VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Distributed System Lab

Lab 2: Introduction to Apache Spark & RDD-based Programming

Student: Nguyen Hoang Anh Thu - 2053478

HO CHI MINH CITY, NOVEMBER 2023

Contents

1 Task 1: Filtering Wrong Records

```python
from pyspark.sql import SparkSession
from datetime import datetime
import pytz

spark_session = SparkSession.builder.appName("Log Processing").getOrCreate()
log_data = spark_session.sparkContext.textFile("FPT-2018-12-02.log")

# filter correct records
def filter_correct_records(line):
    fields = line.split(" ")
    criteria = len(fields) == 7 and float(fields[0]) >= 0 and fields[6].isdigit() and
     int(fields[6]) > 0 and fields[2] != "-"
    return criteria

# convert time format
def convert_time(line):
    fields = line.split(" ")
    input_data = fields[3] + " " + fields[4]
    time_format = "[%d/%b/%Y:%H:%M:%S %z]"
    try:
        timestamp = datetime.strptime(input_data, time_format).replace(tzinfo=pytz.
    UTC).timestamp()
        return timestamp
    except ValueError:
        return None

# filter correct records
filtered_data = log_data.filter(filter_correct_records)
filtered_data = filtered_data.filter(lambda x: convert_time(x) is not None)
sorted_data = filtered_data.sortBy(convert_time)
print("Top 10 correct records:")
for record in sorted_data.take(10):
    print(record)

# filter incorrect records
def filter_incorrect_records(line):
    return not filter_correct_records(line)

# filter incorrect records
filtered_fail_data = log_data.filter(filter_incorrect_records)
filtered_fail_data = filtered_fail_data.filter(lambda x: convert_time(x) is not None)
sorted_fail_data = filtered_fail_data.sortBy(convert_time)
print("\nTop 10 incorrect records:")
for record in sorted_fail_data.take(10):
    print(record)

# stop spark session
spark_session.stop()
```

Listing 1: Task 1 code

Results:

```
CodeCache: size=131072Kb used=12638Kb max_used=12650Kb free=118433Kb
 bounds [0x000000010a9e0000, 0x000000010b650000, 0x00000001129e0000]
 total_blobs=5218 nmethods=4391 adapters=740
 compilation: disabled (not enough contiguous free space left)
Top 10 correct records:
0.000 58.187.29.147 HIT [02/Dec/2018:00:00:00 +0700] /live/prod_kplus_ns_hd/
    prod_kplus_ns_hd.isml/events(1541466558)/dash/prod_kplus_ns_hd-audio_vie
    =56000-49397873671168.dash 28401
0.055 113.23.26.76 HIT [02/Dec/2018:00:00:00 +0700] /live/prod_kplus_1_hd/
    prod_kplus_1_hd.isml/events(1541466464)/dash/prod_kplus_1_hd-video
    =2499968-926210122800.dash 1265928
0.000 118.69.60.62 HIT [02/Dec/2018:00:00:00 +0700] /live/prod_kplus_pm_hd/
    prod_kplus_pm_hd.isml/stb.mpd 39902
0.000 42.118.29.197 HIT [02/Dec/2018:00:00:00 +0700] /live/prod_kplus_ns_hd/
    prod_kplus_ns_hd.isml/prod_kplus_ns_hd.mpd 40102
0.000 14.231.34.1 HIT [02/Dec/2018:00:00:00 +0700] /
    cc8c96ca2e6b3aa3465a5e2d383c8e011543687445/box/_definst_/vtv3-high.m3u8 323
0.001 42.113.129.241 HIT [02/Dec/2018:00:00:00 +0700] /
    d053d51fe6c3935d4c25908089b7c4961543692607/ndvr/vtv3/_definst_/20181201/vtv3-high
    -20181201175215-611181.ts 742224
0.002 14.244.239.143 HIT [02/Dec/2018:00:00:00 +0700] /8
    d7863dc41865b32054cd8478430fb521543689560/box/_definst_/vtv2-high-2063449.ts
    760272
0.000 14.229.126.144 HIT [02/Dec/2018:00:00:00 +0700] /8
    c6bebd4edd78750291593f1756e861a1543686153/box/_definst_/vtv6-high.m3u8 323
0.000 113.22.7.224 HIT [02/Dec/2018:00:00:00 +0700] /live/prod_kplus_ns_hd/
    prod_kplus_ns_hd.isml/events(1541466558)/dash/prod_kplus_ns_hd-audio_vie
    =56000-49397873798144.dash 28840
0.000 113.179.83.143 HIT [02/Dec/2018:00:00:00 +0700] /98
    caef0ff9b853515fe1e3858397badf1543685819/box/_definst_/vtv6-high.m3u8 323

Top 10 incorrect records:
0.000 123.18.156.7 - [02/Dec/2018:00:00:00 +0700] /
    b74394d1bd14f1afb0c4be5b8c4c22481543619685/box/_definst_/vtv1-high.m3u8 166
0.000 113.161.6.128 - [02/Dec/2018:00:00:00 +0700] /4176
    f0256a9c8b9d5cf6854e27ade7461543681587/box/_definst_/vtv3-high.m3u8 166
0.000 42.115.220.10 HIT [02/Dec/2018:00:00:00 +0700] /live/prod_kplus_1_hd/
    prod_kplus_1_hd.isml/stb.mpd 0
0.000 27.3.65.112 - [02/Dec/2018:00:00:00 +0700] /1
    c68cc3b620f13797a8dc7d60ea8bec41543571805/box/_definst_/vtv3-high.m3u8 166
0.000 113.162.4.228 - [02/Dec/2018:00:00:00 +0700] /73
    f2e7f5a352dc48bc3f46ccfe155a701543617776/box/_definst_/vtv3-high.m3u8 166
0.000 14.181.24.168 - [02/Dec/2018:00:00:00 +0700] /9
    f9acfb050af241723b7576ac79879d81543634619/box/_definst_/vtv3-high.m3u8 166
0.000 14.228.49.147 - [02/Dec/2018:00:00:00 +0700] /
    ab65523b72105c0ad027936fddc3c5ec1543562106/box/_definst_/vtv3-high.m3u8 166
0.000 123.17.19.243 - [02/Dec/2018:00:00:00 +0700] /45
    cb0917fe18ec62f2c5af70a9e992b61543632020/box/_definst_/vtv3-high.m3u8 166
0.000 14.189.182.247 - [02/Dec/2018:00:00:00 +0700] /
    e987966426fe66700b9a0775a08f7c2d1543548337/box/_definst_/vtv3-high.m3u8 166
0.000 14.165.116.89 - [02/Dec/2018:00:00:00 +0700] /6
    ab3ab2fcc3cb07a5a9f6cdb7784da9a1543675624/box/_definst_/vtv1-high.m3u8 166
```

Listing 2: Task 1 result

2 Task 2: Preprocessing

```python
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession
from pyspark.mllib.stat import Statistics
from pyspark.mllib.linalg import Vectors
import numpy as np

spark_session = SparkSession.builder.appName("LogAnalysis").getOrCreate()
spark_context = spark_session.sparkContext
log_data = spark_context.textFile("FPT-2018-12-02.log")

def is_valid_record(line):
    fields = line.split(" ")
    criteria = len(fields) == 7 and float(fields[0]) >= 0 and fields[6].isdigit() and
     int(fields[6]) > 0 and fields[2] != "-"
    return criteria

valid_log_data = log_data.filter(is_valid_record)

# classify service
def get_service_type(line):
    content_name = line.split(" ")[5]
    if content_name.endswith(".mpd") or content_name.endswith(".m3u8"):
        return "HLS"
    elif content_name.endswith(".dash") or content_name.endswith(".ts"):
        return "MPEG-DASH"
    else:
        return "Web Service"

classified_log_data = valid_log_data.map(lambda line: (get_service_type(line), 1))
service_counts = classified_log_data.reduceByKey(lambda a, b: a + b)

for service, count in service_counts.collect():
    print(f"{service}: {count} records")

# extract IP
def extract_ip(line):
    ip = line.split(" ")[1]
    return ip

unique_ips = valid_log_data.map(extract_ip).distinct()

# load IP information
ip_info_data = spark_context.textFile("IPDict.csv")
ip_info_dict = ip_info_data.map(lambda line: (line.split(",")[0], (line.split(",")
    [1], line.split(",")[2], line.split(",")[3]))).collectAsMap()
ip_info_broadcast = spark_context.broadcast(ip_info_dict)

# enrich log record
def enrich_record(line):
    fields = line.split(" ")
    ip = fields[1]
    additional_info = ip_info_broadcast.value.get(ip, ("Unknown", "Unknown", "Unknown
    "))
    latency = float(fields[0])
    city = additional_info[1]
    content_size = int(fields[len(fields) - 1])
    return (ip, additional_info, city, latency, fields[5], content_size)

enriched_log_data = valid_log_data.map(enrich_record)

```

```python
# unique ISPs
unique_isps = enriched_log_data.map(lambda log: log[1][2]).distinct().collect()
print(f"Number of unique ISPs: {len(unique_isps)}")

# records from Ho Chi Minh City
hcm_records = enriched_log_data.filter(lambda log: log[2] == "Ho Chi Minh City")
print(f"Number of records from Ho Chi Minh City: {hcm_records.count()}")

# traffic from Hanoi
hanoi_traffic = enriched_log_data.filter(lambda log: log[2] == "Hanoi").map(lambda
    log: log[5]).reduce(lambda a, b: a + b)
print(f"Total traffic from Hanoi: {hanoi_traffic}")

# latency statistics
latencies = enriched_log_data.map(lambda log: log[3])
latencies_vector = latencies.map(lambda latency: Vectors.dense(latency))
latency_stats = Statistics.colStats(latencies_vector)

print(f"Mean Latency: {latency_stats.mean()[0]}")
print(f"Maximum Latency: {latency_stats.max()[0]}")
print(f"Minimum Latency: {latency_stats.min()[0]}")

# Additional logic for median, maximum, and second maximum latency
sorted_latencies = latencies.sortBy(lambda latency: latency).collect()

median_index = int((len(sorted_latencies) + 1) / 2) - 1
median_latency = sorted_latencies[median_index]
print(f"Median Latency: {median_latency}")

max_latency_sorted_list = sorted_latencies[-1]
print(f"Maximum Latency (sorted list): {max_latency_sorted_list}")

second_max_latency = sorted_latencies[-2]
print(f"Second Maximum Latency: {second_max_latency}")

spark_session.stop()
```

Listing 3: Task 2 code

Results:

```
CodeCache: size=131072Kb used=16753Kb max_used=16768Kb free=114318Kb
 bounds [0x00000001071e0000, 0x0000000108260000, 0x000000010f1e0000]
 total_blobs=7137 nmethods=6236 adapters=814
 compilation: disabled (not enough contiguous free space left)
MPEG-DASH: 826313 records
Web Service: 13976 records
HLS: 462938 records
Number of unique ISPs: 125
Number of records from Ho Chi Minh City: 217212
Total traffic from Hanoi: 204245300091
Mean Latency: 0.1516318983569242
Maximum Latency: 199.658
Minimum Latency: 0.0
Median Latency: 0.0
Maximum Latency (sorted list): 199.658
Second Maximum Latency: 119.467
```

Listing 4: Task 2 result

Link GitHub for Lab 2: DS_Lab2_2053478