Apache Spark - PySpark

Author:
Simon Thelin

Supervisor:
NaN

May 24, 2018

Contents

1 Introduction

Data engineering is a broad field and it encounters a lot of areas within development and research. It gives us the foundation to better understand and store the data we are working with and want to work with. Companies of today will continue to grow in this field and the competence within this area is huge all over the world.

1.1 Purpose And Goal

The goal of this project is to learn hos Apache Spark together with PySpark works.

2 Background

This section will cover the core foundation of what this project is about.

2.1 ETL

ETL[1] is a type of data integration that refers to the three steps (extract, transform, load). It can be used to blend data from multiple sources. It's often used to build a data warehouse. During this process, data is taken (extracted) from a source system, converted (transformed) into a format that can be analyzed, and stored (loaded) into a data warehouse or other system. In this project the load part will only be a standard output.

3 Methodology

In this section the methodology for this project will be described briefly.

3.1 Project Design

The design of this project is that is should fast and easy to deploy and run. Below is a description of how the application should be deployment ready.

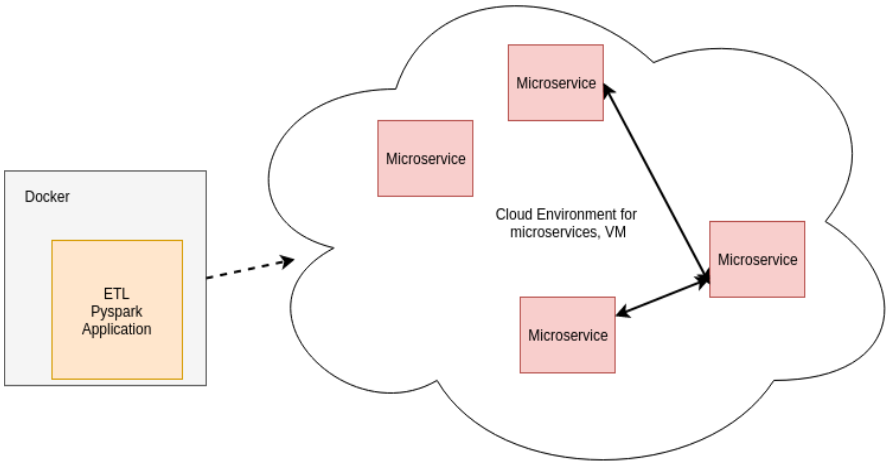

Figure 1: *Project Design Deployment High Level Overview*

3.1.1 Microservice

Microservices is an architectural style that structures an application as a collection of loosely coupled services. It which implement business capabilities such as continuous delivery/deployment of scale.

3.1.2 Docker

Docker is the given tool to create great microservices. Docker encapsulates an application in a good way and makes it very modular. There are of course down sides to a docker as well. But for this project it will work good.

3.2 Bath Processing

For this project the data source is a TSV file. This means that the processing of data is done in one batch and not in a streaming manner.

3.2.1 Apache Spark

Apache Spark[4] is a unified analytic engine for large-scale data processing. It is a general distributed in-memory computing framework implemented in scala[6]. Apache Hadoop[5] is also a famous tool. Hadoop is essentially a distributed data infrastructure. It provides distributes big data collections across multiple nodes within a cluster of commodity servers. Spark is a data-processing tool that operates on those distributed data collections. Hence it does not do distributed storage. Spark is also much faster than hadoop. Apache Spark and Hadoop have different purposes but works good together. A high level overview of Spark foundation below:

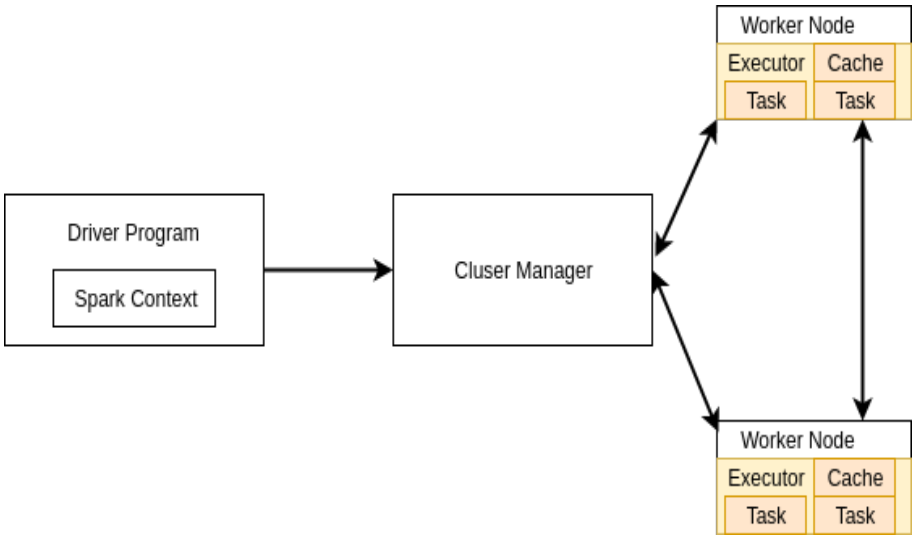

Figure 2: *Apache Spark Concept High Level Overview*

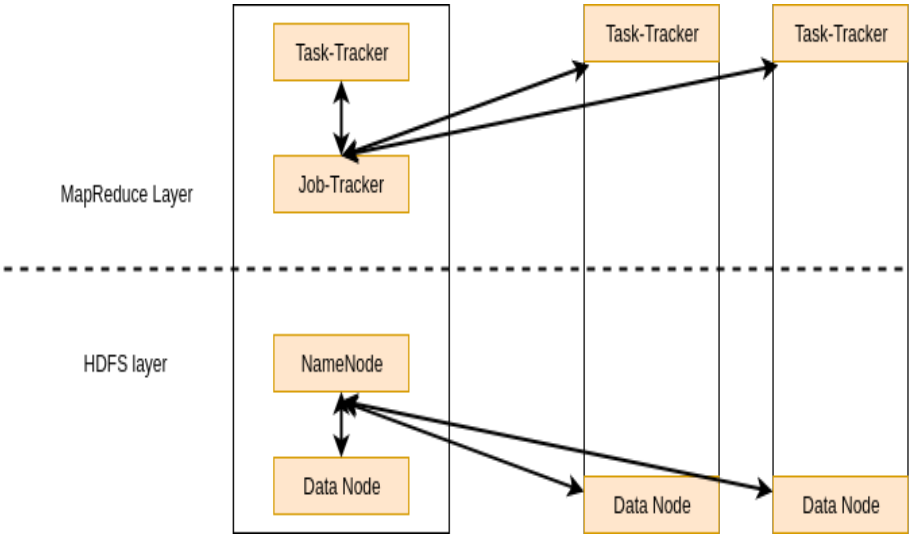

Figure 3: *Apache Hadoop Concept High Level Overview*

3.2.2 PySpark

Pyspark is an API developed in Python[9] for spark programming and writing spark applications in Python style. In PySpark there are some concepts. Such as *Resilient Distributed Datasets*[7]. RDD is the building blocks of Spark. It is the original API that Spark developed and pretty much all the higher level APIs decompose to RDDs. The RDD is a set of Java or Scala objects representing data.

RDD have three main characteristics.

- Compile-time type safe.
- Slow due to that they are based on Scala collections API.

DataFrame[8] has been derived from the RDD concept. Dataframes will provide a higher level of abstraction. Which implicitly allow a query language way of typing to manipulate data. The abstraction represents data and a schema. Which enable better interaction with front end applications.

Pyspark is also very good when you are experimenting with Spark and want to develop something fast and efficient. It could be considered better to use when doing batch analysis rather than doing streaming analysis where Scala would perform better.

3.3 Data Management

The given data format for this project is TSV. The data is stored within a file and is not coming in a streaming format.

3.3.1 Dataset

The dataset has the following values divided by their tab: field header (date, time, user_id, url, IP, user_agent_string).

3.4 Analysis On Data

In this project the analysis will be the one that is stated.

- Top 5 Countries based on number of events.
- Top 5 Cities based on number of events.
- Top 5 Browsers based on number of unique users.
- Top 5 Operating systems based on number of unique users.

4 Results

In this section the results from the implementation will be shown.

4.1 Fetching OS & Browser

To be able to fetch the OS used and the browser from the dataset. The following library has been used: httpagentparser[10]. It has the given features:

- Works on Python 2.7 and Python 3
- Detects OS and Browser. Does not aim to be a full featured agent parser
- Will not turn into django-httpagentparser

How it is implemented in the source code:

```
    agentinfo = udf(getOsBrowser, StringType())
    _df = _df.withColumn("getOsBrowser",
        agentinfo(_df.user_agent_string))
```

Using user defined function[3] to make the call with the column values to the getOsBrowser function. A very nice and easy way that Spark provides.

```
def getOsBrowser(value):
    """
    Calls the httpagentparser and retrieves the os and browser
        information
    :param value: each column value of user_agent_string
    :return: the browser and os as a string
    """
    return str(httpagentparser.simple_detect(value)[0] + "-" +
        httpagentparser.simple_detect(value)[1])
```

In the source code there will be an action that will read in the value, split it and divide it into two columns.

4.2 Fetching Country & City

This has been a lot of headache. It was tried to use an API service but they did not work and HTTP 503 was received over and over again. To make the best possible implementation without having to stop and try to reinvent the wheel. It was decided to use a library file which would

be used to parse the IP values. The library thas has been used is from geolite[2]. The source file called *etl.py, geoip.py* handles the conversion. The process is as followed:

```
    print("Converting IP adress to city and country... ")
    ip1 = _df.select("ip1").collect()

    """ Get the countries and cities from the IP columns """
    ip1 = getCountryCity(ip1)

    """ Create dataframe for countries and cities of the first
        ip column """
    cs1 = spark.createDataFrame(
        ip1,
        StringType()) \
        .withColumnRenamed("value", "location") \
        .withColumn("id", monotonically_increasing_id())

    ip1 = splitCol(cs1, "-", ["location", "country",
        "city"]).drop("location")
```

```
def getCountryCity(list):
    """
    Makes a call to ipquery inside the geoip.py to retreive the
        country and city to a certain IP
    :param list: the list representing all the dataframe column
        values
    :return:
    """
    atrlist = []
    """ Doing transformation from IP to find Country, City...
        """
    for i in list:
        _ips = i.ip1
        try:
            atrlist.append(ipquery(_ips))
        except:
            atrlist.append("NotTraceable-NotTraceable")

    return atrlist
```

```
"""
    http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
    """
```

```
rawdata = pygeoip.GeoIP(cwd + "/GeoLiteCity.dat")

def ipquery(ip):
    """
    Function to parse IP to country, city
    http://www.linuxx.eu/2014/05/geolocate-ip-with-python.html
    Have been sliglty modified
    :param ip:
    :param name: which sort of data that wants to be
       transformed from the IP address
    :return:
    """
    if ip is None:
        return str("None-None")

    data = rawdata.record_by_name(ip)

    if data['country_name'] is not None and data['city'] is not
        None:
        return str(data['country_name'] + "-" + data['city'])
    else:
        return str("NotTraceable-NotTraceable")
```

As seen above udf has not been used. After several tries it kept on failing. Implementations in Scala was also done but the imports did not work. The knowledge to produce an algorithm to parse country and city from a IP is not something that is done very easily. However, the values from the column is parsed as a list. The list of IP values is then converted to country and cities which is then converted to a dataframe.

Also when doing the transforming, it was discovered that there where several IP adresses in the same column value. After analyzing the result it was chosen to not use the column of extra IP adresses.

More details and comments can be found in the source code.

4.3 Production Ready

This service should be as close to production ready as possible. Therefore the project has been encapsulated with docker. This makes it totally production ready as a microservice.

4.3.1 How To Get The Output

First of all, make sure docker is installed on the computer.

With the provided source code there is a readme.md file that explains how to run this project. To run this project as a docker, be in the folder with files and type the following: docker build -t etl-yieldify .

The following will now happen:

```
Sending build context to Docker daemon 72.44MB
Step 1/12 :FROM ubuntu:latest
 ---> 452a96d81c30
Step 2/12 :ENV PATH
   /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
 ---> Using cache
 ---> 25da5da4b74b
Step 3/12 :RUN apt-get update
 ---> Using cache
 ---> e45b16d1b6e2
Step 4/12 :RUN  apt-get install -y python2.7 && apt-get
   install -y python-pip
 ---> Using cache
 ---> ec6eeb22dad8
Step 5/12 :RUN apt-get update && apt-get install -y
   openjdk-8-jdk && apt-get install -y ant && apt-get
   clean && rm -rf /var/lib/apt/lists/ && rm -rf
   /var/cache/oracle-jdk8-installer;
 ---> Using cache
 ---> 5a35bfb52f31
Step 6/12 :ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64/
 ---> Using cache
 ---> f0bc7fa9c089
Step 7/12 :RUN export JAVA_HOME
 ---> Using cache
 ---> c6143fd3f08e
Step 8/12 :WORKDIR /etc/app
 ---> Using cache
 ---> 19003608b661
Step 9/12 :ADD . /etc/app
 ---> c0ae5d9f3033
Step 10/12 :RUN chmod +x /etc/app/run.sh && chmod +x
   /etc/app/input_data
```

```
 ---> Running in 5bda2d8c8b2c
Removing intermediate container 5bda2d8c8b2c
 ---> fe3b305455c7
Step 11/12 :RUN chmod 0644/etc/app/etl.py && chmod 0644
   /etc/app/geoip.py && chmod 0644/etc/app/input_data
 ---> Running in 2af3de3299e6
Removing intermediate container 2af3de3299e6
 ---> 661b18d003fa
Step 12/12 :ENTRYPOINT ["./run.sh"]
 ---> Running in d5c79c3bafa4
Removing intermediate container d5c79c3bafa4
 ---> e14a30d70e7e
Successfully built e14a30d70e7e
Successfully tagged etl-yieldify:latest
```

It may look trivial. But it will install all necesarry dependencies for setting up the initial environment such as ubuntu with java. Java will be needed to run the PySpark application. The above output is when all these have already been installed once before.

Now simply run this docker container by the following command:

```
docker run etl-yieldify
```

The following will then happen:

```
Collecting findspark==1.2.0 (from -r requirements.txt
   (line 1))
  Downloading https://files.pythonhosted.org/packages/18/...
Collecting httpagentparser==1.8.1 (from -r
   requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/de/...
Collecting numpy==1.14.3 (from -r requirements.txt (line 3
   ))
  Downloading
     https://files.pythonhosted.org/packages/c0/...
     (12.1MB)
Collecting pandas==0.23.0 (from -r requirements.txt (line 4
   ))
  Downloading
     https://files.pythonhosted.org/packages/76/...
     (11.8MB)
Collecting py4j==0.10.6 (from -r requirements.txt (line 5))
```

```
  Downloading https://files.pythonhosted.org/packages/...
     (189kB)
Collecting pygeoip==0.3.2 (from -r requirements.txt (line 6
    ))
  Downloading https://files.pythonhosted.org/packages/f2/...
Collecting pyspark==2.3.0 (from -r requirements.txt (line 7
    ))
  Downloading https://files.pythonhosted.org.... (211.9MB)
Collecting python-dateutil==2.7.3 (from -r
    requirements.txt (line 8))
  Downloading
     https://files.pythonhosted.org/packages/cf.... (211kB)
Collecting pytz==2018.4 (from -r requirements.txt (line 9))
  Downloading
     https://files.pythonhosted.org/packages/dc/... (510kB)
Requirement already satisfied: six==1.11.0 in
    /usr/lib/python2.7/dist-packages (from -r
    requirements.txt (line 10))
Building wheels for collected packages: httpagentparser,
    pyspark
  Running setup.py bdist_wheel for httpagentparser: started
  Running setup.py bdist_wheel for httpagentparser:
     finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/c5...
  Running setup.py bdist_wheel for pyspark: started
  Running setup.py bdist_wheel for pyspark: finished with
     status 'done'
  Stored in directory: /root/.cache/pip/wheels/d9/db...
Successfully built httpagentparser pyspark
Installing collected packages: findspark, httpagentparser,
    numpy, pytz, python-dateutil, pandas, py4j, pygeoip,
    pyspark
Successfully installed findspark-1.2.0
    httpagentparser-1.8.1 numpy-1.14.3 pandas-0.23.0
    py4j-0.10.6 pygeoip-0.3.2 pyspark-2.3.0
    python-dateutil-2.7.3 pytz-2018.4
total 70768
drwxr-xr-x 2 root root    4096 May 19 17:27 .
drwxr-xr-x 72root root    4096 May 19 17:27 ..
-rw-r--r-- 1 root root    2522 May 19 16:52 .gitignore
-rw-r--r-- 1 root root    1050 May 19 17:24 Dockerfile
-rw-r--r-- 1 root root 20539238May 1617:44GeoLiteCity.dat
```

```
-rw-r--r-- 1 root root     831 May 19 16:58 README.md
-rw-r--r-- 1 root root    6792 May 19 17:27 etl.py
-rw-r--r-- 1 root root     920 May 19 16:15 geoip.py
-rw-r--r-- 1 root root 51880473 May 15 15:07 input_data
-rw-r--r-- 1 root root     160 May 19 16:45 requirements.txt
-rwxr-xr-x 1 root root     106 May 19 17:24 run.sh
2018-05-19 17:28:46 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
2018-05-19 17:28:53 WARN ObjectStore:6666 - Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
2018-05-19 17:28:53 WARN ObjectStore:568 - Failed to get database default, returning NoSuchObjectException
2018-05-19 17:28:54 WARN ObjectStore:568 - Failed to get database global_temp, returning NoSuchObjectException
2018-05-19 17:29:42 WARN TaskSetManager:66 - Stage 3 contains a task of very large size (736 KB). The maximum recommended task size is 100KB.
2018-05-19 17:29:44 WARN TaskSetManager:66 - Stage 4 contains a task of very large size (736 KB). The maximum recommended task size is 100KB.
2018-05-19 17:29:48 WARN TaskSetManager:66 - Stage 8 contains a task of very large size (736 KB). The maximum recommended task size is 100KB.
2018-05-19 17:29:49 WARN TaskSetManager:66 - Stage 9 contains a task of very large size (736 KB). The maximum recommended task size is 100KB.
2018-05-19 17:29:54 WARN TaskSetManager:66 - Stage 12 contains a task of very large size (736 KB). The maximum recommended task size is 100KB.
[Stage 21:>                 (0 + 8) / 8][Stage 22:>                 (0 + 0) / 8]2018-05-19 17:30:31 WARN TaskSetManager:66 - Stage 22 contains a task of very large size (736 KB). The maximum recommended task size is 100KB.
Perform extraction
Perform transformation
```

```
The dataframe is being cleaned....
date and time column is becomming one timestamp...
The user_agent_string is becomming os and browser...
Converting IP adress to city and country...
Printing Transformed Dataframe Schema
root
 |-- eventID: long (nullable = false)
 |-- timestamp: timestamp (nullable = true)
 |-- user_id: string (nullable = true)
 |-- url: string (nullable = true)
 |-- os: string (nullable = true)
 |-- browser: string (nullable = true)
 |-- country: string (nullable = true)
 |-- city: string (nullable = true)

Perform load
Top 5 cities based on number of events
+--------------+------+
|       country| count|
+--------------+------+
|United Kingdom|122317|
|  NotTraceable| 22555|
|       Ireland| 11286|
|        Sweden| 5377|
|        Norway| 4435|
+--------------+------+
only showing top 5rows

Top 5 cities based on number of events
+------------+-----+
|        city|count|
+------------+-----+
|NotTraceable|22555|
|      Dublin| 5325|
|      London| 4626|
|  Manchester| 3304|
|     Bristol| 2699|
+------------+-----+
only showing top 5rows

Top 5 Browsers based on number of unique users
+--------------------+----------------------+
```

```
|            browser|count(DISTINCT user_id)|
+-------------------+-----------------------+
|         Safari 7.0|                  25521|
|         Safari 8.0|                  16865|
|Chrome 37.0.2062.124|                   4844|
|       Safari 7.0.6|                   2577|
|Microsoft Interne...|                   2347|
+-------------------+-----------------------+
only showing top 5rows

Top 5 Operating systems based on number of unique users
+---------------+-----------------------+
|             os|count(DISTINCT user_id)|
+---------------+-----------------------+
|  IPad iOS 8.0.2|                  10290|
|  IPad iOS 7.1.2|                  10052|
|      Windows 7|                   6471|
|iPhone iOS 8.0.2|                   4483|
|iPhone iOS 7.1.2|                   4414|
+---------------+-----------------------+
only showing top 5rows

Spark application ends
```

A picture of the running of the docker environment with the source code.

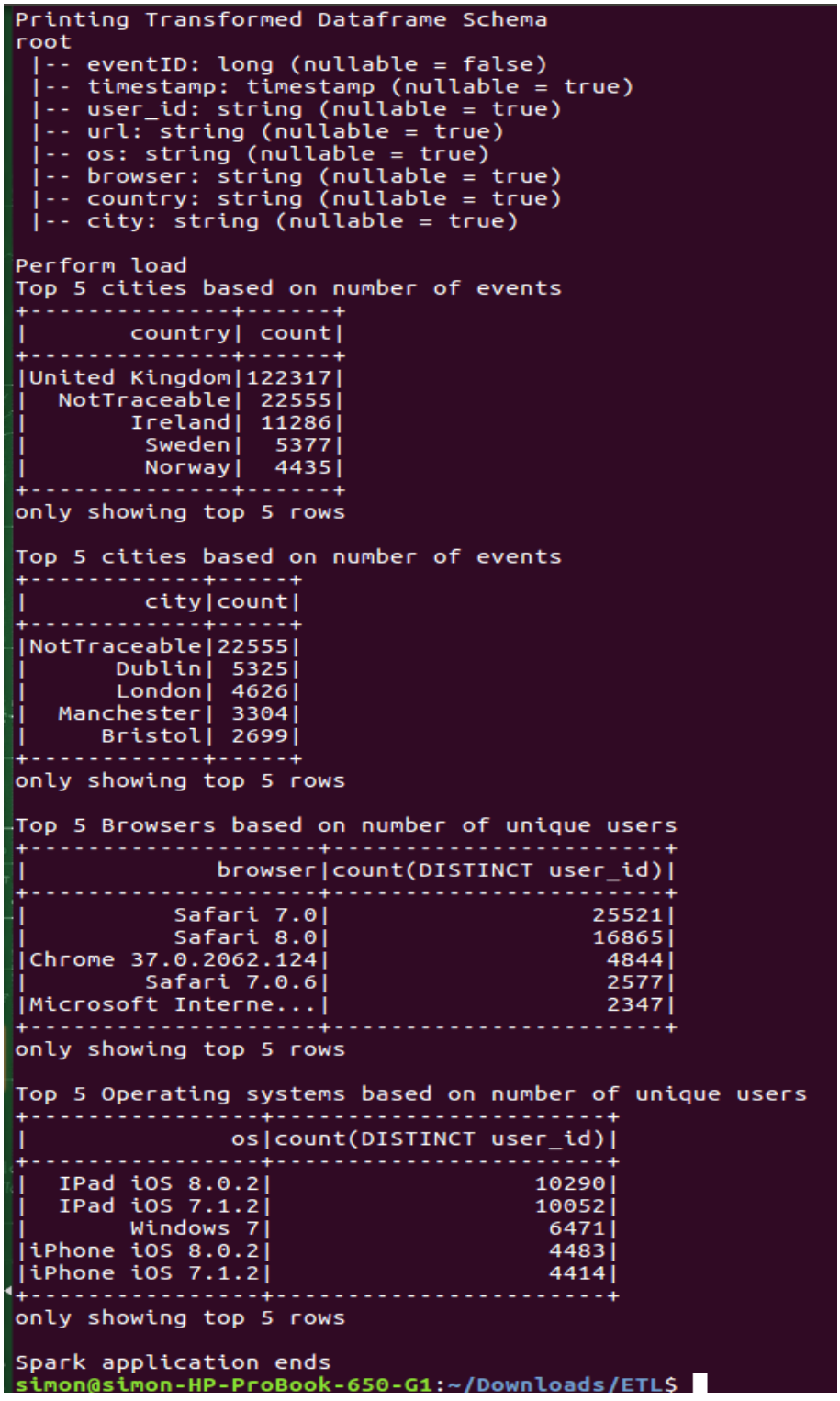

Figure 4: *Output*

5 Conclusion And Perspectives

In this section some conclusions and perspectives on the implementation will be shared.

5.1 API problems

When doing this project there was a lot of problems to get the API for fetching country and city from the IP-adresses. It was done both in Scala and in Python with PySpark but with a lot of errors. In the end a locally stored dat file containing the information has been used. It is not optimal, not perfect to date, but it works and gives a good example of the task at hand.

5.2 Not Traceable

There are some values that are not traceable, this is partially because the file being used to parse city and country may not be perfect. IP addresses can be switched around by time and this may resolve to conflicts.

5.3 Personal Reflection

Before doing this project I had never touched Apache Spark. I had only read about it, so it was a good project to test it out.

6 References

[1] Extract Transform Load. https://en.wikipedia.org/wiki/Extract,_transform,_load. Accessed: 2017-05-15.

[2] Geolite. https://dev.maxmind.com/geoip/legacy/geolite/. Accessed: 2017-05-15.

[3] UDF. https://docs.databricks.com/spark/latest/spark-sql/udf-in-python.html. Accessed: 2017-05-15.

[4] Apache Spark. https://spark.apache.org/. Accessed: 2017-05-15.

[5] Apache Hadoop. http://hadoop.apache.org/. Accessed: 2017-05-15.

[6] PySpark. https://www.scala-lang.org/. Accessed: 2017-05-15.

[7] RDD. https://spark.apache.org/docs/1.1.1/api/python/pyspark.rdd.RDD-class.html. Accessed: 2017-05-15.

[8] Dataframes. http://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html. Accessed: 2017-05-15.

[9] Python. https://www.python.org/. Accessed: 2017-05-15.

[10] OS & Browser Parser. https://pypi.org/project/httpagentparser/. Accessed: 2017-05-15.