2_spark_maps_and_lazy_evaluation

March 27, 2022

1 Maps

In Spark, maps take data as input and then transform that data with whatever function you put in the map. They are like directions for the data telling how each input should get to the output.

The first code cell creates a SparkContext object. With the SparkContext, you can input a dataset and parallelize the data across a cluster (since you are currently using Spark in local mode on a single machine, technically the dataset isn't distributed yet).

Run the code cell below to instantiate a SparkContext object and then read in the log_of_songs list into Spark.

```
In [ ]: ###
        # You might have noticed this code in the screencast.
        #
        # import findspark
        # findspark.init('spark-2.3.2-bin-hadoop2.7')
        #
        # The findspark Python module makes it easier to install
        # Spark in local mode on your computer. This is convenient
        # for practicing Spark syntax locally.
        # However, the workspaces already have Spark installed and you do not
        # need to use the findspark module
        #
        ###

        import pyspark
        sc = pyspark.SparkContext(appName="maps_and_lazy_evaluation_example")

        log_of_songs = [
                "Despacito",
                "Nice for what",
                "No tears left to cry",
                "Despacito",
                "Havana",
                "In my feelings",
                "Nice for what",
                "despacito",
                "All the stars"
```

```
]

# parallelize the log_of_songs to use with Spark
distributed_song_log = sc.parallelize(log_of_songs)
```

This next code cell defines a function that converts a song title to lowercase. Then there is an example converting the word "Havana" to "havana".

```
In [ ]: def convert_song_to_lowercase(song):
            return song.lower()

        convert_song_to_lowercase("Havana")
```

The following code cells demonstrate how to apply this function using a map step. The map step will go through each song in the list and apply the convert_song_to_lowercase() function.

```
In [ ]: distributed_song_log.map(convert_song_to_lowercase)
```

You'll notice that this code cell ran quite quickly. This is because of lazy evaluation. Spark does not actually execute the map step unless it needs to.

"RDD" in the output refers to resilient distributed dataset. RDDs are exactly what they say they are: fault-tolerant datasets distributed across a cluster. This is how Spark stores data.

To get Spark to actually run the map step, you need to use an "action". One available action is the collect method. The collect() method takes the results from all of the clusters and "collects" them into a single list on the master node.

```
In [ ]: distributed_song_log.map(convert_song_to_lowercase).collect()
```

Note as well that Spark is not changing the original data set: Spark is merely making a copy. You can see this by running collect() on the original dataset.

```
In [ ]: distributed_song_log.collect()
```

You do not always have to write a custom function for the map step. You can also use anonymous (lambda) functions as well as built-in Python functions like string.lower().

Anonymous functions are actually a Python feature for writing functional style programs.

```
In [ ]: distributed_song_log.map(lambda song: song.lower()).collect()

In [ ]: distributed_song_log.map(lambda x: x.lower()).collect()
```