01_spark

November 10, 2020

0.1 Migrating from Spark to BigQuery via Dataproc – Part 1

- Part 1: The original Spark code, now running on Dataproc (lift-and-shift).
- Part 2: Replace HDFS by Google Cloud Storage. This enables job-specific-clusters. (cloud-native)
- Part 3: Automate everything, so that we can run in a job-specific cluster. (cloud-optimized)
- Part 4: Load CSV into BigQuery, use BigQuery. (modernize)
- Part 5: Using Cloud Functions, launch analysis every time there is a new file in the bucket. (serverless)

0.1.1 Copy data to HDFS

The Spark code in this notebook is based loosely on the code accompanying this post by Dipanjan Sarkar. I am using it to illustrate migrating a Spark analytics workload to BigQuery via Dataproc.

The data itself comes from the 1999 KDD competition. Let's grab 10% of the data to use as an illustration.

[1]:
```
!wget http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz
```

```
--2020-11-10 15:42:13--
http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz
Resolving kdd.ics.uci.edu (kdd.ics.uci.edu)… 128.195.1.86
Connecting to kdd.ics.uci.edu (kdd.ics.uci.edu)|128.195.1.86|:80… connected.
HTTP request sent, awaiting response… 200 OK
Length: 2144903 (2.0M) [application/x-gzip]
Saving to: ‘kddcup.data_10_percent.gz’

kddcup.data_10_perc 100%[===================>]   2.04M  4.47MB/s    in 0.5s

2020-11-10 15:42:14 (4.47 MB/s) - ‘kddcup.data_10_percent.gz’ saved
[2144903/2144903]
```

[2]:
```
!hadoop fs -put kddcup* /
```

[3]:
```
!hadoop fs -ls /
```

```
Found 3 items
-rw-r--r--   2 root hadoop    2144903 2020-11-10 15:42
```

```
/kddcup.data_10_percent.gz
drwxrwxrwt   - hdfs hadoop          0 2020-11-10 15:38 /tmp
drwxrwxrwt   - hdfs hadoop          0 2020-11-10 15:37 /user
```

0.1.2 Reading in data

The data are CSV files. In Spark, these can be read using textFile and splitting rows on commas.

[4]:
```
from pyspark.sql import SparkSession, SQLContext, Row

spark = SparkSession.builder.appName("kdd").getOrCreate()
sc = spark.sparkContext
data_file = "hdfs:///kddcup.data_10_percent.gz"
raw_rdd = sc.textFile(data_file).cache()
raw_rdd.take(5)
```

[4]:
```
['0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00
,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,
1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00
,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00
,1.00,0.00,0.00,39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00
,1.00,0.00,0.00,49,49,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.']
```

[5]:
```
csv_rdd = raw_rdd.map(lambda row: row.split(","))
parsed_rdd = csv_rdd.map(lambda r: Row(
    duration=int(r[0]),
    protocol_type=r[1],
    service=r[2],
    flag=r[3],
    src_bytes=int(r[4]),
    dst_bytes=int(r[5]),
    wrong_fragment=int(r[7]),
    urgent=int(r[8]),
    hot=int(r[9]),
    num_failed_logins=int(r[10]),
    num_compromised=int(r[12]),
    su_attempted=r[14],
    num_root=int(r[15]),
    num_file_creations=int(r[16]),
    label=r[-1]
    )
)
parsed_rdd.take(5)
```

[5]:
```
[Row(dst_bytes=5450, duration=0, flag='SF', hot=0, label='normal.',
num_compromised=0, num_failed_logins=0, num_file_creations=0, num_root=0,
protocol_type='tcp', service='http', src_bytes=181, su_attempted='0', urgent=0,
wrong_fragment=0),
 Row(dst_bytes=486, duration=0, flag='SF', hot=0, label='normal.',
num_compromised=0, num_failed_logins=0, num_file_creations=0, num_root=0,
protocol_type='tcp', service='http', src_bytes=239, su_attempted='0', urgent=0,
wrong_fragment=0),
 Row(dst_bytes=1337, duration=0, flag='SF', hot=0, label='normal.',
num_compromised=0, num_failed_logins=0, num_file_creations=0, num_root=0,
protocol_type='tcp', service='http', src_bytes=235, su_attempted='0', urgent=0,
wrong_fragment=0),
 Row(dst_bytes=1337, duration=0, flag='SF', hot=0, label='normal.',
num_compromised=0, num_failed_logins=0, num_file_creations=0, num_root=0,
protocol_type='tcp', service='http', src_bytes=219, su_attempted='0', urgent=0,
wrong_fragment=0),
 Row(dst_bytes=2032, duration=0, flag='SF', hot=0, label='normal.',
num_compromised=0, num_failed_logins=0, num_file_creations=0, num_root=0,
protocol_type='tcp', service='http', src_bytes=217, su_attempted='0', urgent=0,
wrong_fragment=0)]
```

0.1.3 Spark analysis

One way to analyze data in Spark is to call methods on a dataframe.

[6]:
```
sqlContext = SQLContext(sc)
df = sqlContext.createDataFrame(parsed_rdd)
connections_by_protocol = df.groupBy('protocol_type').count().orderBy('count',␣
 ↪ascending=False)
connections_by_protocol.show()
```

```
+-------------+------+
|protocol_type| count|
+-------------+------+
|         icmp|283602|
|          tcp|190065|
|          udp| 20354|
+-------------+------+
```

Another way is to use Spark SQL

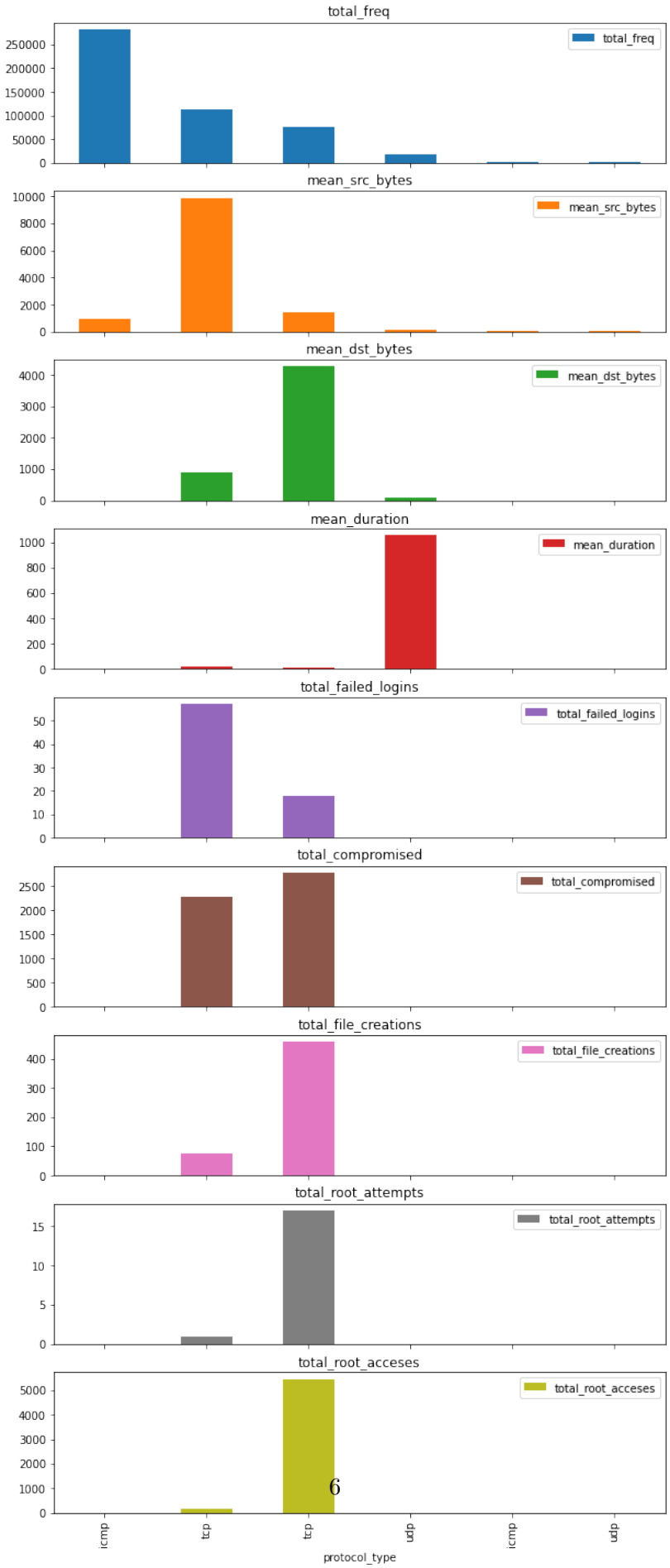
[7]:
```
df.registerTempTable("connections")
attack_stats = sqlContext.sql("""
                            SELECT
                              protocol_type,
                              CASE label
                                WHEN 'normal.' THEN 'no attack'
```

```
                              ELSE 'attack'
                            END AS state,
                            COUNT(*) as total_freq,
                            ROUND(AVG(src_bytes), 2) as mean_src_bytes,
                            ROUND(AVG(dst_bytes), 2) as mean_dst_bytes,
                            ROUND(AVG(duration), 2) as mean_duration,
                            SUM(num_failed_logins) as total_failed_logins,
                            SUM(num_compromised) as total_compromised,
                            SUM(num_file_creations) as total_file_creations,
                            SUM(su_attempted) as total_root_attempts,
                            SUM(num_root) as total_root_acceses
                          FROM connections
                          GROUP BY protocol_type, state
                          ORDER BY 3 DESC
                          """)
attack_stats.show()
```

```
+-------------+---------+----------+--------------+--------------+-------------+
-------------------+-----------------+--------------------+-------------------+-
-----------------+
|protocol_type|    state|total_freq|mean_src_bytes|mean_dst_bytes|mean_duration|
total_failed_logins|total_compromised|total_file_creations|total_root_attempts|t
otal_root_acceses|
+-------------+---------+----------+--------------+--------------+-------------+
-------------------+-----------------+--------------------+-------------------+-
-----------------+
|         icmp|   attack|    282314|        932.14|           0.0|          0.0|
0|                0|                   0|                0.0|                  0|
|          tcp|   attack|    113252|       9880.38|         881.41|        23.19|
57|             2269|                  76|                1.0|
152|
|          tcp|no attack|     76813|       1439.31|        4263.97|        11.08|
18|             2776|                 459|               17.0|
5456|
|          udp|no attack|     19177|         98.01|          89.89|      1054.63|
0|                0|                   0|                0.0|                  0|
|         icmp|no attack|      1288|         91.47|           0.0|          0.0|
0|                0|                   0|                0.0|                  0|
|          udp|   attack|      1177|          27.5|          0.23|          0.0|
0|                0|                   0|                0.0|                  0|
+-------------+---------+----------+--------------+--------------+-------------+
-------------------+-----------------+--------------------+-------------------+-
-----------------+
```

[8]:
```
%matplotlib inline
```

```
ax = attack_stats.toPandas().plot.bar(x='protocol_type', subplots=True,␣
↪figsize=(10,25))
```


total_freq
total_freq
250000
200000
150000
100000
50000
0
mean_src_bytes
mean_src_bytes
10000
8000
6000
4000
2000
0
mean_dst_bytes
mean_dst_bytes
4000
3000
2000
1000
0
mean_duration
mean_duration
1000
800
600
400
200
0
total_failed_logins
total_failed_logins
50
40
30
20
10
0
total_compromised
total_compromised
2500
2000
1500
1000
500
0
total_file_creations
total_file_creations
400
300
200
100
0
total_root_attempts
total_root_attempts
15
10
5
0
total_root_acceses
total_root_acceses
5000
4000
3000
2000
1000
0
icmp
tcp
tcp
udp
icmp
udp
protocol_type