Data Compression Technique On Hadoop's Next Generation: Yarn

Vanisha Mavi[1], Nidhi Tyagi[2]

[1]*M.Tech student, CSE, AKTU Lucknow*

[2]*Professor MIET, Meerut*

Abstract - Data Compression is a strategy for encoding decides that permits considerable diminishment in the aggregate number of bits to store or transmit a document [1]. There is a complete range of different data compression techniques available both online and offline working such that it becomes really difficult to choose which technique serves the best. In this paper we represent number of techniques to compress and decompress the text data and summarize the design, development, and current state of deployment of Hadoop. It also gives an overview of YARN concepts, related issues and challenges, tools and techniques.

Keywords - Big Data, YARN, Data compressions.

1.1 INTRODUCTION

Big data described as a collection of structured, unstructured and semi structured data [1].

2.1 Hadoop

Hadoop is a distributed system infrastructure researched and developed by the Apache Foundation [2]. The users can develop the distributed applications although they do not know the lower distributed layer so that the users can make full use of the power of the cluster to perform the high-speed computing and storage. The core technology of Hadoop is the Hadoop Distributed File System (HDFS) and the MapReduce.

3.1 Compression,

Compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation [3]. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying unnecessary information and removing it. The process of reducing the size of a data file is referred to as data compression [4]. In the context of data transmission, it is called source coding (encoding done at the source of the data before it is stored or transmitted) in opposition to channel coding.

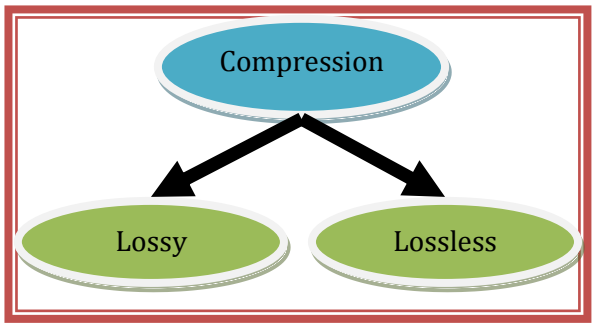

Figure 1:-Types of compression

3.2 What to compress?

- **Compressing input files:** If the input file is compressed, then the bytes read in from HDFS is reduced, which means less time to read data. This time conservation is beneficial to the performance of job execution. If the input files are compressed, they will be decompressed automatically as they are read by MapReduce, using the filename extension to determine which codec to use. For example, a file ending in .gz can be identified as gzip-compressed file and thus read with GzipCodec.
- **Compressing output files:** Often we need to store the output as history files. If the amount of output per day is extensive, and we often need to store history results for future use, then these accumulated results will take extensive amount of HDFS space. However, these history files may not be used very frequently, resulting in a waste of HDFS space. Therefore, it is necessary to compress the output before storing on HDFS.
- **Compressing map output**: Even if your MapReduce application reads and writes uncompressed data, it may benefit from compressing the intermediate output of the map phase. Since the map output is written to disk and transferred across the network to the reducer nodes, by using a fast compressor such as LZO or Snappy, you can get performance gains simply because the volume of data to transfer is reduced.

3.3 Common input format

Compression format	Tool	Algorithm	File extension	Split table
gzip	gzip	DEFLATE	.gz	no
bzip2	bzip2	bzip2	.bz2	yes
LZO	Lzop	LZO	.lzo	Yes, if indexed
Snappy	n/a	Snappy	. Snappy	no

Table 1:- Common input formats

3.4 Choosing a Data Compression Format

Whether to compress your data and which compression formats to use can have a significant impact on performance. Two of the most important places to consider data compression are in terms of MapReduce jobs and data stored in HBase. For the most part, the principles are similar for each [5].

- You need to balance the processing capacity required to compress and uncompress the data, the disk IO required to read and write the data, and the network bandwidth required to send the data across the network. The correct balance of these factors depends upon the characteristics of your cluster and your data, as well as your usage patterns.
- Compression is not recommended if your data is already compressed (such as images in JPEG format). In fact, the resulting file can actually be larger than the original.
- GZIP compression uses more CPU resources than Snappy or LZO, but provides a higher compression ratio. GZip is often a good choice for *cold data*, which is accessed infrequently. Snappy or LZO are a better choice for *hot data*, which is accessed frequently.
- BZip2 can also produce more compression than GZip for some types of files, at the cost of some speed when compressing and decompressing. HBase does not support BZip2 compression.
- Snappy often performs better than LZO. It is worth running tests to see if you detect a significant difference.
- For MapReduce, if you need your compressed data to be splittable, BZip2, LZO, and Snappy formats are splittable, but GZip is not. Splittability is not relevant to HBase data.
- For MapReduce, you can compress either the intermediate data, the output, or both. Adjust the parameters you provide for the MapReduce job accordingly. The following examples compress both the intermediate data and the output. MR2 [6] is shown first, followed by MR1 [7].

✓ **MRv2 Coding**

hadoop jar hadoop-examples-.jar sort "-Dmapreduce.compress.map.output=true"

"Dmapreduce.map.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec"

"-Dmapreduce.output.compress=true"

"Dmapreduce.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec" –outKey

org.apache.hadoop.io.Text -outValue org.apache.hadoop.io.Text input output

✓ **MRv1 Coding**

hadoop jar hadoop-examples-.jar sort "-Dmapred.compress.map.output=true"

"-Dmapred.map.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec"

"-Dmapred.output.compress=true"

"-Dmapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec" -outKey

org.apache.hadoop.io.Text-outValue org.apache.hadoop.io.Text input output

4.1 YARN

YARN (yet another resource negotiator) is a key element of the hadoop data processing architecture that provides different data handling mechanism, including interactive SQL (Structured query language) and batch processing. In YARN, the Resource Manager will be the resources distributor while the Application Master is responsible for the communication with the Resource Manager and cooperate with the Node-manager to complete the tasks [8]. Yarn can be considered as operating system of hadoop ecosystem. It improves the performance of data processing in Hadoop by separating the resource management and scheduling capabilities of map reduce from its data processing components. YARN combines a central resource manager that reconciles the way applications use Hadoop system resources with node manager agents that monitor the processing operations of individual cluster nodes. Separating HDFS from MapReduce with YARN makes the Hadoop environment more suitable for operational applications that can't wait for batch jobs to finish. YARN. Given the limitations of MapReduce, the main purpose of YARN is to divide the tasks for the JobTracker. In YARN, the Resources are managed by the Resource Manager and the jobs are traced by the

Application Master. The Task Tracker has become the NodeManager. Hence, the global Resource Manager and the local NodeManager compose the data computing framework. In YARN, the Resource Manager will be the resources distributor while the Application Master is responsible for the communication with the Resource Manager and cooperate with the NodeManager to complete the tasks [9].

4.2 YARN architecture

Compared with the old MapReduce Architecture, it is easy to find out that YARN is more structured and simple. Then, the following section will introduce the YARN architecture.

There are following four core components of the YARN Architecture [10]:

- **ResourceManager**

  According to the different functions of the ResourceManager, is a designer has divided it into two lower level components: The Scheduler and the Application Manager. On the one hand, the Scheduler assigns the resource to the different running applications based on the cluster size, queues, and resource constraints. The Scheduler is only responsible for the resources allocation but is not responsible for the monitoring the application implementation and task failure. On the other hand, the Application Manager is in charge of receiving jobs and redistributing the containers for the failure objects.

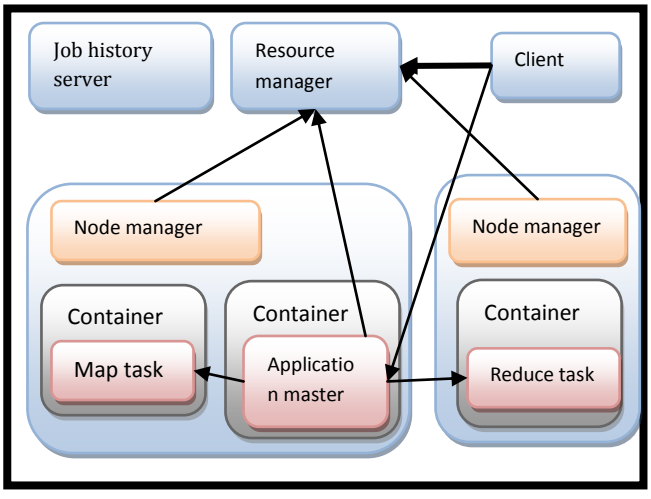

Figure 2:-Yarn Architecture

- **NodeManager**

  The NodeManager is the frame proxy for each node. It is responsible for launching the application container, monitoring the usage of the resource, and reporting all the information to the Scheduler.

- **Application Master**

  The Application Master is cooperating with the NodeManager to put tasks in the suitable containers to run the tasks and monitor the tasks. When the container has errors, the Application Master will apply for another resource from the Scheduler to continue the process.

- **Container**

  In YARN, the Container is the source unit which is the available node splitting the organization resources. Instead of the Map and Reduce source pools in MapReduce, the Application Master can apply for any numbers of the Container.

Due to the same property Containers, all the Containers can be exchanged in the task execution to improve efficiency.

4.3 Difference between MapReduce and YARN (Yet another resource negotiator):- Drawbacks of MapReduce become the benefits of YARN (yet another resource negotiator).

Here are the drawbacks of MapReduce:-

- The MAPREDUCE Model is unable to process the streaming data and in-memory interface operations. Some of the limitations are as follows:-
  - SCALABILITY
  - AVALIABILITY
  - RESOURCE UTILIZATION
  - NON-MAP REDUCE APPLICATION
- Map Reduce cannot be used with real time data processing applications, such as:-weather forecasting, fraud detection, etc. because it is batch oriented processing.
- It doesn't work with a job that runs in isolation and communication needs.
- Cannot be used when lot of data shuffling is done.
- It is not suitable for processing a large amount of short online transaction.

Now, here are the advantages of YARN:-

- YARN is a key elements of the Hadoop data processing architecture that provides different data handling mechanisms, including interactive SQL(Structured query language) and Batch –processing.
- It improves the performance of data processing in Hadoop by separating the resource management and scheduling capabilities of map reduce from its data processing components.
- Provides effective resource utilization.
- Provides a mechanism to YARN through which multiple applications, sharing common resource and can be run simultaneously.
- It allows Hadoop ecosystem to run those application which map reduce model is not follow.
- It doesn't have job tracker and task tracker nodes.
- It provides backward compatibility i.e. application developed on MapReduce can directly run on YARN.

Conclusions

Reasons to compress:

a) Data is mostly stored and not frequently processed. It is usual DWH scenario. In this case space saving can be much more significant then processing overhead
b) Compression factor is very high and thereof we save a lot of IO [11].

c) Decompression is very fast (like Snappy) and thereof we have a some gain with little price

d) Data already arrived compressed

Reasons not to compress:

a) Compressed data is not splittable. Have to be noted that many modern format are built with block level compression to enable splitting and other partial processing of the files.

b) Data is created in the cluster and compression takes significant time. Have to be noted that compression usually much more CPU intensive then decompression.
c) Data has little redundancy and compression gives little gain.

This paper says, YARN can serve as both a solid production framework and also as an invaluable playground for the research community and also data compression done on.

References

[1] ”A. Jenifer Jothi Mary1, Dr. L. Arockiam2”, “A Study on Basic Concepts of Big Data”.

[2] Vinod Kumar, Vavilapall, Arun C Murthy, Chris Douglas, Sharad Agarwali, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O’Malley, Benjamin Reed,Eric Baldeschwieler, “Apache Hadoop YARN: Yet Another Resource Negotiator”, SoCC’13, 1–3 Oct. 2013, Santa Clara, California, USA, ACM978-1-4503-2428-1.

[3] “Khalid Sayood”, “Introduction to Data Compression”.

[4] “ Rajinder Kaur , Mrs. Monica Goyal”, “A Survey on the different text data compression techniques” , International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), Volume 2, Issue 2, February 2013.

[5] “S. Shanmugasundaram , R. Lourdusamy” , “A Comparative Study of Text Compression Algorithms” , International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011.

[6] “A. Singh , Y. Bhatnagar” , “Enhancement of data compression using Incremental Encoding”, International Journal of Scientific & Engineering Research, Volume 3, Issue 5, May-2012.

[7] “Harpreet Kaur”, “A Review of Data Compression Techniques and Data Compression Symmetry”, IJCST Vol. 4, Iss ue 2, April - June 2013

[8] ”Ashish Sharma, Snehlata Vyas”, “Hadoop2 Yarn”, IPASJ International Journal of Computer Science (IIJCS), Volume 3, Issue 9, September 2015.

[9] ”Harshawardhan S. Bhosale, Prof. Devendra P. Gadekar”, “A Review Paper on Big Data and Hadoop”, International Journal of Scientific and Research Publications, Volume 4, Issue 10, October 2014.

[10] “Arinto Murdopo, Jim Dowling“, “Next Generation Hadoop: High Availability for YARN”.

[11] “H.Altarawneh , M. Altarawneh” , " Data Compression Techniques on Text Files: A Comparison Study ”, International Journal of Computer Applications, Volume 26– No.5, July 2011.