LAB REPORT

Practical Work 4 – Word Count (MapReduce)

1. Introduction

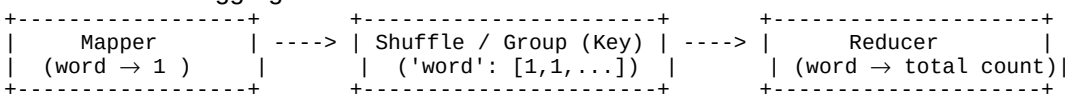
This practical demonstrates the implementation of the Word Count problem using a simplified MapReduce workflow. The program simulates the three major phases of MapReduce: Map, Shuffle/Group, and Reduce. The goal is to understand how data flows between stages and how distributed counting is typically done in real MapReduce systems.

2. MapReduce Framework Choice

A lightweight Python implementation was chosen to simulate MapReduce. This avoids the need for Hadoop or Spark while still illustrating how Map, Shuffle, and Reduce stages work. Python enables clear separation of logic and readability.

3. Word Count Design

The Word Count system follows the standard MapReduce architecture. Each line of text is processed by the Mapper to produce (word, 1) pairs. These are grouped by key during the Shuffle phase, and the Reducer aggregates the counts for each word.



4. System Organization

The system is divided into distinct functions: mapper(), grouper(), reducer(), and run_word_count(). This structure mirrors real-world MapReduce systems where tasks are modular and executed in sequence.

```
WordCount/
■■■ wordcount.py
■   ■■■ mapper()
■   ■■■ grouper()
■   ■■■ reducer()
■   ■■■ run_word_count()
```

5. Implementation

Below is a key portion of the Word Count implementation.

```
def mapper(line):
    line = line.lower()
    words = re.findall(r'\b\w+\b', line)
    return [(word, 1) for word in words]

def grouper(mapped_pairs):
    grouped = {}
    for key, value in mapped_pairs:
        grouped.setdefault(key, []).append(value)
    return grouped

def reducer(word, counts):
```

```
        return (word, sum(counts))
```

6. Conclusion

This practical successfully models the MapReduce paradigm using Python. Although the system runs sequentially, it captures the core idea behind real distributed processing frameworks and helps build a foundational understanding of large-scale data analysis.