Spark Basics

Jim Harner

6/7/2019

The `sparklyr` R frontend to Spark is based on a `dplyr` interface to Spark SQL. Once these required packages are loaded, a Spark connection is established.

```
library(dplyr, warn.conflicts = FALSE)
library(sparklyr)
# start the sparklyr session locally or to the master container
master <- "local"
# master <- "spark://master:7077"
sc <- spark_connect(master = master)
```

Spark is a general-purpose cluster computing system, which:

- has high-level APIs in Java, Scala, Python and R;
- supports multi-step data pipelines using directed acyclic graphs (DAGs);
- supports in-memory data sharing across DAGs allowing different jobs to work with the same data.

Spark provides a unified framework to manage big data processing with a variety of data sets that are diverse in nature, e.g., text data, graph data, etc., as well as the source of data (batch vs. real-time streaming data).

Spark supports a rich set of higher-level tools including:

- *Spark SQL* for running SQL-like queries on Spark data using the JDBC API or the Spark SQL CLI. Spark SQL allows users to extract data from different formats, (e.g., JSON, Parquet, or Hive), transform it, and load it for *ad-hoc* querying, i.e., ETL.
- *MLlib* for machine learning, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and the underlying optimization algorithms. MLlib uses the DataFrame API and thus takes advantage of the Spark SQL engine.
- *Structured Streaming* for real-time data processing. Spark streaming uses a fault-tolerant stream processing engine built on the Spark SQL engine. Thus, you can express your streaming computation the same way you would express a batch computation on static data. Using the DataFrame API, the Spark SQL engine will take care of running it incrementally and continuously and updating the final result as streaming data continues to arrive.

SparkR is part of the officially supported Spark distribution. However, we will focus on the sparklyr package.

5.1 Sparklyr Basics

The `sparklyr` package is being developed by RStudio. It is undergoing rapid expansion. See RStudio's sparklyr for information.

The `sparklyr` R package provides a `dplyr` backend to Spark. Using `sparklyr`, you can:

- filter and aggregate Spark DataFrames and bring them into R for analysis and visualization;
- develop workflows using `dplyr` and compatible R packages;
- write R code to access Spark's machine learning library, MLlib;
- create Spark extensions.

Using `sparklyr`, connections can be made to local instances or to remote Spark clusters. In our case the connection is to a local connection bundled in the `rstudio` container.

The `sparklyr` library is loaded in the setup above and a Spark connection is established. The Spark connection `sc` provides a `dplyr` interface to Spark.

5.1.1 dplyr

The `dpyr` verbs, e.g., `mutate`, `filter`, can be used on Spark DataFrames. A more complete discussion is given in Section 5.2.

We will use the `flights` data in the `nycflights13` package as an example. If its size becomes an issue, execute each chunk in sequence in notebook mode.

```
library(nycflights13)
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  19 variables:
##  $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr  "UA" "UA" "AA" "B6" ...
##  $ flight        : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num  1400 1416 1089 1576 762 ...
##  $ hour          : num  5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num  15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

The `flights` R data frame is a tibble, which allows large data to be displayed. This data frame has the date of departure, the actual departure time, etc. See the package documentation for variable definitions.

The `copy_to` function copies an R `data.frame` to Spark as a Spark table. The resulting object is a `tbl_spark`, which is a `dplyr`-compatible interface to the Spark DataFrame.

```
flights_tbl <- copy_to(sc, nycflights13::flights, "flights")
flights_tbl
```

```
## # Source: spark<flights> [?? x 19]
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
```

```
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with more rows, and 12 more variables: sched_arr_time <int>,
## #   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>
```

```
src_tbls(sc)
```

```
## [1] "flights"
```

By default, the `flights` Spark table is cached in memory (`memory = TRUE`), which speeds up computations, but by default the table is not partitioned (`repartition = 0L`) since we are not running an actual cluster. See the `copy_to` function in the `sparklyr` package for more details.

The Spark connection should be disconnected at the end of a task.

```
spark_disconnect(sc)
```

```
## NULL
```