Spark DataFrame SQL

Jim Harner

9/30/2020

Load `sparklyr` and establish the Spark connection.

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

# start the spark 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     9    13     1723           1725        -2     2001           2020
##  2  2013     5     9     1535           1540        -5     1656           1736
##  3  2013     7     7      456            500        -4      630            640
##  4  2013    11     9     1146           1151        -5     1440           1459
##  5  2013     9     2     1505           1510        -5     1639           1641
```

```
##  6  2013     7    25     2234           2030       124        9           2206
##  7  2013     6     9      832            830         2     1043           1049
##  8  2013     2    12     1857           1900        -3     2018           2051
##  9  2013    10    10      811            817        -6     1139           1127
## 10  2013     8     4     1031           1038        -7     1145           1156
## # ... 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     1    27     1855           1858        -3     2022           2055
##  2  2013     6    24     2021           1729       172     2351           2059
##  3  2013     2    26      623            620         3     1051           1102
##  4  2013     4    11     2233           1929       184      100           2229
##  5  2013     1    20     1628           1629        -1     1938           1931
##  6  2013     9    15      758            808       -10      912            929
##  7  2013    12    19     1838           1815        23     2026           2005
##  8  2013     3     1     1809           1725        44     1920           1855
##  9  2013     6    17      941            945        -4     1252           1305
## 10  2013     2    24     2202           2115        47       37           2358
## # ... 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 to HDFS

We can save the results of our analysis or the tables that you have generated in Spark into HDFS 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 2020-10-01 03:47
##                             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.

5.3.5 Hive Functions

Many of Hive's built-in functions (UDF) and built-in aggregate functions (UDAF) can be called by `dplyr`'s `mutate` and `summarize` funcitons.

`datediff` and `current_date` are Hive UDFs to figure the difference between the `flight_date` and the current system date:

```
flights_sdf %>%
  mutate(flight_date = paste(year, month, day, sep="-"),
         days_since = datediff(current_date(), flight_date)) %>%
  group_by(flight_date, days_since) %>%
  tally() %>%
  arrange(days_since)
```

```
## # Source:     spark<?> [?? x 3]
## # Groups:     flight_date
## # Ordered by: days_since
##    flight_date days_since     n
##    <chr>            <int> <dbl>
##  1 2013-12-31        2466   776
##  2 2013-12-30        2467   968
##  3 2013-12-29        2468   888
##  4 2013-12-28        2469   814
##  5 2013-12-27        2470   963
##  6 2013-12-26        2471   936
##  7 2013-12-25        2472   719
##  8 2013-12-24        2473   761
##  9 2013-12-23        2474   985
## 10 2013-12-22        2475   895
## # ... with more rows
```

```
spark_disconnect(sc)
```