Overview

This practical analyses the JSON data file made from Twitter using Hadoop API. In order to finish the practical, understanding about the structure of Hadoop API was needed. Also, the structure of a JSON file and javax.json API had to be fully understood. Lastly, based on the explanation about JSON on Twitter, I had to find the result that I needed. For this weeks practical I attempted in all extensions and successfully done it.

MapReduce

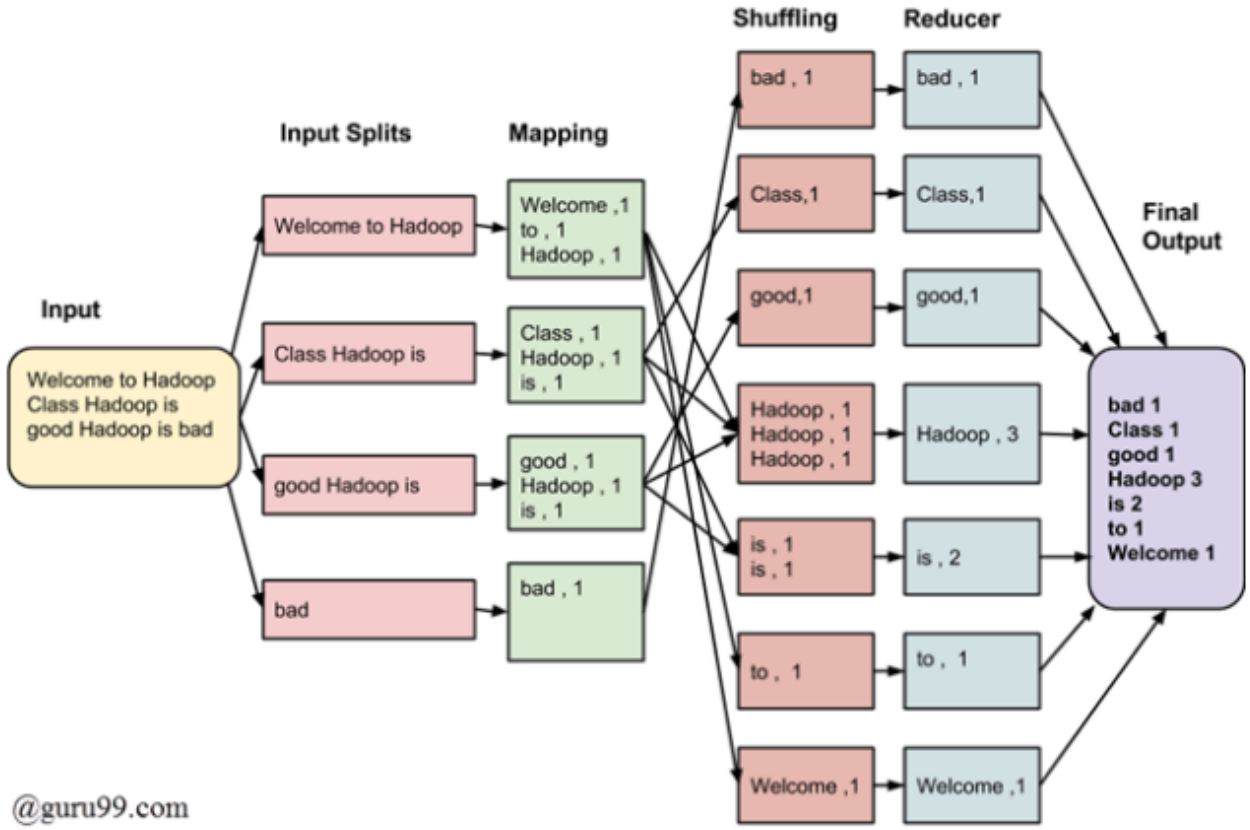

Image from https://www.guru99.com/introduction-to-mapreduce.html

The basic way of Map-Reduce in Hadoop was to read each sentence, map them and reduce (reduce contains shuffle and sort) them. However, in this practical, we had to Map-Reduce a JSON file originated from Twitter.


```
{"created_at":"Thu Jun 02 02:01:00 +0000 2016","id":738188637259632643,"id_str":"738188637259632643",
{"created_at":"Thu Jun 02 02:01:00 +0000 2016","id":738188637259628550,"id_str":"738188637259628550",
{"created_at":"Thu Jun 02 02:01:00 +0000 2016","id":738188637255405569,"id_str":"738188637255405569",
{"created_at":"Thu Jun 02 02:01:00 +0000 2016","id":738188637242789888,"id_str":"738188637242789888",
{"created_at":"Thu Jun 02 02:01:00 +0000 2016","id":738188637242855424,"id_str":"738188637242855424",
{"created_at":"Thu Jun 02 02:01:00 +0000 2016","id":738188637272199172,"id_str":"738188637272199172",
```


(Figure 1. JSON file generated from Twitter)

The JSON file made from Twitter is not an ordinary JSON file. As can be seen in Figure 1, IntelliJ shows that it is a wrong JSON file. Originally a JSON file is supposed to have one root element. However, this problem is not a significant problem when Map- Reducing. The reason is because as I told above, Hadoop reads each sentence and does the mapping. Therefore, in this case, since every sentence parses a JSON root element, even though there is a problem like Figure 1 it doesn't matter.

CS1003 W11 Practical 160005106 Tutor: Alice Lynch 27 Apr 2018

Design

W11Practical.java
This file Map-Reduces the result made when a JSON file is parsed.
The result is to display how many specific words (in this case expanded_url) appeared on Twitter.

- Main method.
  This method uses Map-Reducing by using an object named Job.
  In order to do this Map-Reducing you have to initiate a mapper class and reducer class.

ScanWordsMapper.java
This class extends the Mapper class.
Mapper class contains four generics which are in order key input, value input, key output, and value output.

- map(Key, Value, Context)
  This method gets three parameters which are key,value, and context.
  Key and value have to be same variable types as Mapper generics which are key input and value input. Value is one line of a JSON file. This value will be read as a JsonObject (root element) by using JsonReader Object. After that, the programme will extract the Json object named “entities” form the root element. However, this entities object is nullable, so that the programme has to check whether the entities object is null or not. If this object is not null, the programme will continue extracting “urls” which is a JsonArray from the entities object. Also, urls is nullable. If it is not null, by using the iterator the programme will get the final result, which is the “expanded_url” JsonObject. This result may not be String, hence there is a need of checking whether the result is String value or not. If all of these conditions are satisfied, the programme will save the result as a key with a value of 1

CountWordsReducer.java
This class extends the Reducer class.
Reducer class contains four generics which are in the following order: key input, value input, key output, and value output. An noticeable point is that the Reducer variable type of key input and value input have to be the same variable type as Mapper key output and value output.

- reduce(Key, Iterable values, Context)
  Key and values have to be the same variable types as Reducer generics, which are key input and value input. Also, this key is the word from the mapping result and values are all the counts associated with that word. After receiving these values, the programme will count the occurrence of words. Finally, these counted value and the key will be saved as the result.

Testing

```
Testing CS1003 Week 11 Practical
- Looking for submission in a directory called 'src': Already in it!
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_no_arguments/progRun-expected.out : pass
* TEST - basic/Test02_data-very-small-00/test : pass
* TEST - basic/Test03_data-very-small-01/test : pass
* TEST - basic/Test04_data-very-small-all-files/test : pass
* TEST - basic/Test05_1_minute_of_data/test : pass
* TEST - basic/Test06_10_minutes_of_data/test : pass
* INFO - basic/TestQ_CheckStyle/infoCheckStyle : pass
--- submission output ---
Starting audit...
Audit done.
---
```

Passed all stacscheck.

Evaluation

While doing this weeks practical, I personally think that I fully understood the concept of parsing a JSON file. While for the Map-Reduce, I felt as if I was just scratching the surface.

Conclusion

At first, when I was trying to use Hadoop API, it was quite difficult. However, after briefly understanding what the Map-Reduce is, it was possible to make a straightforward programme.

Extension1 – USAGE

java -cp “lib/*:bin” W11PracticalExt <input_path> <output_path> <true_or false>

Extension1 – Overview

For this extension, I made a programme that can order the urls from the main practical based on occurrence frequency.

Extension1 – Design

I used two ways to get to this goal. The first way is to make a new Mapper class to reverse the order of Key and Value from the output of the basic practical. After reversing the order, by using a new Reducer class, I appended the urls when there are same number of counts. The second way is to make just one Reducer class. In this way, the Reducer class will reverse the order of Key and Value.

Extension1 – Testing

```
"http://twipi.org/084015116ca1", "http://bit.ly/2cma7La"        15


 "http://du3a.org"       107
 "http://d3waapp.org/"   43
 "https://youtu.be/nwmSo7D9S_Y"  39
 "http://ghared.com"     29
 "http://www.totobet1.com"       28
 "http://7asnat.com/"    24
 "http://bbc.co.uk/inauguration" 22
```

Extension2 – USAGE

java -cp "lib/*:bin" W11PracticalExt2 <input_path> <output_directory> <query> <true_or_false>
First argument: input path
Second argument: output path
Third argument: query – (text, user, hashtag and location)
Fourth argument: true or false (boolean value)

Extension2 – Overview

In this extension, I made the programme possible of performing several other queries on the data. This includes showing the text of the most re-tweeted Tweet, displaying the user with the most frequently re-tweeted tweet, the most used hash tag, and the location that had the most tweets created.

Extension2 – Testing

Retweeted text

```
"#615Resist means organize, create, and protest."       2
"#AmericaFirst \n\nDr. Seuss 1941: https://t.co/V6zD4QTIo2"     70
```

Retweeted text – the most retweeted text

```
"It's been the honor of my life to serve you. You made me a better leader and a better man."    13585006
```

Extension3 – Overview

In this extension, I made the programme Map-Reduce by using not only one map but several maps at the same time. Therefore, it is possible to Map-Reduce more than one sentence at one time.

Extension3 – Analyse

- CPU: Intel® Core™ i5-4440 CPU @ 3.10GHz
  - CPU: 4
  - Thread per core: 1
  - Cores per socket: 4
  - Core: 1

```
9140 output: milli seconds - output1
5119 output: milli seconds - output2
4085 output: milli seconds - output3
4167 output: milli seconds - output4
4148 output: milli seconds - output5
4119 output: milli seconds - output6
4229 output: milli seconds - output7
4113 output: milli seconds - output8
4225 output: milli seconds - output9
```

(Figure 2. the outputs by different max map)

	1 map	2 maps	3 maps	4 maps	5 maps	6 maps	7 maps	8 maps	9 maps
Average Time (ms)	9728.9	5536.3	4259.9	4468.9	4110.3	4172.4	4241.7	4220	4169.6

(Table 1. average of 10 times testing for each of the 9 maps)

Extension3 – Conclusion

When I first encountered this extension of this practical, I thought that there would be a relationship between the result and the number of cores in a CPU because Map-Reduce works in a parallel programming environment. However, the result was different with what I expected. The result that I expected was that when I used four maps it would take the shortest time. However, the fastest result was when five maps were used. Since the result was suspicious, I tried nine times more and the average of the result is Table 1. Finally, if I had a better performing CPU, it would have took less time to Map-Reduce.