International Research Journal of Engineering and jTechnology (IRJET) e-ISSN: 2395-056
Volume: 08 Issue: 03 | Mar 2021 www.irjet.net p-ISSN: 2395-0072

# Preview a Novel Approach to Organize Big Data in Columnar Storage Format

**Jadhawar B. A.[1], Dr. Neeraj Sharma[2]**

[1]*Research Scholar, Department of Computer Science & Engineering, Sri Satya Sai University of Technology and Medical Sciences, Sehore , MP, India*
[2]*Associate Professor, Department of Computer Science & Engineering, Sri Satya Sai University of Technology and Medical Sciences, Sehore , MP, India*

---

**Abstract -** *The reason we decided to do research in Hadoop, storage of different documents is provided by HDFS (Hadoop Distributed File System). With the rise of big data, people begin to focus on storage of it. There are different file formats that have been evolved which tries to manage the large data and to increase the read and write performance. These formats are generally columnar in nature instead of row, which helps to analyze the data and fetch only the required columns. Some optimization methods have been introduced on this format which further increases the performance. In this research thesis we will try to implement one such format which will not only increase the performance but also manages data efficiently. We will also do the benchmark of this format and publish the numbers comparing to the already existing formats.*

***Key Words***: High Speed Processing, Reducing file sizes, Columnar Format, *HDFS.*

## 1. INTRODUCTION

The reason we decided to do research in Hadoop, storage of different documents is provided by HDFS (Hadoop Distributed File System). With the rise of big data, people begin to focus on storage of it. There are different file formats that have been evolved which tries to manage the large data and to increase the read and write performance. These formats are generally columnar in nature instead of row, which helps to analyze the data and fetch only the required columns. Some optimization methods have been introduced on this format which further increases the performance. In this research thesis we will try to implement one such format which will not only increase the performance but also manages data efficiently. We will also do the benchmark of this format and publish the numbers comparing to the already existing formats.

Predicate pushdown uses those indexes to determine which stripes in a file need to be read for a particular query and the row indexes can narrow the search to a particular set of 10,000 rows. ORC supports the complete set of types in Hive, including the complex types: structs, lists, maps, and unions.

Many large Hadoop users have adopted ORC. For instance, Facebook uses ORC to save tens of petabytes in their data warehouse and demonstrated that ORC is significantly faster than RC File or Parquet. Yahoo uses ORC to store their production data and has released some of their benchmark results

ORC files are divided in to *stripes* that are roughly 64MB by default. The stripes in a file are independent of each other and form the natural unit of distributed work. Within each stripe, the columns are separated from each other so the reader can read just the columns that are required [7].

Parquet file format were created to make the advantages of compressed, efficient columnar data representation available to any project in the Hadoop ecosystem. Parquet is built from the ground up with complex nested data structures in mind, and uses the record shredding and assembly algorithm described in the Dremel paper. We believe this approach is superior to simple flattening of nested name spaces.

Parquet is built to support very efficient compression and encoding schemes. Multiple projects have demonstrated the performance impact of applying the right compression and encoding scheme to the data. Parquet allows compression schemes to be specified on a per-column level, and is future-proofed to allow adding more encodings as they are invented and implemented.

Parquet is built to be used by anyone. The Hadoop ecosystem is rich with data processing frameworks, and we are not interested in playing favorites. We believe that an efficient, well-implemented columnar storage substrate should be useful to all frameworks without the cost of extensive and difficult to set up dependencies [8].

### 1.1 A brief review of the work already done in the field

In the columnar storage structure, more sorted columns mean better query performance. However, as columns are sorted respectively, the records are no longer horizontal alignment, which causes trouble to the record

reorganization. There is a simple solution to this problem, and that is to copy the ID number to each key column. But owing to the huge cost of the storage space and the searching time, there are few systems that use this method [1]. A new technique to transform procedural code so that it operates on hierarchically nested, columnar data natively, without row materialization. It can be viewed as a compiler pass on the typed abstract syntax tree, rewriting references to objects as columnar array lookups. Also present performance comparisons between transformed code and conventional object-oriented code in a High Energy Physics context [2].

With the development and popularization of e-commerce and social network [3] [4], 21the structured data meeting the relation model have seen continuous increase and shown to reach PB level and even larger scale. For example, the data warehouse constructed by Facebook has stored 300PB structured data in 2014 and expanding by 600TB per day [5]. The search engine system developed by Baidu could handle 100PB of data daily in 2013[6,7]. Supporting interactive query on such large volumes of data necessitates the development of large-scale storage management ability and rapid analysis and calculation capability. These requirements pose new challenges for big data storage and management technology [18, 19].

**ORC files**

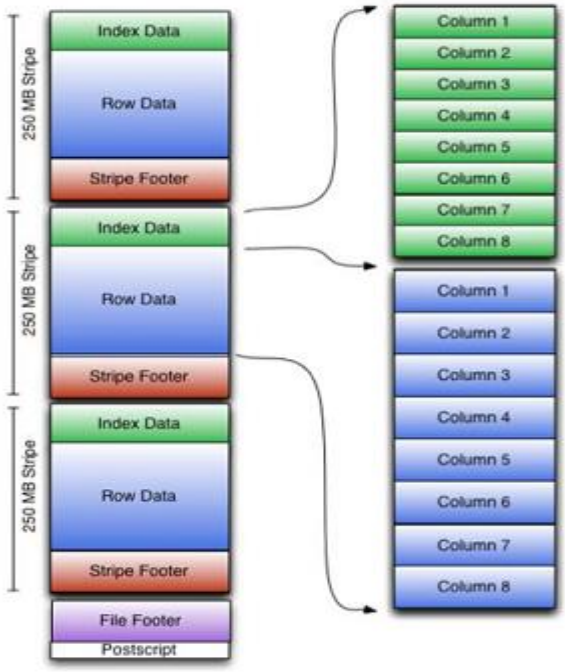


**Fig -1**: **ORC File Layout**

The above Fig.1. Shows ORC file format. The running scenario for this four-part series is a startup, which processes data from different sources, SQL and NoSQL stores, and logs. The challenge with big data, as the domain matures, and for evolving deployments in companies, is to not only to process the data but to also do it efficiently, reducing cost and time required [7].

In the scenario, and for many companies, tables containing billions of rows and numerous columns are unexceptional. Querying and reporting on this data swiftly requires a sophisticated storage format. It ideally stores data compact and enables skipping over irrelevant parts without the need for large, complex, or manually maintained indices. The ORC file format addresses all of these issues.

The Stinger initiative heads the ORC file format development to replace the RCFile. Former should become part of the stable Hadoop releases this year. ORC stores collections of rows in one file and within the collection the row data is stored in a columnar format. This allows parallel processing of row collections across a cluster. Each file with the columnar layout is optimized for compression and skipping of data/columns to reduce read and decompression load.

ORC goes beyond RCFile and uses specific encoders for different column data types to improve compression further, e.g. variable length compression on integers. ORC introduces a lightweight indexing that enables skipping of complete blocks of rows that do not match a query. It comes with basic statistics — min, max, sum, and count — on columns. Lastly, a larger block size of 256 MB by default optimizes for large sequential reads on HDFS for more throughput and fewer files to reduce the load on the name node.

## 1.2 ORC is a columnar file format

The below Fig.2. ORC is a columnar file format. Visualize the structure of an ORC file as an area that is divided into Header, body and footer. The Header contains the text ORC in case some tools require determining the type of file while processing.

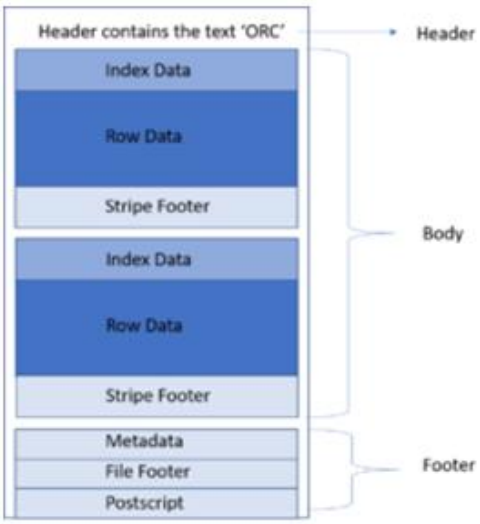


**Fig.2. ORC File Format**

The body contains the actual data as well as the indexes. Actual data is stored in the ORC file in the form of rows of data that are called Stripes. Default stripe size is 250 MB.

Stripes are further divided into three more sections viz the index section that contains a set of indexes for the stored data, the actual data and a stripe footer section. One interesting thing to note here is that both index and data section are stored as columns so that only the columns where the required data is present, is read. Index data consists of min and max values for each column as well as the row positions within each column. ORC indexes help to locate the stripes based on the data required as well as row groups. The Stripe footer contains the encoding of each column and the directory of the streams as well as their location [11, 12].

The footer section consists of three parts viz. file metadata, file footer and postscript. The file Metadata section contains the various statistical information related to the columns and this information is present at a stripe level. These statistics enable input split elimination based on predicate push down which are evaluated for each stripe.

The file footer contains information regarding the list of stripes in the file, number of rows per stripe, and the data type for each column. It also contains aggregates counts at column-level like min, max, and sum.

The Postscript section contains the file information like the length of the file's Footer and Metadata sections, the version of the file, and the compression parameters like general compression used (e.g. none, zlib, or snappy) and the size of the compressed folder [13, 14].

## 2. PROPOSED METHODOLOGY DURING THE TENURE OF THE RESEARCH WORK.

This research work will be carried out in three phases. Following are those phases

**1) Understanding the parquet file format and modifying it to support the research to be done:**

In this phase existing optimized parquet format will be analyzed and extended to expose the API that will help to implement the proposed research. During this phase code of parquet file format will be analyzed to understand the read and write path and how metadata is maintained which will help us to understand where we need to inject the code for the proposed work

**2) Analyzing the best indexes and implementing them:**

In this phase we will analyze different types of indexes like Bitmap index, dictionary based index etc. And will also do some performance benchmarking to understand which index suits which data type. Once this is done suitable indexes will be implemented for different data types and will be injected in parquet file format.

**3) Implementing Sources for apache spark:**

In order to understand the performance we need to implement a custom parquet source with index for apache spark which we will be used to compare the results. The results will be compared with different types of big data use cases and will be published as a part of this thesis

## 3. EXPECTED OUTCOME OF THE PROPOSED WORK

These three are the expected outcomes are

1. File Format.
2. Read Path for file format.
3. Write path for file format.

### 1. File Format

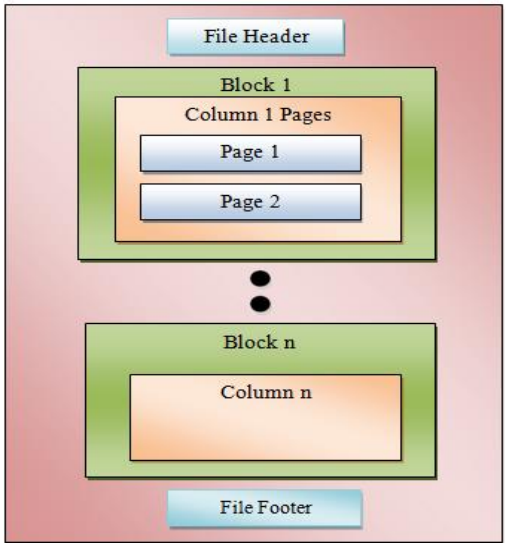


**Fig. 1 File Format**

The below fig.1. Shows File Format for the proposed research work which will not only increase the performance but also manages data efficiently. We will also do the benchmark of this format and publish the numbers comparing to the already existing formats.

### 2. Read path for file format

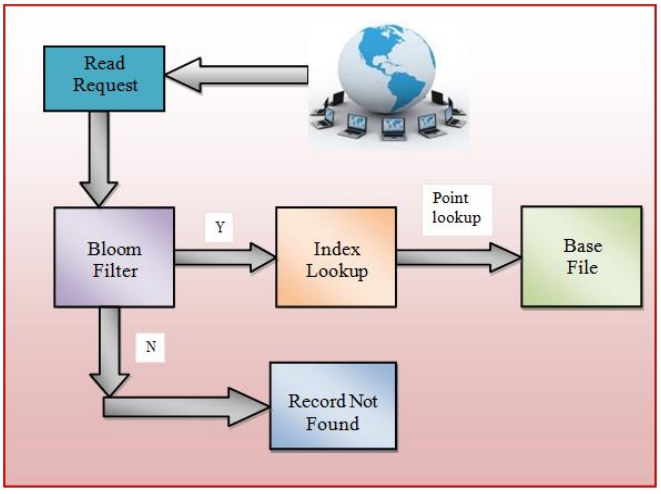


**Fig. 2. Read path for file format**

The above fig.2. Shows Read path for file format. When a read request for a record is submitted to the proposed file format it is first validated against bloom filter (an efficient data structure used to test whether an element is a member of set). If bloom filter returns the availability we will go our search for next step but if it fails we will stop for the search. In the next step we will search for the element in the index which we have created while insertion of the record and find the exact location of element in the actual columnar file.

### 3. Write Path

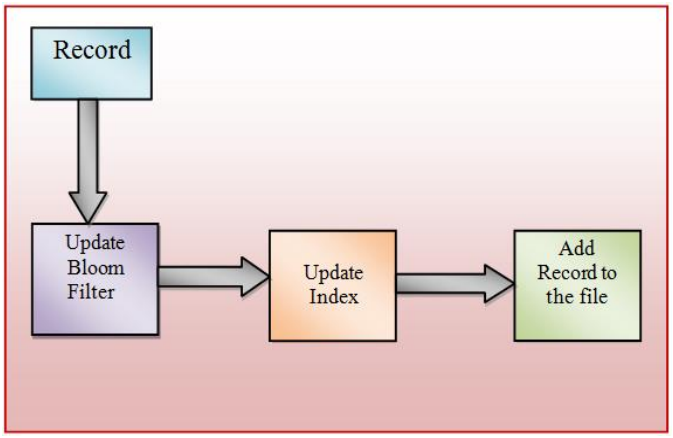


**Fig. 3 Write Path**

The above fig.3. Shows write path for file format. This part explains how the records are written in the proposed file format. When a record is requested to be written in the proposed file format the bloom filter will be updated for that record. Once the bloom filteris updated the index for the proposed file format is updated and then the record is written tothe base file.

### 4. CONCLUSION

We expect to design and create novel approaches for storing data in Big Data in Hadoop. We will also do the benchmark of this format and publish the numbers comparing to the already existing formats.

### ACKNOWLEDGEMENT

I like to acknowledgement my gratitude to Dr. Neeraj Sharma for valuable suggestions in carrying my research work. I also thankful to CSE Department of our University.

### REFERENCES

[1] Tao Xu and Dongsheng Wang —KCGS-Store: A Columnar Storage Based On Group Storing of Key Colum ns", in Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing, DOI 10.1109/CLOUD.2016.39.
[2] Jim Pivarski, Peter Elmer, and Brian Bockelman, Zhe Zhang, —Fast Access to Columnar, Hierarchically Nested Data via Code Transformation,‖ in Proceedings of the 2017 IEEE International Conference on Big Data (BIGDATA), 978-1-5386-2715-0/17/$31.00 ©2017 IEEE.

[3] C. Budak, D. Agrawal, and A. El Abbadi, —Structural trend analysis for online social networks," in Proceedings of the VLDB Endowment, vol. 4, no. 10, pp. 646-656, 2011.
[4] L. Pu, J. Xu, B. Yu and J. Zhang, —Smart cafe: A mobile local computing system based on indoor virtual cloud,|| China Communications, vol. 11, no. 4, pp. 38-49, 2014.
[5] Scaling the Facebook Data Warehouse to 300 PB.
[6] K. Yu, —Large-scale deep learning at Baidu,|| in Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, pp. 2211-2212, 2013.
[7] A. Jacobs, —The Pathologies of Big Data,|| Communications of the ACM, vol. 52, no. 8, pp. 36–44, 2009.
[8] S. Ghemawat, H. Gobioff, and S. T. Leung, —The Google file system,|| in ACM SIGOPS Operating Systems Review, vol. 37, no. 5, pp. 29-43, 2003.
[9] Evans, Chris (Oct 2013). "Big data storage: Hadoop storage basics". **Computer weekly.com.** Computer Weekly. Retrieved 21 June 2016. HDFS is not a file system in the traditional sense and isn't usually directly mounted for a user to view
[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in Proceedings of IEEE Conference on Mass Storage Systems and Technologies, pp. 1-10, 2010.
[11] A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata, "Column-Oriented Storage Techniques for MapReduce," in Proceedings of the VLDB Endowment, vol. 4, no. 7, pp. 419-429, 2011.
[12] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, —RCFile: A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems,|| in Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, pp. 1199– 1208, 2011.
[13]https://issues.apache.org/jira/secure/attachment/12564124/OrcFileInt ro.pptx.
[14] https://github.com/cutting/trevni.
[15] https://github.com/Parquet.
[16] http://parquet.apache.org/documentation/latest
[17] S. Chen, —Cheetah: a high performance, custom data warehouse on top of MapReduce,|| in Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 1459-1468, 2010.
[18] https://parquet.apache.org/documentation/latest/
[19] https://orc.apache.org/

## BIOGRAPHIES

Dr. Neeraj Sharma,Associate Professor, Department Of Computer Science & Engineering, Sri Satya Sai University of Technology and Medical Sciences, Sehore , MP, India.20 year's of teaching experience UG & PG students,08 research papers published in international & national journals, 11 conferences/Webinars & 3 AICTE sponsored workshops.

Bhagyashala Jadhawar, Received Master Degree From Shivaji University of Kolhapur in Computer Science and Engineering, currently she is Research Scholar at Sri Satya Sai University of Technology and Medical Sciences, Sehore , MP, India