Big data architectures for Machine Learning and Data Mining

Installation & Demo scripts - Apache Spark

Sagar Nagaraj Simha
Matrikel Number - 120797, sagar.nagaraj.simha@uni-weimar.de
Master's Computer Science for Digital Media - SS2019
Bauhaus University Weimar

Install Hadoop compatible Apache Spark on a VM:

https://mikestaszel.com/2017/01/04/apache-spark-cluster-vagrant/

https://github.com/mikestaszel/spark_cluster_vagrant

Running:

One can set the node in N_Workers
$vagrant up
The web interface can be found in http://172.28.128.150:8080/

Look at the application running on the UI
Click on the App ID to see that the SparkSession is running on the two nodes

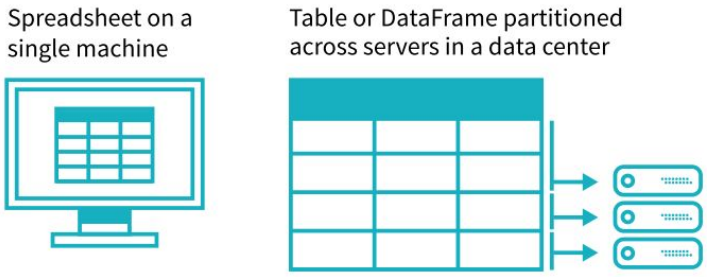
Login to the nodes using the command
$vagrant ssh hn0 (wn0/wn1)

Interactive Data Analysis using the spark python shell:

$./bin/pyspark

A Spark session is represented by the object,
>>spark
DataFrames
>>myRange = spark.range(1000).toDF("number")

Spark DataFrame - **distributed collection.**



Where are RDDs when Data Frame was created?

RDD **Partitions** data abstracted away from user when using DF

Transformation

>>divisBy2 = myRange.where("number % 2 = 0")

< Narrow Transformations and Wide Transformations (Shuffles) >

Lazy Evaluation - predicate pushdown. Provides lineage information which is used in fault recovery

>>divisBy2.explain()

Actions

>>divisBy2.count()

The executor on the Spark UI

See in UI, http://172.28.128.150:4040/executors/

Other actions:

- actions to view data in the console;
- actions to collect data to native objects in the respective language;
- and actions to write to output data sources.

“An individual **action** triggers a Spark job that represents a set of transformations triggered and we can monitor that from the Spark UI”

An End-to-End Example - Flight Data Analytics:

DataSets:

https://github.com/databricks/Spark-The-Definitive-Guide/tree/master/data

Flight data available from the United States Bureau of Transportation statistics.

Raw data file in ‘Datasets’ folder

```
>>flightData2015 = spark.read.option("inferSchema", "true").option("header",
"true").csv("/home/vagrant/Datasets/flight-data/csv/2015-summary.csv")
```

The above is just a transformation. It will be executed on the cluster with **Action**
#Takes the first 3 values and displays

```
>>flightData2015.take(3)
```

Find the cities with the least number of flights:

Planning by Apache Spark before execution
Sort the data according to count values. <Plan>

```
>>flightData2015.sort("count").explain()
```

“explain() - - **helpful tools for debugging** “. **The logical plan of transformations defines lineage for the DataFrame.**

Now, an action to kick off this plan

“ Configuration to restrict the number of prints on screen. To reduce the number of the output partitions from 200 to 2”

```
>>spark.conf.set("spark.sql.shuffle.partitions", "2")
```

```
>>flightData2015.sort("count").take(2)
```

More with DataFrames (same for SQL):

Some interesting statistics from the data.
To find out what the maximum number of flights to and from any given location are

```
>>from pyspark.sql.functions import max
>>flightData2015.select(max("count")).take(1)
```

[Row(max(count)=370002)] → US to US, Domestic flights

Find out the top five destination countries in the data:

```
>>from pyspark.sql.functions import desc
```

```
>>flightData2015.groupBy("DEST_COUNTRY_NAME").sum("count").withColumnRenamed("sum(count)", "destination_total").sort(desc("destination_total")).limit(5).collect()
```

```
>> Replace .collect() by .explain() to see the planned path.
```

We can also write it out to any data source that Spark supports - such as Postrgres SQL

This execution plan is a directed acyclic graph (DAG) of transformations, each resulting in a new immutable DataFrame, on which we call an action to generate a result

Word Count Example:

```
>>input_file = sc.textFile("/home/vagrant/Datasets/shakespeare.txt")
>>map = input_file.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1))
>>counts = map.reduceByKey(lambda a, b: a + b)
>>counts.take(50)
```

Production Applications: From Examples:

Run Interactive exploration all compiled in an executable script into production applications with a tool called spark-submit

Spark-sumbit - Calculation of Pi:

```
$cd spark
$./bin/spark-submit ./examples/src/main/python/pi.py 10
```

################################## END ######################################