Implementing Columnar Databases With R

JUNE 2020

Manikant Prasad

Milliman


Milliman

Contents

Introduction

With the growing data volume, we are now facing use cases where data stored on disk, as files, is too large for the systems to handle and manage, and needs to be stored outside of the system, in a database, so that one can connect to the database, when needed, to retrieve only the chunks needed for the current analysis.

In this paper, we are trying to solve the problem where a system or application, built over R, can communicate to a database with massive data volume, residing on a remote system, to retrieve only the chunks of data needed for current analysis to gain scalability and speed by leveraging the data management and querying speed of the databases. We also perform a simple benchmarking between MariaDB Columnstore and MariaDB Rowstore databases, which are both well-known, stable, and open source databases, to select the right type of database for an analytics use case.

Connecting to MariaDB

Connecting to MariaDB databases, whether it's the Columnstore or Rowstore database, is quite simple and uses the same set of libraries and functions. We assume that you already have the MariaDB databases installed on a system. Please refer to MariaDB Rowstore and Columnstore installation guides, in case you need help installing or updating the databases. We have used MariaDB Columnstore 1.1.6 and MariaDB Server 10.3.9 in this paper for the benchmarking.

System Requirements

A MariaDB or MySQL Open Database Connectivity(ODBC) connector should be installed on the system prior to installing R libraries. (libmariadb-client-lgpl-dev or libmysqlclient-dev (deb), mariadb-connector-c-devel or mariadb-devel (rpm), mariadb-connector-c or mysql-connector-c (brew), unixodbc-dev (unix))

R Packages Required

- odbc package needs to be installed first for the transfer of data between the MariaDB Server and R environment.
- The RMariaDB package provides the database interface and the driver to MariaDB databases by implementing a Database Interface(DBI)-compliant interface to MariaDB and MySQL databases.

To load the packages into your active R environment, use the *library()* function.

```
library(odbc)
library(RMariaDB)
```

Creating Connection Object

For establishing a connection with the MariaDB Server and fetching the connection object, the *dbConnect()* function is used.

```
dbConnect(drv = RMariaDB::MariaDB(), username = "username",
          password = "password", host = "host", port = 3306)
```

Arguments

- *drv*: The DBI Driver object. In the case of MariaDB, it should be RMariDB::MariaDB()
- *username*: Username of the database you want to use for establishing connection with the server
- *password*: Password of the user selected
- *host*: IP of the server where the database has been installed. Use `localhost` in cases where a database is on the same server
- *port*: Port on which the database is reachable. The default port is 3306.

Store the output of *dbConnect()* in a variable to use the connection object created.

Executing Queries

For executing queries that do not produce a result set, like an Update statement, the *dbExecute()* function should be used. It returns the number of affected rows after executing the statement.

```
dbExecute(conn, statement, ...)
```

Arguments

- *conn*: Active connection object created by *dbConnect()*
- *statement*: A string containing a valid SQL statement

Fetching Query ResultSet

For executing queries that return a result set, like a Select statement, the *dbGetQuery()* function is used. It returns the result of the query as a data frame object.

```
dbGetQuery(conn, statement, ...)
```

Arguments

- *conn*: Active connection object created by *dbConnect()*
- *statement*: A string containing a valid SQL statement

Closing Database Connection

Once all the work is done and the database connection is no longer required, one should close the connection to free acquired resources. For closing an active connection, the *dbDisconnect()* function is used. In case a connection was closed or killed prior to calling *dbDisconnect()*, a warning is issued on calling the function.

```
dbDisconnect(conn, ...)
```

Arguments

- *conn*: Active connection object created by *dbConnect()*

Rowstore vs. Columnstore

From the previous section, it is clear that interacting with MariaDB databases, either MariaDB Server or Columnstore, is quite easy and exactly alike. Most of the queries used for different operations on the two databases are almost the same. However, they differ significantly in their working and underlying structures, starting with their storage engines and mechanisms. The MariaDB row-based database, like any other rowstore database, has been designed for online transaction processing(OLTP) workload and generally works better for transaction-oriented applications. Whereas, columnstore databases are generally designed for online analytical processing(OLAP) workload and generally perform a lot better for modern analytics applications.

Rowstore Databases

Rowstore databases have row oriented storage, which basically means all the column values from a row are stored together physically. Almost all the popular rowstore databases follow strong principles of indexes and use that to physically sort data and improve performance through lookup using index tables, which are normally stored as B-trees. Use of indexes speeds up the look-up for specific rows and thus improves performance for transactional queries.

Columnstore Databases

Columnstore databases, as the name suggests, store tables by columns instead of rows. All the values from a particular column are stored in the same file or set of files. It significantly reduces the size of the data being scanned for queries, especially for the analytics tables with hundreds of fields. Normally, columnar databases like MariaDB Columnstore will have better and more optimized compression and decompression strategies.

Benchmarking

We did a simple benchmarking between MariDB Rowstore and Columnstore databases to see and compare the performance of the two on the same dataset and for the same queries. For this, we selected Wikipedia's hourly page views data.

System configuration

For our benchmarking, we used two systems with the same hardware configurations. Both systems were running Ubuntu 18.04 with 32GB of RAM and 50GB of solid-state drive(SSD) storage. The only configuration that was changed was to increase the database sort buffer size to allow heavy sorts. No clustered or non-clustered index was created.

Data

Each row in page views data has four rows, *Category*, *Pagename*, *Number of Requests*, and *Size* of the data returned. Data for each hour had a separate file. We had to add additional columns in each file to denote *year*, *month*, *date*, and *hour* for each row, before uploading them in the database, for better understanding of data.

We created three different tables from the hourly page views data set, each containing page views data for:

- Two hours (12 million rows)
- 12 hours (71 million rows)
- 24 hours (154 million rows)

Queries

We ran four different basic analytical queries, on each table, each performing a different operation.

- Requests per hour: Simple aggregation to sum the number of requests received per hour, an integer column with low cardinality.
- Requests per category and day: Aggregating the requests received for two columns, out of which was a column with high cardinality.
- Top pages in english category: Filtering the data based on a column value and then sorting the data set based on aggregation of the requests received.
- Page1 vs page2 - Applying a simple filter on a column for a set of values.

Interpreting Benchmarking Results

The Table in figure 1 shows the results from the benchmarking and looking at them it is quite clear that MariaDB Columnstore performed drastically well against Rowstore, especially at higher data volumes. Columnstore seems to be quite graceful in handling the growth in volumes of data. The third query, involving an aggregation and sort along with filtering, was the most complex and Columnstore took hardly a minute more to run for a data set 12 times greater than that in the first data set, whereas the Rowstore took almost 13 minutes more, for the third data set, than it took for the first data set. It is quite evident that the indexing here would have been quite helpful for Rowstore, whereas, the Columnstore performed quite well even without any kind of indexing.

Table 1: **MariaDB Columnstore vs MariaDB Rowstore**

Query	Data	MariaDB Columnstore	MariaDB Rowstore
Requests/Hour	2 Hours	1.022s	11.732s
	12 Hours	3.92s	75.36s
	1 Day	7.535s	169.97s
Requests/Category & Day	2 Hours	1.612s	12.741s
	12 Hours	7.225s	93s
	1 Day	16.307s	182.1s
Top Pages in English Category	2 Hours	7.343s	49.412s
	12 Hours	36.016s	308.88s
	1 Day	69.96s	839.64s
Page1 vs Page2	2 Hours	1.221s	10.863s
	12 Hours	12.878s	84.66s
	1 Day	20.667s	127.98s

Using Columnstore database: Restriction

Columnar databases are designed to read or write data in bulk and sequentially. Any use case involving large amounts of random or transactional read or write is going to perform poorly while using a columnar database. For the same reason, updates tend to be quite slow on columnar databases because it involves transactional read and write, for which the columnar databases are not optimized. It is critical to restrict updates to a minimum while using columnar databases.

Conclusion

We saw how easy it is to get going with the MariaDB databases (or even any other database) using R, requiring knowledge of a limited number of packages with little or no complexities involved.

One thing to note here is that all these queries read data sequentially, which most of the analytical queries do, rather than randomly, for which Rowstore is more optimized. Whereas the storage architecture for columnar databases makes them more suitable for sequential read and write. Any query trying to read or write data randomly, like an update statement, will run better for Rowstore databases. For the same reason, columnar databases perform quite well and are now preferred, for analytics use cases, over row-oriented databases. However, for transactional use cases, columnar databases are no replacement for Rowstore databases.

Milliman is among the world's largest providers of actuarial and related products and services. The firm has consulting practices in life insurance and financial services, property & casualty insurance, healthcare, and employee benefits. Founded in 1947, Milliman is an independent firm with offices in major cities around the globe.

milliman.com

CONTACT
Milliman.VALUES@milliman.com