Lab 2

Khrystyna Trowbridge
ktrowbridge@fordham.edu

Rahul Ohlan
rohlan@fordham.edu

Abstract— **this project focused on implementation of Spark RDD running on a 3-Node Cluster using the Google Cloud Platform. The main goal was to evaluate different datasets based on the missed and made shots, ticket parking violation number with given vehicle characteristics, as well as based on the time issued.**

Terminologies:

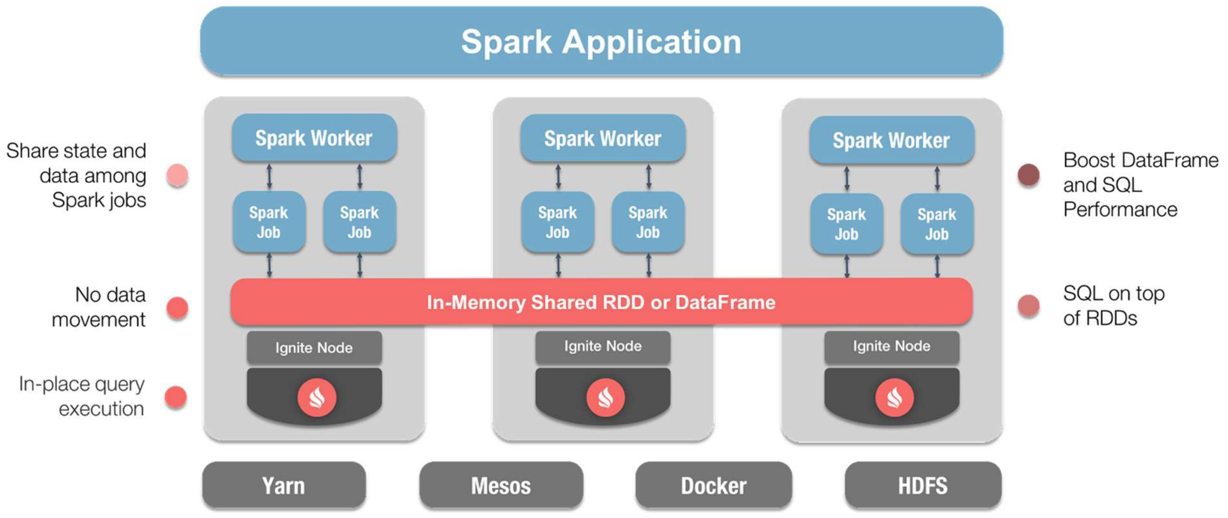
RDD - Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.
Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.
There are two ways to create RDDs − parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark - Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

PySpark - is the Python API for Apache Spark, an open source, distributed computing framework and set of libraries for real-time, large-scale data processing. If you're already familiar with Python and libraries such as Pandas, then PySpark is a good language to learn to create more scalable analyses and pipelines.

Virtual Machines - is the virtualization/emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination.

HDFS - the Hadoop Distributed File System is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data.



INTRODUCTION

The lab consists of three parts. The first part asks to use k-means to calculate the most comfortable zone for each of four players based on hit rate. We were given a dataset of the number of different games, where we had to filter the data by three columns {SHOT DIST, CLOSE DEF DIST, SHOT CLOCK} and count the number of missed and made shots.

For the second part we were given the dataset for parking tickets and were asked to calculate the probability for a black vehicle parking illegally at 34510, 10030, 34050 (street codes) to get a ticket.

The third part asked to find when parking tickets were most likely to be issued based on the NY Parking Violations dataset, while implementing different levels of parallelism - 2, 3, 4, 5.

TASK 1

Description:
NBA shot logs data : https://www.kaggle.com/dansbecker/nba-shot-logs
On shots taken during the 2014-2015 season, who took the shot, where on the floor was the shot taken from, who was the nearest defender, how far away was the nearest defender, time on the shot clock, and much more. The column titles are generally self-explanatory.

For each player, we define the comfortable zone of shooting as a matrix of, {shot dist, close def dist, shot clock} please develop a mapreduce-based algorithm to classify each player's records into 4 comfortable zones.

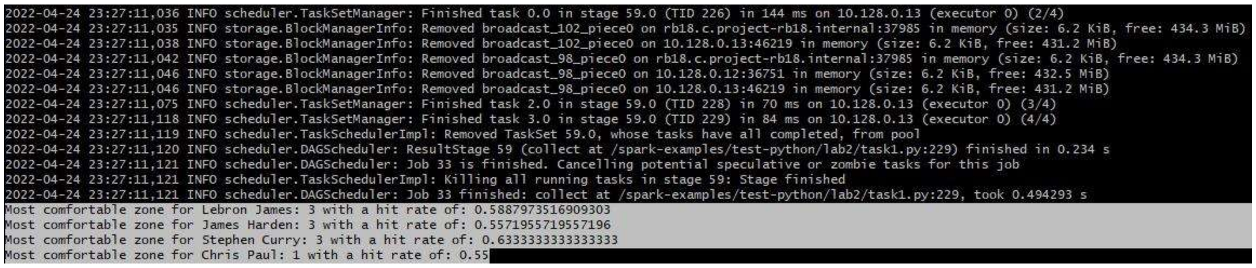
- Considering the hit rate, which zone is the best for James Harden, Chris Paul, Stephen Curry and Lebron James.

Implementation:

Data was loaded into a spark dataframe from the input csv file. To get the clusters, only columns of interest were [SHOT_DIST, CLOSE_DEF_DIST, SHOT_CLOCK]. The values of these columns were changed to float types in order to compute cluster centers using K-Means algorithm. To make the data suitable for k-means, outliers were removed.
Now this dataframe was cached to make it available for use in the future.

Transformations:

1. Use Vector Assembler to get the mentioned features in a separate column of arrays as training data for the k-means algorithm. Pyspark's ML library was used for the implementation. Parameters to Kmeans : clusters(k) = 4
2. Fit the model on the dataframe, to be used for getting predictions on the test data
3. Reuse original dataframe to include "player_names" and "SHOT_RESUT" feature (this is to account for calculating the hit rates)
4. Get data frames for each player by filtering the data frame to include on only 4 given players (Lebron James, James Harden, Stephen Curry, Chris Paul)
5. Make predictions on each of the dataframes to get zones for each datapoint in each of the four datasets
6. Convert dataframe to RDD and apply map to get two features ("SHOT_RESULT") and the clusters ("PREDICTION") as tuples
7. RDD.GroupBy on the cluster zones.
8. RDD. MAP, with a custom function defined to get hit rates for each zone
9. RDD.COLLECT() to get the zone with hit_rate
10. Print the zone with maximum hit rate.



```
2022-04-24 23:27:11,036 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 59.0 (TID 226) in 144 ms on 10.128.0.13 (executor 0) (2/4)
2022-04-24 23:27:11,035 INFO storage.BlockManagerInfo: Removed broadcast_102_piece0 on rb18.c.project-rb18.internal:37985 in memory (size: 6.2 KiB, free: 434.3 MiB)
2022-04-24 23:27:11,038 INFO storage.BlockManagerInfo: Removed broadcast_102_piece0 on 10.128.0.13:46219 in memory (size: 6.2 KiB, free: 431.2 MiB)
2022-04-24 23:27:11,042 INFO storage.BlockManagerInfo: Removed broadcast_98_piece0 on rb18.c.project-rb18.internal:37985 in memory (size: 6.2 KiB, free: 434.3 MiB)
2022-04-24 23:27:11,046 INFO storage.BlockManagerInfo: Removed broadcast_98_piece0 on 10.128.0.12:36751 in memory (size: 6.2 KiB, free: 432.5 MiB)
2022-04-24 23:27:11,046 INFO storage.BlockManagerInfo: Removed broadcast_98_piece0 on 10.128.0.13:46219 in memory (size: 6.2 KiB, free: 431.2 MiB)
2022-04-24 23:27:11,075 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 59.0 (TID 228) in 70 ms on 10.128.0.13 (executor 0) (3/4)
2022-04-24 23:27:11,118 INFO scheduler.TaskSetManager: Finished task 3.0 in stage 59.0 (TID 229) in 84 ms on 10.128.0.13 (executor 0) (4/4)
2022-04-24 23:27:11,119 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 59.0, whose tasks have all completed, from pool
2022-04-24 23:27:11,120 INFO scheduler.DAGScheduler: ResultStage 59 (collect at /spark-examples/test-python/lab2/task1.py:229) finished in 0.234 s
2022-04-24 23:27:11,121 INFO scheduler.DAGScheduler: Job 33 is finished. Cancelling potential speculative or zombie tasks for this job
2022-04-24 23:27:11,121 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 59: Stage finished
2022-04-24 23:27:11,121 INFO scheduler.DAGScheduler: Job 33 finished: collect at /spark-examples/test-python/lab2/task1.py:229, took 0.494293 s
Most comfortable zone for Lebron James: 3 with a hit rate of: 0.5887973516909303
Most comfortable zone for James Harden: 3 with a hit rate of: 0.5571955719557196
Most comfortable zone for Stephen Curry: 3 with a hit rate of: 0.6333333333333333
Most comfortable zone for Chris Paul: 1 with a hit rate of: 0.55
```

Fig.1 Output of the comfortable zone for each player

TASK 2

Description:

The NYC Department of Finance collects data on every parking ticket issued in NYC (10M per year!). This data is made publicly available to aid in ticket resolution and to guide policymakers.

- Given a Black vehicle parking illegally at 34510, 10030, 34050 (street codes). What is the probability that it will get a ticket? (very rough prediction)?

Idea: Conditional probability for a “black” vehicle to be computed given that it was parked on the streets (34510, 10030, 34050) would equal the ratio of total black vehicles ticketed there to total number vehicles irrespective of color ticketed.

Implementation:

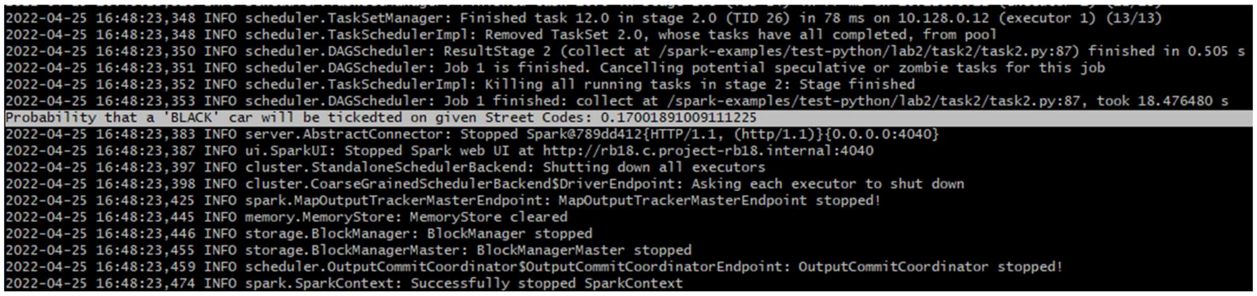
Data was loaded into a spark dataframe from the input csv file. To preprocess the data for further analysis, dataframe was filtered to get only the required features (“Street Code 1”, “Street Code 2”, “Street Code 3” and “Vehicle Color”). After getting rid of null values in the dataframe, the column type for streetcode were changed to integers, to make it easy for computation.

Broadcast Variable:

A. “codes” referencing a list of given streetcodes [34510, 10030, 34050] to make it available at each of the machines in the cluster.
B. “Black” referencing a list of all possible variations of the string “black” present in the data, for example : “BLACK”, “BLK”, “BLAC” etc

Transformations:

1. Data was filtered so that at least one of the columns of street codes (1,2,3) contained the given street codes in the broadcast variable.
2. Only column of essence left in the dataframe was “vehicle color” so the dataframe was converted to RDD.map to contained only tuples of form (vehicle color, 1)
3. RDD.map on a custom function that returns tuples of type (“black”,1) and (“non_black”,1)
4. RDD.ReduceByKey to get an rdd consisting of the number of values of black and non-black vehicles.
5. RDD.Collect() to get the two tuples in the form of a list
6. Print the probability as ratio of number of black vehicles to total number of vehicles



```
2022-04-25 16:48:23,348 INFO scheduler.TaskSetManager: Finished task 12.0 in stage 2.0 (TID 26) in 78 ms on 10.128.0.12 (executor 1) (13/13)
2022-04-25 16:48:23,348 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
2022-04-25 16:48:23,350 INFO scheduler.DAGScheduler: ResultStage 2 (collect at /spark-examples/test-python/lab2/task2/task2.py:87) finished in 0.505 s
2022-04-25 16:48:23,351 INFO scheduler.DAGScheduler: Job 1 is finished. Cancelling potential speculative or zombie tasks for this job
2022-04-25 16:48:23,352 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 2: Stage finished
2022-04-25 16:48:23,353 INFO scheduler.DAGScheduler: Job 1 finished: collect at /spark-examples/test-python/lab2/task2/task2.py:87, took 18.476480 s
Probability that a 'BLACK' car will be tickedted on given Street Codes: 0.17001891009111225
2022-04-25 16:48:23,383 INFO server.AbstractConnector: Stopped Spark@789dd412{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
2022-04-25 16:48:23,387 INFO ui.SparkUI: Stopped Spark web UI at http://rb18.c.project-rb18.internal:4040
2022-04-25 16:48:23,397 INFO cluster.StandaloneSchedulerBackend: Shutting down all executors
2022-04-25 16:48:23,398 INFO cluster.CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
2022-04-25 16:48:23,425 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2022-04-25 16:48:23,445 INFO memory.MemoryStore: MemoryStore cleared
2022-04-25 16:48:23,446 INFO storage.BlockManager: BlockManager stopped
2022-04-25 16:48:23,455 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2022-04-25 16:48:23,459 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
2022-04-25 16:48:23,474 INFO spark.SparkContext: Successfully stopped SparkContext
```

Fig.2 Output for a probability of black vehicle getting a ticket at 34510, 10030, 34050

TASK 3

Description:

The NYC Department of Finance collects data on every parking ticket issued in NYC (10M per year!). This data is made publicly available to aid in ticket resolution and to guide policymakers.

- When are tickets most likely to be issued?
- Report results with different levels of parallelism, 2, 3, 4, 5

Idea: Tickets will most likely be issued in the month and time pairs which occur most frequently in the data. We just concatenate the month and time values and report result with maximum frequency.

Implementation:

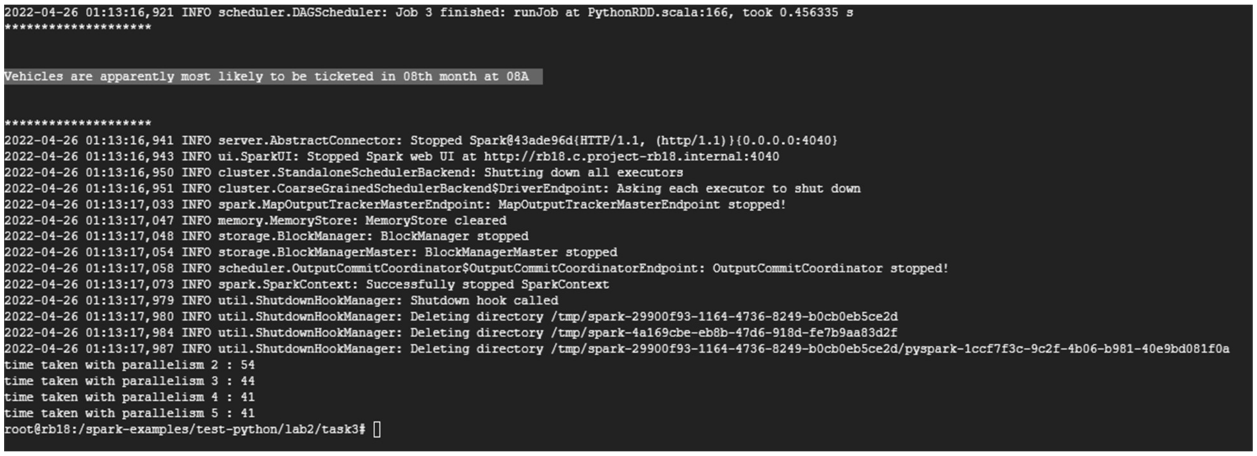
Data was loaded into a spark dataframe from the input csv file. After getting rid of null values, only columns of importance of were the “Issue Date” and the “Violation Time” and hence the data frame was filtered to contain only those two columns.

Transformations:

1. Data frame was converted to RDD and the values from two features (“Issue Date”) and (“Violation Time”) were taken as a tuple in the RDD using the map function
2. RDD.map to format the two tuple, fetch the month and time, concatenate them and return a tuple like (08–10A,1) where first two digits represent the month of the year, and last two represent the “hour” and “AM/PM”. Minutes were discarded as it is not that significant. It was all done using a custom user defined function.
3. RDD.reduceBy to get the frequencies of month-time pairs
4. RDD.sortBy(ascending=False) to sort the data in the reverse order to get the most frequent month-time pair
5. RDD.take(1) to fetch the most frequent pair and print it on the screen.

<u>Parallelization:</u>

Different levels of parallelization were added (2,3,4,5) by adding –conf spark.default.parallelism = K after spark-submit.
The test.sh file’s main script was run 4 times for each of the parallelization levels with timestamps at Beginning at the end of each run. Time cost as difference between (end time and start time) were stored in 4 different variables and reported in the end.



```
2022-04-26 01:13:16,921 INFO scheduler.DAGScheduler: Job 3 finished: runJob at PythonRDD.scala:166, took 0.456335 s
********************


Vehicles are apparently most likely to be ticketed in 08th month at 08A


********************
2022-04-26 01:13:16,941 INFO server.AbstractConnector: Stopped Spark@43ade96d{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
2022-04-26 01:13:16,943 INFO ui.SparkUI: Stopped Spark web UI at http://rb18.c.project-rb18.internal:4040
2022-04-26 01:13:16,950 INFO cluster.StandaloneSchedulerBackend: Shutting down all executors
2022-04-26 01:13:16,951 INFO cluster.CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
2022-04-26 01:13:17,033 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2022-04-26 01:13:17,047 INFO memory.MemoryStore: MemoryStore cleared
2022-04-26 01:13:17,048 INFO storage.BlockManager: BlockManager stopped
2022-04-26 01:13:17,054 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2022-04-26 01:13:17,058 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
2022-04-26 01:13:17,073 INFO spark.SparkContext: Successfully stopped SparkContext
2022-04-26 01:13:17,979 INFO util.ShutdownHookManager: Shutdown hook called
2022-04-26 01:13:17,980 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-29900f93-1164-4736-8249-b0cb0eb5ce2d
2022-04-26 01:13:17,984 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-4a169cbe-eb8b-47d6-918d-fe7b9aa83d2f
2022-04-26 01:13:17,987 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-29900f93-1164-4736-8249-b0cb0eb5ce2d/pyspark-1ccf7f3c-9c2f-4b06-b981-40e9bd081f0a
time taken with parallelism 2 : 54
time taken with parallelism 3 : 44
time taken with parallelism 4 : 41
time taken with parallelism 5 : 41
root@rb18:/spark-examples/test-python/lab2/task3# 
```

Fig.3 Output of the highest frequency of tickets issued