Practical Work 4: Word Count using MapReduce

USTH – Distributed Systems
Do Minh Tien - 23BI14421

December 10, 2025

1 Introduction

This practical work implements the classical Word Count program using a custom MapReduce model in Python. The goal is to understand how Mapper, Shuffle, and Reducer components cooperate in distributed data processing, without relying on heavy frameworks such as Hadoop or Spark.

The assignment requires using a MapReduce system of our choice, and since Python is allowed while C/C++ frameworks are unavailable, we developed our own lightweight MapReduce pipeline.

2 Why We Chose Our MapReduce Implementation

We implemented a minimal MapReduce system in Python because:

- Python is simple, portable, and installed on all USTH lab machines.
- It avoids the heavy installation and configuration of Hadoop.
- It clearly demonstrates the internal logic of Map, Shuffle, and Reduce.
- The pipeline can be executed using Unix tools (cat, sort).
- It satisfies the requirement of "inventing" our own MapReduce version.

This makes the implementation educational, lightweight, and easy to test.

3 MapReduce Architecture for Word Count

The system follows the classical 3-phase MapReduce pipeline:

1. **Map Phase**: Tokenizes input text and outputs `(word, 1)` pairs.
2. **Shuffle/Sort Phase**: Groups pairs by word and sorts them.
3. **Reduce Phase**: Sums all counts for each word and emits final totals.

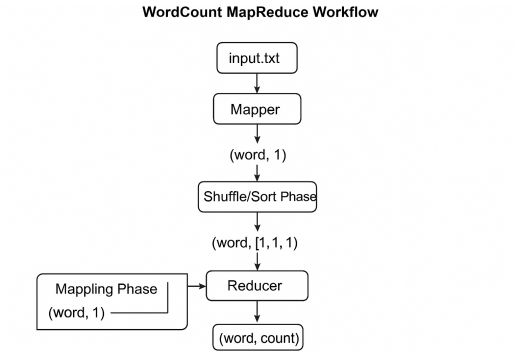

Figure 1: Enter Caption

Figure 2: Word Count MapReduce Workflow

4 Mapper Implementation

The Mapper reads input from standard input, splits into words, and emits a key-value pair for each occurrence:

```
# mapper.py
import sys

for line in sys.stdin:
    line = line.strip().lower()
    for word in line.split():
        print(f"{word}\t1")
```

5 Reducer Implementation

The Reducer receives sorted pairs and aggregates word counts:

```
# reducer.py
import sys

current_word = None
current_count = 0

for line in sys.stdin:
    word, count = line.strip().split("\t")
    count = int(count)

    if word == current_word:
        current_count += count
    else:
        if current_word is not None:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word is not None:
    print(f"{current_word}\t{current_count}")
```

6 How to Run the Program

The entire Word Count pipeline is executed using a simple Unix command:

```
cat input.txt | python3 mapper.py | sort | python3 reducer.py
```

This simulates a full MapReduce workflow with:

- Python scripts as Mapper and Reducer.
- Unix `sort` as the Shuffle/Sort step.

7 Who Does What

- **Mapper**: Processes input chunks, tokenizes into words, and outputs key-value pairs (word, 1) for each occurrence.

- **Shuffle/Sort**: Groups all pairs by key (word) and sorts the keys, preparing for reduction (handled by `defaultdict` and `sorted`).
- **Reducer**: Aggregates values for each key by summing the list of 1s, producing the final count.
- **Main Function**: Splits input into chunks, orchestrates map/shuffle/reduce phases, and collects results into a dictionary.

8 Conclusion

This practical work demonstrates how distributed data processing can be simulated using a lightweight MapReduce model. By implementing the Mapper, Shuffle, and Reducer manually in Python, we gain a clear understanding of how large-scale systems like Hadoop operate internally.