Spark DataFrame SQL

Jim Harner

9/28/2020

Load `sparklyr` and establish the Spark connection.

```
library(dplyr, warn.conflicts = FALSE)
library(sparklyr)

# start the sparklyr session
master <- "local"
# master <- "spark://master:7077"
sc <- spark_connect(master, spark_home = Sys.getenv("SPARK_HOME"),
                    method = c("shell"), app_name = "sparklyr")
```

5.3 Spark DataFrame SQL

`sparklyr` can import a wide range of data directly into Spark from an external data source, e.g., json. In addition, it is possible to query Spark DataFrames directly.

We will be using the `nycflights13` data again. The `flights` and `airlines` R data frames are copied into Spark.

```
library(nycflights13)
flights_sdf <- copy_to(sc, flights, "flights",  overwrite = TRUE)
airlines_sdf <- copy_to(sc, airlines, "airlines", overwrite = TRUE)
```

5.3.1 Joining Spark Data Tables

In Section 5.2.1 the `dplyr` verbs were used to manipulate a Spark DataFrame. However, we often have multiple related Spark SQL tables which we need to combine prior to performing data manipulations.

A workflow was developed in Section 5.2.1 to find the flights with a departure delay greater than 1000 minutes. However, we did not have the carrier names since they were in a different table. Providing this information can be done with a `left_join`.

```
flights_sdf %>%
  left_join(airlines_sdf, by = "carrier") %>%
  select(carrier, name, flight, year:day, arr_delay, dep_delay) %>%
  filter(dep_delay > 1000) %>%
  arrange(desc(dep_delay))
```

```
## # Source:     spark<?> [?? x 8]
## # Ordered by: desc(dep_delay)
##   carrier name                  flight  year month   day arr_delay dep_delay
##   <chr>   <chr>                  <int> <int> <int> <int>     <dbl>     <dbl>
## 1 HA      Hawaiian Airlines Inc.     51  2013     1     9      1272      1301
## 2 MQ      Envoy Air                3535  2013     6    15      1127      1137
## 3 MQ      Envoy Air                3695  2013     1    10      1109      1126
```

```
## 4 AA      American Airlines Inc.    177  2013     9    20      1007      1014
## 5 MQ      Envoy Air                3075  2013     7    22       989      1005
```

Notice that three of the top five largest delays were associated with Envoy Air, which was not obvious based on the two-letter abbreviation.

`dplyr` has various verbs that combine two tables. If this is not adequate, then the joins, or other operations, must be done in the database prior to importing the data into Spark

5.3.2 Querying a Spark DataFrame

It is also possible to use Spark DataFrames as tables in a "database" using the Spark SQL interface, which forms the basis of Spark DataFrames.

The `spark_connect` object implements a DBI interface for Spark, which allows you to use `dbGetQuery` to execute SQL commands. The returned result is an R data frame.

We now show that the above workflow can be done in R except that R data frames are used.

```
library(DBI)
flights_df <- dbGetQuery(sc, "SELECT * FROM flights")
airlines_df <- dbGetQuery(sc, "SELECT * FROM airlines")
flights_df %>%
  left_join(airlines_df, by = "carrier") %>%
  select(carrier, name, flight, year:day, arr_delay, dep_delay) %>%
  filter(dep_delay > 1000) %>%
  arrange(desc(dep_delay))
```

```
##   carrier                   name flight year month day arr_delay dep_delay
## 1      HA Hawaiian Airlines Inc.     51 2013     1   9      1272      1301
## 2      MQ              Envoy Air   3535 2013     6  15      1127      1137
## 3      MQ              Envoy Air   3695 2013     1  10      1109      1126
## 4      AA American Airlines Inc.    177 2013     9  20      1007      1014
## 5      MQ              Envoy Air   3075 2013     7  22       989      1005
```

Of course, this assumes the Spark DataFrames can be imported into R, i.e., they must fit into local memory.

The `by` argument in the `left_join` is not needed if there is a single variable common to both tables. Alternately, we could use `by = c("carrier", "carrier")`, where the names could be different if they represent the same variable.

5.3.3 Sampling

We can sample random rows of a Spark DataFrame using:

- `sample_n` for a fixed number;
- `sample_frac` for a fixed fraction.

```
sample_n(flights_sdf, 10)
```

```
## # Source: spark<?> [?? x 19]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     4    21     1827           1830        -3     2052           2100
##  2  2013     3     4      609            610        -1      845            855
##  3  2013    12    23     2334           2250        44       40           2356
##  4  2013     4    21     1216           1220        -4     1510           1505
##  5  2013     4    10      632            630         2      819            820
```

```
##  6  2013    12    11      555            600        -5      844            825
##  7  2013    12    27      800            750        10     1110           1115
##  8  2013     4     9     1520           1519         1     1808           1820
##  9  2013     3    10     1727           1725         2     2029           2050
## 10  2013     5    15     1905           1909        -4     2234           2235
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
sample_frac(flights_sdf, 0.01)
```

```
## # Source: spark<?> [?? x 19]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7    11      750            752        -2      938            953
##  2  2013     1    31     2109           2000        69     2239           2147
##  3  2013    10     3     1433           1435        -2     1741           1732
##  4  2013     1    21      849            900       -11      938           1020
##  5  2013     4    29     1312           1317        -5     1642           1710
##  6  2013     1     2     1848           1701       107     2112           1856
##  7  2013     7     2     2115           2054        21        4           2354
##  8  2013    10    20     1026           1029        -3     1315           1315
##  9  2013     1    13     1929           1700       149     2232           2006
## 10  2013     3     9     1828           1833        -5     2046           2124
## # ... with more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Sampling is often done during the development and testing cycle to limit the size of the data.

5.3.4 Writing Data

We can save the results of our analysis or the tables that you have generated in Spark into persistent storage. Parquet is a commonly used persistent store for various data processing systems in the Hadoop ecosystem. It has a columnar storage format which Spark SQL supports for both reading and writing, including the schema of the original data.

As an example, we can write the `airlines_sdf` Spark DataFrame out to a Parquet file using the `spark_write_parquet` function.

```
library(rhdfs)
```

```
## Loading required package: rJava
```

```
##
## HADOOP_CMD=/opt/hadoop/bin/hadoop
```

```
##
## Be sure to run hdfs.init()
```

```
hdfs.init()
spark_write_parquet(airlines_sdf,
                    path = "hdfs://hadoop:9000/user/rstudio/airlines_parquet",
                    mode = "overwrite")
hdfs.ls("/user/rstudio")
```

```
##   permission   owner   group size          modtime
## 1 drwxr-xr-x rstudio rstudio    0 2020-09-29 04:07
##                             file
```

```
## 1 /user/rstudio/airlines_parquet
```

This writes the Spark DataFrame to the given HDFS path and names the Parquet file `airlines_parquet`.

You can use the `spark_read_parquet` function to read the same table back into a subsequent Spark session:

```
spark_read_parquet(sc, "airlines2_sdf",
                   "hdfs://hadoop:9000/user/rstudio/airlines_parquet")
```

```
## # Source: spark<airlines2_sdf> [?? x 2]
##    carrier name
##    <chr>   <chr>
##  1 9E      Endeavor Air Inc.
##  2 AA      American Airlines Inc.
##  3 AS      Alaska Airlines Inc.
##  4 B6      JetBlue Airways
##  5 DL      Delta Air Lines Inc.
##  6 EV      ExpressJet Airlines Inc.
##  7 F9      Frontier Airlines Inc.
##  8 FL      AirTran Airways Corporation
##  9 HA      Hawaiian Airlines Inc.
## 10 MQ      Envoy Air
## # ... with more rows
```

Note that `airlines2_sdf` is a Spark DataFrame. Use the `spark_write_csv` and `spark_write_json` functions among others to write data to HDFS as `csv` or `json` files, respectively.

```
spark_disconnect(sc)
```