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

Step 1: Create a directory in HDFS and copy the log file from local machine to HDFS. After that, I will check whether the log file has been copied:

```
hadoop fs -ls /user/S2053478/test1/
```

```
Thus-MacBook-Air:Downloads thunguyen$ hadoop fs -ls /user/S2053478/test1
2023-11-13 21:24:58,378 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... usi
ng builtin-java classes where applicable
Found 1 items
-rw-r--r--   2 thunguyen supergroup  282042368 2023-11-13 21:24 /user/S2053478/test1/FPT-2018-12-02.log
```

Step 2: Create a RDD (Resilient Distributed Datasheet).

```
>>> log_data = spark_session.sparkContext.textFile("FPT-2018-12-02.log")
>>> log_data.count()
1896813
```

Step 3: Create a function to filter as the criteria and a function to convert time format:

```
>>> 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
...
>>> filtered_data = log_data.filter(filter_correct_records)
>>> 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
...
>>> filtered_data = filtered_data.filter(lambda x: convert_time(x) is not None)
>>> sorted_data = filtered_data.sortBy(convert_time)
>>> filtered_data.count()
1303227
```

Listing 1: Total number of records

Step 4: The same as step 3 but for the incorrect records:

```
>>> def filter_incorrect_records(line):
...     return not filter_correct_records(line)
...
>>> 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)
>>> filtered_fail_data.count()
```

```
593586
```

Listing 2: Number of wrong records

Step 5: Print out the 10 of correct and incorrect records.
For the correct records:

```
>>> for record in sorted_data.take(10):
...     print(record)
...
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
```

Listing 3: Top 10 correct records

For the incorrect records:

```
>>> for record in sorted_fail_data.take(10):
...     print(record)
...
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 4: Top 10 incorrect records

2 Task 2: Preprocessing

Step 1: Create a function to classify services:
The *classified_log_data* RDD:

- Apply the below function to each line in the *filtered_data* RDD that I did at the Task 1.
- Create a new RDD of Key-Value pairs (as the *service_counts* variable) which Key is the service group, and Value is the number of records for that group.

```
>>> 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 = filtered_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")
...
MPEG-DASH: 826313 records
Web Service: 13976 records
HLS: 462938 records
```

Step 2: Print out the list of unique IPs by creating a function to get the IPs in the RDD data.

```
>>> def extract_ip(line):
...     ip = line.split(" ")[1]
...     return ip
...
>>> unique_ips = filtered_data.map(extract_ip).distinct()
>>> unique_ips.count()
3952
```

Step 3: Build a RDD containing the map of IPs
Assume that I have copied the *IPDict.csv* from local machine to HDFS.

```
$ hadoop fs -ls /user/S2053478/test1/
2023-11-13 22:00:08,188 WARN util.NativeCodeLoader: Unable to load native-hadoop
    library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r--   2 thunguyen supergroup  282042368 2023-11-13 21:24 /user/S2053478/test1/
    FPT-2018-12-02.log
-rw-r--r--   2 thunguyen supergroup     278374 2023-11-13 21:59 /user/S2053478/test1/
    IPDict.csv
```

Step 4: Analyse data

```
>>> ip_data = spark_session.sparkContext.textFile("IPDict.csv")
>>> ip_info = ip_data.map(lambda line: (line.split(",")[0], (line.split(",")[1], line.split(",")[2], line.split(",")[3]))).collectAsMap()
>>> ip_broadcast = spark_session.sparkContext.broadcast(ip_info)
>>> def enrich_record(line):
...     fields = line.split(" ")
...     ip = fields[1]
...     additional_info = ip_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 = filtered_data.map(enrich_record)
```

Step 5: Print number of records from Ho Chi Minh City:

```
>>> unique_isps = enriched_log_data.map(lambda log: log[1][2]).distinct().collect()
>>> print(f"Number of unique ISPs: {len(unique_isps)}")
Number of unique ISPs: 125
```

```
>>> 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()}")
Number of records from Ho Chi Minh City: 217212
```

Listing 5: Number of records from Ho Chi Minh City

```
>>> 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}")
Total traffic from Hanoi: 204245300091
```

Listing 6: Total traffic from Hanoi

Step 6: Calculate the latencies's values.

```
>>> from pyspark.mllib.stat import Statistics
>>> from pyspark.mllib.linalg import Vectors
>>> 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]}")
Mean Latency: 0.1516318983569242
>>> print(f"Maximum Latency: {latency_stats.max()[0]}")
Maximum Latency: 199.658
>>> print(f"Minimum Latency: {latency_stats.min()[0]}")
Minimum Latency: 0.0
```

Listing 7: Mean maximum and minimum latencies

Link GitHub: DS_Lab2