CarbonData Introduction

CarbonData is a fully indexed columnar and Hadoop native data-store for processing heavy analytical workloads and detailed queries on big data. CarbonData allows faster interactive query using advanced columnar storage, index, compression and encoding techniques to improve computing efficiency, which helps in speeding up queries by an order of magnitude faster over PetaBytes of data.

In customer benchmarks, CarbonData has proven to manage Petabyte of data running on extraordinarily low-cost hardware and answers queries around 10 times faster than the current open source solutions (column-oriented SQL on Hadoop data-stores).

Some of the salient features of CarbonData are :

- Low-Latency for various types of data access patterns like Sequential, Random and OLAP.
- Fast query on fast data.
- Space efficiency.
- General format available on Hadoop-ecosystem.

CarbonData Features

CarbonData file format is a columnar store in HDFS. It has many features that a modern columnar format has, such as split-table, compression schema, complex data type etc and CarbonData has following unique features:

- Unique Data organization: Though CarbonData stores data in Columnar format, it differs from traditional Columnar formats as the columns in each row-group(Data Block) is sorted independent of the other columns. Though this arrangement requires CarbonData to store the row-number mapping against each column value, it makes it possible to use binary search for faster filtering and since the values are sorted, same/similar values come together which yields better compression and offsets the storage overhead required by the row number mapping. For more details, see Unique Data Organization.
- Advanced Push Down Optimizations: CarbonData pushes as much of query processing as possible close to the data to minimize the amount of data being read, processed, converted and transmitted/shuffled. Using projections and filters it reads only the required columns form the store and also reads only the rows that match the filter conditions provided in the query. For more details, see Advanced PushDown Optimizations.
- Multi Level Indexing: CarbonData uses multiple indices at various levels to enable faster search and speed up query processing. For more details, see Multi Level Indexing.
- Dictionary Encoding: Most databases and big data SQL data stores employ columnar encoding to achieve data compression by storing small integers numbers (surrogate value) instead of full string values. However, almost all existing databases and data stores divide the data into row groups containing anywhere from few thousand to a million rows and employ dictionary encoding only within each row group. Hence, the same column value can have different surrogate values in different row groups. So, while reading the data, conversion from surrogate value to actual value needs to be done immediately after the data is read from the disk. But CarbonData employs global surrogate key which means that a common dictionary is maintained for the full store on one machine/node. So CarbonData can perform all the query processing work such as grouping/aggregation, sorting etc on light weight surrogate values. The conversion from surrogate to actual values needs to be done only on the final result. This procedure improves performance on two aspects. Conversion from surrogate values to actual values is done only for the final result rows which are much less than the actual rows read from the store. All query processing and computation such as grouping/aggregation, sorting, and so on is done on lightweight surrogate values which requires less memory and CPU time compared to actual values. For more details, see Dictionary Encoding.
- Deep Spark Integration: It has built-in spark integration for Spark 1.6.2, 2.1 and interfaces for Spark SQL, DataFrame API and query optimization. It supports bulk data ingestion and allows saving of spark dataframes as CarbonData files. For more details, see Seamless Integration with Big Data Eco-System.
- Update Delete Support: It supports batch updates like daily update scenarios for OLAP and Base+Delta file based design. For more details, see U pdate and Delete Support.
- Bucketing : It is a technique that is used for uniform distribution of data across files in CarbonData. It enhances the performance of join queries. While loading the data, records are placed into buckets based on hashing algorithm. During the execution of join queries the records can be fetched from buckets with out need of shuffling.This feature is used to distribute/organize the table/partition data into multiple files placing similar records in same file.
- Global Multi Dimensional Keys(MDK) based B+Tree Index for all non- measure columns: Aids in quickly locating the row groups(Data Blocks) that contain the data matching search/filter criteria.
- Min-Max Index for all columns: Aids in quickly locating the row groups(Data Blocks) that contain the data matching search/filter criteria.
- Data Block level Inverted Index for all columns: Aids in quickly locating the rows that contain the data matching search/filter criteria within a row group(Data Blocks).
- Store data along with index: Significantly accelerates query performance and reduces the I/O scans and CPU resources, when there are filters in the query. CarbonData index consists of multiple levels of indices. A processing framework can leverage this index to reduce the task it needs to schedule and process. It can also do skip scan in more finer grain units (called blocklet) in task side scanning instead of scanning the whole file.
- Operable encoded data: It supports efficient compression and global encoding schemes and can query on compressed/encoded data. The data can be converted just before returning the results to the users, which is "late materialized".
- Column group: Allows multiple columns to form a column group that would be stored as row format. This reduces the row reconstruction cost at query time.
- Support for various use cases with one single Data format: Examples are interactive OLAP-style query, Sequential Access (big scan) and Random Access (narrow scan).

The ID 'Unique Data Organization' is not a valid bookmark ID. IDs must begin with a letter ([A-Za-z]) followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("_"), and periods (".").