Implement a distributed system that calculates the word frequency in a text

Background

Knowing the frequency of words in a text is a very useful basic task in many applications, such as search engines, machine learning, statistics, etc.

Objective

Given some text files, count how many times each word appears in the file.
The output should be a list similar to a dictionary data structure with keys being words from the text and the values being their frequency in the text.

Bonus

The dictionary should be sorted based on the value, meaning that words that appear more frequently come first.

Example

Input

file1.txt

```
At Aalborg University, we believe that knowledge can and must change the world. Our
search for knowledge is always in concert with the wider world, engaging with real
problems and missions to achieve sustainable solutions.
```

file2.txt

```
These missions are the driving force behind our work.
```

Output

```
freq = [the: 3, knowledge: 2, and: 2, world: 2, our: 2, with: 2,
missions: 2, at: 1, aalborg: 1, university: 1, we: 1, believe: 1,
that: 1, can: 1, must: 1, change: 1, search: 1, for: 1, is: 1,
always: 1, in: 1, concert: 1, wider: 1, engaging: 1, real: 1,
problems: 1, to: 1, achieve: 1, sustainable: 1, solutions: 1, these:
1, are: 1, driving: 1, force: 1, behind: 1, work: 1]
```

What to do

Client

- Implement a client application that runs on your laptop that accesses several files from a folder called `input`.
  - The files will be named `file1.txt, file2.txt, …, fileN.txt.`
- The client counts how many files are in the folder `input` and creates as many threads as the number of files.
- Each thread has a unique ID that starts at 1 and goes up to N, where N is the number of files.
- The thread with ID=K will be responsible for reading the `fileK.txt`.

- Implement a **gRPC service** called **`FrequencyCalculator`**.
- Each thread calls a **gRPC function** called **`Calculate`**, which takes as input a string and returns a list of <key, 1> pairs, where key is a word and value is simply 1.
    - The input to the function will be the content of the file.
- The main thread waits for all N threads to finish and to return their lists.
- The main thread then puts the content of all N lists into one unique list.
    - *Notice that now there might be duplicates of <key, 1> pairs in the list.*
- The main thread then calls another **gRPC function** called **`Combine`**, which combines all <key, 1> pairs with the same key by adding their values.
    - The input to the function will be the list containing all <key, 1> pairs.
    - The output will be a list of <key, value> pairs, but now there should not be any duplicated <key, value> pairs anymore since the value represents the sum of all 1s for each specific word.
- The main thread receives the output of the function and prints it out.

Server

The server is a multithreaded implementation that can serve multiple clients at the same time.
The server simply runs the gRCP service and the functions defined in the Client part.

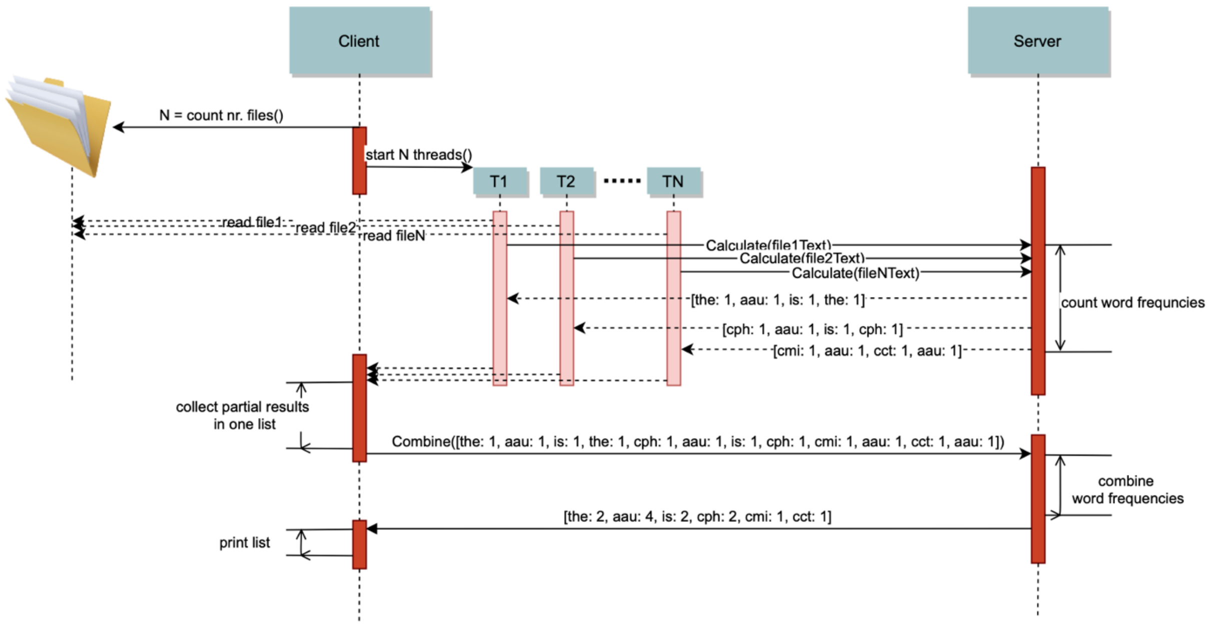
Client
Server
N = count nr. files()
start N threads()
T1
T2
TN
read file1
read file2
read fileN
Calculate(file1Text)
Calculate(file2Text)
Calculate(fileNText)
[the: 1, aau: 1, is: 1, the: 1]
[cph: 1, aau: 1, is: 1, cph: 1]
[cmi: 1, aau: 1, cct: 1, aau: 1]
count word frequncies
collect partial results in one list
Combine([the: 1, aau: 1, is: 1, the: 1, cph: 1, aau: 1, is: 1, cph: 1, cmi: 1, aau: 1, cct: 1, aau: 1])
combine word frequencies
[the: 2, aau: 4, is: 2, cph: 2, cmi: 1, cct: 1]
print list

Figure 1: Sequence diagram of the operations described by the assignment.

What to deliver

- Source code of your implementation (on Moodle)
- 1-page report describing your implementation and reflections on:
    - What did you like about this implementation.
    - What you did not like about this implementation and why (think of efficiency, programming difficulties/learning curve, etc.)
    - How could it have been done differently/better.