ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 11, November 2018

A Novel Approach to Process Small HDFS Files using Apache Spark

Priyanka Dukale [1]

P.G. Student, Department of Computer Engineering, Vishwabharati College of Engg., Ahmednagar, Mahaarastra, India[1]

ABSTRACT: Hadoop is an open source distributed computing platform and HDFS is Hadoop Distributed File System. The HDFS has a powerful data storage capacity. Therefore, it is suitable for cloud storage system. However, HDFS was originally developed for the streaming access on large software; it has low storage efficiency for massive small files. To solve this problem, the HDFS file storage process is improved. The files are judged before uploading to HDFS clusters. If the file is a small file, it is merged and the index information of the small file is stored in the index file with the form of key-value pairs. The MapReduce based simulation shows that the improved HDFS has lower NameNode memory consumption than original HDFS and Hadoop Archives (HAR files). This memory consumption can be optimized significantly if MapReduce based file processing is reduced by Spark based file processing. Thus, it can improve the access efficiency as well.

KEYWORDS: Distributed file system, Hadoop small files, HAR Files, HDFS, MapReduce, Spark.

I. INTRODUCTION

The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS is a distributed file system that provides high-performance access to data across Hadoop clusters. Like other Hadoop-related technologies, HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.

HDFS is typically deployed on low-cost commodity hardware, so server failures are common. The file system is designed to be highly fault-tolerant, however, by facilitating the rapid transfer of data between compute nodes and enabling Hadoop systems to continue running if a node fails. That decreases the risk of catastrophic failure, even in the event that numerous nodes fail. When HDFS takes in data, it breaks the information down into separate pieces and distributes them to different nodes in a cluster, allowing for parallel processing. The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others. As a result, the data on nodes that crash can be found elsewhere within a cluster, which allows processing to continue while the failure is resolved.

HDFS is built to support applications with large data sets, including individual files that reach into the terabytes. It uses a master/slave architecture, with each cluster consisting of a single NameNode that manages file system operations and supporting Data Nodes that manage data storage on individual compute nodes.HDFS is originated primarily to tackle the problem of handling large files. As per core design, HDFS processes large files with exceptional performance. However, problem arises while processing large number of small files.Performance of HDFS decreases significantly while handling such usual cases. Various approaches have been proposed to address this issue through heterogeneous aspects. Some of them have been discussed in next section as well. A novel approach is proposed in this paper on a top of all these existing approaches. This approach looks towards brand new Spark technology as a more optimal way to process HDFS files with the in memory computation capabilities of Spark.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 11, November 2018

II. RELATED WORK

1.Harball Archive: Harball archive contains the metadata entry for the index of file it contains. This index serves as a meta-meta data layer for the data in the archive, causing slight overhead in referencing files. File referencing request goes through the metadata of the archive to the index of metadata that the archive contains. The overhead of file referencing is negligible as it is done in main memory. [1]

2.HDFS I/O Speed Optimization: HDFS is designed to store large files and suffers performance penalty while storing large amount of small files. This performance penalty can be reduced drastically by optimizing HDFS I/O speed of small files based on the original HDFS. The basic way is let one block save many small files and let the datanode save some meta-data of small files in its memory, this will reduce the read and write request received by the namenode; furthermore, by sorting files with directory and file name when reading and writing files will further optimize increase the speed of reading. [2]

3. Prefetching Technique: Access patterns play crucial role in performance if HDFS is being accessed by heterogeneous users. These performance issues have the remedy of an efficient Prefetching technique for improving the performance of the read operation where large number of small files is stored in the HDFS Federation. This Prefetching algorithm learns through the files access patterns progressively. Even though some existing solutions incorporated prefetching, it was purely based on the locality of reference. The proposed solution based on the file access patterns improves the read access time by 92 percent compared to the time taken withthe prefetching based on the locality of reference and 94 percent compared to the time taken without prefetching. The files are prefetched into the cache on webserver, which eliminates cache coherence problems. [3]

4. New Hadoop Archive (NHAR): NHAR redesigns the architecture of HAR in order to improving performance of small-file accessing in Hadoop. Since reading file from HAR need to access 2 indexes which affect access performance. To resolve this problem, the architecture of HAR has been modified to minimize the metadata storage requirements for small files and to improve the access performance. This approach can achieve obvious improvements on small I/O performance. [4]

5. Enhanced HDFS: In enhanced HDFS, the performance of handling interaction-intensive tasks is significantly improved. The modifications to the HDFS are: (1) changing the single namenode structure into an extended namenode structure; (2) deploying caches on each rack to improve I/O performance of accessing interactive-intensive files; and (3) using PSO-based algorithms to find a near optimal storage allocation plan for incoming files. Structurally, only small changes were made to the HDFS, i.e. extending single namenode to a hierarchical structure of name nodes. However, the experimental results show that such a small modification can significantly improved the HDFS throughput when dealing with interaction-intensive tasks and only cause slight performance degradation for handling large size data accesses. [5]

6. MapReduce Approach: MapReduce approach to handle small files, considers mainly two parameters. Firstly, execution time to run file on Hadoop Cluster and secondly the memory utilization by Namenode. By considering these parameters, proposed algorithm improves the result compared to existing approaches. It can handle both sequence file and text file efficiently and avoid files whose size is greater that threshold. [6].

7. Prefetching and Caching Approach: This approach focuses on small file problem of HDFS. Storing a huge number of small files results in high memory usage of namenode and increased access cost. The proposed approach combines a large number of small files into single combined file to reduce the memory usage of Namenode. The memory usage is more in HDFS because Namenode stores file as well as block metadata of each file. Thus, as the number of files increases, the memory usage also increases. The memory used by proposed approach is less as namenode stores merely file metadata of each small file. The block metadata is stored by the namenode for single combined file and not for every single file. As consequence, the memory usage of namenode is reduced. [7]

8. Improved HDFS File Process: This approach judges file before uploading them to HDFS cluster. Small files are merged and index information of the small file is stored in the index file as key-value pairs. This Hadoop architecture has lower Namenode memory consumption that original HDFS and Hadoop Archive (HAR) files, improving access efficiency. HDFS file stored process is improved by targeting low storage and access efficiency. Small file is evaluated before uploading to HDFS and is merged and the index information of the small file is stored in the index file, in the form of key-value pairs. The comparative analysis of storage and access efficiency is done against the result of original

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 11, November 2018

HDFS, HAR and improved HDFS. The improved NameNode memory consumption of the improved HDFS is least as compared to others. [8]

III. PROPOSED ALGORITHM

The proposed approach highlights the usability of Spark far faster file processing. The architectural model can be divided into three layers i.e. User Layer, Data Processing Layer and Storage Layer.

A. User Layer: This layer provides user with an ability to perform certain operations on HDFS files. User layer gives enduser an ease of access to add/ update/ delete files of HDFS. User interface of Ambari can be used to access the HDFS in a user friendly manner. Ambari has an authentication-oriented mechanism to protect HDFS files from unauthorized access.

B. Data Processing Layer:

This layer represents an operational unit of the complete architectural model. This unit can be split into four sequential parts as explained below:-

2.1. File Judging Unit:

This unit categorizes an input file based on its size. It determines whether an input file is small or big against the predefined file size threshold value. If the input file size value is below threshold level, it is sent to file merging unit, whereas for above threshold value, file is straightaway sent to HDFS client for processing it in a traditional HDFS way, as the conventional HDFS has best possible capability to process big size HDFS files.

2.2. Spark based File Processing Unit:

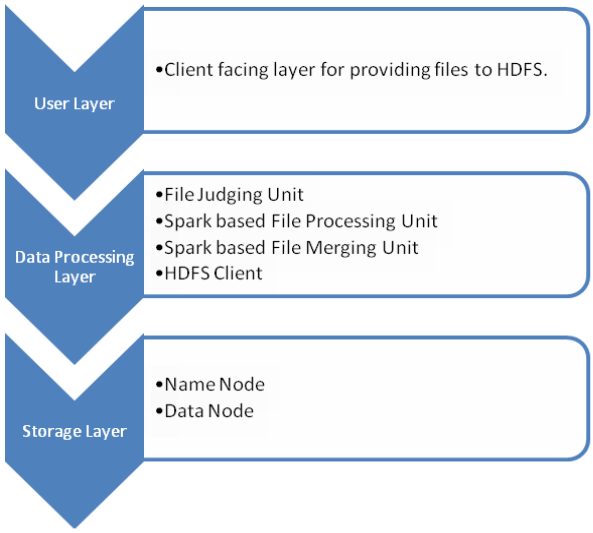
This unit receives files small size files from File Judging Unit, calculates file size and starts creating incremental offset values until the pre-defined block size exceeds. Spark is used here for file retrieval and in-memory offset calculation. The calculated files are the passed to Spark Based File Merging Unit.

2.3. Spark based File Merging Unit

After receiving computed files, this unit performs merging of small computer files using datasets of Spark. This merging operation refers predefined block sizes and offset values to assure creation of optimal file block sizes.

2.4. HDFS Client

File blocks having reached offset size are received by HDFS client and then stored in HDFS. This blocks are accompanied by the big sized files, already landed into HDFS directly from File Judging Unit.



Block Diagram of Proposed System

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 6, Issue 11, November 2018

IV. SIMULATION RESULTS

In this section simulation carried out on Map Reduce and Spark. It compares how the use of Spark instead of Map Reduce is efficient in handling small files in HADOOP.

Fast Data Processing:

As we are promising that Spark is faster than map reduce because it runs in-memory on the cluster, and that it isnt tied to Hadoops MapReduce two-stage paradigm. This makes repeated access to the same data much faster. Spark can run as a standalone or on top of Hadoop YARN, where it can read data directly from HDFS.

More Interactive Interface:

As we know Hadoop MapReduce can be programmed in Java which is comparatively difficult. Spark has comfortable APIs for Java, Scala and Python, and also includes Spark SQL for the SQL savvy so we easily write user defined functions. It even includes an interactive mode for running commands with immediate feedback. So this is one of the reason we prefer Apache Spark instead of Map Reduce.

Compatibility:

We are going to run Spark on top of Hadoop YARN which supports Hadoop Input format so we can integrate it with all data sources and file formats that are supported by Hadoop

Data Processing:

As we observed Spark can be used for real time processing and batch processing so it can process graphs and use the machine-learning libraries. So we need not to split tasks for different platform which we get at one place

IV. CONCLUSION AND FUTURE WORK

Aiming at the low store and access efficiency on small files in HDFS cloud storage, the HDFS file stored process is improved. If the file is a small file by judging before uploading to HDFS clusters, it is merged and the index information of the small file is stored in the index file with the form of keyvalue pairs. The file storage and access efficiency is analyzed through Spark. The results show that the improved NameNode memory consumption of the Spark based Hadoop Processing is the least. It can also gives 100 times faster performance than Map reduce. It can save the NameNode memory when storing the small files. Thus, the Spark based Hadoop Processing can optimize the access efficiency of small files.

REFERENCES

1. Mackey, S. Sehrish and J. Wang,"Improving Metadata Management for Small Files in HDFS", in 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), 2010.
2. L. Jiang, B. Li and M. Song, "The Optimization of HDFS Based on Small Files", EURASIP Journal on Embedded System by Springer-Link 2013: 2013(6).
3. A.K,A.R.A,S.M.C,C.Babu and P.B, "Efficient Prefetching Technique for Storage of Heteregeneous small files in Hadoop Distributed File System Federation," in Fifthe International Conference on Advanced Computing (ICoAC), 2013.
4. C. Vorapongkitipun and N. Nupairoj, "Improving Performance of Small- File Accessing in Hadoop," in 11th International Joint Conference on Computer Science and Software Engineeting (JCSSE), 2014.
5. X. Hua, W. Hao and S. Ren, "Enhancing Throughput of Hadoop Distributed File System for Interaction- Intensive Tasks," in 22nd Euromicro Intternational Conference on Parallel, Distributed, and Network- Based Processing, 2014.
6. P. Gohil, B. Panchal and J. S. Dhobi, "A Novel Approach to Improve the Performance of Hadoop in Handling of Small Files," in Electrical, Computer and Communication Technologies (ICECCT) IEEE International Conference, 2015.
7. M. A. Mehta and A. Patel, "A Novel Approach for Efficient Handling of Small Files in HDFS," in IEEE International Advance Computing Conference (IACC), 2015.
8. L. Changtong, "An Inproved HDFS for Small Files," in 18th International Conference on Advanced Communication Technology (ICACT), 2016.