International Journal of Computer Applications (0975 – 8887)
Volume 146 – No.1, July 2016

An Efficient Approach for Storing and Accessing Small Files with Big Data Technology

Bharti Gupta
Research Scholar
Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India

Rajender Nath
Professor
Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India

Girdhar Gopal
Assistant Professor
Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India

Kartik
Student
Kurukshetra Institute of Technology & Management, Kurukshetra, Haryana, India

ABSTRACT
Hadoop is an open source Apache project and a software framework for distributed processing of large datasets across large clusters of computers with commodity hardware. Large datasets include terabytes or petabytes of data where as large clusters means hundreds or thousands of nodes. It supports master slave architecture, which involves one master node and thousands of slave nodes. NameNode acts as the master node which stores all the metadata of files and various data nodes are slave nodes which stores all the application data. It becomes a bottleneck, when there is a need to process numerous number of small files because the NameNode utilizes the more memory to store the metadata of files and data nodes consume more CPU time to process numerous number of small files. This paper presents a novel technique to handle small file problems with Hadoop technology based on file merging, caching and correlation strategies. The experimental results shows that the proposed technique reduces the amount of data storage at NameNode, average memory usage of DataNodes and improves the access efficiency of small files in Hadoop Distributed File System up to 88.57% as compared with the general solution Hadoop Archive.

General Terms
Big Data Analytics, Small files in Hadoop.

Keywords
Hadoop, HDFS, Map Reduce, small files in Hadoop, small files storage in Hadoop.

1. INTRODUCTION
In recent years, Hadoop has become one of the most popular high performance distributed computing paradigm for large scale data analytics. Hadoop is an open source software framework developed for reliable, scalable, distributed computing and storage. Hadoop architecture consists of two main layers that are Hadoop Distributed File System (HDFS) and MapReduce programming model. HDFS, the primary storage system of Hadoop is a distributed file system designed to run on commodity hardware and store extremely large files suitable for streaming data access patterns. HDFS is highly fault tolerant and is able to scale up from a single server to thousands of machines, each offering the same functionality that is local computation and storage. HDFS protects the data by replicating data blocks into multiple nodes, with a default replication factor of 3. HDFS has a master/slave architecture. It consists of two types of nodes: NameNode which is known as “master” and several DataNodes which are called as “slaves” [1]. MapReduce is a programming model to process large datasets and make use of computing resources of each server's CPU. It comprises of two phases: Map phase and Reduce phase. Every job must contain one map function followed by optional reduce function, these steps need to follow this certain order. MapReduce incorporates JobTracker and TaskTrackers [2]. The storage system of Hadoop is not physically separated from the processing system [3]. Hadoop is excellent when it comes to handle large files of data. HDFS divides the input data into data blocks of minimum 64 MB in size. NameNode stores the metadata of these data blocks and DataNodes store the actual data blocks. These data blocks are processed by MapReduce. But with the increasing scale of small files, Hadoop is inefficient in handling numerous number of small files whose size ranges from 10 KB to 5 MB. These small files are generated by word docs, pdf files, flash files, power point, MP3 and so on [4].

These numerous small files can bring serious performance issues with Hadoop because storing these many small files into Hadoop becomes an overhead in memory usage of metadata stored in NameNode, so it impacts on the size of memory in the NameNode. Thus, a large number of small files take significant amount of NameNode's main memory [5]. To process these many small files more number of MapReduce tasks are created, it creates an overhead between MapReduce tasks and CPU time.

This research intends to make the use of Hadoop based file merging and caching technique that has proven effective and efficient experimentally. The technique reduces the memory use of NameNode and DataNodes to store data blocks. The research objective is to create effective number of MapReduce tasks to process HDFS data blocks, which drastically reduces MapReduce task overhead and the total CPU time. An efficient approach for storing and accessing small files is developed and implemented on Hadoop framework. This approach reduces the memory utilization of NameNode to store metadata files. This technique can be used for all type of resources on file system which makes it more general in use.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents the problem formulation. Section 4 proposes a novel technique to handle small files problem. Section 5 presents the experimental results and discussions. Conclusions and future work are drawn in section 6.

2. RELATED WORK
As the applications of the HDFS increases, the pitfalls of the HDFS are also being discovered. Among them is the poor performance when the HDFS handles small and frequently interacted files. They pointed out that HDFS is designed for processing big data transmission rather than transferring a large number of files [6]. Shvachko from Yahoo! described

the detailed design and implementation of HDFS. They realized that their asumption that applications would mainly create large files was flawed, and the new applications for HDFS would need to store a large number of smaller files [1]. They described as there is only one NameNode in Hadoop which keeps all the metadata in main memory, a large number of small files produce significant impact on metadata performance of HDFS [7] [8].

Research on small file problem of HDFS has attracted the significant attention and it is believed that there are three issues that need to be solved in a more appropriate way. The first issue is the identification of 'how small is small'. Liu et al. treated the files smaller than 16 MB are small files, no proof or justification was provided for the same. The second issue is the classification of small files. He discussed the small files into two types: (i) files that are pieces of a large logical file (ii) files that are inherently small. The third issue is the solutions to the small file problem. Solutions are classified into two categories: general solutions and special solutions [3]. Hadoop Archive (HAR) [7], SequenceFile [3] and MapFile [1] are typical general solutions to small files optimization for Hadoop used by various researchers.

HAR is a file archiving facility that packs a number of small files into large HDFS blocks so that the original files can be accessed in parallel transparently and efficiently. It contains metadata (in the form of _index and _masterindex) and data (part-*) files. Part files contain the content of files which are part of the archives [4].

A SequenceFile is a flat file and provides a persistent data structure for binary key- value pairs. It uses the file name as the key and file contents as the value, and supports the compressing and decompressing at record level or block level. Small files can be put into a single sequence file that is processed using MapReduce operating on sequence file [3].

A MapFile is a type of sorted SequenceFiles with an index to lookup operation by key. It consists of two files, a data file and a smaller index file. All of the sorted key-value pairs are stored in the data file and the key location information is stored in index file. MapFile does not search for entire file when looking for a specific key [1].

Researchers merged the page files into a large one and built an index for each book to store book pages for digital libraries. There was no detailed scheme to improve the access efficiency [9].

They discussed a special solution which combined the small files into a large one and built a hash index for each small file which stores the small data of Geographic Information System on HDFS. Merging of neighboring files and reserving several versions of data were considered [10].

3. PROBLEM FORMULATION

As it is evident from the related work discussed in section 2, when small files are stored on HDFS, disk utilization is not a bottleneck. In general, small file problem occurs when memory of NameNode is highly consumed by the metadata and BlockMap of huge numbers of files. NameNode stores file system metadata in main memory and the metadata of one file takes about 250 bytes of memory. For each block by default three replicas are created and its metadata takes about 368 bytes [11]. Let the number of memory bytes that NameNode consumed by itself be denoted as α. Let the number of memory bytes that are consumed by the BlockMap be denoted as β. The size of an HDFS block is denoted as S. Further assume that there are N files, whose lengths are denoted as L_1, L_2, …., L_N, then the memory consumed by the NameNode is given by

$$M_{NN} = 250 * N + (368 + \beta) * \sum_{i=1}^{N} \frac{L_i}{S} + \alpha \qquad (1)$$

In order to reduce the memory consumption of NameNode, the number of small files that NameNode manages and number of blocks need to be reduced [13]. The various techniques [1] [3], [7], [10] to handle the small files problem have been proposed in the literature but they still suffer from many limitations such as (i) In HAR, creating an archive generates a copy of original files, which puts extra pressure on disk space as once a .har is created it takes as much space as the original files. Don't mistake .har files for compressed files. (ii) Reading through files in a HAR is no more efficient than reading through files in

HDFS, and in fact may be slower since each HAR file access requires two index file reads as well as the data file read as shown in diagram.

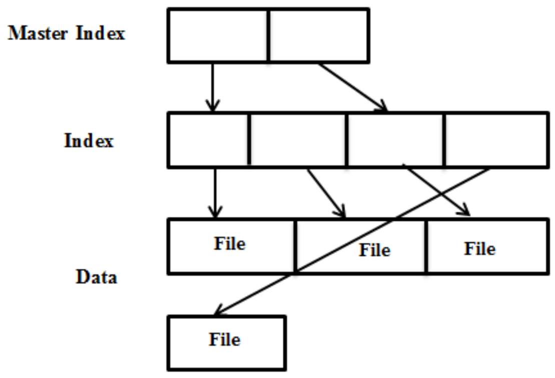

Figure 1: Layout of the HAR File

(iii) When a .har file is given as an input to MapReduce job, the small files inside the .har file will be processed individually by separate mappers which is inefficient and no mechanism is provided to improve access efficiency. (iv) SequenceFile does not support update/delete method for a specified key; it only supports append method whereas MapFile only supports append method for a specified key. (v) The existing techniques such as HAR, SequenceFile, and MapFile do not consider file correlations while storing files. (vi) The special solution provided by the Liu et al. uses the index technique only, which further needs improvement.

4. PROPOSED APPROACH

To address the problems mentioned in the last section a novel technique to handle the small files is proposed based on file merging and caching techniques. The proposed technique is composed of five phases: (i) File merging strategy (ii) Local index file strategy (iii) Fragmentation of files (iv) Uploading of files to HDFS (v) File caching. These phases are discussed in detail below.

Phase 1: File merging strategy: In this phase, merging operation is carried out at client side. The merging strategy merges all the small files into a single big file and does not perceive the existence of original small files, thus to reduce the consumption of NameNode memory.

Phase 2: Local index file strategy: A local index file is created for each original file to indicate its offset and length in the merged file. It consists of four parameters “Location of small file”, “Starting Page No.”, “End Page No.”, “Merged File Name”.

Phase 3: Fragmentation of files: Files will be fragmented when merged, so that no internal fragmentation of files occur in HDFS blocks.

Phase 4: Uploading of files to HDFS: Both of the files, local index file and merged file are written to HDFS which avoid overhead involved in keeping the information at NameNode. NameNode keeps the information of merged file and index file only. File correlations are considered when storing the files to improve the access efficiency.

Phase 5: File caching strategy: The caching strategy is used to cache local index file and correlated files. Based on the strategy, communications with HDFS are drastically reduced thus to improve the access efficiency, when downloading files. When a requested file misses in cache, the client needs to query NameNode for file metadata. According to the metadata, the client connects with appropriate DataNodes where blocks locate. When the local index file is firstly read, based on the offset and length, the requested file is split from the block, and is returned to the client [13].

5. EXPERIMENTAL RESULTS AND DISCUSSIONS

The experiment is conducted on a cluster with 3 machines. One machine acts as the NameNode and the other two machine acts as the DataNodes. Each of these machine has Intel i7 processor, 4 GB RAM, 500 GB hard disk and operating system is Ubuntu 12.04. Hadoop version is 2.0.6 and the java version is 1.8. The number of replicas is set to 3 and the HDFS block size is 64 MB. The workload consists of 246 small files in the range of 10 KB to 5 MB. These are of size 260 MB. The performance of the proposed technique is compared with the existing general solution HAR to analyze the memory usage of the NameNode, average memory usage of DataNodes and time taken to process files. The following parameters are considered to measure the performance of proposed technique and existing general solution HAR:

- Memory Usage of the NameNode to store metadata.
- Average Memory Usage of DataNodes to store data blocks.
- Time taken in the MapReduce phases to process files. Total time taken by CPU to process files.

Table 1. Memory Usage of the NameNode

Approach	Memory Usage (MegaBytes)
Existing HAR Approach	624
Proposed Approach	331

Memory Usage of NameNode has reduced upto a great extent from 624 MegaBytes to 331 MegaBytes as compared between existing approach HAR and the proposed approach as shown in table 1 and figure 2.

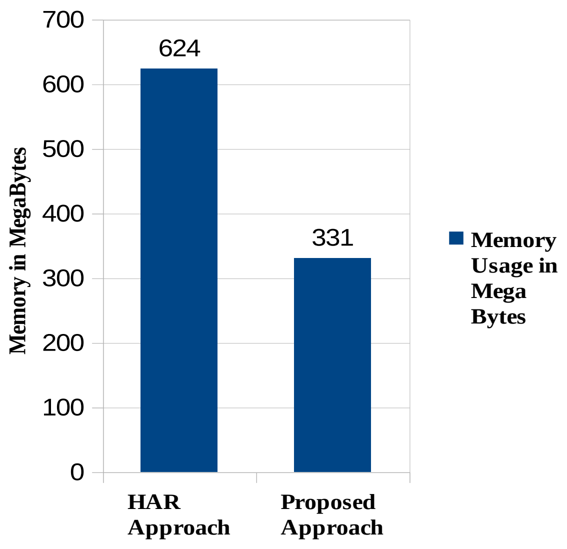

Figure 2: Memory Usage of the NameNode

Table 2. Average Memory Usage of the DataNodes

Approach	Memory Usage (MegaBytes)
Existing HAR Approach	1711
Proposed Approach	1129

Table 2 and figure 3 shows that Average Memory Usage of the DataNodes has reduced upto a great extent from 1711 MegaBytes to 1129 MegaBytes as compared between existing approach HAR and the proposed approach.

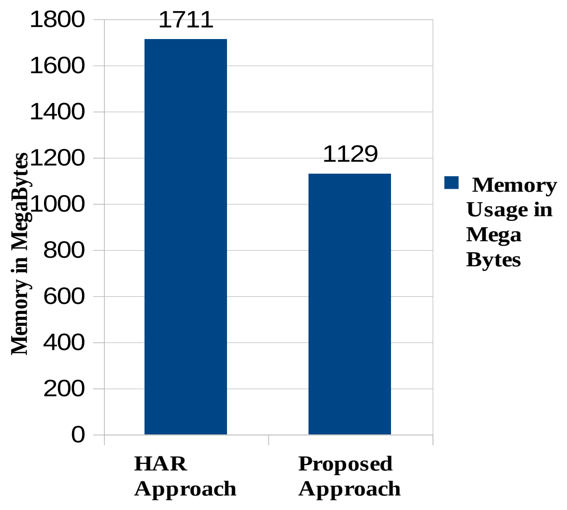

Figure 3: Average Memory Usage of the DataNodes

Table 3. Map, Reduce and Total CPU Time Comparison

Approach	Map Time (sec)	Reduce Time (sec)	Total CPU Time (sec)
Existing HAR Approach	64	18	70
Proposed Approach	33	9	8

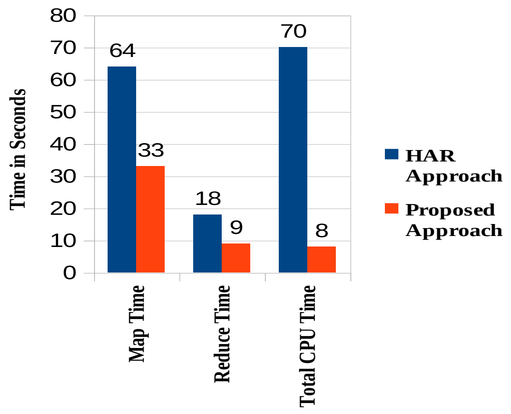

Figure 4: Map, Reduce and Total CPU Time Comparison

The experiments has improved the efficiency of storing and accessing small files in HDFS. The processing time of small files has drastically reduced upto 88.57% as compared with existing approach HAR as shown in table 3 and figure 4.

6. CONCLUSION AND FUTURE WORK

This paper presents a solution to handle small files problem in Hadoop based on file merging and caching techniques. The possibility of making the searching faster with the help of prefetching and index file has been exploited in this work. The experimental evaluation has demonstrated that the proposed technique has reduced the NameNode memory consumption and has improved the efficiency of storing and accessing small files in HDFS. It has been observed from the experimental results that the memory usage of the NameNode to store metadata has reduced from 624 MegaBytes to 331 MegaBytes and average memory usage of the DataNodes to store data blocks has reduced from 1711 MegaBytes to 1129 MegaBytes. It is further observed that the processing performance of small files has been improved up to 88.57 % by the proposed approach, which is very promising. As for future work, small file storage solutions on HDFS will be mainly studied for other types of files which will include structurally related files and logically related files. Based on the file correlation analysis, small files are classified as multiple types and customized approaches will be supplied to different types to further improvement of efficiency. In the future work, the cut-off point between large and small files will be further studied.

7. REFERENCES

[1] Shvachko, K., Hairong, K., Radia, S., Chansler, R. 2010. The Hadoop Distributed File System. In proceedings of IEEE 26th Symposium in Mass Storage Systems and Technologies (MSST). 1-10.

[2] Yuan, Yu, Cui, C., Wu, Y., and Chen, Z. 2013. Performance analysis of Hadoop for handling small files in single node. Computer Engineering and Application. Vol. 49, no. 3. 57-60.

[3] White T. 2009. The Small Files Problem. http://www.cloudera.com/blog/2009/02/the small files problem.

[4] Dong, B., Qiu J., and Zheng Q. 2010. A Novel Approach to improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files. IEEE International Conference on Services Computing, 978-0-7695-4126-6/10. 65-72.

[5] White, T. 2010. Hadoop: The Definitive Guide. 2nd ed. O'Reilly Media, Sebastopol, CA. 41-45.

[6] Jiang, L., Li, B., and Song, M. 2010. The optimization of hdfs based on small files. In 3rd IEEE International Conference on Broadband Network and Multimedia Technology, IC-BNMT. 912-915.

[7] Mackey, G., Sehrish, S., and Wang, J. Aug 31- Sep 4, 2009. Improving metadata management for small files in HDFS. In proceedings of IEEE International Conference on Cluster Computing and Workshops. New Orleans, USA. 1-4.

[8] Min, L., and Yokata, H. 2010. Comapring hadoop and fat-btree based access method for small file I/o applications. Web-age information management, Lecture notes in computer science. Vol. 6184, Springer. 182-193.

[9] Shen, C., Lu, W., Wu, J., and Wei, B. 2010. A digital library architecture supporting massive small files and efficient replica maintenance. In Proceedings of the 10th annual joint conference on digital libraries. ACM. 391-2.

[10] Liu, X., Han, J., Zhong, Y., Han, C., and He, X. 2009. Implementing webgis on hadoop: a case study of improving small file i/o performance on HDFS. In IEEE international conference on cluster computing and workshops, CLUSTER'09. 1-8.

[11] Shvachko, K. 2007. Name-node memory size estimates and optimization proposal. https://issues.apache.org/jira/browse/HADOOP-17S.

[12] Dong, B., Zheng, Q., Tian, F., Chao, K.M., Ma, R., Anane, R. July 2012. An optimized approach for storing and accessing small files on cloud storage. In Proceedings of Journal of Network and Computer Applications 35. 1847-1862.

[13] Gupta, B., Nath, R., Gopal, G. April, 2016. A Novel Techniques to Handle Small Files with Big Data Technology. In Proceedings of Vivechana : A National Conference on Advances in Computer Science and Engineering (ACSE) held at Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India on 29-30 April 2016.