CS839 NoSQL Systems

Assginment 2

29 March 2023

Submitted By

Group 19

1. IMT2020502 - Monjoy Narayan Choudhury
2. IMT2020509 - Hardik Khandelwal
3. IMT2020548 - Tejas Sharma

Way to run given files

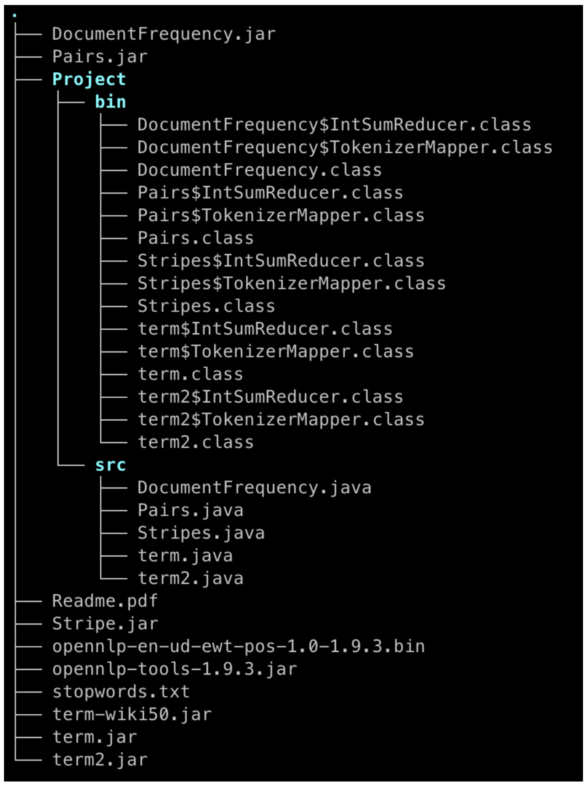
1. We have given JAR files which have been compiled in Eclipse with the default class specifically mentioned. So in place of the command given in the doc, we use the following command:
hadoop jar <JAR file> <Input directory> <Output directory>

The output is stored in the output directory specified. The input directory in the above command refers to the directory where the Wikipedia articles are stored.

Name of jar files/File Structure

1. Problem 1: part1: Pairs.jar
2. Problem 1: part2: Stripe.jar
3. Problem 2: part1: DocumentFrequency.jar
4. Problem 2: part2: term.jar, term2.jar which is implemented using stripes as discussed in class.

2. However note the following that all dependencies in the code are on an absolute path so to actually run them on your system you will have to remake these jars yourselves in eclipse. The code files lies in the project folder inside src.



```
.
├── DocumentFrequency.jar
├── Pairs.jar
├── Project
│   ├── bin
│   │   ├── DocumentFrequency$IntSumReducer.class
│   │   ├── DocumentFrequency$TokenizerMapper.class
│   │   ├── DocumentFrequency.class
│   │   ├── Pairs$IntSumReducer.class
│   │   ├── Pairs$TokenizerMapper.class
│   │   ├── Pairs.class
│   │   ├── Stripes$IntSumReducer.class
│   │   ├── Stripes$TokenizerMapper.class
│   │   ├── Stripes.class
│   │   ├── term$IntSumReducer.class
│   │   ├── term$TokenizerMapper.class
│   │   ├── term.class
│   │   ├── term2$IntSumReducer.class
│   │   ├── term2$TokenizerMapper.class
│   │   └── term2.class
│   └── src
│       ├── DocumentFrequency.java
│       ├── Pairs.java
│       ├── Stripes.java
│       ├── term.java
│       └── term2.java
├── Readme.pdf
├── Stripe.jar
├── opennlp-en-ud-ewt-pos-1.0-1.9.3.bin
├── opennlp-tools-1.9.3.jar
├── stopwords.txt
├── term-wiki50.jar
├── term.jar
└── term2.jar
```

Figure 1: Files

Problem 1: Part-of-speech tagging using Pairs and Stripes

Pairs Approach

In the pairs approach (as discussed in class), The mapper finds each co-occurring pair and outputs it with a count of 1. The reducer just adds up the frequencies for each pair. This requires the use of complex keys (a pair of words). However in the given case where we have to count the value of POS tags for all documents present, it makes no sense to use pairs as a paradigm explained above. According to that we would have created a co-occurance words and their POS tag which in the end is not in coherence with what the problem statement wants.
What we decided to do is that we create a pair of the form (POS TAG, 1) based on the POS tag obtained from a document and emit such pairs out. The reducer will merge this to give the total number as the output.

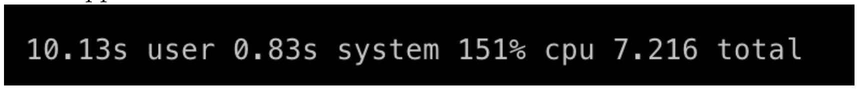
Stripes Approach

In the stripes approach, there is a map which contains the number of occurrences of each tag in a particular line. Then for each tag present in the map, the mapper emits a key-value pair of (tag, map).
The key value emitted by the mapper is a POS-tag. Thus, each reducer must add only those corresponding key values in the map that is passed. For example, if the key value given to a reducer is NOUN, then the reducers task is to add the number of occurrences of all NOUN in the map across all key-value pairs. This result is stored in another map. This map contains the tag and the number of occurrences of that tag in the given input.
We then emit tag, number of occurrences of that tag as the output from the reducer.

Performance Comparison

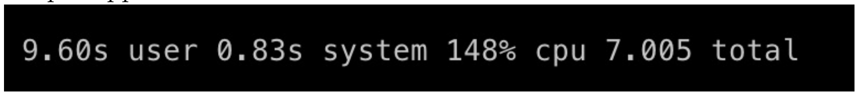
Time taken:

1. We tried to run on the whole dataset but this took more than 5 minutes. Moreover, it was also causing the laptop to overheat. Thus we decided to only run for the wikipedia 50 dataset.
2. Pairs approach -



```
10.13s user 0.83s system 151% cpu 7.216 total
```

3. Stripes approach -



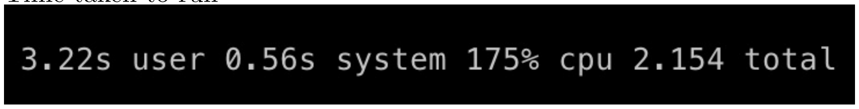
```
9.60s user 0.83s system 148% cpu 7.005 total
```

Problem 2: Indexing Documents via Hadoop

Part 1: Computing Document Frequency

1. For mapper we first tokenize the sentence, filter the stop-words, and put the **lowercase version** version of the word to the Porter-Stemmer(). We then emit the following key-value pair for every word that comes out of the stemmer - (word, docId).
2. For Reducer we take the intermediate key and create a set that contains the docIDs where the word is present. We populate this set by adding all docIDs which contain the key. The size of this set is the document frequency of this word. We emit the result as the following - (word, documentFrequency)
3. We did not perform other preprocessing tasks like removing punctuation. This explains why we observed outputs for words like -"work" and "work."

4. Time taken to run -



```
3.22s user 0.56s system 175% cpu 2.154 total
```

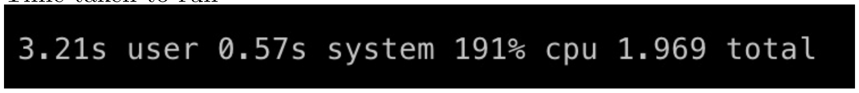
Part 2: Computing Term Frequency and Score

1. In mapper, we perform the same stemming and preprocessing that we performed for the previous task. Using the stripes approach, we are emitting the following key-value pair : (DocName + term, term frequency).

2. For reducer, first the setup function is called that reads the tsv file generated by the previous part and stores and term and its document frequency in a hashmap. Then, in reduce function, we use the output that we got from the previous task to calculate the total term frequency for each combination of term and document. After aggregating, we calculate the score for this combination and emit the following pair : (DocName + term, score).

3. We also observed that the outputs are by default tab-separated and we don't have to specify it.

4. Time taken to run -



```
3.21s user 0.57s system 191% cpu 1.969 total
```

5. **NOTE:** The above doesn't follow a typical stripes program so we decided to modify the code in the following way:

   (a) For the mapper, we emit the key-value pairs as the following - (docID, map) where the map corresponds to the term frequency of each word found in that document.

   (b) For the reducer, we take the output from the mapper and create a map that stores the word and the term frequency of that word. This map is computed by iterating through all values of all maps which are in the input to the reducer. The reducer then takes the values of this map and calculates the score of TF-IDF given by the formula and the document frequency (already calculated and cached) and emits the following for each word - (docID + word, score for that word)