CSSE434 Research Lab - Spark

Hadoop Reddit Analyzer

Adam Gastineau, Lewis Kelley, Alex Pieragowski

Objective
After completing this lab, you will be able to use Spark to transform large amounts data in numerous ways using Java, Python, or Scala faster than traditional Hadoop allows. You will also know how to run your Java/Python/Scala code in the command line.

Requirements
1. HDP 2.6 Sandbox or HDP Cluster
2. SSH into a Linux machine
3. Eclipse Neon or later

Spark (what it is/what we use it for)
Spark is an Apache utility that allows you to transform large amounts of data much faster than you can with traditional Hadoop commands. Spark is able to do this through optimization algorithms that use MapReduce commands in the most efficient way possible. Spark programs can be written in Java, Python, or Scala, and then run in Hadoop. Spark allows the user to write SQL queries and more complex analytics with its various libraries. This can be used in machine learning, streaming, or other data transformation applications.

Spark can use a number of data sources, including HDFS and Hive. You can pull data from and push data to these data stores within your Java/Python/Scala code. After pulling data, you perform whatever transformations you want on the data, and push it back to the data source. Everything can be done within the Java/Python/Scala code except the actual running. To do that you create a jar of your class (much like you have to do for UDF in Hive or Pig), put it in the proper location on your cluster, and run it in the command line with the proper arguments and parameters.

Introduction & Set-Up
You can download Spark from its website here. You can leave the release and package types as the defaults. After downloading and unzipping, you can link your project with Spark with the dependency located on that same webpage. This requires that your project is built using Maven so you can edit the pom.xml. You may need to download other Spark projects if you want to use different tools, but the original download includes SQL, streaming, machine learning, and graphing libraries built-in.

Shell/how to access
Spark doesn't have a shell to run commands because it's mainly written in an object-oriented language and run on a Hadoop cluster. Instead, similarly to how you add jar files to Hadoop for Hive and Pig, you need to put the jar of your java class on the local filesystem where you can access and run it. From there you can run it with the built-in spark running script in the following format:

```
./bin/spark-submit --class <main class> --master <master url> --deploy-mode <mode>
--conf <key=value> <jar file> <arguments>
```

Where <main class> is the class in the project that contains the main method, the <master url> is the url for the cluster, <deploy mode> is cluster or client, <conf> are configuration options, and <arguments> are any variable values that you need in the Java/Python/Scala project.

WordCount
Spark can be used for numerous text processing applications. The one we will use to demonstrate this capability is WordCount. First you need to create a Java class (or Python or Scala, but this example will be in Java) in your Maven project with the dependencies discussed earlier. This class needs a main method, where we'll put most of our code for this example. In this example we used Java 8, which is necessary to use lambdas, but you could also work around that if your version doesn't match (just keep an eye out for those errors).

First, we need to do some error checking. If the person running the code doesn't actually send a file, then everything will break. Add an if statement that sends an error message and exits if the given arguments are empty.

Now we can start our actual SparkSession, which means this class can now be run as a Spark application. There are many possible arguments when instantiating the SparkSession, but for now just stick to:

```
SparkSession spark = SparkSession
                        .builder()
                        .appName("WordCount")
                        .getOrCreate();
```

Now we get to start on the things that make Spark unique, such as its Resilient Distributed Dataset functionality, which we will use to import the data like so:

```
JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD();
```

Where each line of the file is a different entry in the dataset. Next we want to break these lines into words so that we can count the words.

```
JavaRDD<String> words = lines.flatMap(a -> Arrays.asList(Pattern.compile("
").split(a)).iterator());
```

This line acts as a regex that breaks lines into words and inserts each word into the dataset. Next we will write a statement that acts much like a map class in a MapReduce job:

```
JavaPairRDD<String, Integer> wordMap = words.mapToPair(s -> new Tuple2<>(s, 1));
```

Much like a mapper, this one line assigns each word a value of one to say that there is one instance of it. These values will be combined in a Reduce-like function next.

```
JavaPairRDD<String, Integer> counts = wordMap.reduceByKey((i1, i2) -> i1 + i2);
```

As you can see, this line combines the mapped values from the previous statements and gets the actual counts of the words. Now there is only one final step: assigning the output, printing it to the console, and closing the connection to Spark.

```
List<Tuple2<String, Integer>> output = (List<Tuple2<String, Integer>>) counts.collect();
for (Tuple2<String, Integer> tuple : output) {
            System.out.println(tuple._1() + "\t " + tuple._2());
          }
spark.stop();
```

Follow-up Project

Continue your practice with Spark by creating a project that is able to read values from two input text files: pets.txt and people.txt. You should read these into JavaRDD's like we did in the example and write a new PairFunction to write them into JavaPairRDD's. From there, join the two JavaPairRDD's and output the result. This project aims to show you how you can map text data into Spark data structures and how easy it is to join from there.

Pet name, species:

```
Lucky,bird
Polly,dog
Whiskers,fish
Spot,cat
```

Person name, pet name:

```
Leo,Lucky
Charlotte,Whiskers
Tina,Spot
Charlie,Polly
```

Your application should be able to join these two files and return the following output:

```
Lucky,bird,Leo
Polly,dog,Charlie
Whiskers,fish,Charlotte
Spot,cat,Tina
```

Issues/Things to Look Out For

- Similar to other Hadoop tools, make sure that your Spark version will work properly with your Hadoop, Hive, and other tool versions. This is especially an issue when you start using more Spark tools, like SparkSQL and mllib.
- Our team found that support for Java isn't as extensive as it is for Scala. This wasn't an issue with the small projects in this lab, but might be more of an issue if working on something more complex.

CSSE434 Research Lab - Spark

Hadoop Reddit Analyzer

Adam Gastineau, Lewis Kelley, Alex Pieragowski

Documentation

- Spark home: https://spark.apache.org/
- Installation: https://spark.apache.org/downloads.html
- Quick start guide: https://spark.apache.org/docs/latest/quick-start.html
- Java API: https://spark.apache.org/docs/latest/api/java/index.html
- Wikipedia article: https://en.wikipedia.org/wiki/Apache_Spark