Distributed System Course
Lab 2 - An Introduction to Apache Spark & RDD-based Programming (Exercise)

1. Objective:

The objective of this laboratory is to start playing around with Apache Spark. Specifically, we start with Resilient Distributed Dataset (RDD) data structure, which is an important brick of Spark programming.

2. Preparation:

- Download a dataset file from: https://drive.google.com/file/d/1AwiiKSyGIeOptY2ckKJsNaROWCZET88g/view?usp=sharing

3. Problem Description:

The provided dataset is a log file of a system, which contains the user traffic information going through the server. The log file is raw so that it may contain wrong or value-missing records. The log file is provided on your local machine. In this laboratory, your task is to manipulate RDD operations to filter, preprocess, analyze and compute some statistics on this log.

Records format: Each line of the log file corresponds to a user record. Each record includes 6 fields, namely latency, user's IP, caching status, resquest's time, content name, and file size.

For examples, we have the following 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

1.001	**Latency**	Seconds
58.187.29.147	**user IP**	
HIT	**Caching Status**	There are 4 statuses: ● HIT, HIT1: the content has been already cached by the server. ● MISS: the content has not been cached yet. ● - : the content has been cached by local devices.
[02/Dec/2018:00:00:00 +0700]	**Request Time**	
/live/prod_kplus_ns_hd/prod_kplus_ns_hd.isml/events(1541	**Content Name**	

466558)/dash/prod_kplus_ns_hd-audio_vie=56000-49397873671168.dash		
28401	**Content Size**	Size of the message Byte

4. Task 1: Filtering Wrong Records

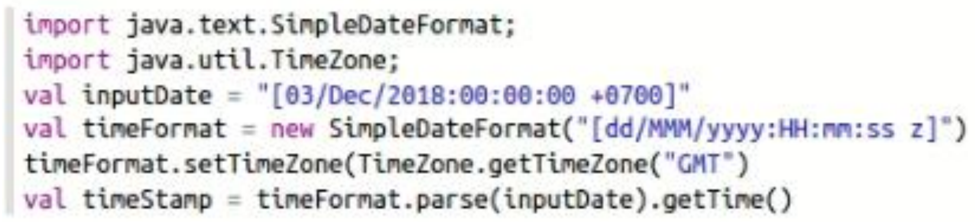
The beginning of a data analysis pipeline is always the filtering step. A record is wrong if it satisfies one of the following criteria:

- The record line does not contain 6 fields (It may have more or less than 6 fields).
- The latency of records is positive (non negative and not equal zero).
- The size of content is positive.
- The hit status cannot be dash ("-").

Requirements:

- Print out the total number of records and the number of wrong records.
  (***Hint:*** You can use the following methods:
    - Spark: textFile(), map(), filter()
    - Scala: split()

)
- Now we have two record lists, namely wrong and correct lists. Print out the top 10 of each list. Record lists should be ascendingly sorted by response time.
  (***Hint:*** In order to sort by time, you should convert the response time into timestamp format (seconds or milliseconds).
    - Spark: sortBy(), top()
    - Scala: To convert the string of time to timestamp, you can do as the following example.

```
import java.text.SimpleDateFormat;
import java.util.TimeZone;
val inputDate = "[03/Dec/2018:00:00:00 +0700]"
val timeFormat = new SimpleDateFormat("[dd/MMM/yyyy:HH:mm:ss z]")
timeFormat.setTimeZone(TimeZone.getTimeZone("GMT")
val timeStamp = timeFormat.parse(inputDate).getTime()
```

)

5. Task 2: Preprocessing

In order to analyze the log file, we need to extract and transfer the raw information into more meaningful formats. Specifically, we need to do some pre-process step as follows:

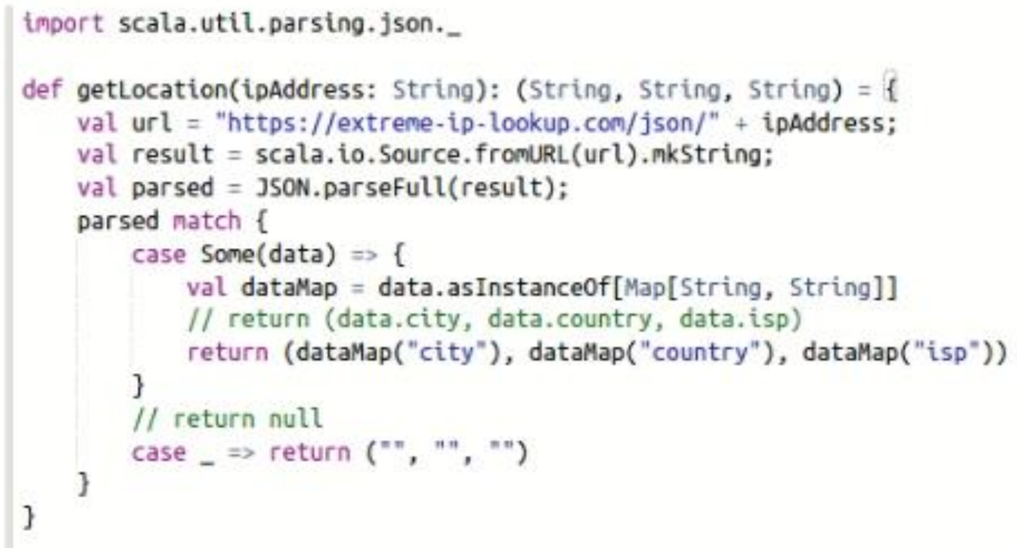
- Mapping the user IP with geo-location information and ISP. In more details, we focus on the information as ISP, city, country, continent.
- The log file mainly contains the records of two services, which are web service, video streaming service. The video streaming service is based on two streaming protocols, namely HLS and MPEG-DASH. The records can be classified into 3 groups based on their content name. More details, we have the following rules:
    - HLS: the extension of the content name is ".mpd" or ".m3u8".
    - MPEG-DASH: the extension of the content name is ".dash" or ".ts".
    - Web service: They are the remaining records.

Requirements:

- Print out the number of records for each service group.
  (***Hint:*** You can use the following methods:
    - Spark: map(), filter(), count()
    - Scala: split(), contains()

)
- Print out a list of unique IPs from the log.
  (***Hint:*** You can use the following methods:
    - Spark: distinct(), map()

)
- Build a RDD, which contains the map of IPs and their additional information.
  (***Hint:*** To get the additional information of IPs, there are 2 options:
    - You can use the public API: *https://extreme-ip-lookup.com/json/*, and the following function.



```
import scala.util.parsing.json._

def getLocation(ipAddress: String): (String, String, String) = {
    val url = "https://extreme-ip-lookup.com/json/" + ipAddress;
    val result = scala.io.Source.fromURL(url).mkString;
    val parsed = JSON.parseFull(result);
    parsed match {
        case Some(data) => {
            val dataMap = data.asInstanceOf[Map[String, String]]
            // return (data.city, data.country, data.isp)
            return (dataMap("city"), dataMap("country"), dataMap("isp"))
        }
        // return null
        case _ => return ("", "", "")
    }
}
```

  - Or you can read from file csv “IPDict.csv”.

)

- Print out a list of the ISPs (the list should have unique values).
- Print out the number of records, which come from Ho Chi Minh City
- Print out the mean, median, maximum and minimum of latency
- Print out total traffic (total of content size), which come from Ha Noi

6. Task 3: Analysis (Optional)

The caching performance of system is evaluated by Hit Rate metric, which is simply computed by the following equation:

$$HitRate = \frac{HIT + HIT1}{HIT + HIT1 + MISS}$$

Requirements:

- Print out the number of HIT, MISS, and HIT1 requests.
- Compute the *HitRate* of the system.
- Find the best ISP, which has the best *HitRate*.

7. Task 4: Improving Performance of Spark (Optional)

There are some aspects of the caching mechanism and shared variables, which can affect the performance of Spark jobs. In this exercise, we will do some experiments to evaluate these Spark features.

(Notice: The gaps between the improvement strategies and the original ones depend on several conditions such as network bandwidth, the computational capacity of workers, and the volume of the dataset.)

a. Cache/Persistence:

At Task 2, you should have the RDD, which contains IP and their additional information (geo-location, ISP, etc.).

Your task: In this task, you should use the cache/persist method to cache this RDD and do some queries with this RDD (for instance, the last requirement of task 2). You also compare the computation time when using these methods and when not.

b. Shared variables:

When a "function" passed to a Spark operation is executed on a remote cluster node, it works on separate copies of all the variables used in the function. These variables are copied to each node of the cluster, and no updates to the variables on the nodes are propagated back to the driver program.

Spark provides a type of shared variables called accumulators. Accumulators are shared variables that are only "added" to through an associative operation and can therefore be efficiently supported in parallel. Instead of using reduce() for simple statistics, you should use accumulators.

Your task: The fifth requirement of the task 2 should be re-implemented by using an accumulator. Print out the execution time when using the accumulator and when not.

For more information about these methods, you can refer this link:
https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html