DAS839-NoSQL Assignment-2 (Part B) Report

Siddhesh Deshpande-IMT2022080
Abhinav Kumar-IMT2022079
Jinesh Pagaria-IMT2022044
Krish Patel-IMT2022097

March 23, 2025

Problem 4: Co-occurring Word Matrix Generation

a) Top 50 Most Frequently Occurring Words (Pairs Approach)

1. Distributed Cache was used to make stopwords.txt file available to all the mappers so that stopwords can be ignored. Inserted all the stopwords in a hashset so that we can efficiently check if the word is a stopword or not.

2. Min-Heap Data Structure was used in the reducer side to keep track of top 50 most frequently occuring words efficiently.

Runtime and Output:

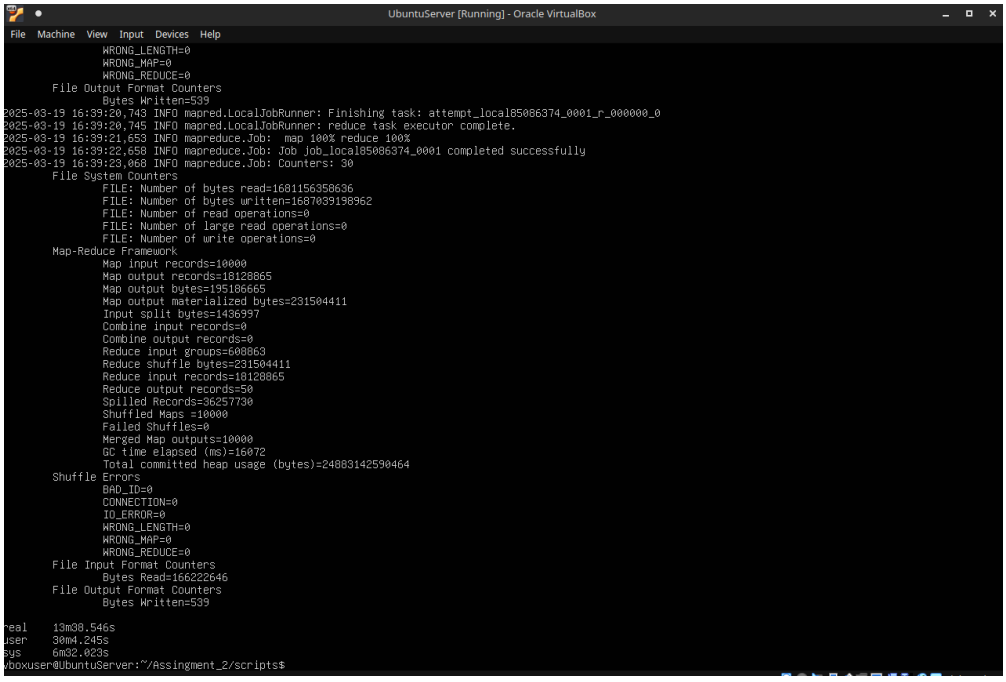

Figure 1: Q4.a

b) Co-occurring Word Matrix (Pairs Approach)

Objective: Create a co-occurring word matrix using the pairs approach for distances $d = \{1, 2, 3, 4\}$.

Runtime Analysis:

- $d = 1$:
- $d = 2$:
- $d = 3$:
- $d = 4$:

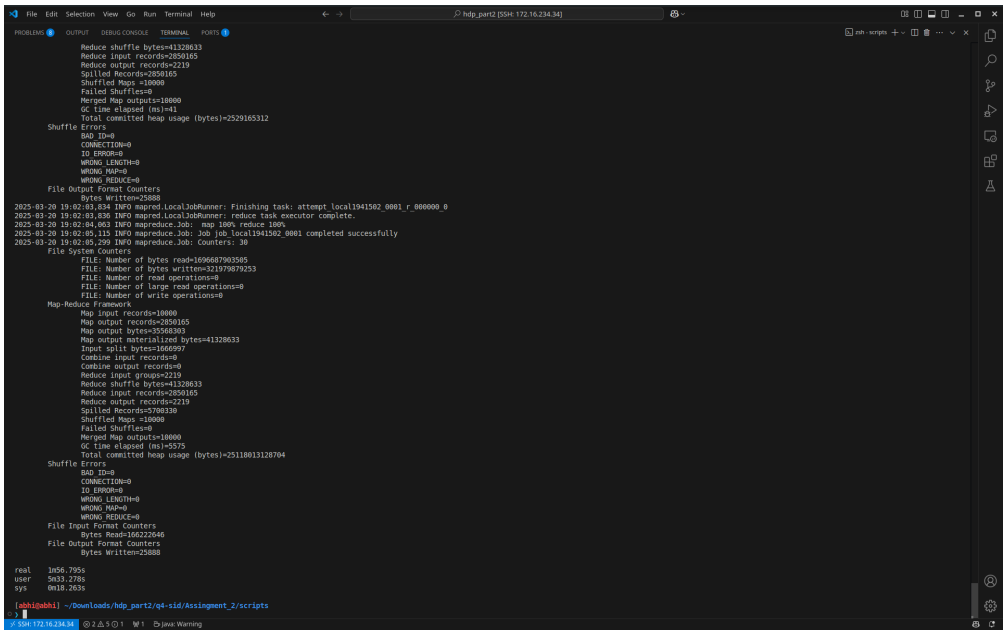

Figure 2: Execution time for $Q4.b$ with $d = 1$

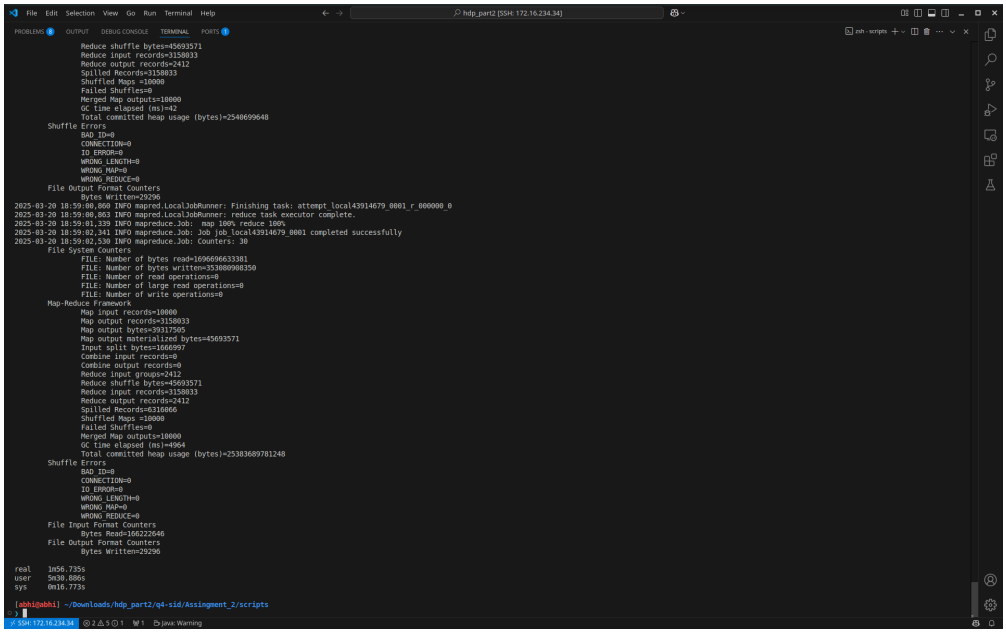

Figure 3: Execution time for $Q4.b$ with $d = 2$

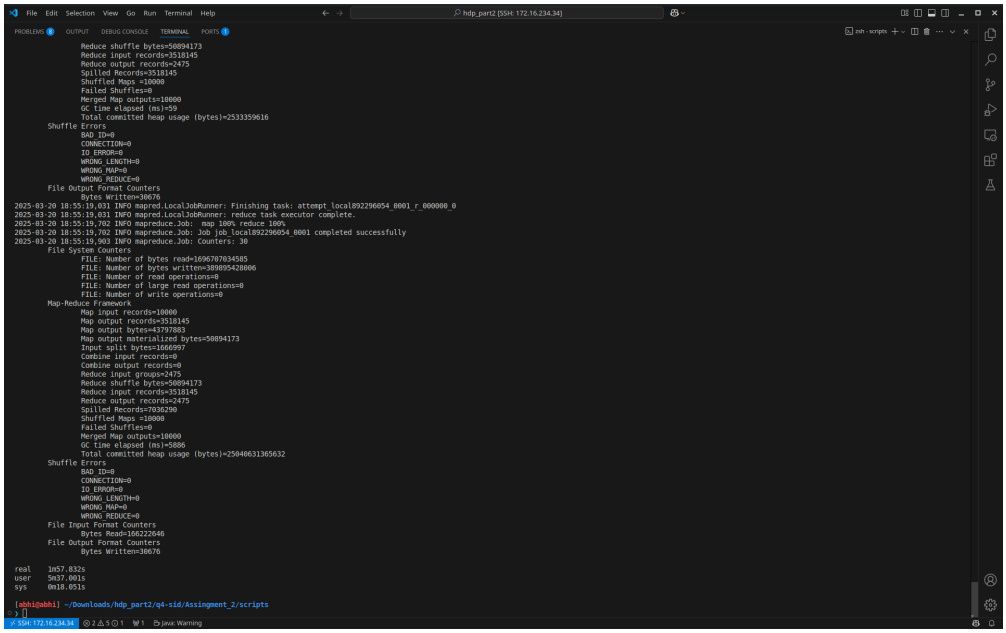

Figure 4: Execution time for $Q4.b$ with $d = 3$

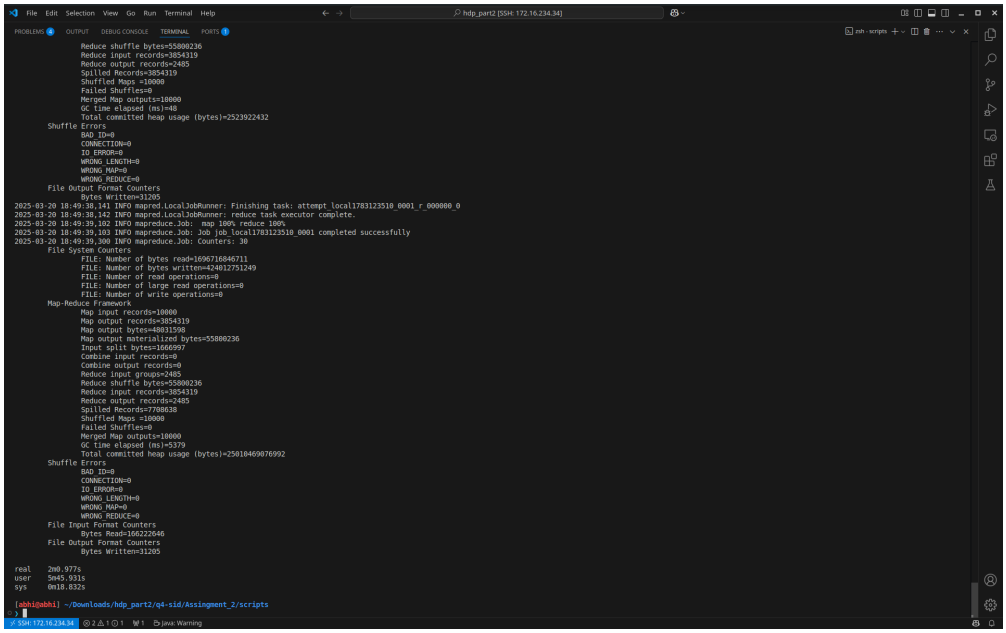

Figure 5: Execution time for $Q4.b$ with $d = 4$

c) Co-occurring Word Matrix (Stripe Algorithm)

Objective: Construct the co-occurring word matrix using the stripe algorithm with distances $d = \{1, 2, 3, 4\}$.
Runtime Analysis:

- $d = 1$:
- $d = 2$:

- $d = 3$:
- $d = 4$:

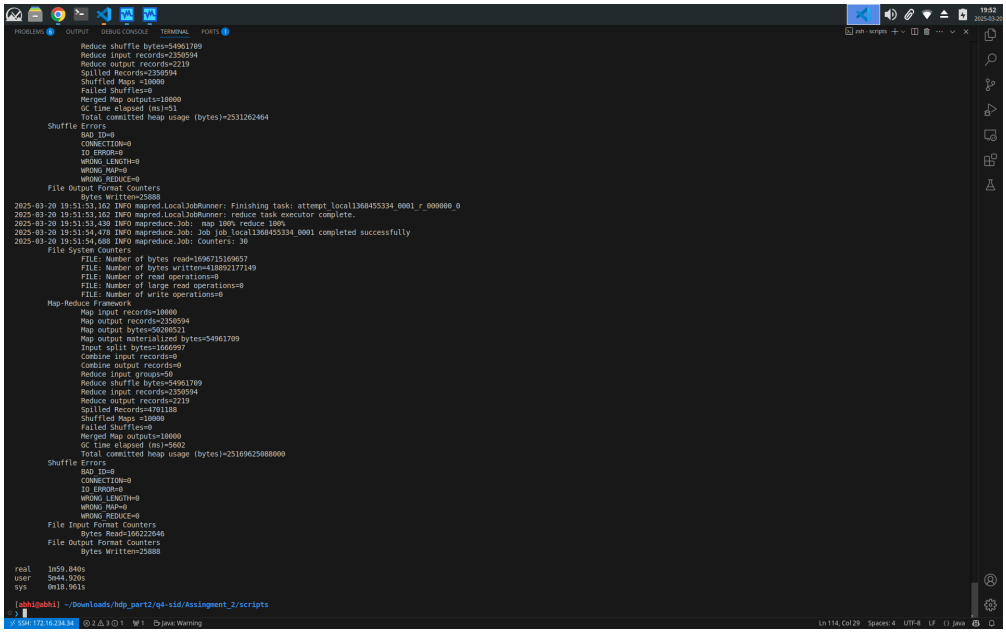

Figure 6: Execution time for $Q4.c$ with $d = 1$

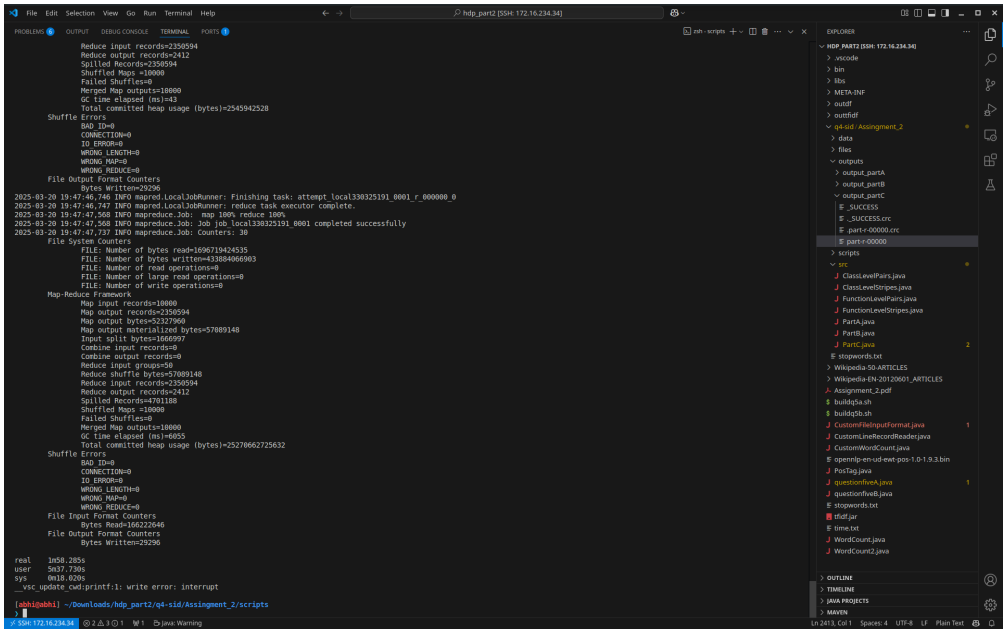

Figure 7: Execution time for $Q4.c$ with $d = 2$

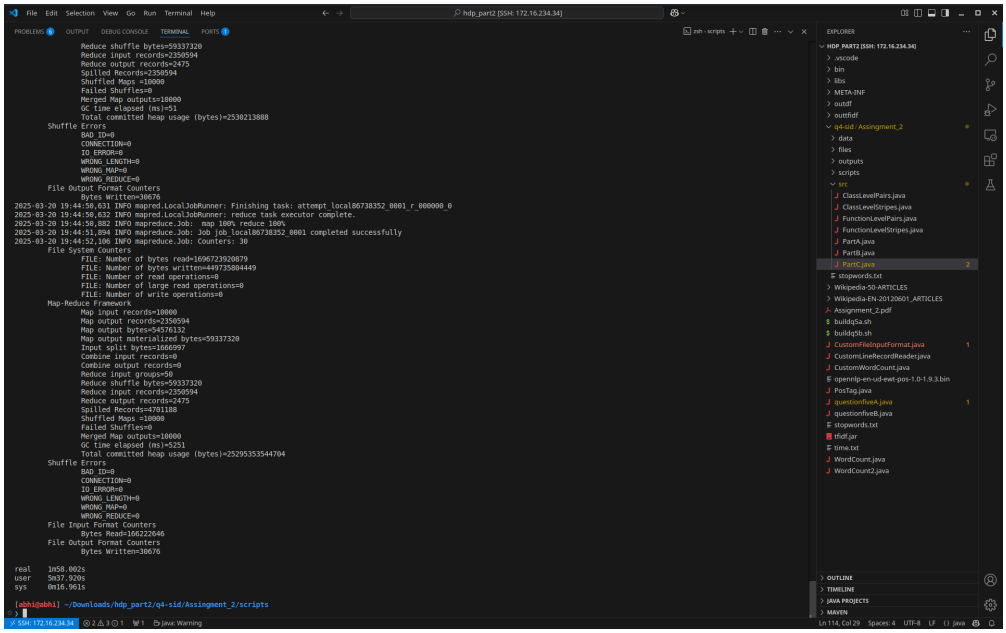

Figure 8: Execution time for $Q4.c$ with $d = 3$

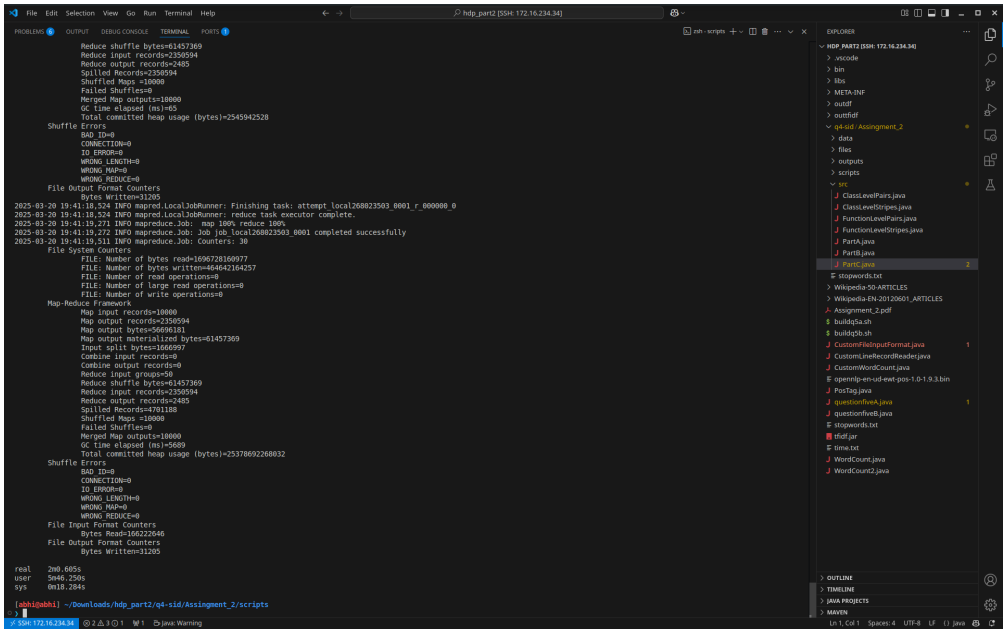

Figure 9: Execution time for $Q4.c$ with $d = 4$

d) Local Aggregation Performance Comparison

Objective: Compare performance between Map-class and Map-function level aggregation.

1 Results for Q4.d

1.1 Function Level Pairs

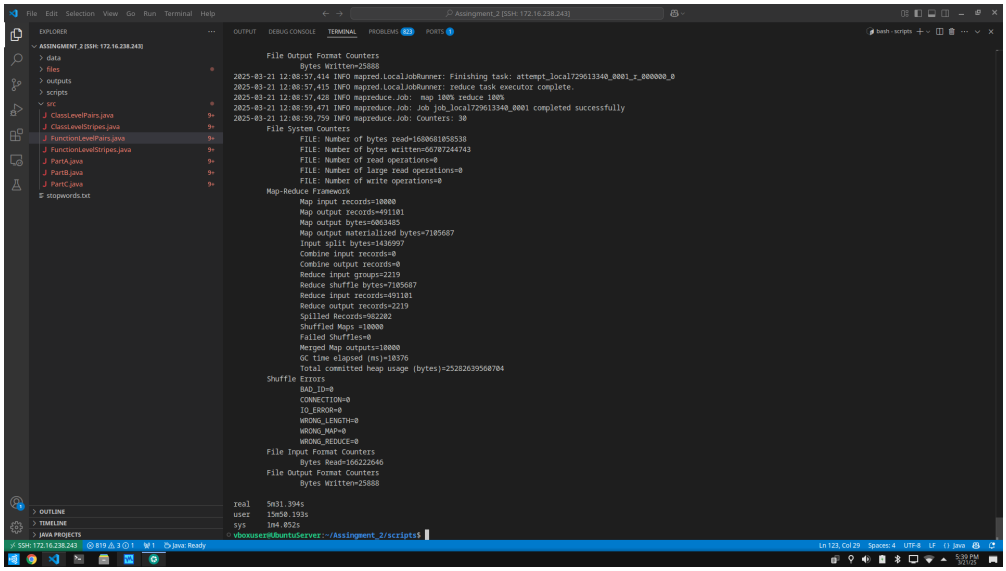

Figure 10: Function - level pairs $d = 1$

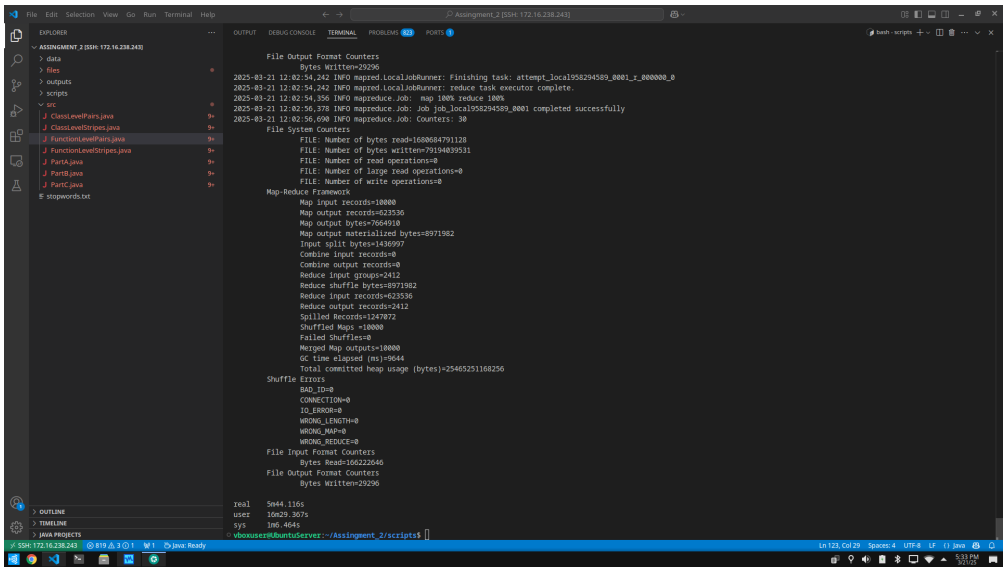

Figure 11: Function - level pairs $d = 2$

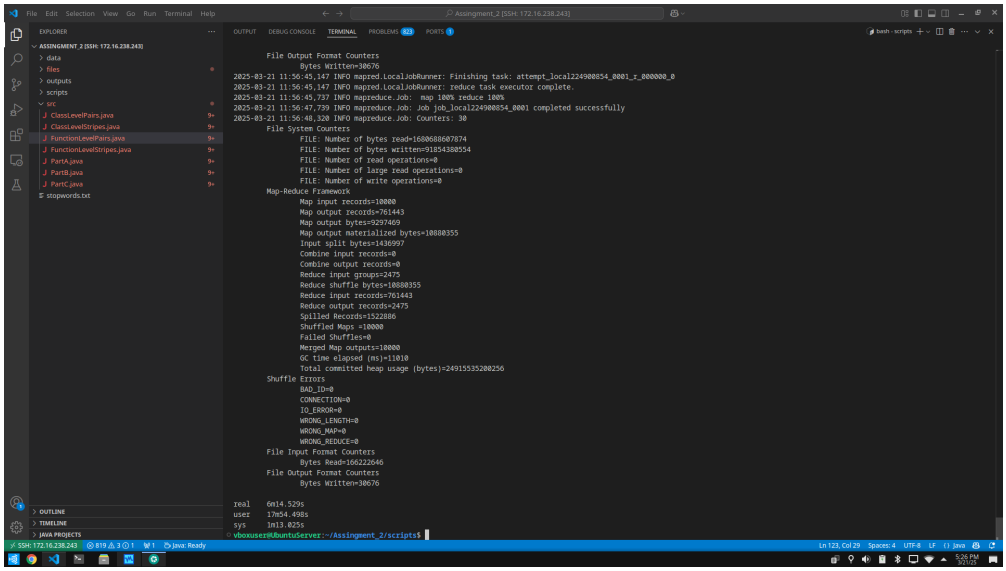

Figure 12: Function - level pairs $d = 3$

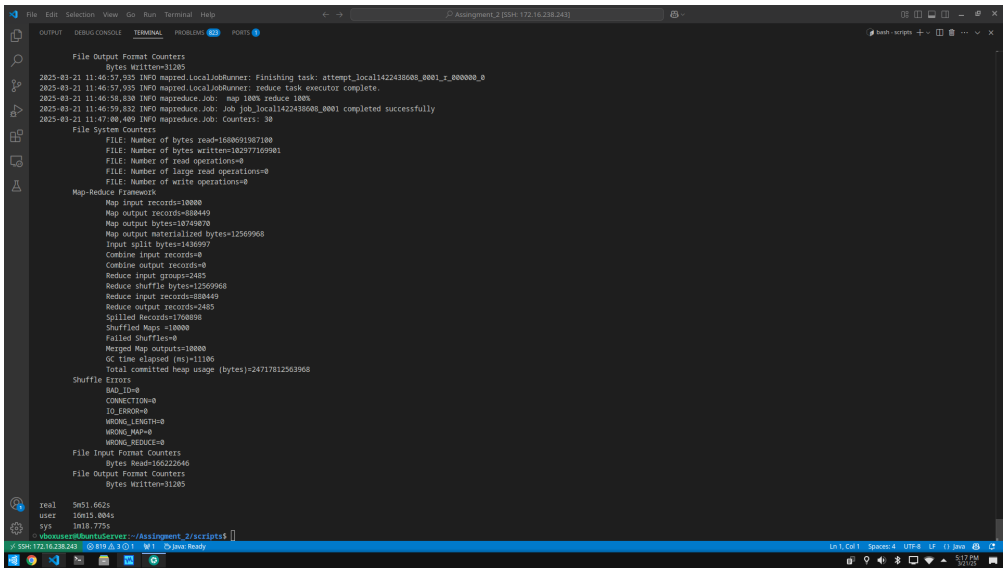

Figure 13: Function - level pairs $d = 4$

1.2 Class Level Pairs

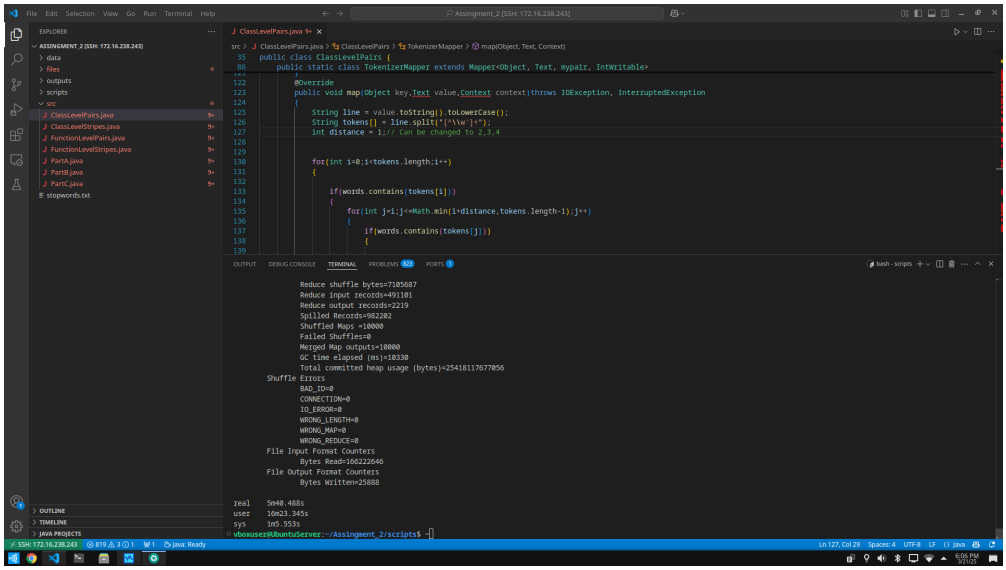

Figure 14: Class - level pairs $d = 1$

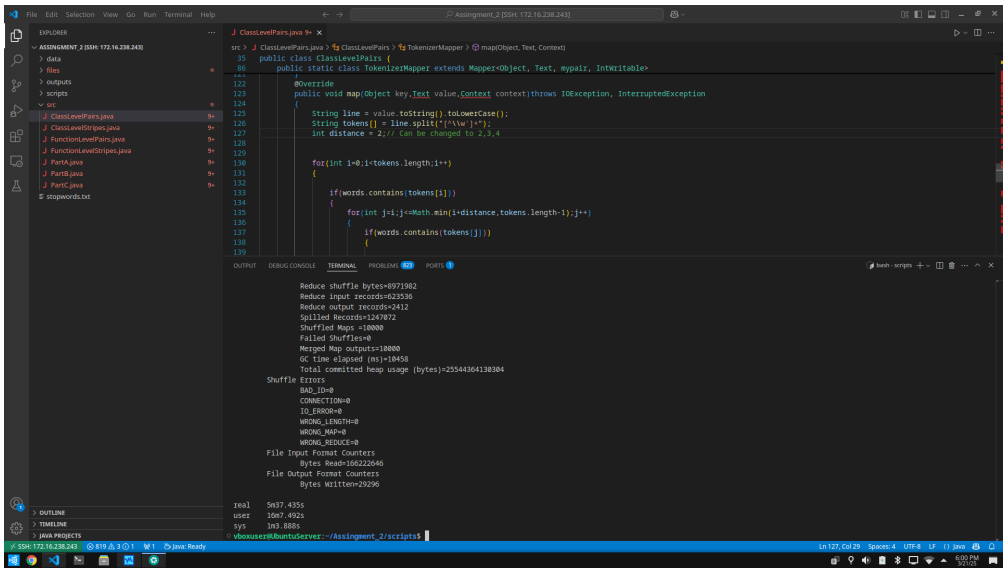

Figure 15: Class - level pairs $d = 2$

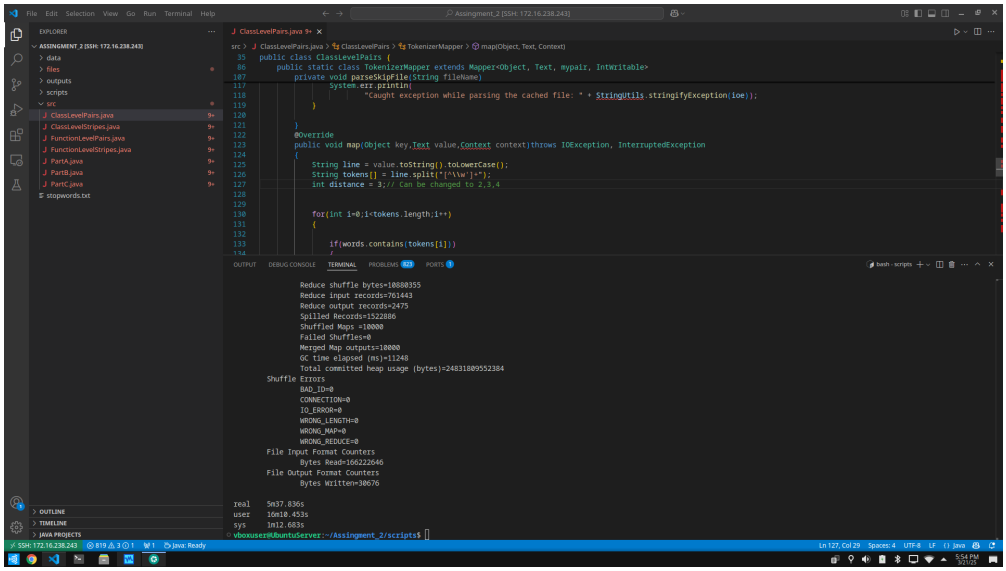

Figure 16: Class - level pairs $d = 3$

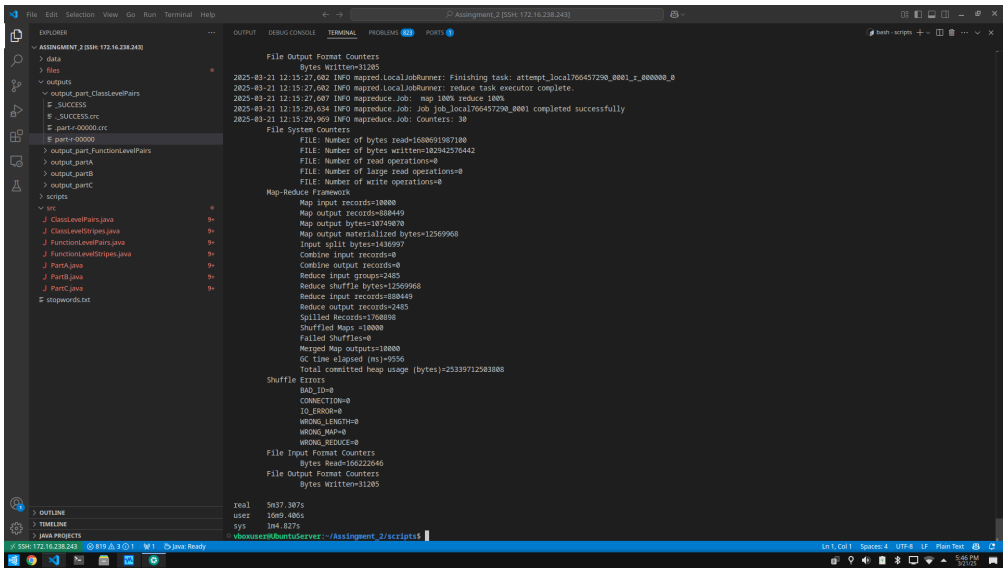

Figure 17: Class - level pairs $d = 4$

1.3 Function Level Stripes

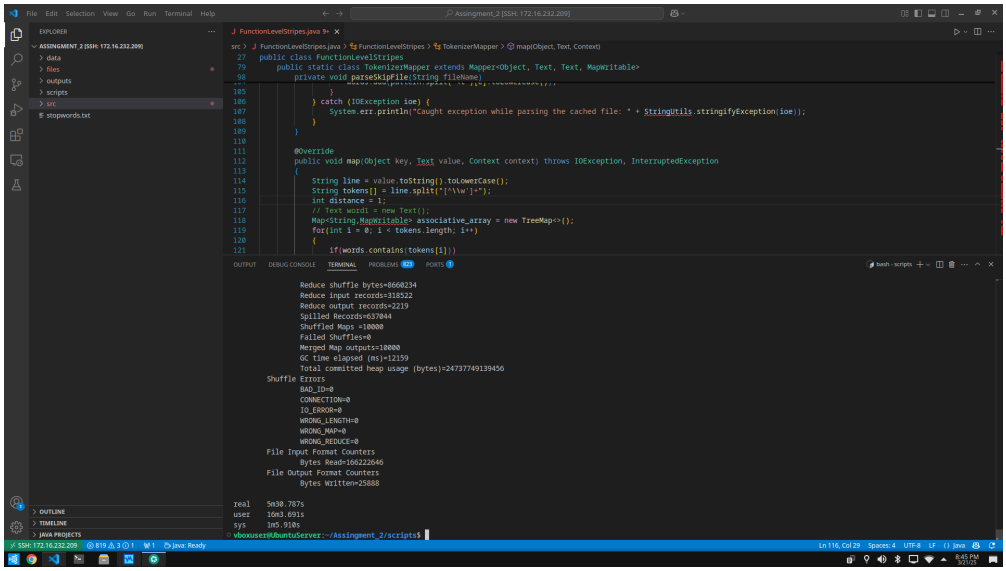

Figure 18: Function - level stripes $d = 1$

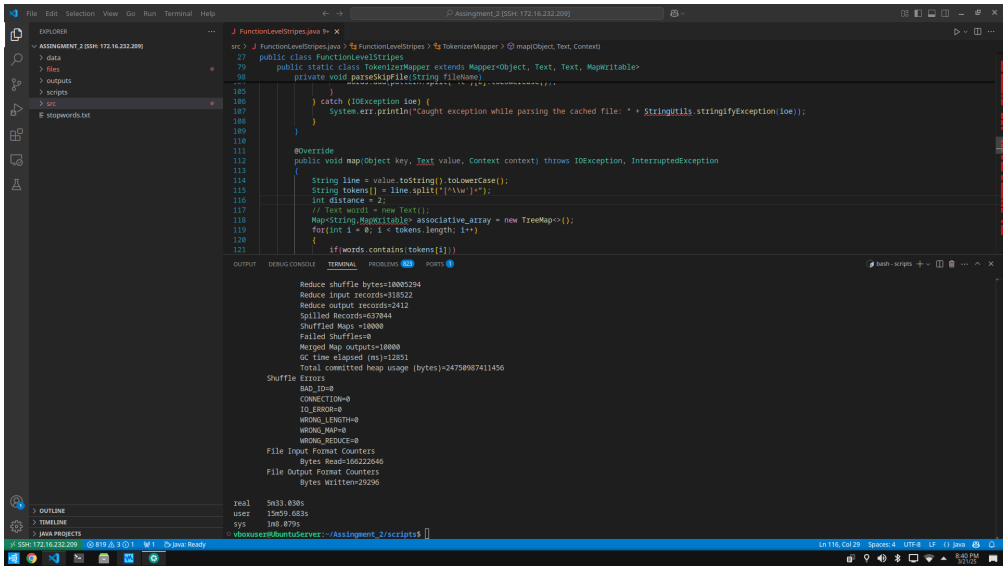

Figure 19: Function - level stripes $d = 2$

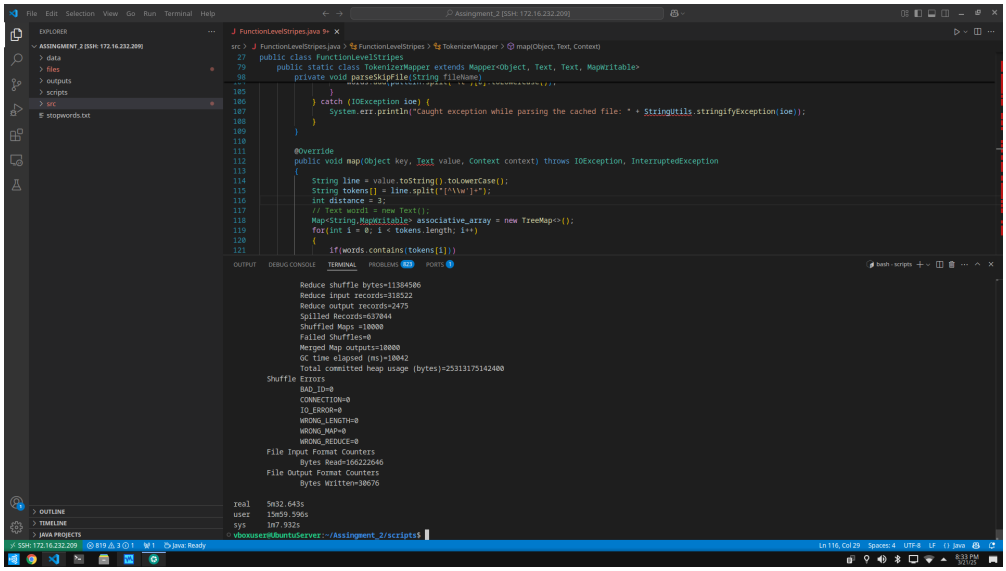

Figure 20: Function - level stripes $d = 3$

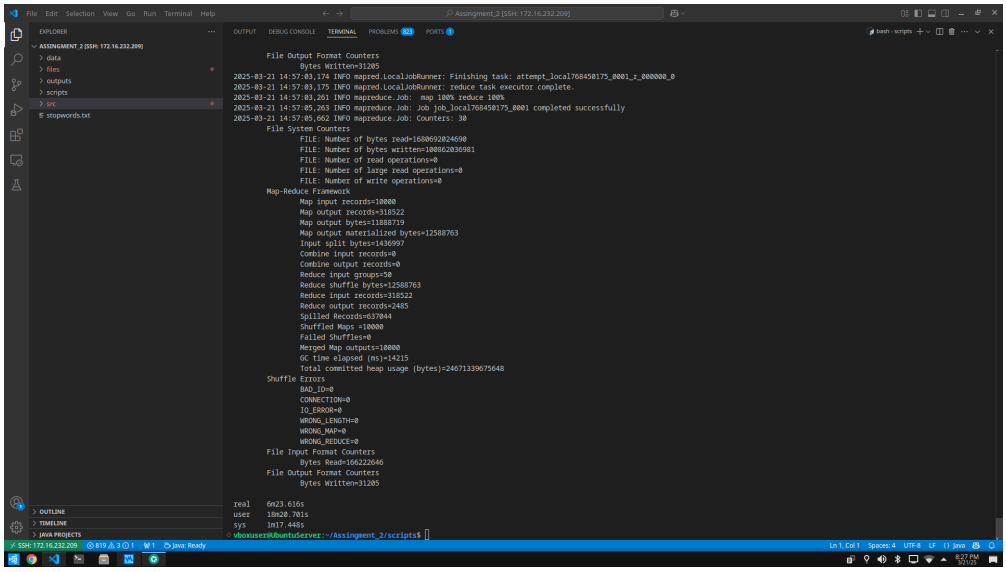

Figure 21: Function - level stripes $d = 4$

1.4 Class Level Stripes

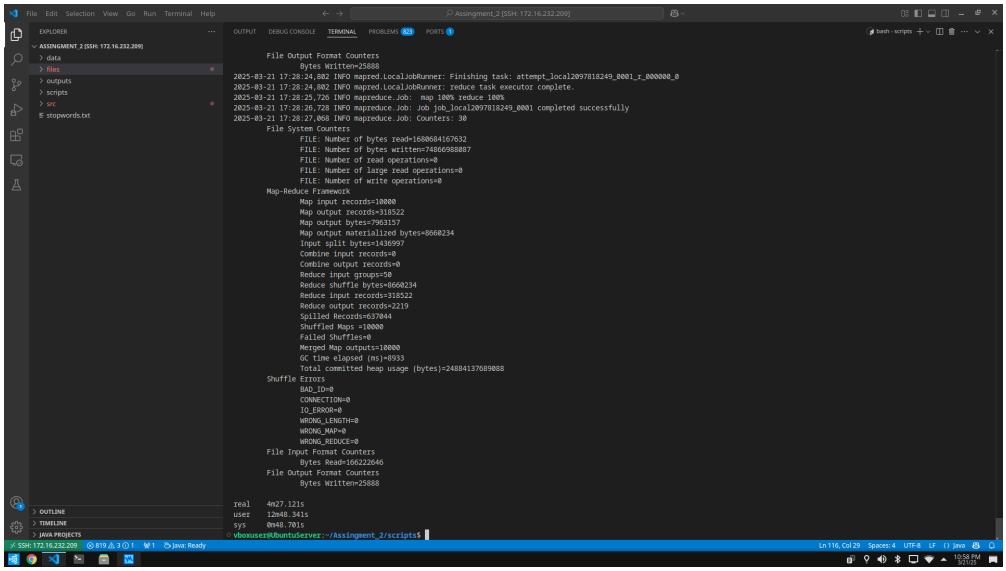

Figure 22: Class - level stripes $d = 1$

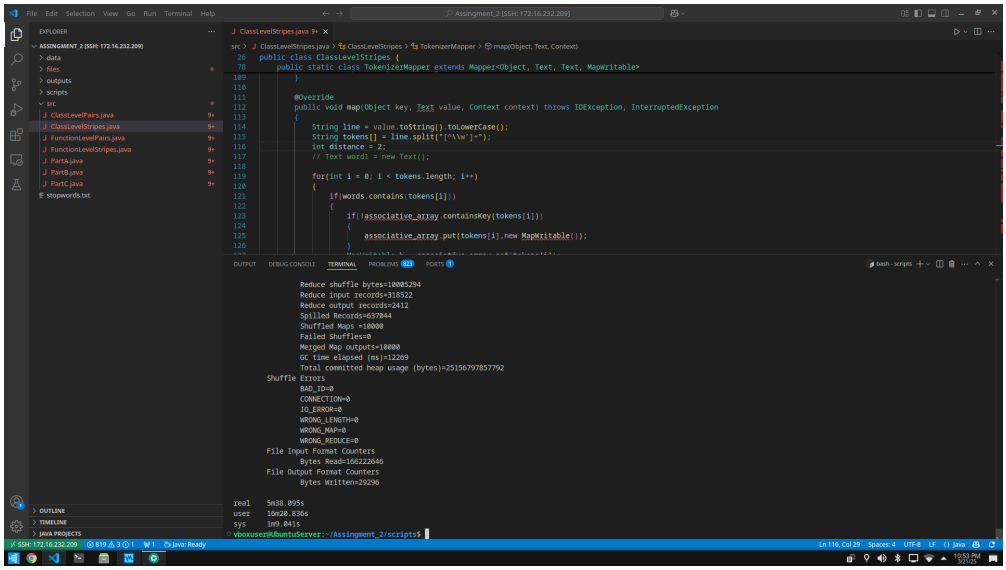

Figure 23: Class - level stripes $d = 2$

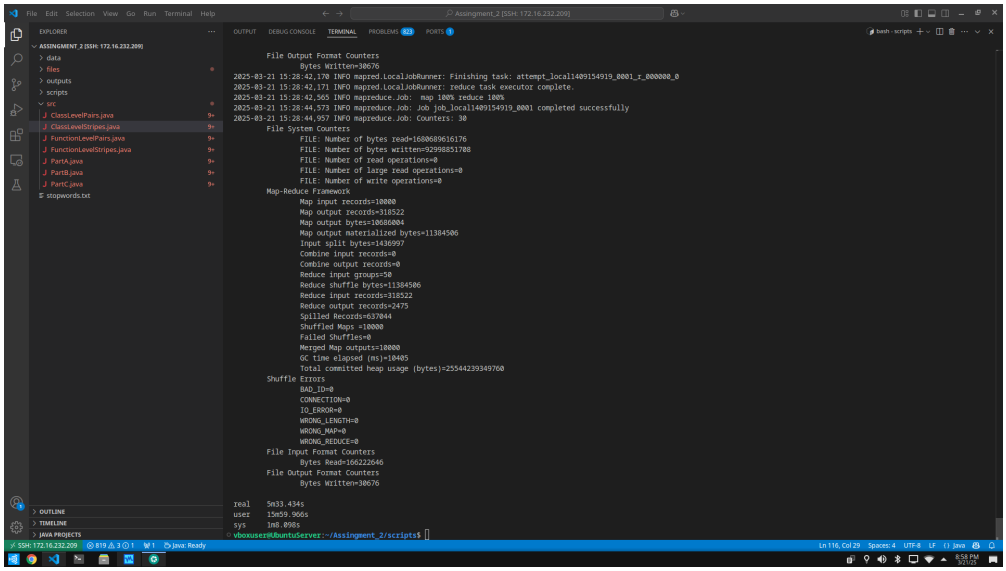

Figure 24: Class - level stripes $d = 3$

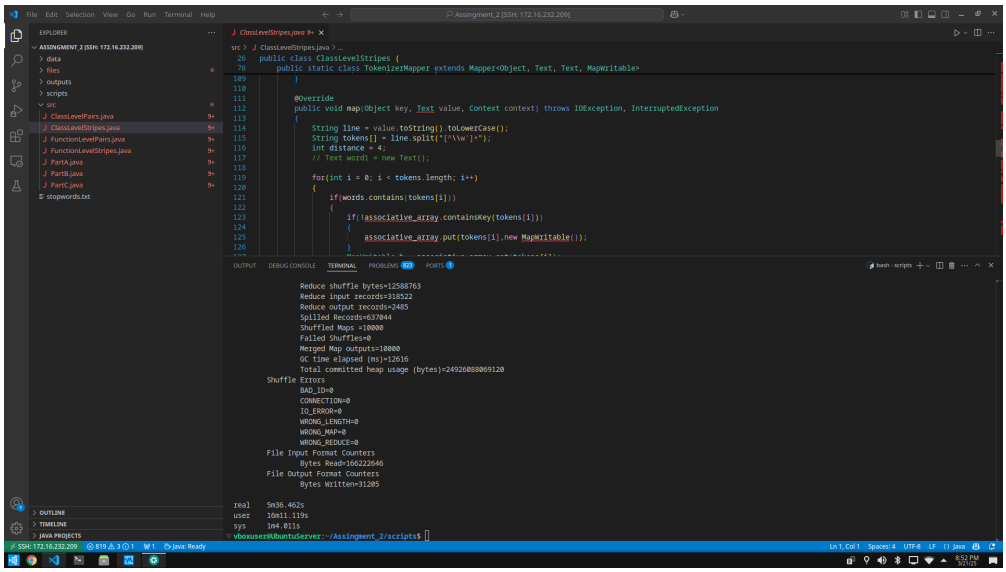

Figure 25: Class - level stripes $d = 4$

Problem 5: Indexing Documents via Hadoop

a) Document Frequency (DF) Calculation

Objective: Implement MapReduce to calculate the DF of each distinct term, filtering out stop words.

Execution Steps:

1. Preprocess the Wikipedia dump.
2. Use MapReduce to count DF.
3. Store the DF values in a TSV file.

Output Format:

```
TERM<TAB>DF
data    3456
hadoop  2890
map     2678
```

Map :

- We read *stopwords.txt* and store all the words in the file in a Set for filtering.
- We read the input, split the line into words and stem using *PorterStemmer* from *openlp-tools* and convert to lowercase, We allow those words which are not in *stopwords.txt* and only distinct words using Set datastructure.
- We filter words using sets by sending (word,1) when word occurs the first time (not in set) and add it to the set i.e. we send a word only if it is not in the set and also stopwords are filtered using sets.

Reduce :

- We declare a priority queue inorder to obtain top 100 words with highest document frequency.
- We receive values for a key which we sum the values (we get 1 for each document) and store the key and sum in the priority queue and if the number of elements in priority queue is more than 100 we poll it i.e. the key value pair with least document frequency is popped.
- We output all the key sum pairs stored in the priority queue as it contains only top 100 words which have highest document frequency.

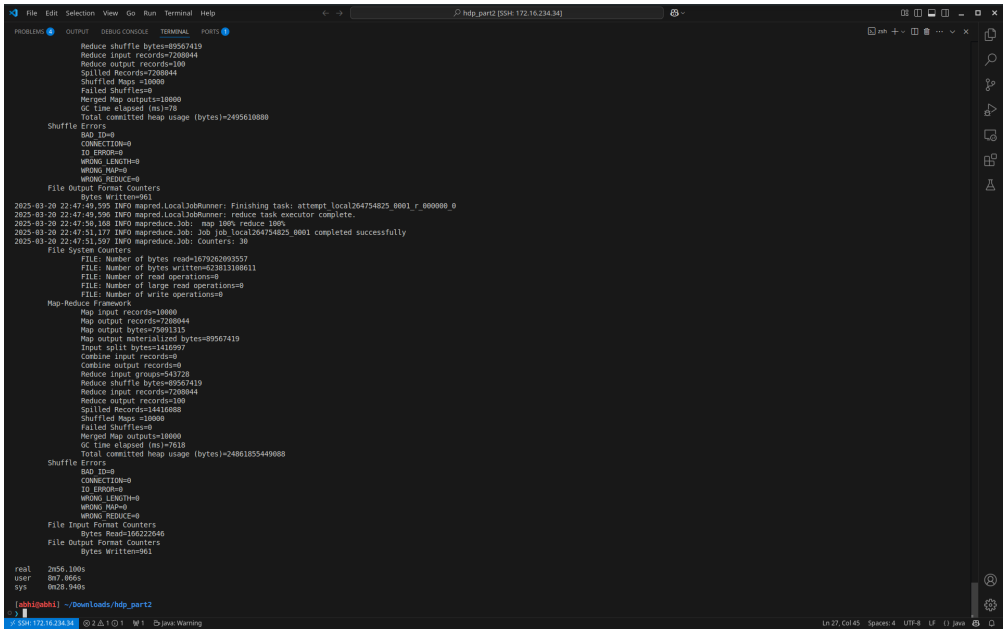

Figure 26: Q5.a

b) Term Frequency-Inverse Document Frequency (TF-IDF)

Objective: Calculate the TF-IDF score using MapReduce with stripes algorithm.

Execution Steps:

1. Preprocess the Wikipedia dump.

2. Use MapReduce with stripes to compute TF.
3. Cache the DF file using `Job.addCacheFile`.
4. Calculate TF-IDF:
$$\text{SCORE} = \text{TF} \times \log\left(\frac{10000}{\text{DF}+1}\right)$$
5. Store results in a TSV file.

Output Format:

```
ID<TAB>TERM<TAB>SCORE
123     data    0.678
124     hadoop  0.592
125     map     0.502
```

Map :

- We extract the id of file from its name. We then read the output file from previous map reduce ans store the words in a set (top 100 words containing highest document frequency).
- we read the lines from input files ans split them into words and stem them using *PorterStemmer* from *opennlp-tools* and convert them to lower case and if the word is in the set having words with highest document frequency the we store it in a Map (if map already has the word we add 1 to its Term Frequency or else we put (word,1) in the map).
- we release the map (having word, tf) as value and document id as key.

Reduce :

- We then read the output file from previous map reduce ans store the words and their document frequency in a Map (top 100 words containing highest document frequency).
- for a key(id) we receive a Map (stripes) we iterate over the map (value) and calculates its score using formula :
$$\text{SCORE} = \text{TF} \times \log\left(\frac{10000}{\text{DF}+1}\right)$$
and here DF is document frequency is obtained from map containing words with highest document frequency.
- for each word we release a key value pair as (id, (word,score)) (word is obtained from map (value) format : `ID<TAB>TERM<TAB>SCORE`).

Execution Time:

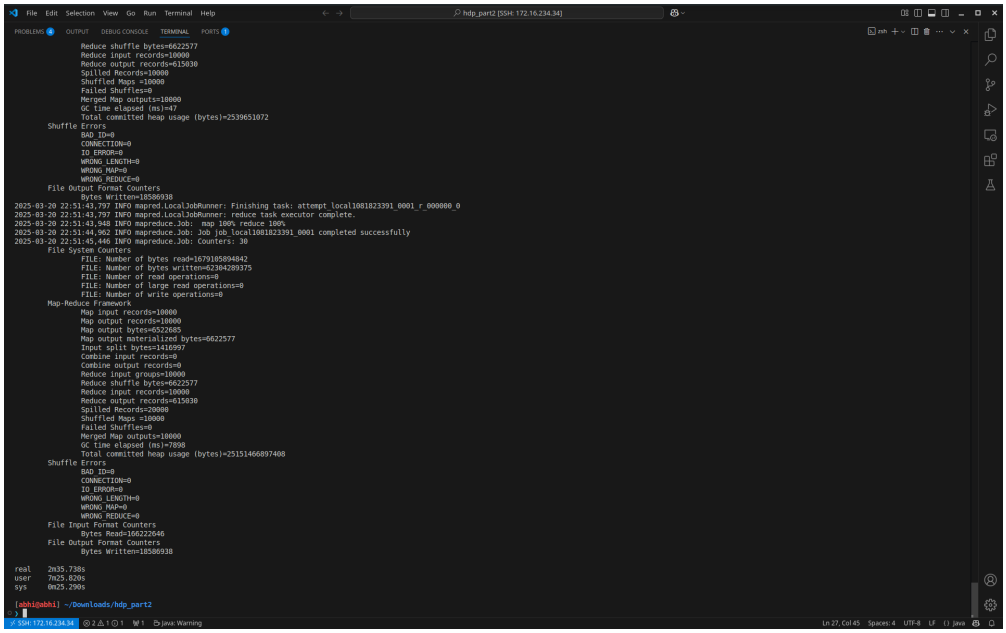

Figure 27: Q5.b

System Configuration

The Following is the configuration of the system that was used for running the mapreduce task for problem 4 and 5.

1. RAM:10000 MB
2. No of Cores : 9 Cores
3. CPU : Intel i7 12th gen

Note-These are the configurations of the virtual machine that we used for running mapreduce tasks.