Working with a real-life problem

Understanding more transformation and actions

Prerequisite:

You need to run your HDFS and Spark Cluster at minimum with Pyspark functional.

Login your namenode/master as hadoop user and run pyspark. Open a file named a.txt and save the file.

 # vim /home/hadoop/a.txt

This is our second file what we are going to put into our HDFS

This will be just starting

Python file operation is interesting and we are going to start soon

 # pyspark

Word Count

```
>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

>>> counts.collect()

[('soon', 1), ('start', 1), ('HDFS', 1), ('just', 1), ('our', 2), ('Python', 1), ('going', 2), ('operation', 1), ('is', 2), ('file', 2), ('will', 1), ('what', 1), ('are', 2), ('to', 2), ('be', 1), ('into', 1), ('interesting', 1), ('we', 2), ('and', 1), ('This', 2), ('second', 1), ('put', 1), ('starting', 1)]
```

Understanding FlatMap Function

```
>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" "))

>>> counts.collect()

['This', 'is', 'our', 'second', 'file', 'what', 'we', 'are', 'going', 'to', 'put', 'into', 'our', 'HDFS', 'This', 'will', 'be', 'just', 'starting', 'Python', 'file', 'operation', 'is', 'interesting', 'and', 'we', 'are', 'going', 'to', 'start', 'soon']
```

Understanding Map Function

```
>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1))

>>> counts.collect()
```

[('This', 1), ('is', 1), ('our', 1), ('second', 1), ('file', 1), ('what', 1), ('we', 1), ('are', 1), ('going', 1), ('to', 1), ('put', 1), ('into', 1), ('our', 1), ('HDFS', 1), ('This', 1), ('will', 1), ('be', 1), ('just', 1), ('starting', 1), ('Python', 1), ('file', 1), ('operation', 1), ('is', 1), ('interesting', 1), ('and', 1), ('we', 1), ('are', 1), ('going', 1), ('to', 1), ('start', 1), ('soon', 1)]

Understanding ReduceByKey

>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

>>> counts.collect()

[('soon', 1), ('start', 1), ('HDFS', 1), ('just', 1), ('our', 2), ('Python', 1), ('going', 2), ('operation', 1), ('is', 2), ('file', 2), ('will', 1), ('what', 1), ('are', 2), ('to', 2), ('be', 1), ('into', 1), ('interesting', 1), ('we', 2), ('and', 1), ('This', 2), ('second', 1), ('put', 1), ('starting', 1)]

Let us simulate some different problem statements

Number of words per first letter wise

>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word[0], word)).groupByKey().mapValues(lambda word: len(word))

>>> counts.collect()

[('o', 3), ('t', 2), ('T', 2), ('P', 1), ('j', 1), ('g', 2), ('H', 1), ('w', 4), ('i', 4), ('s', 4), ('p', 1), ('b', 1), ('f', 2), ('a', 3)]

What happened if we do reduceByKey?

>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word[0], word)).reduceByKey(lambda a, b: a + b)

>>> counts.collect()

[('o', 'ourouroperation'), ('t', 'toto'), ('T', 'ThisThis'), ('P', 'Python'), ('j', 'just'), ('g', 'goinggoing'), ('H', 'HDFS'), ('w', 'whatwewillwe'), ('i', 'isintoisinteresting'), ('s', 'startsoonsecondstarting'), ('p', 'put'), ('b', 'be'), ('f', 'filefile'), ('a', 'andareare')]

Map Function

>>> counts=sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word[0], word))

>>> counts.collect()

[('T', 'This'), ('i', 'is'), ('o', 'our'), ('s', 'second'), ('f', 'file'), ('w', 'what'), ('w', 'we'), ('a', 'are'), ('g', 'going'), ('t', 'to'), ('p', 'put'), ('i', 'into'), ('o', 'our'), ('H', 'HDFS'), ('T', 'This'), ('w', 'will'), ('b', 'be'), ('j', 'just'), ('s', 'starting'), ('P', 'Python'), ('f', 'file'), ('o', 'operation'), ('i', 'is'), ('i',

'interesting'), ('a', 'and'), ('w', 'we'), ('a', 'are'), ('g', 'going'), ('t', 'to'), ('s', 'start'), ('s', 'soon')]]

MapValues Function

>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word[0], word)).groupByKey().mapValues(lambda word: len(word))

>>> counts.collect()

[('o', 3), ('t', 2), ('T', 2), ('P', 1), ('j', 1), ('g', 2), ('H', 1), ('w', 4), ('i', 4), ('s', 4), ('p', 1), ('b', 1), ('f', 2), ('a', 3)]

Number of unique words per first letter wise

>>> counts = sc.textFile("/home/hadoop/a.txt").flatMap(lambda line: line.split(" ")).map(lambda word: (word[0], word)).groupByKey().mapValues(lambda word: len(set(word)))

>>> counts.collect()

[('o', 2), ('t', 1), ('T', 1), ('P', 1), ('j', 1), ('g', 1), ('H', 1), ('w', 3), ('i', 3), ('s', 4), ('p', 1), ('b', 1), ('f', 1), ('a', 2)]

Number of unique words per first letter wise – Efficient Method

*>>> counts = sc.textFile("/home/hadoop/a.txt").repartition(6).flatMap(lambda line: line.split(" ")).**distinct()**.map(lambda word: (word[0], word)).groupByKey().mapValues(lambda word: len(word))*

>>> counts.collect()

[('o', 2), ('g', 1), ('i', 3), ('H', 1), ('b', 1), ('t', 1), ('w', 3), ('T', 1), ('P', 1), ('j', 1), ('p', 1), ('f', 1), ('s', 4), ('a', 2)]

Working with a real life problem

Prerequisite:

You need to have run HDFS and Spark Cluster with Pyspark workable.

Problem Description:

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment but peppered it with multiple attacks.

All the raw responses are kept in file named **corrected**

Detail schema is given in **Columns.xlsx**

The following analytics shall be done:

1. Select tcp network interactions with more than 1 second duration and no transfer from destination
   Note: The output column can be "duration | dst_bytes"

2. Find protocol type wise network interaction count
   Note: The output column can be "protocol_type | count"

3. Count how many interactions last more than 1 second, with no data transfer from destination, grouped by protocol type. Output can be "protocol_type | count"

4. Count them by label and protocol type. Output can be "label | protocol_type | count"

Note that: If label is not normal then it is considered as attack

Instruction to be followed:

1. Raw Capture file shall be loaded in HDFS /RAW/ directory daily from external program

2. Process all the outputs in Apache Spark in standalone cluster mode

3. Store all the results in HDFS /RAW/ directory in CSV format

4. Schedule the spark job to run every day at 03:00PM

Solution:

netconnectionanalyzer.py

```
from pyspark import SparkConf, SparkContext, SQLContext
import sys
from pyspark.sql.functions import *
from pyspark.sql import column
from pyspark.sql.functions import col
from hdfs import Config
import sys
```

```
import os, math
import subprocess
from pyspark.sql.functions import lit
from pyspark.sql import Row

## Constants
APP_NAME = "Net Connection Analyzer"
HDFS_RAWFILE_DIR = "/RAW/"
HDFS_BASE_URL = "hdfs://bdrenfdludcf01:9000"



def get_label_type(label):
   if label != "normal.":
      return "attack"
   else:
      return "normal"



if __name__ == "__main__":

   # Folder creation for placing allt the spark data
   cmd_a = "mkdir -p " + "/tmp/SPARK_PROCESS/"
   os.system(cmd_a)

   # Configure Spark
   conf = SparkConf().setAppName(APP_NAME).set("spark.local.dir",
"/tmp/SPARK_PROCESS/")
   #https://spark.apache.org/docs/latest/configuration.html
   sc = SparkContext(conf=conf)
```

```
sqlContext = SQLContext(sc)
client = Config().get_client('bdrenhdfs')
files = client.list(HDFS_RAWFILE_DIR)
totalfilecount = len(files)

if totalfilecount == 0:
    print("There is no files to be processed, application exiting...")
    sys.exit(0)

filecount = 0

for filename in files:
    print(filename)
    if filename.find("corrected") >= 0:
        filecount = filecount + 1
        # df_trips = sqlContext.read.format("csv").option("header",
'true').load(HDFS_BASE_URL + HDFS_RAWFILE_DIR + filename)
        raw_data = sc.textFile(HDFS_BASE_URL + HDFS_RAWFILE_DIR + filename)

if filecount == 0:
    print("There is no files to be processed by Spark, application exiting...")
    system.exit(0)

csv_data = raw_data.map(lambda l: l.split(","))
row_data = csv_data.map(lambda p: Row(
    duration=int(p[0]),
    protocol_type=p[1],
    service=p[2],
    flag=p[3],
```

```
    src_bytes=int(p[4]),
    dst_bytes=int(p[5])
  ))
  interactions_df = sqlContext.createDataFrame(row_data)
  interactions_df.registerTempTable("interactions")
  # Select tcp network interactions with more than 1 second duration and no transfer from destination
  tcp_interactions = sqlContext.sql(
    "SELECT duration, dst_bytes FROM interactions WHERE protocol_type = 'tcp' AND duration > 1000 AND dst_bytes = 0")
  tcp_interactions.show()
  tcp_interactions.coalesce(1).write.mode("overwrite").csv(HDFS_BASE_URL + HDFS_RAWFILE_DIR + "analysis_1.csv")

  interaction_df_2 = interactions_df.select("protocol_type", "duration", "dst_bytes").groupBy("protocol_type").count()
  interaction_df_2.show()
  interaction_df_2.coalesce(1).write.mode("overwrite").csv(HDFS_BASE_URL + HDFS_RAWFILE_DIR + "analysis_2.csv")

  interaction_df_3 = interactions_df.select("protocol_type", "duration", "dst_bytes").filter(
    interactions_df.duration > 1000).filter(interactions_df.dst_bytes == 0).groupBy("protocol_type").count()
  interaction_df_3.show()
  interaction_df_3.coalesce(1).write.mode("overwrite").csv(HDFS_BASE_URL + HDFS_RAWFILE_DIR + "analysis_3.csv")

  row_labeled_data = csv_data.map(lambda p: Row(
  duration=int(p[0]),
  protocol_type=p[1],
```

```
   service=p[2],
   flag=p[3],
   src_bytes=int(p[4]),
   dst_bytes=int(p[5]),
   label=get_label_type(p[41])
   ))
   interactions_labeled_df = sqlContext.createDataFrame(row_labeled_data)
   interaction_df_4 = interactions_labeled_df.select("label").groupBy("label").count()
   interaction_df_4.show()
   interaction_df_4.coalesce(1).write.mode("overwrite").csv(HDFS_BASE_URL +
HDFS_RAWFILE_DIR + "analysis_4.csv")
   #Better to call shutdown hook to release all resources properly.
   sc.stop()
```

Shell Script which will capture logs and hosts our spark python program in schedule.

autoschedule.sh

```
#!/bin/bash
. /home/hadoop/.bashrc
changedname=`date '+%Y%m%d%H%M%S'`
cd /tmp/
export filename=`echo 'networkanalyzer_'$changedname $$'.log'`
((
#python /home/hadoop/development/<anypthoncodeyouneedtorun.py>
/usr/local/spark/bin/spark-submit  --master spark://bdrenfdludcf01:7077
/home/hadoop/development/netconnectionanalyzer.py
)2>&1)| tee -a $filename
```

You need to run following command and save our script in schedule as hadoop user.

```
#crontab -e
00 15 * * * /home/hadoop/development/autoschedule.sh
```