Columnar Storage Formats

Avrilia Floratou
Microsoft, Sunnyvale, CA, USA

Definitions

Row Storage: A data layout that contiguously stores the values belonging to the columns that make up the entire row.

Columnar Storage: A data layout that contiguously stores values belonging to the same column for multiple rows.

Overview

Fast analytics over Hadoop data has gained significant traction over the last few years, as multiple enterprises are using Hadoop to store data coming from various sources including operational systems, sensors and mobile devices, and web applications. Various Big Data frameworks have been developed to support fast analytics on top of this data and to provide insights in near real time.

A crucial aspect in delivering high performance in such large-scale environments is the underlying data layout. Most Big Data frameworks are designed to operate on top of data stored in various formats, and they are extensible enough to incorporate new data formats. Over the years, a plethora of open-source data formats have been designed to support the needs of various applications. These formats can be *row* or *column* oriented and can support various forms of serialization and compression. The *columnar* data formats are a popular choice for fast analytics workloads. As opposed to row-oriented storage, columnar storage can significantly reduce the amount of data fetched from disk by allowing access to only the columns that are relevant for the particular query or workload. Moreover, columnar storage combined with efficient encoding and compression techniques can drastically reduce the storage requirements without sacrificing query performance.

Column-oriented storage has been successfully incorporated in both disk-based and memory-based relational databases that target OLAP workloads (Vertica 2017). In the context of Big Data frameworks, the first works on columnar storage for data stored in HDFS (Apache Hadoop HDFS 2017) have appeared around 2011 (He et al. 2011; Floratou et al. 2011). Over the years, multiple proposals have been made to satisfy the needs of various applications and to address the increasing data volume and complexity. These discussions resulted in the creation of two popular columnar formats, namely, the Parquet (Apache Parquet 2017) and ORC (Apache ORC 2017) file formats. These formats are both open-source and are currently supported by multiple proprietary and open-source Big Data frameworks. Apart from columnar organization, the formats provide efficient encoding and compression techniques

S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_248-1

and incorporate various statistics that enable predicate pushdown which can further improve the performance of analytics workloads.

In this article, we first present the major works in disk-based columnar storage in the context of Big Data systems and Hadoop data. We then provide a detailed description of the Parquet and ORC file formats which are the most widely adopted columnar formats in current Big Data frameworks. We conclude the article by highlighting the similarities and differences of these two formats.

Related Work

The first works on columnar storage in the context of Hadoop, namely, the RCFile (Row Columnar File) (He et al. 2011) and the CIF (Column Input File format) (Floratou et al. 2011) layouts, were mostly targeting the MapReduce framework (Dean and Ghemawat 2008). These two columnar formats have adopted significantly different designs. The CIF file format stored each column in the data as a separate file, whereas the RCFile adopted a PAX-like (Ailamaki et al. 2001) data layout where the columns are stored next to each other in the same file.

These design choices led to different tradeoffs. The CIF file format was able to provide better performance as it allowed accessing only the files that contain the desired columns but required extra logic in order to colocate the files that contain adjacent columns. Without providing colocation in HDFS, reconstructing a record from multiple files would result in high network I/O. The RCFile, on the other hand, did not require any such logic to be implemented as all the columns were stored in the same HDFS block. However, the RCFile layout made it difficult to enable efficient skipping of columns due to file system prefetching. As a result, queries that were accessing a small number of columns would pay a performance overhead (Floratou et al. 2011).

The successor of the RCFile format, namely, the ORC (Optimized Row Columnar) file format, was developed to overcome the limitations of the RCFile format. It was originally designed to speed up Hadoop and Hive, but it is currently incorporated in many other frameworks as well. The ORC file format still uses a PAX-like (Ailamaki et al. 2001) layout but can more efficiently skip columns by organizing the data in larger blocks called row groups. A detailed description of the ORC file format is provided in the following section.

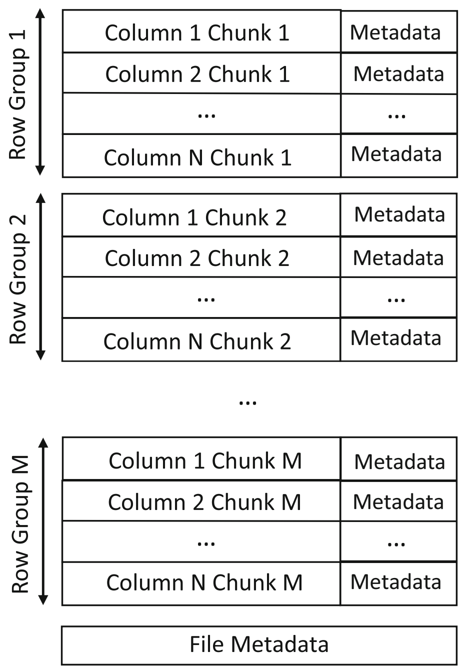
The Parquet (Apache Parquet 2017) file format is another popular, open-source columnar format. The Parquet file format was designed to efficiently support queries over deeply nested data. The format is based on the Dremel distributed system (Melnik et al. 2010) that was developed at Google for interactively querying large datasets. Similar to the ORC format, the parquet format has also adopted a PAX-like layout (Ailamaki et al. 2001). The following section presents Parquet's layout in detail.

Detailed comparisons of various columnar formats can be found in Huai et al. (2013) and in Floratou et al. (2014). The work by Floratou et al. (2014) compares the ORC and Parquet file formats in the context of Hive (Apache Hive 2017) and Impala (Kornacker et al. 2015) for SQL workloads.

Other related open-source technologies are Apache Arrow (2017) and Apache Kudu (2017). Apache Arrow is a columnar in-memory format that can be used on top of various disk-based storage systems and file formats to provide fast, in-memory analytical processing. Note that as opposed to the file formats discussed above, Arrow is an in-memory representation and not a disk-based file format. Apache Kudu is an open-source storage engine that complements HDFS (Apache Hadoop HDFS 2017) and HBase (Apache Hbase 2017). It provides efficient columnar scans as well as inserts and updates. Kudu provides mutable storage, whereas the file formats described above (e.g., Parquet) are immutable, and thus any data modifications require rewriting the dataset in HDFS.

Columnar File Formats for Big Data Systems

In this section, we describe in more detail two popular, open-source columnar formats



Columnar Storage Formats, Fig. 1 The parquet file format

that have been incorporated in various Big Data frameworks, namely, the **ORC** and the **Parquet** file formats. Both ORC and Parquet are Apache projects (Apache ORC 2017; Apache Parquet 2017) and are actively used by multiple organizations when performing analytics over Hadoop data.

The Parquet File Format

The Parquet file format (Apache Parquet 2017) is a self-described columnar data format specifically designed for the Hadoop ecosystem. It is supported by many Big Data frameworks such as Apache Spark (Zaharia et al. 2012) and Apache Impala (Kornacker et al. 2015), among others. The format inherently supports complex nested data types as well as efficient compression and encoding schemes. In the next section, we describe the format layout in more detail.

File Organization

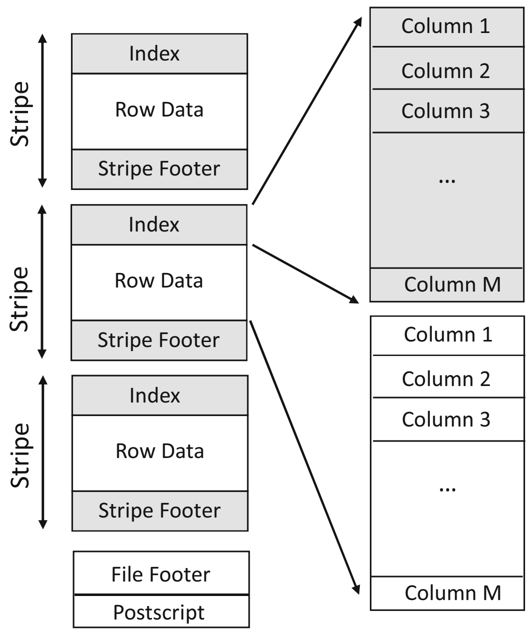
The Parquet format supports both primitive (e.g., integer, Boolean, double, etc.) and complex data types (maps, lists, etc.). The structure of a Parquet file is shown in Fig. 1. The file consists of a set of `row groups` which essentially represent horizontal partitions of the data. A `row group` consists of a set of `column chunks`. Each `column chunk` contains data from a particular data column and it is guaranteed to be contiguous in the file. The `column chunk` consists of one or more `pages` which are the unit of compression and encoding. Each `page` contains a header that describes the compression and encoding method used for the data in the page.

As shown in Fig. 1, the file consists of N columns spread across M `row groups`. The `row groups` are typically large (e.g., 1 GB) to allow for large sequential I/Os. Since an entire `row group` might need to be accessed, the HDFS block size should be large enough to fit the `row group`. A typical configuration is to have 1 GB HDFS blocks that contain one `row group` of 1 GB.

Metadata is stored at all the levels in the hierarchy, i.e., file, `column chunk`, and `page`. The bottom of the file contains the file metadata which include information about the data schema as well as information about the `column chunks` in the file. The metadata contains statistics about the data such as the minimum and maximum values in a column chunk or page. Using these statistics, the Parquet file format supports predicate pushdown which allows skipping of pages and reduces the amount of data that needs to be decompressed and parsed. The Parquet readers first fetch the metadata to filter out the column chunks that must be accessed for a particular query and then read each column chunk sequentially.

Compression

As described in the previous section, the unit of compression in the Parquet file format is the `page`. The format supports common compression codecs such as GZIP and Snappy (Snappy Compression 2017). Moreover, various types of encoding for both simple and nested data types are also supported. The interested reader can find more information about the various encoding techniques in Parquet Encodings (2017).



Columnar Storage Formats, Fig. 2 The ORC file format

The ORC File Format

The Optimized Row Columnar (ORC) format (Apache ORC 2017) is a column-oriented storage layout that was created as part of an initiative to speed up Apache Hive (2017) queries and reduce the storage requirements of data stored in Apache Hadoop (2017). Currently the ORC file format is supported by many Big Data solutions including Apache Hive (2017), Apache Pig (2017), and Apache Spark (Zaharia et al. 2012), among others.

File Organization

The ORC format is an optimized version of the previously used Row Columnar (RC) file format (He et al. 2011). The format is self-describing as it includes the schema and encoding information for all the data in the file. Thus, no external metadata is required in order to interpret the data in the file. The format supports both primitive (e.g., integer, Boolean, etc.) and complex data types (maps, lists, structs, unions). Apart from the columnar organization, the ORC file supports various compression techniques and lightweight indexes.

The structure of an ORC file is shown in Fig. 2. As shown in the figure, the file consists of three major parts: a set of `stripes`, a `file footer`, and a `postscript`. The `postscript` part provides all the necessary information to parse the rest of the file such as the compression codec used and the length of the file footer. The `file footer` contains information related to the data stored in the file, such as the number of rows in the file, statistics about the columns, and the data schema. The process of reading the file starts by first reading the tail of the file that consists of the `postscript` and the `file footer` that contain metadata about the main body of the file.

An ORC file stores multiple groups of row data as `stripes`. Each `stripe` has a size of about 250 MB and contains only entire rows so a row cannot span multiple stripes. Internally, each `stripe` is divided into index data, row data, and stripe footer in that order. The data columns are stored next to each other in a PAX-like (Ailamaki et al. 2001) organization. Depending on the query's access pattern, only the necessary columns are decompressed and deserialized. The stripe footer contains the location of the columns in data as well as information about the encoding of each column. The indexes contain statistics about each column in a row group (set of 10,000 rows). For example, in the case of integer columns, the column statistics include the minimum, maximum, and sum values of the column in the given row group. These statistics can be used to skip entire sets of rows that do not satisfy the query predicates. Recently, bloom filters can additionally be used to better prune row groups (ORC Index 2017).

Finally, it is worth noting that column statistics are also stored in the `file footer` at the granularity of `stripes`. These statistics enable skipping entire `stripes` based on a filtering predicate.

Compression

Depending on the user's configuration, an ORC file can be compressed using a generic compression codec (typically zlib (ZLIB Compression 2017) or snappy (Snappy Compression 2017)).

The file is compressed in chunks so that entire chunks can be directly skipped without decompressing the data. The default compression chunk size is 256 KB but it is configurable. Having larger chunks typically results in better compression ratios but requires more memory to be allocated during decompression.

The ORC file makes use of various run-length encoding techniques to further improve compression. The interested reader can find more information about the supported encoding methods in ORC Encodings (2017)

File Format Comparison

The Parquet and ORC file formats have both been created with the goal of providing fast analytics in the Hadoop ecosystem. Both file formats are columnar and they have adopted a PAX-like layout. Instead of storing each column at a separate file, the columns are stored next to each other in the same HDFS block. By having large `row groups` and HDFS blocks, both formats exploit large sequential I/Os and can skip unnecessary columns.

Both formats support various compression codecs at a fine granularity so that the amount of data that is decompressed is minimized. They also support various encoding schemes depending on the data type.

Finally, the file formats incorporate statistics at various levels in order to avoid decompressing and deserializing data that do not satisfy the filtering predicates. Typically, the statistics include the maximum and the minimum values of a column. The ORC file additionally supports bloom filters at the `row group` level, while the Parquet file supports statistics at the `page` level as well.

Regarding the data types supported, the Parquet file format has been designed to inherently support deeply nested data using the record shredding and assembly algorithm presented in Melnik et al. (2010). The ORC file, on the other hand, flattens the nested data and creates separate columns for each underlying primitive data type.

Conclusions

The increased interest in fast analytics over Hadoop data fueled the development of multiple open-source data file formats. In this work, we focused on the columnar storage formats that are used to store HDFS data. In particular, we first reviewed the work on disk-based columnar formats for Big Data systems and then presented a detailed description of the open-source ORC and Parquet file formats. These two columnar data formats are currently supported by a plethora of Big Data solutions. Finally, we highlighted the differences and similarities of the two file formats.

Cross-References

- ▸ Caching for SQL-on-Hadoop
- ▸ Wildfire: HTAP for Big Data

References

Ailamaki A, DeWitt DJ, Hill MD, Skounakis M (2001) Weaving relations for cache performance. In: Proceedings of the 27th international conference on very large data bases (VLDB'01), pp 169–180

Apache Arrow (2017) Apache Arrow. https://arrow.apache.org/

Apache Hadoop (2017) Apache Hadoop. http://hadoop.apache.org

Apache Hadoop HDFS (2017) Apache Hadoop HDFS. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

Apache Hbase (2017) Apache HBase. https://hbase.apache.org/

Apache Hive (2017) Apache Hive. https://hive.apache.org/

Apache Kudu (2017) Apache Kudu. https://kudu.apache.org/

Apache ORC (2017) Apache ORC. https://orc.apache.org/

Apache Parquet (2017) Apache Parquet. https://parquet.apache.org/

Apache Pig (2017) Apache Pig. https://pig.apache.org/

Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Floratou A, Patel JM, Shekita EJ, Tata S (2011) Column-oriented Storage Techniques for MapReduce. Proc VLDB Endow 4(7):419–429

Floratou A, Minhas UF, Özcan F (2014) SQL-on-Hadoop: full circle back to shared-nothing database architectures. Proc VLDB Endow 7(12):1295–1306

He Y, Lee R, Huai Y, Shao Z, Jain N, Zhang X, Xu Z (2011) 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 (ICDE'11). IEEE Computer Society, pp 1199–1208

Huai Y, Ma S, Lee R, O'Malley O, Zhang X (2013) Understanding insights into the basic structure and essential issues of table placement methods in clusters. Proc VLDB Endow 6(14):1750–1761

Kornacker M, Behm A, Bittorf V, Bobrovytsky T, Ching C, Choi A, Erickson J, Grund M, Hecht D, Jacobs M, Joshi I, Kuff L, Kumar D, Leblang A, Li N, Pandis I, Robinson H, Rorke D, Rus S, Russell J, Tsirogiannis D, Wanderman-Milne S, Yoder M (2015) Impala: a modern, open-source SQL engine for hadoop. In: CIDR

Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T (2010) Dremel: interactive analysis of web-scale datasets. Proc VLDB Endow 3(1–2):330–339

ORC Encodings (2017) ORC Encodings. https://orc.apache.org/docs/run-length.html

ORC Index (2017) ORC Index. https://orc.apache.org/docs/spec-index.html

Parquet Encodings (2017) Parquet Encodings. https://github.com/apache/parquet-format/blob/master/Encodings.md

Snappy Compression (2017) Snappy Compression. https://en.wikipedia.org/wiki/Snappy_(compression)

Vertica (2017) Vertica. https://www.vertica.com/

Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. NSDI, USENIX

ZLIB Compression (2017) ZLIB Compression. https://en.wikipedia.org/wiki/Zlib