Spark Apply

Jim Harner

1/12/2021

Load `sparklyr` and establish the Spark connection.

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

# start the sparklyr session locally or to the master container
if(system("test \"/bin/spark-class/\" && echo 1 || echo 0") == 1){
  master <- "spark://master:7077"
} else{
  master <- "local"
}
sc <- spark_connect(master = master)
```

5.4 Spark Apply

`sparklyr` provides support to run arbitrary R code at scale within Spark through the function `spark_apply`. Thus, much of R's functionality can be distributed across an R cluster. Apache Spark, even with Spark Packages, has a limited range of functions available.

`spark_apply` applies an R function to a Spark DataFrame, typically. Spark objects are partitioned so they can be distributed across a cluster. You can use `spark_apply` with the default partitions or you can define your own partitions with the `group_by` argument. Your R function must return another Spark DataFrame. `spark_apply` will run your R function on each partition and output a single Spark DataFrame.

5.4.1 Apply an R function to a Spark Object

Let's apply the identify function, `I()`, over a list of numbers we created with the `sdf_len` function.

```
sdf_len(sc, length = 5, repartition = 1) %>%
  spark_apply(function(e) I(e))
```

```
## # Source: spark<?> [?? x 1]
##       id
##    <int>
## 1      1
## 2      2
## 3      3
## 4      4
## 5      5
```

5.4.2 Group By

A common task is to apply your R function to specific groups in your data, e.g., computing a regression model for each group. This is done by specifying a `group_by` argument.

The following example initially counts the number of rows in the `iris` data frame for each species.

```r
iris_tbl <- copy_to(sc, iris)
iris_tbl %>%
  spark_apply(nrow, group_by = "Species")
```

```
## # Source: spark<?> [?? x 2]
##   Species     result
##   <chr>        <int>
## 1 versicolor      50
## 2 virginica       50
## 3 setosa          50
```

Now compute the R^2 for a linear model for each species.

```r
iris_tbl %>%
  spark_apply(
    function(e) summary(lm(Petal_Length ~ Petal_Width, e))$r.squared,
    names = "r.squared",
    group_by = "Species")
```

```
## # Source: spark<?> [?? x 2]
##   Species     r.squared
##   <chr>           <dbl>
## 1 versicolor      0.619
## 2 virginica       0.104
## 3 setosa          0.110
```

5.4.3 Distributed Packages

With `spark_apply` you can use nearly any R package inside Spark.

As an example, we use the `broom` package to create a tidy data frame from the linear regression output.

```r
spark_apply(
  iris_tbl,
  function(e) broom::tidy(lm(Petal_Length ~ Petal_Width, e)),
  names = c("term", "estimate", "std.error", "statistic", "p.value"),
  group_by = "Species")
```

```
## # Source: spark<?> [?? x 6]
##   Species    term        estimate std.error statistic  p.value
##   <chr>      <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 versicolor (Intercept)     1.78    0.284      6.28 9.48e- 8
## 2 versicolor Petal_Width     1.87    0.212      8.83 1.27e-11
## 3 virginica  (Intercept)     4.24    0.561      7.56 1.04e- 9
## 4 virginica  Petal_Width    0.647    0.275      2.36 2.25e- 2
## 5 setosa     (Intercept)     1.33    0.0600    22.1  7.68e-27
## 6 setosa     Petal_Width    0.546    0.224      2.44 1.86e- 2
```

The ability to use R packages in Spark is a killer feature, i.e., it expands the capability of Spark to most of the 13,000+ R packages. There are limitations, however. For example, referencing free variables using closures will not work.

```r
spark_disconnect(sc)
```