Practical Work 4: Word Count

Le Anh Quang

December 20, 2024

Introduction

This report outlines the implementation of the Word Count task using a MapReduce approach in Python. The script deployed with the Message Passing Interface (MPI) for parallel computation. This implementation are following the step of the MapReduce paper by Google

Reason of Implementation

The implementation was chosen for some following reasons:

- Use of `mpi4py`: This library is used for simulate the worker in MapReduce. In this implementation, each M map task also be the R reduce task
- Use of consistent hashing (`mmh3.hash`) for partitioning function: the pairs are partitioned into R regions
- A straightforward design that clearly separates the mapper, partitioner, and reducer functionalities nearly from scratch.

Implementation Details

Mapper

The mapper reads chunks of data, split the text into words, and maps each word to the intermediate key-value pair $(word, 1)$. These pairs are sent to the appropriate processes for further handling.

Partitioner

Partitioner will handle partition the data from mappers to each reducers using formula

$$\text{hash(word)} \mod R$$

where hash(word) is the hash value of the word, and R is the number of partitions. In this case, MurmurHash is used because it is non-cryptographic hash

and it generate the same hash for same word that process in parallel. Then the pair will be sent to the corresponding reducer

Shuffle and Sort

In this part, the partitioned data will be shuffle then sort in the alphabet order for easier counting in the reduce step.

Reducer

The reducer will count all the duplicated word in the given data, then export the data in the output file

Conclusion

The Word Count implementation demonstrates the power of parallel processing with MapReduce. The design ensures modularity and efficient task distribution among processes.