DS/CMPSC 410 MiniProject #1

Spring 2021

Instructor: John Yen

TA: Dongkuan Xu, Rupesh Prajapati

Learning Objectives

- Be able to identify frequent 2 port sets and 3 port sets that are scanned by scanners in the Darknet dataset
- Be able to improve the frequent port set mining algorithm by adding suitable filtering
- Be able to improve the performance of frequent port set mining by suitable reuse of RDD, together with appropriate persist and unpersist on the reused RDD.

Total points: 100

- Exercise 1: 10 points
- Exercise 2: 10 points
- Exercise 3: 20 points
- Exercise 4: 20 points
- Exercise 5: 10 points
- Exercise 6: 30 points (run spark-submit on a large Dataset)

Submit the following items for this mini project deliverable:

- Completed Jupyter Notebook (including answers to Exercise 9.1 to 9.5; in HTML or PDF format)
- The python file (.py) used for spark-submit
- The output file that contains counts of 2-port sets and 3-port sets.
- The log file of spark-submit that shows the CPU time for completing the spark-submit job.

Due: midnight, April 2, 2021

In [1]:

```
import pyspark
import csv
import pandas as pd
```

In [2]:

```
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField, StructType, StringType, LongType
from pyspark.sql.functions import col, column
from pyspark.sql.functions import expr
from pyspark.sql.functions import split
from pyspark.sql import Row
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler, IndexToString
from pyspark.ml.clustering import KMeans
```

In [3]:
```
ss = SparkSession.builder.master("local").appName("Lab9 FrequentPortSets").getOrCreate()
```

Exercise 9.1 (10 points)

- Complete the path below for reading "sampled_profile.csv" you downloaded from Canvas, uploaded to Lab9 folder. (5 points)
- Fill in your Name (5 points):

In [4]:
```
Scanners_df = ss.read.csv("/storage/home/kky5082/ds410/Lab9/sampled_profile.csv", header= True, inferSchema=True )
```

We can use printSchema() to display the schema of the DataFrame Scanners_df to see whether it was inferred correctly.

In [5]:
```
Scanners_df.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- id: integer (nullable = true)
 |-- numports: integer (nullable = true)
 |-- lifetime: double (nullable = true)
 |-- Bytes: integer (nullable = true)
 |-- Packets: integer (nullable = true)
 |-- average_packetsize: integer (nullable = true)
 |-- MinUniqueDests: integer (nullable = true)
 |-- MaxUniqueDests: integer (nullable = true)
 |-- MinUniqueDest24s: integer (nullable = true)
 |-- MaxUniqueDest24s: integer (nullable = true)
 |-- average_lifetime: double (nullable = true)
 |-- mirai: boolean (nullable = true)
 |-- zmap: boolean (nullable = true)
 |-- masscan: boolean (nullable = true)
 |-- country: string (nullable = true)
 |-- traffic_types_scanned_str: string (nullable = true)
 |-- ports_scanned_str: string (nullable = true)
 |-- host_tags_per_censys: string (nullable = true)
 |-- host_services_per_censys: string (nullable = true)
```

Part A Transfosrm the feature "ports_scanned_str" into an array of ports.

The original value of the column is a string that connects all the ports scanned by a scanner. The different ports that are open by a scanner are connected by dash "-". For example, "81-161-2000" indicates the scanner has scanned three ports: port 81, port 161, and port 2000. Therefore, we want to use split to separate it into an array of ports by each scanner. This transformation is important because it enables the identification of frequent ports scanned by scanners.

The original value of the column "ports_scanned_str"

In [6]:
```
Scanners_df.select("ports_scanned_str").show(30)
```

```
+--------------------+
|   ports_scanned_str|
+--------------------+
|               13716|
|         17128-17136|
|               35134|
|               17140|
|               54594|
|               17130|
|               54594|
|               37876|
|               17142|
|17128-17130-17132...|
|               54594|
|               12941|
|               30188|
|23-80-81-1023-232...|
|               54594|
|17128-17132-17136...|
|               17136|
|               54594|
|               17134|
|                 445|
|               34226|
|               17130|
|               17134|
|           137-17130|
|               17142|
|               17142|
|17128-17130-17132...|
|                  23|
|               54594|
|               54594|
+--------------------+
only showing top 30 rows
```

Convert the Column 'ports_scanned_str' into an Array of ports scanned by each scanner (row)

In [7]:

```
Scanners_df2=Scanners_df.withColumn("Ports_Array", split(col("ports_scanned_str"), "-") )
Scanners_df2.persist().show(10)
```

```
+-------+-------+--------+--------+-----+-------+------------------+-----------
---+--------------+----------------+----------------+----------------+-----+---
--+-------+-------+-------------------------+--------------------+-------------
-------+------------------------+--------------------+
|    _c0|     id|numports|lifetime|Bytes|Packets|average_packetsize|MinUniqueDe
sts|MaxUniqueDests|MinUniqueDest24s|MaxUniqueDest24s|average_lifetime|mirai| zm
ap|masscan|country|traffic_types_scanned_str|   ports_scanned_str|host_tags_per
_censys|host_services_per_censys|         Ports_Array|
+-------+-------+--------+--------+-----+-------+------------------+-----------
---+--------------+----------------+----------------+----------------+-----+---
--+-------+-------+-------------------------+--------------------+-------------
-------+------------------------+--------------------+
|1645181|1645181|       1|     0.0|   60|      1|                60|
1|             1|               1|               1|             0.0|false|false
|  false|     BR|                       15|               13716|
null|                    null|             [13716]|
|2091467|2091467|       2|  199.84|  752|     12|                62|
1|             1|               1|               1|           66.61|false|false
|  false|     CN|                    11-16|         17128-17136|
null|                    null|      [17128, 17136]|
| 888618| 888618|       1|     0.0|   60|      1|                60|
1|             1|               1|               1|             0.0|false|false
|  false|     US|                       15|               35134|
null|                    null|             [35134]|
|1512937|1512937|       1|  793.37| 1561|     23|                67|
1|             1|               1|               1|          396.69|false|false
|  false|     JP|                    11-16|               17140|
null|                    null|             [17140]|
| 654939| 654939|       1|   48.69|  571|      3|               190|
1|             2|               1|               2|           24.34|false|false
|  false|     TR|                       16|               54594|
null|                    8081|             [54594]|
|  73109|  73109|       1| 1056.83| 1924|     26|                74|
1|             1|               1|               1|         1056.83|false|false
|  false|     CN|                       11|               17130|
null|                    null|             [17130]|
| 923577| 923577|       1|  348.42|  465|      3|               155|
1|             2|               1|               2|          174.21|false|false
|  false|     JO|                       16|               54594|
null|                    null|             [54594]|
|1349341|1349341|       1|     0.0|   60|      1|                60|
1|             1|               1|               1|             0.0|false|false
|  false|     US|                       15|               37876|
null|                    null|             [37876]|
|1959916|1959916|       1|  814.15| 1631|     24|                67|
1|             1|               1|               1|          407.07|false|false
|  false|     TW|                    11-16|               17142|
null|                    null|             [17142]|
| 565394| 565394|       7| 2505.48| 5422|     71|                76|
1|             3|               1|               3|          119.31|false|false
|  false|     CN|                    11-16|17128-17130-17132...|
null|                    null|[17128, 17130, 17...|
+-------+-------+--------+--------+-----+-------+------------------+-----------
---+--------------+----------------+----------------+----------------+-----+---
--+-------+-------+-------------------------+--------------------+-------------
-------+------------------------+--------------------+
only showing top 10 rows
```

For Mining Frequent Port Sets being scanned, we only need the column `Ports_Array`

In [8]:
```
Ports_Scanned_RDD = Scanners_df2.select("Ports_Array").rdd
```

In [9]:
```
Ports_Scanned_RDD.persist().take(5)
```

Out[9]:
```
[Row(Ports_Array=['13716']),
 Row(Ports_Array=['17128', '17136']),
 Row(Ports_Array=['35134']),
 Row(Ports_Array=['17140']),
 Row(Ports_Array=['54594'])]
```

Because each port number in the Ports_Array column for each row occurs only once, we can count the total occurance of each port number through flatMap.

In [10]:
```
Ports_list_RDD = Ports_Scanned_RDD.map(lambda row: row[0] )
```

In [11]:
```
Ports_list_RDD.persist()
```

Out[11]:
```
PythonRDD[27] at RDD at PythonRDD.scala:53
```

In [12]:
```
Ports_list2_RDD = Ports_Scanned_RDD.flatMap(lambda row: row[0] )
```

In [13]:
```
Port_count_RDD = Ports_list2_RDD.map(lambda x: (x, 1))
Port_count_RDD.take(2)
```

Out[13]:
```
[('13716', 1), ('17128', 1)]
```

In [14]:
```
Port_count_total_RDD = Port_count_RDD.reduceByKey(lambda x,y: x+y, 1)
Port_count_total_RDD.persist().take(5)
```

Out[14]:
```
[('13716', 14),
 ('17128', 31850),
 ('17136', 31617),
 ('35134', 13),
 ('17140', 31865)]
```

In [15]:
```
Port_count_total_RDD.count()
```

Out[15]:
```
65536
```

In [16]:
```
Sorted_Count_Port_RDD = Port_count_total_RDD.map(lambda x: (x[1], x[0])).sortBy
Key( ascending = False)
```

```
In [17]: Sorted_Count_Port_RDD.persist().take(50)
```

```
Out[17]: [(32014, '17132'),
          (31865, '17140'),
          (31850, '17128'),
          (31805, '17138'),
          (31630, '17130'),
          (31617, '17136'),
          (29199, '23'),
          (25466, '445'),
          (25216, '54594'),
          (21700, '17142'),
          (21560, '17134'),
          (15010, '80'),
          (13698, '8080'),
          (8778, '0'),
          (6265, '2323'),
          (5552, '5555'),
          (4930, '81'),
          (4103, '1023'),
          (4058, '52869'),
          (4012, '8443'),
          (3954, '49152'),
          (3885, '7574'),
          (3874, '37215'),
          (3318, '34218'),
          (3279, '34220'),
          (3258, '33968'),
          (3257, '34224'),
          (3253, '34228'),
          (3252, '33962'),
          (3236, '33960'),
          (3209, '33964'),
          (3179, '34216'),
          (3167, '34226'),
          (3155, '33970'),
          (3130, '33972'),
          (2428, '50401'),
          (1954, '34222'),
          (1921, '34230'),
          (1919, '33966'),
          (1819, '33974'),
          (1225, '3389'),
          (1064, '1433'),
          (885, '22'),
          (878, '5353'),
          (604, '21'),
          (594, '8291'),
          (554, '8728'),
          (512, '443'),
          (382, '5900'),
          (330, '8000')]
```

The value of the threshold below should be identical to your choice of threshold for Exercise 9.3

In [18]:

```
threshold = 1000
Filtered_Sorted_Count_Port_RDD= Sorted_Count_Port_RDD.filter(lambda x: x[0] > threshold)
Filtered_Sorted_Count_Port_RDD.persist().count()
```

Out[18]:

```
42
```

In [19]:

```
Top_Ports = Filtered_Sorted_Count_Port_RDD.map(lambda x: x[1]).collect()
```

In [20]:

```
Top_1_Port_count = len(Top_Ports)
```

In [21]:

```
print(Top_1_Port_count)
```

```
42
```

Exercise 9.2 (10 points)

Compute the total number of scanners in Ports_list_RDD with the total number of scanners that scan more than one port. What is the impact of this filter on the size of the RDD? Complete the following code to find out the answers. Then, fill the answer in the cell marked as Answer to Exercise 9.2.

In [22]:

```
# Filter out those scanners that scan only 1 port
multi_Ports_list_RDD = Ports_list_RDD.filter(lambda x: len(x)>1 )
```

In [23]:

```
Ports_list_RDD.count()
```

Out[23]:

```
227062
```

In [24]:

```
Ports_list_RDD.take(5)
```

Out[24]:

```
[['13716'], ['17128', '17136'], ['35134'], ['17140'], ['54594']]
```

In [25]:

```
multi_Ports_list_RDD.count()
```

Out[25]:

```
73663
```

Answer to Exercise 9.2

- Original number of scanners: ## 227062
- Number of scanners that scan more than one port: ## 73663
- Impact of the filtering on the size of filtered scanners: ## The size flitered scanners is 1/3 of the original scanners

Exercise 9.3 (20 points)

- Choose a threshold (suggest a number between 500 and 1000) (5 points)
- Complete the following code for finding 2 port sets (7 points)
- Add suitable persist and unpersist to suitable RDD (8 points)

In [26]:
```
multi_Ports_list_RDD.take(2)
```

Out[26]:
```
[['17128', '17136'],
 ['17128', '17130', '17132', '17134', '17136', '17138', '17140']]
```

In [29]:
```
# Initialize a Pandas DataFrame to store frequent port sets and their counts
Freq_Port_Sets_df = pd.DataFrame( columns= ['Port Sets', 'count'])
# Initialize the index to the Freq_Port_Sets_df to 0
index = 0
# Set the threshold for Large Port Sets to be 100
threshold = 1000
multi_Ports_list_RDD.persist()
for i in range(0, Top_1_Port_count-1):
    Scanners_port_i_RDD = multi_Ports_list_RDD.filter(lambda x: Top_Ports[i] in
x)
    Scanners_port_i_RDD.persist()
    for j in range(i+1, Top_1_Port_count-1):
        Scanners_port_i_j_RDD = Scanners_port_i_RDD.filter(lambda x:  Top_Ports
[j] in x)
        two_ports_count = Scanners_port_i_j_RDD.count()
        if two_ports_count > threshold:
            Freq_Port_Sets_df.loc[index]=[ [Top_Ports[i], Top_Ports[j]], two_po
rts_count]
            index = index +1
    Scanners_port_i_RDD.unpersist()
```

In [30]:
```
print(Freq_Port_Sets_df)
```

```
          Port Sets  count
0    [17132, 17140]  16317
1    [17132, 17128]  16279
2    [17132, 17138]  16299
3    [17132, 17130]  16336
4    [17132, 17136]  16148
..              ...    ...
259  [33960, 33964]   1005
260  [33960, 34226]   1034
261  [33964, 34226]   1004
262  [34216, 34226]   1015
263  [34226, 33972]   1023

[264 rows x 2 columns]
```

In [31]:
```
tri_Ports_list_RDD=multi_Ports_list_RDD.filter(lambda x: len(x)>2)
```

In [32]:
```
tri_Ports_list_RDD.count()
```

Out[32]:
```
49549
```

In [33]:
```
index
```

Out[33]:
```
264
```

Exercise 9.5 (20 points)

- Use the same threshold as Exercise 9.4 (5 points)
- Complete the following code to find frequent 3 port sets (7 points)
- Add persist and unpersist to suitable RDD (8 points)

In [36]:

```
# Set the threshold for Large Port Sets to be 100
threshold = 1000
tri_Ports_list_RDD.persist()
for i in range(0, Top_1_Port_count-1):
    Scanners_port_i_RDD = tri_Ports_list_RDD.filter(lambda x: Top_Ports[i] in x
)
    Scanners_port_i_RDD.persist()
    for j in range(i+1, Top_1_Port_count-1):
        Scanners_port_i_j_RDD = Scanners_port_i_RDD.filter(lambda x: Top_Ports[
j] in x)
        two_ports_count = Scanners_port_i_j_RDD.count()
        Scanners_port_i_j_RDD.persist()
        if two_ports_count > threshold:
            Scanners_port_i_RDD.unpersist()
            for k in range(j+1, Top_1_Port_count -1):
                Scanners_port_i_j_k_RDD = Scanners_port_i_j_RDD.filter(lambda x
: Top_Ports[k] in x)
                Scanners_port_i_j_RDD.unpersist()
                three_ports_count = Scanners_port_i_j_k_RDD.count()
                if three_ports_count > threshold:
                    Freq_Port_Sets_df.loc[index] = [ [Top_Ports[i], Top_Ports[j
], Top_Ports[k]], three_ports_count]
                    index = index + 1
                    # print("Ports: ", Top_Ports[i], ", ", Top_Ports[j], ",  ",
Top_Ports[k], ": Count ", three_ports_count)
```

In [41]:

```
Freq_Port_Sets_DF = ss.createDataFrame(Freq_Port_Sets_df)
```

Exercise 9.5 (10 points)

Complete the following code to save your frequent 2 port sets and 3 port sets in an output file.

In [44]:

```
output_path = "/storage/home/kky5082/ds410/Lab9/output"
Freq_Port_Sets_DF.rdd.saveAsTextFile(output_path)
```

Exercise 9.6 (30 points)

- Remove .master("local") from SparkSession statement
- Change the input file to "/gpfs/scratch/juy1/Day_2020_profile.csv"
- Change the output file to a different directory from the one you used in Exercise 9.4
- Export the notebook as a .py file
- Run spark-submit on ICDS Roar (following instructions on Canvas)