RECORD-AWARE PARTIAL COMPRESSION SCHEME IN SPARK

Dalavai Durga, M. Tech Scholar, Deportment of Computer Science &Engineering, JNTUACEA, Ananthapuramu, A.P, India.

ABSTRACT

In Real-time, digital data is increasing in all aspects like banking, healthcare, education and, social science, etc. A Large amount of data will be stored in the form of text. For effective data analysis representation, the compression process is used which reduces the data size, storage space and transmission cost of data.

In an existing system, the Content-aware Partial Compression scheme (CaPC) was developed which improves the performance and reduces the resources for textual big data analysis in Hadoop. But this scheme should not offer better compression ratio and performance. To get the maximum value of the compression for data analysis, the two-layered architecture of the Record-aware Partial Compression scheme (RaPC) is developed in Hadoop. But the main disadvantage with Hadoop is, it supports only batch processing and it can't support stream processing. The batch processing is efficient to process a large amount of data, but it depends on the data size and computational power of the system, so there will be a delay in output and overall performance will be slow.

The proposed work includes the replacement of Hadoop with Spark which supports stream processing. This stream processing will be worked as continuous input and output, which improves the speed of the data and the data will be processed in less amount of time. Spark uses in-memory processing to process the data. The memory is still considered to be an expensive hardware resource, so this memory consumption will be reduced by using CaPC or RaPC Layer-1 scheme. In CaPC or RaPC Layer-1 compression, the compressed data can be processed directly without decompression.

I. INTRODUCTION

The exponential growth of data, big data challenges storage and transport issues. The remaining big data challenges will be described in [12]. To overcome this problem various compression schemes introduced. In that, the Record-aware Partial Compression scheme (RaPC) is a two-layered architecture implemented in Hadoop.

The Hadoop framework is based on a simple map-reduce programming model and it focusing on enables a computing solution that is scalable, flexible, fault-tolerant and cost-effective. The main concern is to maintain speed in processing large datasets to reduce waiting time between queries and run the program.

Apache Software Foundation has introduced the Spark for speeding up the Hadoop computational computing software process. In-memory cluster computing is the main feature of Spark and it increases the processing speed of an application. In in-memory computation, the data is kept in random access memory (RAM) instead of some slow disk drives and is processed in parallel. Using this we can detect a pattern, analyze large data. This has become popular because it reduces the cost of memory. So, in-memory processing is economic for applications. The two main columns of in-memory computation are-

RAM storage

Parallel distributed processing.

In an existing system, RaPC is implemented in Hadoop. The problem identified with Hadoop is it supports only batch processing and it can't support stream processing; it means unbounded sequence of data arriving continuously. Hadoop map reduce is read and write from disk. as a result, it slows down the computation. To overcome this problem the RaPC scheme is implemented in Spark is explored in this paper.

The proposed system solves the storage and transport issues. It supports stream processing and in-memory processing. Apart from these problems to compare with Hadoop, the spark is easy to use and easy to program because it has rich APIs in Java, Scala, Python and R. In this paper, the detailed description of the compression scheme and experimental results will be detailed.

II.RELATED WORKS

This section provides a review of the literature of the introduction of spark is explored in [1]. The Hadoop limitations are the focus in [2]. There are some special cases about where the compressed data can be searched in [3]. The universal text pre-processing for data compression in [4]. Studies from [5] demonstrated that applying fully reversible transformations (e.g., BWT) on text for better compression ratios. The detail about various conventional compression schemes is mentioned in [6]. The basic schemes of RaPC, Content-aware Partial Compression (CaPC) and RaC is studied in [7] [8]. The implementation of CaPC in Hadoop is studied in [9]. The comparison of Hadoop and Spark is mentioned in [10]. The various compression techniques are studied in [11]. In this paper, the problem of data transmission cost, storage space, and data size.

III. PRELIMINARIES

This section provides important information that leads to understanding the proposed work in this paper.

A. Compression

This section provides important information that leads to an understanding of the proposed work in this paper. The main concern of big data is, to maintain data storage and processing speed of the digital data from various sources. Many sources contain irrelevant and redundant. If this unnecessary data can be removed then the data size will be reduced and easy to manage. Various techniques to achieve this task are called compression techniques. These techniques can be utilized to text, image, audio, video, etc. data. Compressed data can improve processing speed and time. The figure illustrates the real-time transfer of the given data.

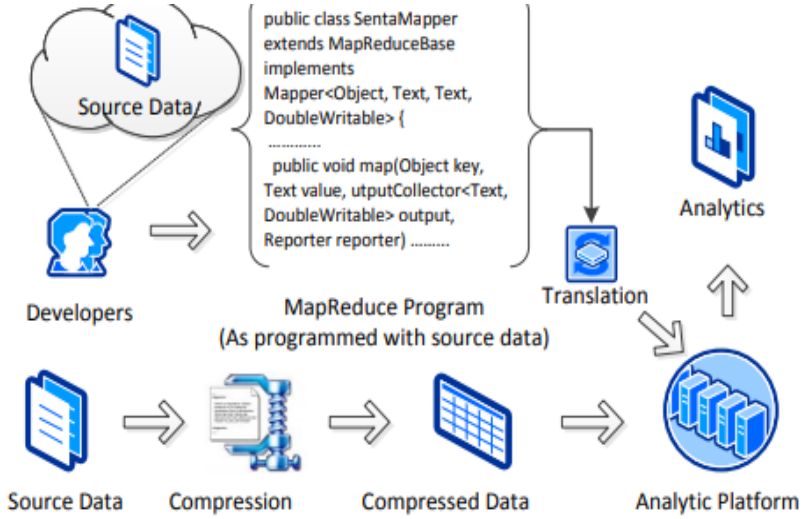


Figure 1: An illustration of the conceptual architecture.

B. Encoding

Arithmetic coding and Huffman coding schemes produce optimal codewords for encoding and these are widely used for the encoding scheme. Arithmetic coding creates dependent codewords. The Huffman encoding produces independent codewords and it also creates prefix-free codewords for symbols. These codewords have measured in bits. So, comparing with arithmetic coding Huffman coding is better for the encoding scheme. The main task of Huffman coding is, it generates shorter codes for more frequently occurring symbols in a given message. This coding scheme uses the Huffman tree for distributing symbols in the input data.

C. Context-aware Partial Compression

The Context-aware Partial Compression (CaPC) scheme is the basic source for the implementation of RaPC scheme. It inherits the main features of CaPC with an improved design. The CaPC design is implemented in RaPC layer-1. The main feature of this scheme is it produces the code for valid strings that are compatible with ASCII codes and UTF-8 format. The detail description of this scheme is described in [9].

D. Record-aware Compression

The Record-aware Compression (RaC) scheme is another source for the imple4mentation of RaPC scheme. It inherits the main features of RaC, which are implemented in RaPC layer-2. The main feature of this scheme is block-based compression, it means it takes variable-length of input data and gives a fixed size of compression data as output. The information about this scheme is explained in [8].

E. Record-aware Partial Compression

The Record-aware Partial Compression (RaPC) scheme is a two-layer scheme. The algorithm designed for improving MapReduce performance and reducing Hadoop cluster resource usage. The RaPC Layer-1 is the outer layer of this compression scheme. In this, maximally reduces the data size by using an encoding. The inner layer that means RaPC Layer-2 used to compress Layer-1 compressed data by using a word-based context-free compression scheme. The detail information will be given in [8].

IV.PROBLEM FORMULATION

Here, the problem is mainly concern about data processing and storage. Hadoop supports batch processing, which means that the data will be processed that has been already stored over a period of time. One more problem of this batch processing is it depends on the data size and computational power of the system. So, finally it reduces the data processing speed and finally, there will delay in output.

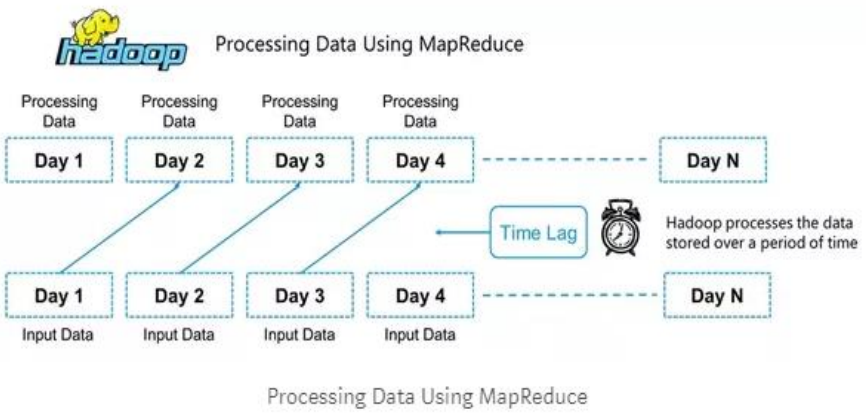


Figure 2. Batch processing in Hadoop.

As shown in the figure.2, the batch processing is working in Hadoop. This is the main disadvantage of batch processing. This type of data processing does not work efficiently for large data sets. To overcome this problem, stream processing is replaced. The RaPC scheme is already implemented in Hadoop but for better performance, this scheme is implemented in Spark, which supports stream processing and in-memory processing.

V. PROPOSED SYSTEM

The proposed system is, the RaPC scheme is implemented in Spark. Only Spark will be solving the problems in the existing system. The Spark will be supporting stream processing and in-memory processing. Here, the stream processing means the data will be processed continuously as shown in figure.3 and there will be no delay in output**.**

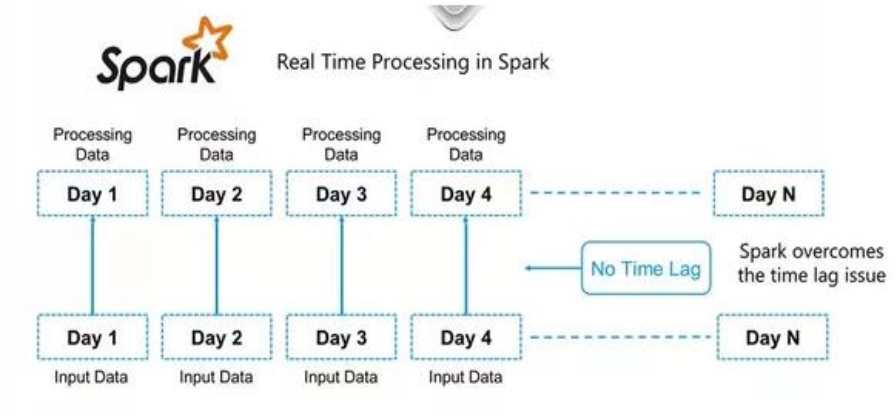


Figure 3. Stream processing in Spark.

Another feature of Spark is in-memory processing used RAM instead of disk drives and processed in parallel. Now it has become more popular for cost reduction in memory. As shown in figure 4, Spark is better than Hadoop for iterations.

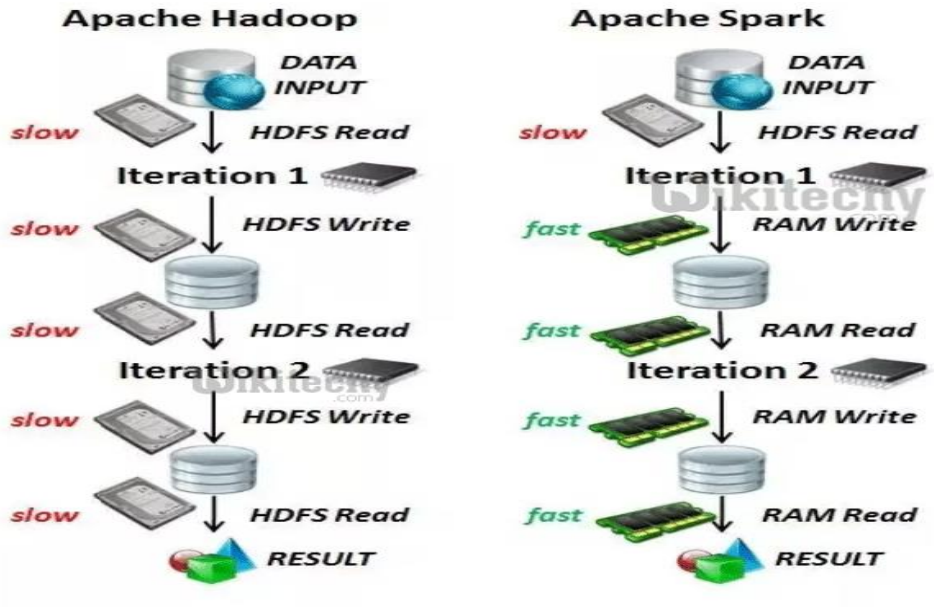


Figure 4. Iterations in Hadoop and Spark.

A. Huffman Encoding

The main task of Huffman encoding is how the variable length codes can be packed together**.** In this each character represented by 8-bits and directly separate one character from the next by breaking off 8-bit chunks. Finally, a serial data stream of ones and zeros received.

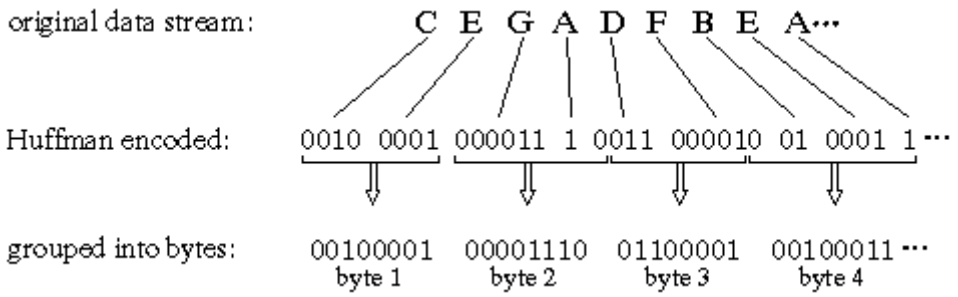


Figure 5. Huffman encoding

As shown in figure 5, an example of Huffman encoding is simplified. This allows the binary data stream to be separated into codes and there is no need for delimiters or other markers between the codes. The detailed description will be given in [13][14].

```
Huffman (C)
  n = |C|
  Q = C
   for i = 1 to n-1
      allocate new node z
      z.left = x = Extract-min (Q)
      z.right = y = Extract-min (Q)
       insert (Q, z)
   return Extract-Min (Q)  // returns the root

Complexity = O(n lg n)
```

Figure 6. Huffman tree algorithm.

RaPC layer1 technique assigns integer values to all unique words and then encodes it for compression. The Huffman coding is used for this encoding and this will be implemented in spark.

VI. EXPERIMENTAL RESULTS

In this, as part of streams used dataset which consists of some text documents and this text documents will be uploaded to spark and spark will compress data using 'RaPC Layer-1 Compression' technique.

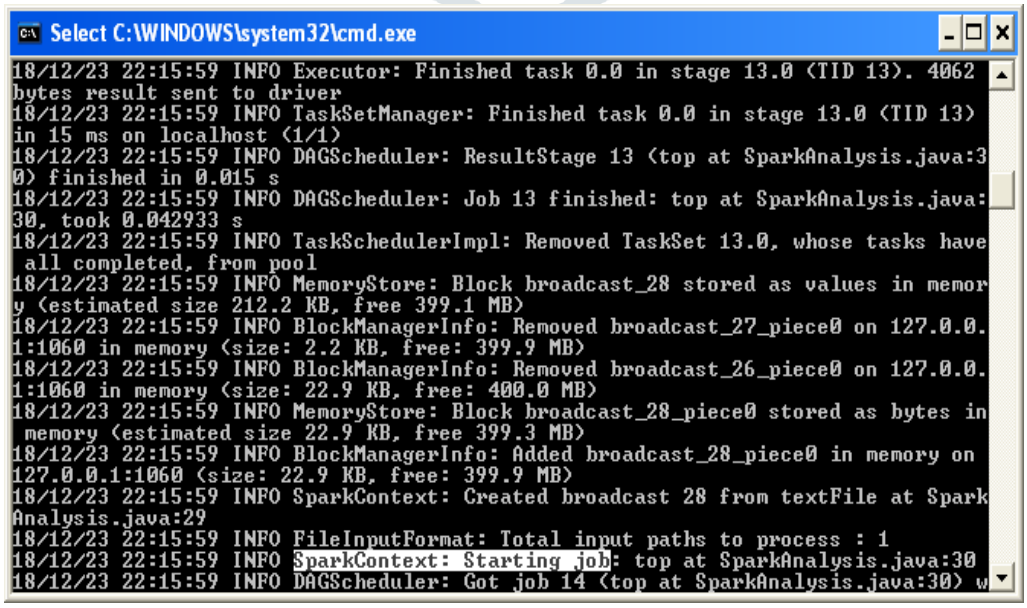


Figure 7. Spark processing details

As shown in figure 7, the Huffman encoding will be done for all uploaded datasets in spark. After it shows the encoding values for all text documents as shown in figure 8.

Figure 8. Encoding view

As shown in figure 9, it is clear that the difference between the storage space for normal and encoded text.

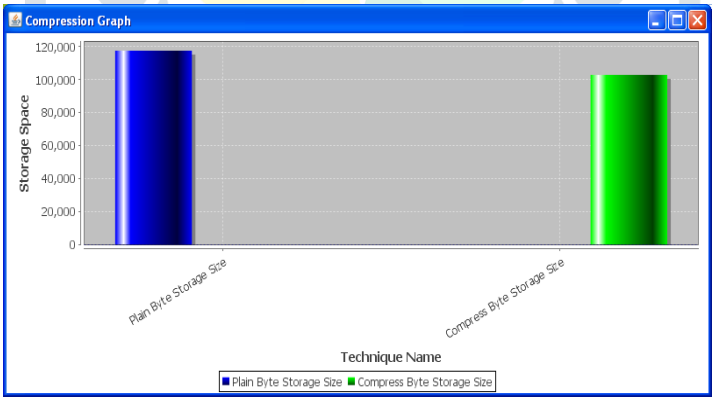
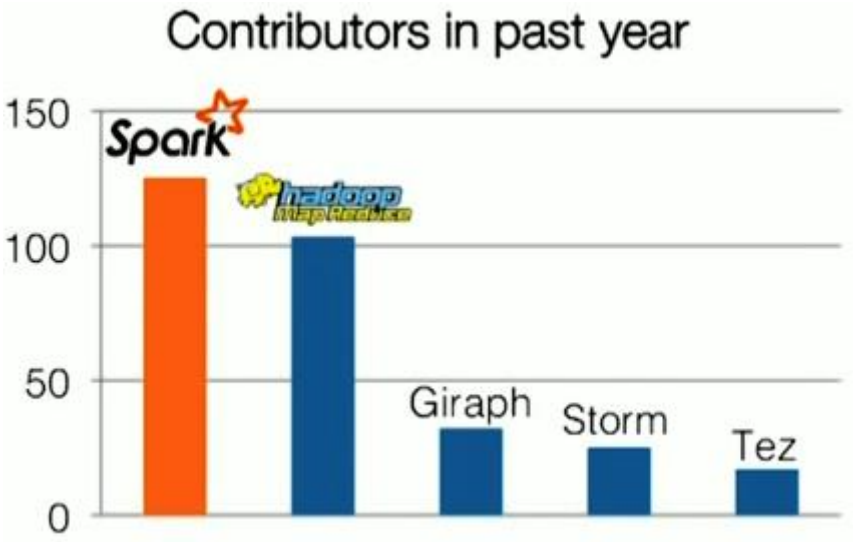


Figure 9. Compression graph

In real-world, Spark places top position in the business applications. The following figure shows the contributors in past year of the Hadoop and spark.



Hence, the RaPC scheme gives better performance and efficient result. Finally, the proposed system achieves the efficient results for storage and data processing.

VII. CONCLUSION AND FUTURE WORK

Leveraging novel compression schemes for improving performance and reducing resource requirements for big data analysis is our overall contribution to the field of big data research. In this thesis, several compression schemes having different properties suitable for a variety of big data analysis scenarios.

Using modern compressors to achieve a high level of compression without considering direct processability. That provides a more generic solution. Whereas modern processor and memory speed have improved greatly, there has been little progress on improving I/O performance (especially the I/O system between commodity hard disks and memory which still presents a bottleneck. Therefore, the adoption of the procedures of reading compressed data from hard disks to memory, then decompressing it in memory has the potential advantage of better performance over reading original data straight into memory. This has been proved by RaPC scheme.

In this process, a dynamic coding scheme in which the codeword length increases along with the compression according to the variety of information to be encoded. RaPC Layer-1 compression follows the same principles in CaPC and AHC. The output of RaPC Layer-1 is directly processable data and achieves a data size reduction of approximately 50%.

Moreover, as compression algorithms often have to achieve a balance between compression ratio and decompression speed, we investigated whether a scheme with higher compression ratio and lower decompression speed can achieve better overall performance than one with a lower compression ratio and higher decompression speed. Apart from MapReduce, Spark is an increasingly popular framework which focuses on in-memory processing and supporting Directed Acyclic Graph (DAG)work-flows. As memory is still considered to be an expensive hardware resource (comparing to commodity hard disks), using RaPC Layer-1 compression can greatly reduce memory consumption, since the RaPC Layer-1 compressed data can be processed directly without decompression.

Future work:

The general idea of RaPC schemes can be adopted by any big data analytic platform. In this work, the implementation supporting libraries primarily for Apache Spark. Apart from spark, today there are a number of open source streaming frameworks available like Flink, Samza, Kafka etc. Interestingly, almost all of them are quite new and have been developed in last few years only. In future the work intends to have the implementation of RaPC scheme in these streaming frameworks.

REFERENCES

1 https://data-flair.training/blogs/spark-tutorial/

2. https://data-flair.training/blogs/hadoop-tutorial/

3. R. Venturini, Compressed Data Structures for Strings on Searching and Extracting Strings from Compressed Textual Data. Atlantis Press, 2014.

4. J. Abel and W. Teahan, “Universal text preprocessing for data compression,” Computers, IEEE Transactions on, vol. 54, no. 5, pp. 497–507, May 2005.

5 M. Burrows and D. Wheeler, “A Block-sorting Lossless Data Compression Algorithms,” Digital Equipment Corporation, Palo Alto, CA, Tech. Rep. 124, May 1994.

6. https://cora.ucc.ie/bitstream/handle/10468/2697/DongD_PhD2016.pdf;sequence=2

7. 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, Content-aware Partial Compression for Big Textual Data Analysis Acceleration Dapeng Dong and John Herbert.

8. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science, Record-aware Two-level Compression for Big Textual Data Analysis Acceleration Dapeng Dong and John Herbert.

9. IEEE TRANSACTIONS ON BIG DATA: Content-aware Partial Compression for Textual Big Data Analysis in Hadoop Dapeng Dong.

10. 2016 Symposium on Colossal Data Analysis and Networking (CDAN) Big Data Management Processing with Hadoop MapReduce and Spark Technology: A Comparison

11. Data Compression Techniques for Big Data 1 Ms. Poonam Bonde, 2 Mr. Sachin Barahate.

12. 2014 International Journal of Computer Science and Information Technologies,” Big Data Storage and Challenges” M.H. Padgavankar1, Dr.S.R. Gupta2

13. 2010 International Conference on Methods and Models in Computer Science (ICM2CS-2010),” Implementation of data compression using Huffman coding”, K. Ashok Babu.

14. Xrysovalantis Kavousianos, Emmanouil Kalligeros, Dimitris Nikolos, "Multilevel-Huffman Test-Data Compression for IP Cores with Multiple Scan Chains", IEEE Transactions ON Very Large-Scale Integration (VLSI) Systems, vol. 16, no. 7, July 2008.