A Novel Technique to Handle Small Files with Big Data Technology

Bharti Gupta
M.Tech Scholar, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India
Email: gupta.bharti2503@gmail.com
Rajender Nath
Professor, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India
Email: rnath2k3@gmail.com
Girdhar Gopal
Assistant Professor, Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India
Email: girdhar.gopal@kuk.ac.in

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 proposes a novel technique to handle small file problems with Hadoop technology based on file merging, caching and correlation strategies. The proposed technique has reduced the amount of data storage at NameNode as has been proved by theoretical validation.

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

1. Introduction

In today's world, it is increasingly inseparable from the network, people visit hundred of pages, upload photos or videos on the daily basis, by which data content on the network is growing at exponential rate. Traditional architectures has become unable to process these vast amount of data. Therefore, for processing and storage of vast amount of data many new technologies like Hadoop has been reported in the literature [12]. Hadoop is an open source software framework for distributed processing of large datasets across large clusters of computers. Hadoop architecture consists of two main layers that are Hadoop Distributed File System (HDFS) and MapReduce programming model. HDFS 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 which consists of two types of nodes, namely, a NameNode called “master” and several DataNodes called “slaves” [8]. 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. In Map phase, mapper must be able to ingest the input and process

that input record and then that processed record is forwarded to Reduce phase, where tasks are reduced. The mapper takes in a key/value pair and generates intermediate key/value pairs. The reducer merges all the pairs associated with the same intermediate key and produce the final output that is list of key/values. 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 [12]. The storage system of Hadoop is not physically separated from the processing system [10]. 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 gradually becomes powerless. It is inefficient in handling numerous number of small files whose size ranges from 10 KB to 5 MB [2].

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 will take significant amount of NameNode's main memory [11]. To process these many small files more number of MapReduce tasks are created, it creates an overhead between MapReduce tasks and CPU time [4].

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 theoretical validation of the proposed technique. Section 6 concludes the paper.

2. Related Work

Researcher [8] described the detailed design and implementation of HDFS. They realized their assumption that applications would generally create large files was flawed and new applications for HDFS would need to store a large number of small files. They [6] [7] 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.

They [5] discussed that HDFS is designed to read/write large files and there is no optimization mechanism for small files. There would be a mismatch in accessing patterns if HDFS is used to read/write a large number of small files. Research on small file problem of HDFS has put up three issues that need to be solved in a more appropriate manner. The first issue is the identification of 'how small is small'. They 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 [10] 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. HAR [6] , SequenceFile [10] and MapFile [8] are typical general solutions to small files optimization for Hadoop used by various researchers.

HAdoop Archive (HAR) 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 the metadata files and data files. The file data is stored in multiple part files, these part files are indexed for keeping the original separation of data intact. The metadata files can record the original directory information and the file states. HAR can reduce the memory consumption of NameNode to a great extent [6].

A SequenceFile is a flat file which consists of binary key-value pairs. It uses the file name as the key and contents of file as value. Small files can be put into a single sequence file that is processed using MapReduce operating on sequence file [10].

A MapFile is sorted SequenceFile 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 [8].

They [5] 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. They [13] discussed that HDFS had inefficient issue with small files and traditional methods had low efficiency performance and high consumption of resources. They discussed a proposed approach to efficiently store and process large number of small files by the procedure of file merger. It consisted of two modules. First, small files written and merger and the other one was small file read and separation. In the file processing strategy before the merging of small files index of 16 bytes in file head got added. The size of the merged new file could not be more than HDFS minimum file chunk that is 64 MB.

3. Problem Formulation

As it is evident from the related work discussed in the 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 [9]. 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

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

In order to relieve the memory consumption of NameNode, the number of small files that NameNode manages and number of blocks need to be reduced [3]. The various techniques [6], [5], [8], [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 and no mechanism is provided to improve access efficiency. (ii) 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. (iii) The existing techniques such as HAR, SequenceFile, and MapFile do not consider file correlations while storing files. (iv) The special

solution provided by the Liu et al. uses the index technique only, which further needs improvement.

4. Proposed Technique to Handle Small Files Problem

To address the problems mentioned in the last section a novel technique to handle the small files has been 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”. These two phases are carried out with the following algorithm.

Algorithm : File Merging and local index file creation

Begin

Input:

F_{small}: Set of small files to be merged.

F_{merged}: Name of merged file.

count = Number of files uploaded to HDFS for merging.

constantCount = count;

offset=0;

While(count>0)

For each F_{small} Do Begin

name=F_{small}.name;

P_{start}=offset;

P_{End}= F_{small}.length;

merged_name=F_{merged}.name;

If((F_{merged}.size() + F_{small}.length) > $HDFS_{BlockSize}$)

count=constantCount- count;

BoundryFilling(count);

Else read(F_{small})

write(F_{merged});

insert(name, P_{Start}, P_{End}, merged_name)

offset = P_{End};

count--;

End For

End While

Output:

F_{merged}: Final merged file.

F_{Index}: Local index file for set of merged small files.

Phase 3: Fragmentation of files: Files will be fragmented when merged, so that no internal fragmentation of files occur in HDFS blocks as shown in figure 1.

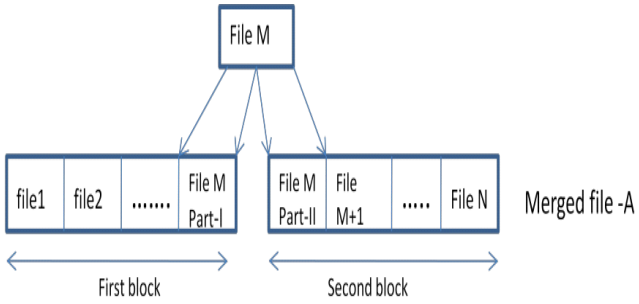

Fig. 1 (a): Merged File before applying the Boundary Filling

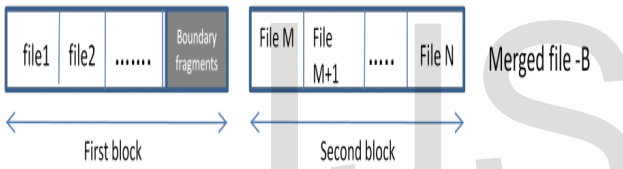

Fig. 1 (b): Merged File after applying the Boundary Filling

Following algorithm is used to avoid the internal fragmentation.

Algorithm: Boundary Filling

Begin

Input:

F_{small}: Set of small files to be merged.

$newF_{merged}$: New name of merged file.

count =Number of small files remained to be uploaded to HDFS.

constantCount= count;

offset=0;

While(count>0)

For each F_{small} Do Begin

name=F_{small}.name;

P_{Start} = offset;

P_{End}= F_{small}.length;

merged_name= $newF_{merged}$.name;

If(($newF_{merged}$.size()+F_{small}.length)>$HDFS_{BlockSize}$)

count=constantCount- count;

boundryFillingAlgo(count); //RECURSION

Else

readFully(F_{small})

writefully($newF_{merged}$);

insert (name, P_{Start}, P_{End}, merged_name);

offset = end;

count--;

End For

End While

Output:

F_{merged}: Set of small merged file.

F_{Index}: Local index file for set of merged small files.

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 queries the NameNode for file metadata. According to the metadata, the client connects with appropriate DataNodes where blocks locate. When the local index file is read, based on the offset and length, the

requested file is split from the block, and is returned to the client.

5. Theoretical Validation Of The Proposed Technique

Suppose there are N small files, which are merged into K merged files whose lengths are denoted as L_{M1}, L_{M2}, …, and L_{MK}. The computational formula of the consumed memory of NameNode in file merging and caching technique is given as

$$M_{NN} = 250 * N + (368 + \beta) * \sum_{i=1}^{K} \lceil \frac{L_{Mi}}{S} \rceil + \alpha \quad (2)$$

The number of blocks required to store the files in HDFS is calculated as

$$N_{block} = \sum_{i=1}^{K} \lceil \frac{L_{Mi}}{S} \rceil \quad (3)$$

Memory consumption of NameNode does not have relations with the number of original files N, but is relevant with the number of merged files K, which is much smaller than N. Thus, number of blocks in (3) are much smaller than the number of blocks produced with equation (1) with small files. So It concludes that the file merging and caching strategy has effectively reduce the memory consumption of NameNode, and improve storage efficiency.

As the NameNode only maintains the metadata of merged files and does not perceive the existence of original small files, thus it does not put any extra pressure on disk space, thus improves access efficiency.

The proposed solution not only uses the index technique but it also uses the file caching and file correlation techniques, thus improves access efficiency.

6. Conclusion

This paper has proposed a novel technique to handle small files by using Hadoop – a big data technology. Hadoop is inefficient in handling small files because storing these small files into HDFS becomes an overhead in memory usage of metadata stored in NameNode as more number of Mappers and Reducers are required to process these files. The proposed technique is based on file merging and caching strategies. Theoretical validation of the proposed technique has shown that it has reduced the NameNode memory consumption and has improved the efficiency of storing and accessing small files in HDFS.

References

[1] J. Dean, S. Ghemawat, “MapReduce: simplified data processing on large clusters”, Commun. ACM 51, pp. 107-113, 2008.

[2] Bo Dong, Jie Qiu and Qinghua Zheng, “A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files”, IEEE Internatioal Conference on Services Computing, 978-0-7695-4126-6/10, pp. 65-72, 2010.

[3] Bo Dong, Qinghua Zheng, Feng Tian,Kuo-Ming Chao, Rui Ma, Rachid Anane, “An optimized approach for storing and accessing small files on cloud storage”, In Proceedings of Journal of Network and Computer Applications 35, pp. 1847-1862, July 2012.

[4] Hadoop, The Hadoop Architecture and Design, June 2014, Available: http://hadoop.apache.org/docs/r0.23.11/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, Last Accessed On April 5, 2016.

[5] X Liu, J Han, Y Zhong, C Han, X He, “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, pp. 1-8, 2009.

[6] G. Mackey, S. Sehrish and J. Wang, "Improving metadata management for small files in HDFS", In Proceedings of IEEE International Conference on Cluster computing and workshops,New Orleans, USA, pp. 1-4, August 31- September 4, 2009.

[7] L. Min, H. Yokota, "Comparing 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, pp. 182–193, 2010.

[8] K. Shvachko, K. Hairong, S. Radia, R. Chansler, "The Hadoop Distributed File System", In Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10, 2010.

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

[10] Tom White, "The Small Files Problem", 2009, Available: http://www.cloudera.com/blog/2009/02/the small files problem.

[11] T. White, "Hadoop: The Definitive Guide: The Definitive Guide", O'Reilly Media, Sebastapol, CA, 2010.

[12] Yu Yuan, Chaoyuan Cui, Yun Wu, Zhuhong Chen, "Performance analysis of Hadoop for handling small files in single node", Computer Engineering and Application, vol. 49, no. 3, pp. 57 - 60, 2013.

[13] Yang Zhang, Dan Liu, "Improving the Efficiency of Storing for Small Files in HDFS", In Proceedings of IEEE International Conference on Computer Science and Service System CSSS , 978-0-7695-4719-0/12 , pp. 2239-2242, 2012.