Ege Sari
s1034535

Big Data Project Report

Preface

I have chosen the informal way to describe what I have done and what I have learned for this report. Because I believe for the readers it is easier and more enjoyable for them to read and evaluate this report. I hope I am correct, otherwise, this may be ended up with a bad grade.

What did I want to do?

We were given to perform an analysis job on the Web Archive. At first, I wanted to show the correlation between the words “Ukraine” and “energy”. This means I want to show the frequency of the words “Ukraine” and “energy” occurring on the same web page between the timeline February 2022 and May 2022. My expectation for this analysis is to show there is a correlation between these words and also I was wondering in which month the frequency is the highest. But unfortunately, I couldn’t perform this task because the Web Archive in the cluster includes only the timeline until April 2021 (and some of the days of April).

What did I have done?

After that, I changed my experiment. My new goal was to find the frequency of “bitcoin” in 2020 month by month. But this time, I was not planning to look at the body of the webpage, my scope only contains the title. Also, I have limited the scope to three websites: bbb.com, nytimes.com (new york times), and wsj.com (wall street journal).

What was my motivation?

I have chosen the word bitcoin because in 2021 we have seen that it has gone to the moon and somehow it gained popularity, especially among my generation. But I don’t believe that it happened in one night. Moreover, we know that after the declaration of pandemia in March 2020, bitcoin fell to a historical low point.

I have limited my scope to only the title of the webpage, because the other way, looking at the body of the webpage, the content would be too complex and take more time.

The reason I have chosen the year 2020 is related to this, after March 2020 it went to the bottom and in November 2020, people were talking about it will go skyrocketing. So I thought I should look for the year 2020 for the background story of 2021 skyrocketing.

I have chosen three websites because they are big websites which means I will have access to more samples. And they have a special scope for bitcoin especially the new york times and wall street journal which means I will have access to the aimed sample.

What is my expected result?

I believed that at the end of the analysis, the frequency in early months like January and February would be low because the level of bitcoin was stable. Yet in March, I expected to see the frequency would be high because the level of bitcoin just crashed. For the mid-months, from April to November I expected again low frequency, and the last months, November and December would have a high frequency. I expected this way because, for the mid-months, there was no change, yet we see an uptrend at the end of the year.

The results are discussed at the end of this report, you can find the reality of my expectations.

Hang on, what is WARC Record?

Before starting to build any program for my analysis, I had to learn what WARC records and WARC files are and understand how they work. At first, I have literally no knowledge, I just knew that a WARC file contains one or more WARC records. I have learned what are they, and what they contain, inside of WARC file, and inside the headers of WARC records.

While learning the header structure of the WARC record, and how to parse it, I realized that the information I was looking for (the title, the URL, the date…) was inside the webpage itself, it was buried in HTML. So I have to learn how to parse HTML in Spark and extract the parts I need. Luckily, there are very nice documented cookbooks for that. So learning the two parts, WARC records/files, and HTML parsing summaries the before-programming part.

After learning these, I realized that before running my program on the cluster, I should somehow test my program on one or more WARC files. I have chosen three WARC files. I have found them using the CDX Index service. I have used bbc.com WARC files because I knew that they contain at least one WARC file using the link bbc.com. A quick spoiler, neither of the three WARC files contain a WARC record with a title containing the word “bitcoin”. But we will see the details of this issue, and the solution for it in the next sections.

I have copied and moved the WARC files from redbad container to big-data container. Now they are ready to be worked on them.

Then, let’s start

At first, I sketched what will I do to achieve my goal. If we think about our analysis, there will be mainly two operations: Filtering and aggregation. I want to filter the data, the link, and the title. Then I will aggregate all the remaining results grouped by the month, which means the frequency of the word bitcoin per month.

For the very first step, we should take the WARC file and split it to WARC records :

```
val fname = z.textbox("Filename:")
val warcfile = s"file:///opt/hadoop/rubigdata/${fname}.warc.gz"
```

Filename:

CC-MAIN-20210411030031-20210411060031-00586

```
val warcs = sc.newAPIHadoopFile(
                warcfile,
                classOf[WarcGzInputFormat],             // InputFormat
                classOf[NullWritable],                  // Key
                classOf[WarcWritable]                   // Value>
    )//.cache()
```

INTERMEZZO: Of course, we define and configure our Spark session. An important thing to notice in this configuration is we have set the driver memory to 4 GB and executor memory to 8 GB. Otherwise, we encounter java heap space errors, which we will discuss later. Another important thing is, while the code below commented on the dynamic allocation section for Zeppelin, we use dynamic allocation for the cluster:

```
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession

val sparkConf = new SparkConf()
                    .setAppName("RUBigData WARC4Spark 2021")
                    .set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
                    .registerKryoClasses(Array(classOf[WarcRecord]))
                    //.set("spark.dynamicAllocation.enabled", "true")
                    .set("spark.driver.memory", "4g")
                    .set("spark.executor.memory", "8g")
implicit val sparkSession = SparkSession.builder().config(sparkConf).getOrCreate()
val sc = sparkSession.sparkContext
```

After splitting the Warc file to the records, now we can look at the header of the records and the information buried in HTML.

Yes, finally parsing!

We should parse the header first. I have thought that I can extract the date and the link of the webpage from the header of the WARC record. Actually, it is true, indeed I can extract the date and the link. But after comparing the date I get from parsing HTML, the date from the WARC header does not match. Hence I have decided to use the date extracted from HTML.

But for the link I have done the reverse, I have extracted it from the WARC header. Because after comparing with the link I get from parsing HTML, the link from the WARC header sometimes does not match. Moreover, sometimes the link in HTML is NULL. Hence I have decided to use the link extracted from the WARC header. In addition to these, by extracting the link from the WARC header, I can apply filter earlier, and I have applied it. Also as you will see in the next section, I could apply filtering by SQL commands for the link together with the date and title. Yet I wanted to show the diversity of what I have learned during the course. So in addition to the advantage of early filtering, I have applied a filter for the link earlier to demonstrate what I have learned.

Below you can see the first part, extracting the link from the WARC header. After the extraction I have put it in the array as the second element of the tuple :

```
import org.apache.commons.lang3.StringUtils
import org.jsoup.Jsoup
import org.jsoup.nodes.{Document,Element}

val clean =
     warcs.map{ wr => wr._2 }.
        filter{ _.isValid() }.
        map{ _.getRecord() }.
        //filter{_.getHeader().getDate().contains("2018-09-07")}. //filter for date
        filter{_.getHeader().getHeaderValue("WARC-Type") == "response" }.
        map{warc => {
            val url = warc.getHeader().getUrl()
            (warc, url)
            }
        }.
        filter{x => x._2.contains("bbc.com") || x._2.contains("wsj.com") ||x._2.contains("nytimes.com")}.
```

Also, you can see an example filtering by date on the Header. Yet it is commented because as we mentioned before, the date in the header is not reliable.

After we filtered the WARC records, now we extract the title and the date from the HTML of the webpage and we put them in our sequence :

```
    map{wh => (wh._1.getHttpStringBody(), wh._2)}.
    filter{ _._1.length > 0 }.
    map{body => (Jsoup.parse(body._1), body._2)}.
    map{ js => {
                    val t = js._1.title()
                    val link =js._2
                    val time = js._1.select("time").attr("datetime") //for selecting and filtering time
                    (time, t, link)
                }
        }.
    filter {x => x._1.length>0 && x._2.length>0 && x._3.length>0}
```

To achieve the extraction, I have encountered some problems. They are related to parsing HTML. It was a bit hard, I have to learn the Jsoup cookbook. Here the implementation of the extraction for the link is not visible, yet it was hard because I have to find a place where all three webpages (bbc, nytimes and wsj) have common that contains the URL of the webpage. And I have found it, you can extract it with “js_1.select(“meta [property$=og:ru]l”)”. But indeed it was hard to find this.

Now I have partially filtered data, I have turned it into a data frame. Because using data frames is easy, efficient and moreover, I have learned them in this course :

```
val webDF =clean.toDF("time", "title", "link") //.cache()
webDF.createOrReplaceTempView("web")
webDF.show(100)
```

Now I can run some SQL on it and continue filtering based on the date and the title. Do not forget that still, the column time contains the date with the type String :

```
val filtered = spark.sql(
    """
    SELECT time, title, link
    FROM web
    WHERE time LIKE '%2020%'
    AND title LIKE '%Bitcoin%'

    """.stripMargin)
filtered.show(10)
filtered.createOrReplaceTempView("filtered");
```

Here we have a filtered nice data namely “filtered”. Now we can make an aggregation on it to find the frequency distributed monthly. But we should fix an issue. You must remember about values in the time column. They are string. Moreover, they have no unique format: Some of them are like “`2020-02-02T18:47:...`” or some of them are like “`2020-07-02`”. Or even some of them are usual sentences like “11 April 2020”. So we can not aggregate them in this format. We have to find a solution that converts all these values into a standard format. And it is enough for this format to store the month value only.

The solution is UDF

I remember that I have found some built-in defined functions for these types of conversions. Yet I wanted to demonstrate what I have learned. So I have chosen to make a user-defined function (UDF), which will take the values from the “time” column and turn them into two-digit strings, like “04” which means the 4th month, April.

The main problem I have encountered here was serialization. Unfortunately, you can not use non-serializable functions in Spark. And also the functions do not extend to Serializable class. My solution was to create a serializable object (an object that extends to Serializable class). And it did work. Again I have spent some effort researching it but I have found a proper way for that.

Below you will see my implementation and its explanation:

```
import spark.implicits._
import java.lang.StringBuilder
import org.apache.spark.sql.functions.udf

object timeConverter extends Serializable{

    def toMonth(t:String): String =
    {
        if(t.length>=7  && !t.toLowerCase().contains("a") && !t.toLowerCase.contains("e") && !t.toLowerCase().contains("u")){
            val month = s"${t(5)}${t(6)}"
            month}
        else{
            val month = "0"
            month
        }

    }

    val toMonthUDF = udf((f: String) => toMonth(f))
}
```

The function is very simple if its length is bigger than 7 (which means there will be a month, because of the format YYYY-MM contains 7 characters), and if it does not contain any letter of “a”, “e” and “u” (and this is for the month values in words, and if you think there is no month that does not contain any of these letters), then we get the month by extracting the 6th and 7th digit and return it. If these conditions are not met, we return zero.

The reader may question that this approach may cause to lose a huge amount of data, and make the frequency not real. Also, I have questioned that but I have some arguments against it, proving this approach is safe.
The first argument is about the distribution of these “outliers”. Only a small percentage (maybe %2) contains an irregular value (i.e. literal string like June 2020, wrong format MM-YYYY or corrupted). So it does not affect the generality.
The second argument is about our target cluster. I have looked for many bbc, nytimes, and wsj webpages for their time value. All of them contain the format of YYYY-MM-DDTHH:MM… Some of them (especially before 2012, I don’t know why) contain only YYYY-MM-DD format, which is also fine.
Hence this approach is safe.
Below you can find the application of this function to our data frame “filtered” :

```
val converted = filtered.withColumn("month", timeConverter.toMonthUDF($"time"))
converted.createOrReplaceTempView("converted");
converted.show(100)
```

```
|2020-02-25T19:46:...|                  | Fbuz74.ru|http://fbuz74.ru/...|   02|
|2020-11-14T08:58:...|The CE Academy Ar...|http://fermynwood...|   11|
|2020-12-30T17:28:...|Feature Requests ...|http://forum.xeth...|   12|
|2020-08-17T03:15:...|Sensualica Chatur...|http://freeporndr...|   08|
|2020-08-21T09:49:...|Ventajas de InSal...|http://fundaciong...|   08|
|2020-10-04T00:01:...|4 octombrie 2020 ...|http://fundatiase...|   10|
|          2020-12-20|   navy strength gin|http://gianpaolog...|   12|
|          2020-04-03|The Proven Method...|http://goldbackfu...|   04|
```

As you can see, in the left column, we see the different formats for time. Yet on the right column, their month value is extracted correctly.

This is the end

At this point, the only thing left is an aggregation. What we are going to do is we are going to group the data by month. But we will not get the months with a value of 0. Because they are invalid. Then we will count the values in each group. And we will have our frequency by month. Also, we can order in ascending order so that we can start with January. You can see the implementation below :

```
val grouped = spark.sql(
    """
    SELECT month, Count(title) as frequency
    FROM monthh
    WHERE month != '0'
    GROUP BY month
    ORDER BY month ASC
    """.stripMargin)

grouped.show(10)
```

Unfortunately, I can not show the results, because I couldn’t find a WARC record containing the word “Bitcoin” in 2020. But I have tested my implementation with other combinations : with other words, years and site links. It works correctly, which made me happy.

It has been a long way

Before jumping into the problems I have encountered in the cluster, I want to talk a bit about the problems I have encountered in Spark. Some of them are already mentioned like extractions and Java heap space error. It is best to list them all and their possible solutions.

The first one is about Java heap space error. For this two solutions worked. First I have enlarged the executor and the driver memory. Second, I have removed caching.

The second one is about Serialization. It took me some time to find a solution. The clearest solution is creating the function inside of an object that extends to Serializable class.

The third one is about the Jsoup parser. It was hard to find a common attribute contained in all web pages. But <time datetime/> row worked for the time. And if I were going to use, I could use the column <meta property =og:url /> row in HTML.

The fourth one is about finding the correct sample. As I have mentioned none of the three WARC files contained the word “bitcoin”. But since I have to test my code, I have looked for the titles and the links in the WARC records. I have test with the title containing “b”, the date 2020 and it was working. I have tried different combinations. For example I have run with the title containing “workers”, and the link “bbc.com” and indeed it gave me the correct output because there were only two web pages with the link “bbc.com” and one of them carries the title containing “workers”. So I can say that my implementation worked.

It is definitely not the end

After I have finished my code, I was very excited, I just want to run it on the cluster and see how does that work. But first I should understand how the cluster works. This may sound simple and easy but it was not for me.

First of all, I have to change my code, so that there will be nothing unnecessary (for example show()), and it should be compatible with the sbt. For example, the file path for the source must be correct and it took me a day for it. Also, I have to think of a sink format. I have found turning the final “grouped” data frame to csv file is a good idea. So I have made “grouped.write.csv().
In summary, it took me some time especially debugging to be made compatible because the help was limited, and the online sources were not enough. Yet the TAs, the teacher, and my colleague students helped me.

Secondly, sometimes the cluster was so full that it took hours for my program to be submitted. I have tried running in the early mornings when there are fewer/no people running on the cluster.

And in the end, my code worked on the cluster also. But it took too long. For example, again I have run my code on the same WARC files I have run on Spark, but this time it took more than 1.5 hours, and in the end, I canceled it.

Why it didn't work as expected?

I have thought about this and made some research and run tests. Here is the list of possible reasons why it takes too much time :

1.) It can be noticed that I use lots of contains() and LIKE. Because I have to find the corresponding values for the link, the date, and the title. But I know that the function contains() and SQL LIKE really takes too much time, according to my research.
2.) In addition to using contains() for each filtering, I use multiple contains() for each filtering. For example, I use three different contains() for filtering the link
3.) Moreover, the parameters that are used in contains() and SQL LIKE are relatively long strings like "bitcoin" which contains 7 chars.
4.) Maybe the code structure is not optimal. For example, I could filter the date and the title right after extracting them. But actually, I don't think it will matter much. Because still, we have to use contains() rather than SQL LIKE, so the time complexity will remain.
5.) Also, I think I can delete one column from the data frame, but again I don't think that it will reduce the time complexity that much.
6.) Unfortunately, I have always worked with the Strings and I believe this also affects the time/space complexity.

Even though I have indicated the possible problems, there are no solutions for them. For the possible problems [1], [2], and [3] there are no solutions because for my project I have to apply that many filters. So at this point, it seems I have exchanged the complexity of my project for time complexity; I could choose to filter on only one link, a smaller word, and maybe an older date. But I was a bit afraid to do too simple program.
For the problem [6], I thought maybe I could turn all these strings to bytes, but then the program would be too complex, and that scared me.

At this point, the reader might ask this question: "But wait for a second, you have indicated that your code runs on Spark?" Yes, this is not wrong but it is not complete.
First, as I have mentioned before, while I was doing some tests on Spark, I used too many contains() and SQL LIKEs. Most of the time, I have filtered at most two from three (title, link, date). Also, the words I was using for contains() or SQL LIKE were very short, most of the time two chars like "ba". Additionally, I was not using multiple contains() for one filter, for example, I was just looking if the data contains the link "bbc" or not, I was not looking for also containing wsj and nytimes.
Second, on Spark, I have worked iteratively, which means I have worked node by node, running cells, not sequential. Between the runtime of two cells, I was doing debugging, researching, thinking about better/novel approaches for efficiency, etc. Hence I have never waited for 1.5 hours. It probably would never take that long because of what I mentioned in the previous paragraph, yet maybe it had taken 20-30 minutes in total. But I have never measured the total time. That's why I didn't notice unexpected things while working and testing on Spark.

What I have learned from this project?

Before starting this project, of course, I knew what Spark was. I have already worked on it while doing Assignments 3 and 4. Yet, I was always given code and instructions. But this time all the engineering and architecture part was left to me. I believe I have successfully put a good architecture.

Beyond Spark itself, also I have learned extra information and experience about specific packages. For example, I have learned how to use Jsoup or parse WARC record headers.

I believe that this project was a merging of what I have learned during the course, especially from the practicals. In my opinion, it is very valuable.

In addition to all of these, I have learned that rather than writing the program, applying it successfully to big data is much more important and hard. I think I will never forget that.