CMPE 300 - Analysis of Algorithms
Fall 2022
MPI Programming Project

Due Date: 20.12.2022 Tuesday 23:59

General Instructions

In this project you are expected to calculate the data for an n-gram language model, more specifically a bigram language model. Of course the most important aspect of this project will be to utilize the MPI framework and do your calculations in parallel. You can only use built-in data structures for calculations, additional libraries cannot be used. In the following section, an overview of the n-gram language model is given.

Background

N-gram

First we will explain what an n-gram is. In the topic of natural language processing, consecutive sequences of *n* words are called n-grams. The parameter *n* denotes the number of consecutive words. Given a text, beginning from the first word in the text, each consecutive n word sequence obtained by the sliding window technique is considered as an n-gram. When n=1, n-gram is called unigram (1-gram); when n=2 it is called bigram (2-gram); when n=3 it is called trigram (3-gram); and so on.

As an example, consider the text “I go to the university by bus”. The n-grams in this text for n=1, 2, and 3 are as follows:

- Unigrams: “I”, “go”, “to”, “the”, “university”, “by”, “bus”
- Bigrams: “I go”, “go to”, ““to the”, “the university”, “university by”, “by bus”
- Trigrams: “I go to”, “go to the”, “to the university”, “the university by”, “university by bus”

N-gram Language Model

Language models aim at calculating the probability distribution over textual data. A language model is used for two purposes. Given a sequence of words, a language model is used i) to calculate the probability of this sequence and ii) to predict words that can follow this sequence. It is used in various areas such as speech recognition to aid in predicting the most probable

next word during a speech act, spelling correction to come up with the most probable correction, machine translation, and many more applications.

For instance, consider the word sequence “its water is so transparent that”. We may want to know the probability of this sequence, i.e. the probability that we can see this sequence in an ordinary English text. This can be expressed as the probability $P(its\ water\ is\ so\ transparent\ that)$, and can be written as follows by using the chain rule of probability:

P(its water is so transparent that) = P(its) * P(water|its) * P(is|its water) * P(so|its water is) * P(transparent|its water is so) * P(that|its water is so transparent)

The conditional probabilities on the right-hand side above can be divided into two parts using the law of conditional probability. As an example, consider the probability $P(transparent|its\ water\ is\ so)$. It can be written as follows:

$$P(transparent|its\ water\ is\ so) = \frac{P(its\ water\ is\ so\ transparent)}{P(its\ water\ is\ so)}$$

This conditional probability is normally estimated by using the frequencies of the two sequences in the numerator and denominator above in a corpus (large collection of texts). So, the conditional probability above can be written as follows:

$$P(transparent|its\ water\ is\ so) = \frac{Freq(its\ water\ is\ so\ transparent)}{Freq(its\ water\ is\ so)}$$

$Freq(seq)$ denotes the number of times the sequence seq occurs in the corpus.

However, it is not practically possible to obtain frequencies for long sequences even if we have a very large corpus. For instance, the sequence “its water is so transparent” may not appear in a corpus. This problem can be solved by using a simplifying assumption (Markov assumption): A conditional probability can be simplified as having just a few words on the conditioned part. That is, for instance, the probability $P(transparent|its\ water\ is\ so)$ can be simplified as $P(transparent|its)$ (bigram case) or $P(transparent|its\ water)$ (trigram case), and so on.

So, the probability $P(its\ water\ is\ so\ transparent\ that)$ can be written as follows. Suppose that we use bigrams, i.e. when estimating the probability of a word, considering only the previous word is sufficient.

P(its water is so transparent that) = P(its) * P(water|its) * P(is|water) * P(so|is) * P(transparent|so) * P(that|transparent)

In this project you will not be expected to calculate sentence probabilities, you will only calculate conditional probabilities of bigrams. The information above is given only as

background knowledge. If you are interested in the details, you can take a look at Chapter 3 of the book "Speech and Language Processing" by Daniel Jurafsky & James H. Martin.

Problem Definition

You will be given a preprocessed text document which contains a single sentence per line consisting of only words. The sentences will contain special tokens <s> (start of sentence) and </s> (end of sentence) in the beginning and end to make computing the bigram counts easier. The file must be read from the command line arguments using the argument "--input_file". An example input file is given below.

```
<s> i love learning new technologies </s>
<s> new technologies replace the old ones </s>
<s> learning new programming skills is necessary </s>
```

The task is to count the frequencies of the bigrams and unigrams in these sentences using the MPI framework. For instance, for the document given above, you will count the bigrams "<s> i", "i love", "love learning", etc., and also the unigrams "<s>", "i", "love", "learning", etc.

Then these frequencies will be used to obtain conditional probabilities of bigrams that will be given as a test input to the program. The test data will be read separately from a file. Each line will contain two words separated by a single space. The file must be read from the command line arguments using the argument "--test_file". An example input file is given below.

```
new technologies
the old
new skills
```

The examples in the file above correspond to the conditional probabilities P(technologies|new), P(old|the), and P(skills|new).

The program is supposed to work in a master slave/worker architecture. There will be **P processes** where the rank of the master process is zero and the ranks of the worker processes are positive. This means that there will be **P-1 workers**. The master node will be responsible for reading and distributing the input data evenly to the processes. You can assume that the text file fits into memory. The workers are responsible for calculating the bigram data in parallel. At the end, the data will be gathered in the master node and the requested calculations (Requirement 4) will be made in the master process.

The program has to satisfy the following requirements.

Requirement 1 (10 points): The first requirement is to read the document file and distribute the data **evenly** to the worker processes. This operation will be handled by the master process. Make the data distribution as equal as possible so that there is no imbalance between the work distributed to the worker processes. For instance, if there are 22 sentences in the input file and there are 4 worker processes, then two processes should get 5 sentences each and the remaining two processes should get 6 sentences each.

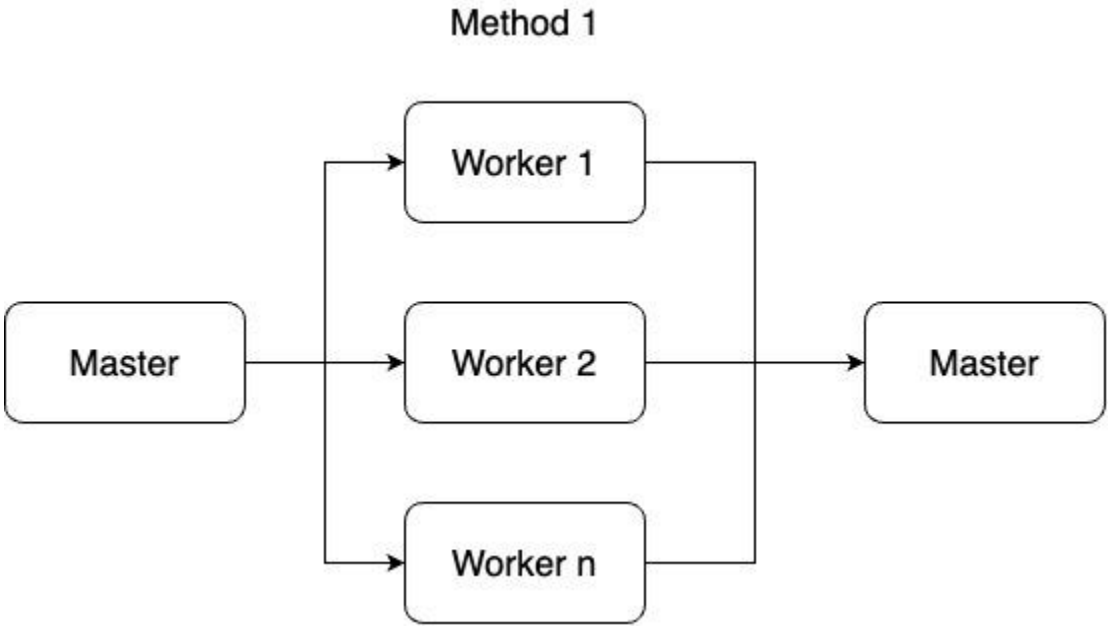

Figure 1: High level workflow for the first data merging method

Requirement 2 (25 points): After the data is distributed to the workers evenly, each worker will receive its data. The worker process **should print its rank and the number of sentences it has received**. Then, each worker will count the bigrams and unigrams in the sentences it received. This will be done by all the worker processes in parallel. Then, the bigram and unigram counts of the workers will be merged by summing the counts of the same bigrams and unigrams.

There will be two types of workflows for merging the calculated data. The workflow which is asked to be implemented in this requirement can be seen in Figure 1. After each worker has finished calculating the data, they will send it to the master process. The merge operation is the master process's responsibility and will be done at the master process. The program will receive an argument **"--merge_method"** with the value **"MASTER"**. Hence, the program will be called with "--merge_method MASTER". An example program call is given below.

```
mpiexec -n 5 python main.py --input_file data/sample_text.txt --merge_method MASTER --test_file data/test.txt
```

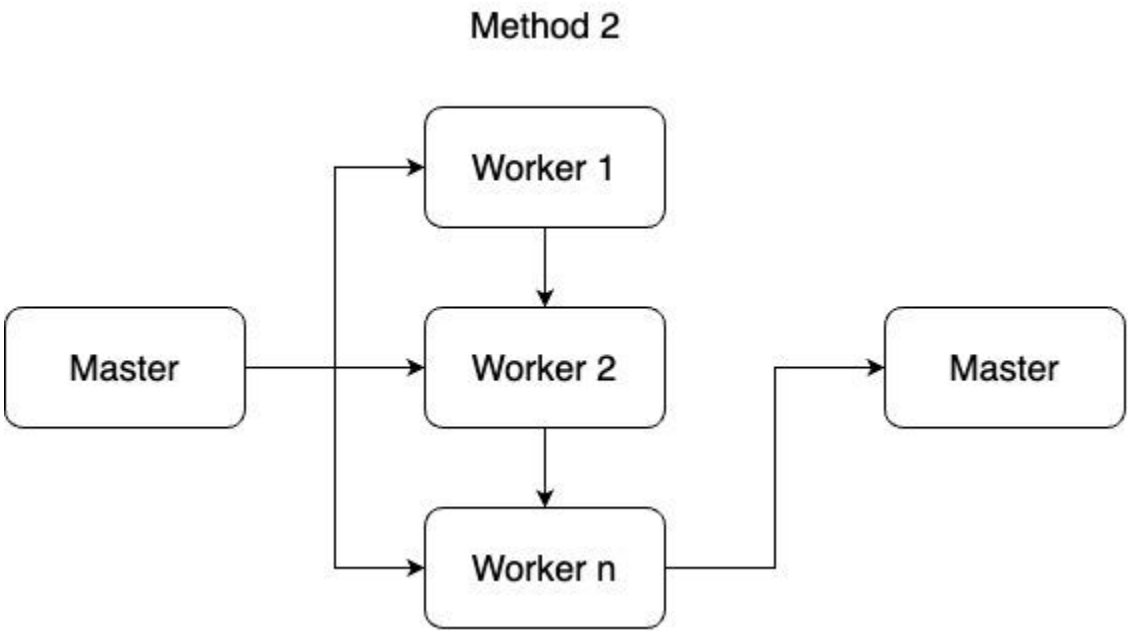

Figure 2: High level workflow for the second data merging method

Requirement 3 (25 points): In this requirement, you will implement the data merging operation sequentially between the workers as in Figure 2. Instead of passing the calculated data to the master node, each process will gather the data from the previous worker, merge that data with its own data, and pass it to the next worker. The last worker will in the end pass the final data to the master node. **Please note that Requirement 2 and Requirement 3 are independent from each other and only one method works according to the given parameter "--merge_method".** The argument value for this requirement will be "WORKERS" so the program will be called with "--merge_method WORKERS".

```
mpiexec -n 5 python main.py --input_file data/sample_text.txt --merge_method WORKERS --test_file data/test.txt
```

Requirement 4 (20 points): The master process will compute the conditional probabilities of the bigrams that are read from the input test file. The program **should print the bigrams and their conditional probabilities** to the console. Explain how you calculate the probabilities in the report.

Requirement 5 (20 points): Write a project report for your application. Include all of your design decisions, assumptions, and results clearly and thoroughly.