weibin Guo, and Guisheng Fan, "**A Method to Improve the Performance for Storing Massive Small Files in Hadoop**", 7th International Conference on Computer Engineering and Networks, 2017.

[9] Qin, Dongxue, "**Study on Processing of Massive Small Files Based on Hadoop**", Liaoning University, China, 2011.

[10] Sharma, Garima, and Anita Ganpati, "**Performance evaluation of fair and capacity scheduling in Hadoop YARN**", Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on. IEEE, 2015.

[11] Fu, Songling, et al., "**Performance Optimization for Managing Massive Numbers of Small Files in Distributed File Systems**", IEEE Transactions on Parallel and Distributed Systems, Vol. 26, No. 12 pp. 3433-3448, 2015.

[12] Vorapongkitipun, Chatuporn, and Natawut Nupairoj, "**Improving performance of small-file accessing in Hadoop**", Computer Science and Software Engineering (JCSSE), 2014 11th International Joint Conference on. IEEE, 2014.

[13] Dev, Dipayan, and Ripon Patgiri, "**HAR: Archive and metadata distribution! Why not both?**", Computer Communication and Informatics (ICCCI), 2015 International Conference on. IEEE, 2015.

[14] Huang, Yicheng, et al., "**Towards model-based approach to Hadoop deployment and configuration**", Web Information System and Application Conference (WISA), 2015 12th. IEEE, 2015.

[15] Chintapalli, Sanket Reddy, "**Analysis of Data Placement Strategy based on Computing Power of Nodes on Heterogeneous Hadoop Clusters**", Diss. Auburn University, 2014.

[16] Xiong, A. P., and J. Y. Ma., "**HDFS distributed metadata management research**", International Conference on Applied Science and Engineering Innovation, 2015.

[17] Xie, Jiong, et al., "**Improving mapreduce performance through data placement in heterogeneous Hadoop clusters**", Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. IEEE, 2010.

[18] Eltabakh, Mohamed Y., et al., "**CoHadoop: flexible data placement and its exploitation in Hadoop**", Proceedings of the VLDB Endowment 4.9 (2011): 575-585, 2011.

[19] Shahrivari, Saeed. "**Beyond batch processing: towards real-time and streaming big data**", Computers 3.4 (2014): 117-129, 2014.

[20] Zhou, Fang, "**Assessment of Multiple MapReduce Strategies for Fast Analytics of Small Files**", 2015.

[21] White, Tom, "**The small files problem**", Cloudera Blog, blog. cloudera. Com/blog/2009/02/the-small-files problem, 2009.

[22] Gohil, Parth, and Bakul Panchal, "**Efficient ways to improve the performance of HDFS for small files**", Computer Engineering and Intelligent Systems 5.1 (2014): 45-49, 2014.

[23] Team, Apache HBase, "**Apache hbase reference guide**", Apache, version 2.0, 2015.

[24] Harter, Tyler, et al., "**Analysis of HDFS under HBase: a Facebook messages case study**", FAST. Vol. 14, 2014.

[25] Deyhim, Parviz, "**Best Practices for Amazon**", Technical report, 2013. Vorapongkitipun, C., & Nupairoj, N., "**Improving performance of small-file accessing in Hadoop**", In, 2014 11th IEEE International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 200-205, May, 2014.

الملخص باللغة العربية

هو إطار مفتوح المصدر مكتوب بلغه الجافا يستخدم لمعالجة (Hadoop)هادوب من عنصرين رئيسيين: الأول هو البيانات الضخمة بطريقة موزعة. يتكون هادوب حيث (MapReduce) والثاني هو ماب رديوس (HDFS)نظام الملفات الموزعة يستخدم العنصر الأول في تخزين الملفات المرتبطة بالتطبيق الواحد في وحدات موزعة بينما يستخدم العنصر الثاني في معالجة الملفات الموزعة من خلال إسناد العمليات المرتبطة بالتطبيق الواحد إلى الوحدات المختلفة لتنفيذها ومن ثم تجميع النتائج الفرعية لبناء النتيجة النهائية للتطبيق. وقد أثبتت الدراسات المختلفة أن هادوب يعمل بشكل جيد مع الملفات الكبيرة بينما يقل أداء هادوب مع الملفات الصغيرة وذلك لأن الملفات الصغيرة التي تمثل التطبيق الواحد تستهلك ذاكرة أكبر تستغرق زمن أكبر في التوزيع على وحدات كم (DataNodes)من أماكن لتخزين التنفيذ ومن ثم يزيد من وقت تنفيذ التطبيق الواحد. تقدم هذة الورقة البحثية دراسة مرجعية لتأثير الملفات الصغيرة على أداء هادوب ومناقشة الطرق المقترحة