Improving Performance of Hadoop in Handling Small Files

Arshid Ahmed shah[1], Mangala C.N[2]
[1, 2] Department of computer science and engineering
[1, 2] East West Institute of Technology, Bengaluru, Karnataka

Abstract- *Hadoop is an open Source software stack for processing unstructured and semi-structured data over the cluster of Commodity hardware. It is a reliable, scalable and low cost solution for storing and processing huge amounts of data in a distributed form. It is designed for processing and handling large files and faces performance penalty while dealing with large number of small files. The large number of small files put a heavy burden on the namenode memory resulting in high Namenode memory usage. Also processing large number of small files requires many map phases which increase the overall map execution time. This Paper describes the MapReduce merge mechanism to merge large number of small files into a single merge file and then store and process the corresponding Merge File. The small files are given as input to a Map phase with key as filename and value as key size which adds the files greater than some threshold (85 % of the block size of 64 MB) to the list. This list is forwarded to the Reduce phase that Outputs the Merged File. This approach is quite effective than previous solutions with less node metadata memory requirement and Improved Map execution time.*

Keywords- Hadoop; MapReduce; HDFS; Namenode; Small Files .

I. INTRODUCTION

The advancement of Internet and Popularity of Web resulted in the generation of massive amounts of unstructured data. In order to process this mammoth data, apache hadoop was developed by a team at yahoo headed by Doug cutting and mike cafarella as an offshoot of the Nutch project. Hadoop is a scalable, reliable, fault tolerant, and cluster oriented distributive approach for storing unstructured data on the commodity hardware. It is a product of research by Google's published papers MapReduce and Google File System (GFS) [1]. Hadoop Uses a Map Reduce Programming paradigm to process the data by dividing the job into sub tasks and giving the subtasks to the nodes in a cluster [2]. The sub tasks are either map or reduce tasks where Reduce task are performed after all the Map tasks are processed. This programming model works on key value pairs and outputs the key value pairs making hadoop a batch processing system. The map Task takes input splits as an input, processes the input splits, generates Key, value as an output. These Map outputs are given to one or more reducers. The reducers reduce the data and store an output in HDFS.

Another component of hadoop is HDFS, a scalable, reliable, fault tolerant, and distributed file storage system which stores data in blocks and replicates the blocks in multiple nodes (default replication number is 3). This makes HDFS fault tolerant in that if one node is lost, the data can still be recovered or accessed from its replica on other nodes. HDFS includes Namenode which stores the file system metadata and is single point of failure.

The HDFS works well with the large files and faces a performance penalty while dealing with large number of small files as large number of small file put a heavy burden on the namenode resulting in increased Namenode start time, increased Namenode Memory requirement and more number of map phases. The above problem is termed as small file problem in hadoop terminology. This paper gives an map reduce approach to merge thousands of small files into a few merged files improving Namenode Memory Usage, improved namenode load time and less map task requirement.

This paper is divide into number of sections, section II gives the brief introduction about the background of hadoop, section III describes the small file problem, section IV explains the existing solutions and their limitations, section V describes the Proposed solution, section VI explains the experimental setup and analyses the different output parameters, and section VII depicts conclusion and future work.

II. BACKGROUND

Hadoop is an open source software stack for processing data over a cluster of commodity hardware. The two main components of hadoop are MapReduce and HDFS [2] [3]. Map reduce is a programming model while as HDFS is a distributed storage file system. HDFS is reliable, fault tolerant and scalable file system which stores data in the cluster in distributed form.

In hadoop cluster HDFS works in master slave paradigm consisting of a two types of nodes, NameNode and DataNode. Namenode stores metadata information of the file system while as DataNodes store the actual data blocks in their storage disks.

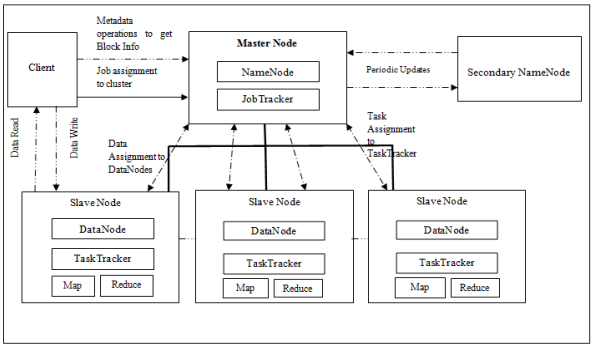

Figure 1: HDFS Architecture

A. NameNode

Namenode stores the file system metadata in the main memory and is the sole proprietor of file system Image log. If Namenode fails, all the data in HDFS is lost. The hadoop cluster includes two Namenodes, one is primary and other is secondary. The secondary Namenode acts as a Backup but is not in sync with primary namenode. To improve Fault tolerance of NameNode, HDFS federation is employed where Multiple Namenodes store the part of file System metadata which means failure of one Namenode has no effect on other Namenodes and only that part of file metadata is lost.

B. DataNode

DataNodes store the Actual data in their respective disks and act as slave nodes. The data nodes continuously talk with Namenode by sending the heart beat signal at regular intervals. The data nodes store the multiple replicas of the same blocks so that if a block on one DataNode is lost, it can be recovered or accessed from its replica on another data node.

C. Job Tracker

In hadoop 1.x Job tracker assigns and schedules the jobs to the hadoop cluster. When job is submitted to the client it converts the input to input-splits and starts the multiple task trackers. Each task tracker is given either a Map or Reduce Task. In Hadoop 2.x YARN manages the scheduling while as MapReduce Processes the task using map and reduce phase.

D. HDFS Client

In hadoop cluster, client node loads the data, submit jobs and retrieve the results. It acts as interface to the hadoop cluster and starts jobs and can read, write and delete data including files and directories from HDFS.

E. Task Tracker

Task tracker works on slave nodes and processes the actual job in map and reduces phases. They periodically send the job status to the Job tracker. If any task fails, job tracker assigns the corresponding task to a new Task tracker with same data replicas.

F. Resource Manager

Hadoop 2.x includes YARN (yet another resource Negotiator) where resource manager schedules the tasks and MapReduce process the same task in individual map and reduce tasks.

III. SMALL FILE PROBLEM IN HADOOP AND EXISTING APPROACHES

Hadoop is designed to work with large files and thus large number of small files decreases the hadoop performance by increasing the namenode memory usage and Map Reduce execution time [7]. For example storing 10 million files require almost 3GB of RAM [6] and therefore storing billions of files requires terabytes of memory increasing the execution time and NameNode Load time. Number of solution s have been design to solve the problem out of which few are discussed below

1) Hadoop Archive [9]

Hadoop Archive merges large number of small files into a single archive file reducing the number of files in HDFS. Reading files in HAR is slow because two indexes are to be accessed before reading the file.

One is the master index for indexing the individual HAR file and other is the indexes of the individual files. Thus in order to increase the access speed, the file Locality should be improved in HAR File [9].

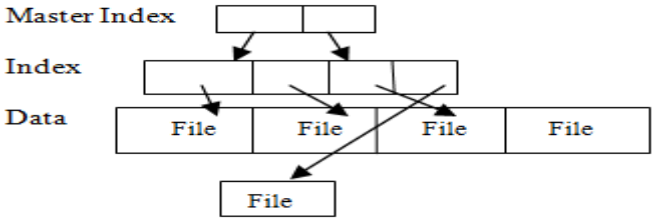

Figure 2: HAR File

2) Consolidator [10]

It takes the files from one or more folders and merges them into a single file. It has a disadvantage that original filenames are lost.

3) SEQUENCE files [10]

This technique creates a sequential file of individual files with key as name and value as content. The problem with sequential approach is the time to create a sequence is very high and there is no way to list all the files in the sequential file.

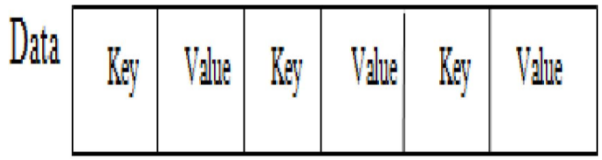

Figure 3: Sequential File

4) Hbase [10]

Hbase stores data in Map Files which are indexed sequential files and is a good choice for performing the streaming analysis .It provides a better interface and faster look up for files.

IV. PROPOSED APPROACH

This section describes the proposed approach by firstly giving an overview and then describing the merge algorithm.

The large number of small files decreases the hadoop performance in terms of memory usage of Namenode and increase in execution time of MapReduce. The proposed approach uses the Map reduce merge algorithm to merge small files into a merge file.

In the proposed approach the small files are given as an input to the map phase in the form of key value pairs, where key is the file name and value is the file size. The Map phase adds the files to the list; if file size is less than a threshold (90 % of hadoop block size 128 MB) .The Map phase continues to add files to the list until default file size is reached. The list is then passed to the reducer which merges the files in the list into single file and stores it in the HDFS.

MERGE ALGORITHM

MAP Phase

1) Read the input file name and size.
2) Check whether filename is greater than threshold (90 % of HDFS block size)
3) Add the small files less than threshold to the list
4) Pass the list to the reducer
5) Fetch new files

REDUCE Phase

1) Merge the files in the outputted list from Map Phases.
2) Store the merged file in HDFS.

V. EXPERIMENT AND RESULTS

The experiments for the proposed solutions were setup on live cluster with 3 nodes. Here 3 nodes indicate 1 Namenode and 2 DataNodes. All the three nodes are configured with hadoop 2.7.2.

A) Node 1 is based on Intel® Core i7 processor with 6 GB of RAM, 250 GB hard disk, and Operating system is Ubuntu 14.04.4 LTS .
B) Node 2 is based on Intel® Core i3 processor with 4 GB of RAM, 1TB hard disk, and Operating system is Ubuntu 14.04.4 LTS.
C) Node 3 is based on Intel® Core i5 processor with 4 GB of RAM, 1TB hard disk, and Operating system is Ubuntu 14.0.4 LTS.

A) Execution Time

In this solution we have considered the files in the size of KB's to few MB's. The total size of the files is 872 MB and the hadoop default size is taken as 128MB. The Files which are greater than 128 MB are ignored by MapReduce Merge Algorithm.

Without using any merging technique each file is stored directly in the HDFS. Each file is allocated to at least one block, thus having a lot of blocks require large number of map tasks which increases the overall execution time. The

Table 1 shows the execution time for different scenarios. The proposed solution takes less time to execute the same job.

LET,

A = time to convert small file into single file (Minutes)
B = Execution time to run word count program on converted files (Minutes)
C = total time to execute an application on small files (Minutes)

Thus, C=A+B

Table 1: Execution time Comparison with different approaches

Approach	A	B	C
HDFS	0	40	40
HAR	5.4	2.3	7.7
Sequential	20	3.5	23.5
Proposed Merge	4.3	1.5	5.8

The figure 4 shows the comparison of previous and proposed solutions. In HAR approach there is an inconsistency with the block size and sequence approach takes a lot of time to convert the small files into sequential file.

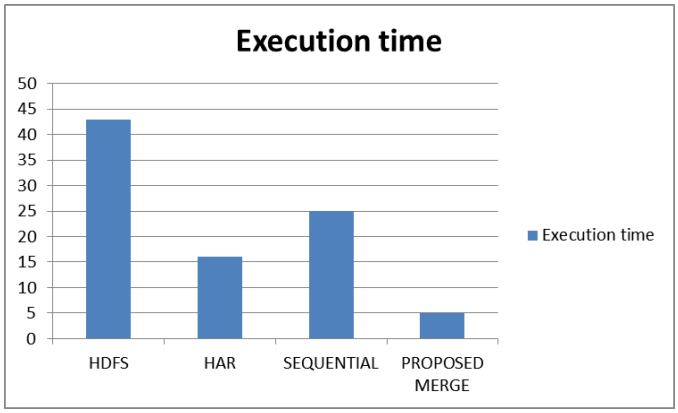

Figure 4: Execution time with different approaches

B) Namenode Memory usage

We used the Htop utility to evaluate how much memory the namenode Process takes using its process ID. The namenode memory analysis for storing 872 files has provided the results shown in Figure 5. It is clear that the proposed approach takes memory in the range of HAR and sequential files but very less as compared to the original HDFS

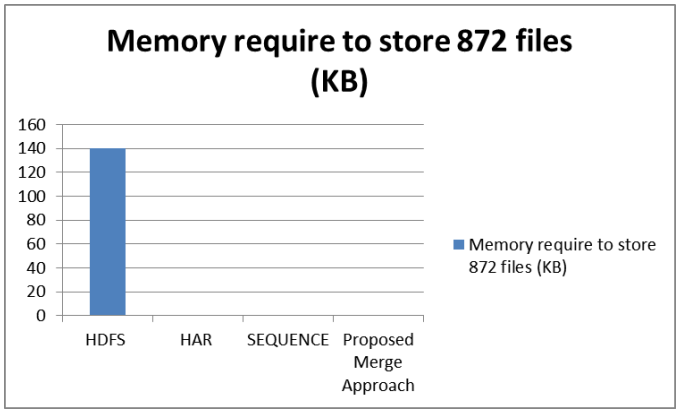

Figure 5: Memory usages with different approaches

VI. CONCLUSION AND FUTURE WORK

Improving hadoop performance in dealing with small files is one of the core active researches in hadoop community. The Mentioned solution improves performance as compared to the existing solutions. Firstly it reduces the Namenode memory requirement and secondly it reduces the Map execution time. The solution has been applied to CSV and Text files and has been found to be more effective than previous solutions.

The work can be continued forward for image files or other small file types. Image files are also type of small files and also suffer from the performance issues as with the CSV or Text files.

REFERENCE

[1] S. Ghemawat, H. Gobioff, and S.Leung, "The Google file System," Proceedings of ACM symposium on Operating System Principles, PP 29-43, October 2003.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[3] Apache hadoop Framework, http://hadoop.apache.org

[4] HDFS Architecture Guide. http://hadoop.apache.org/common/docs/current/hdfs/design.html, 2009.

[5] T. White, "The small files problem [online]." Available: http://blog.cloudera.com/blog/2009/02/the-small-files-Problem

[6] The-small-files-problem, Cloudera http://blog.cloudera.comlblog/2009/02/the-small-files-probleml,February2009

[7] Small-file-problem-in-hadoop.html, http://amilaparanawithana.blogspot.in/2012/06/small-file-problem hadoop.html, June 2012

[8] Http://issues.apache.org/jira/browse/HADOOP-1687

[9] Hadoop archives, http://hadoop.apache.org/common/docs/current/hadoop _archives.html.