Practical Work 4: Word Count (MapReduce)

Nghiêm Xuân Sn - ID

December 6, 2025

1 Implementation Choice

For this practical work, I chose to **implement a custom, lightweight MapReduce framework** using Python.

Reasoning:

- As per the lab requirement ("Invent yourself"), creating a simulation helps us understand the core mechanics of MapReduce (Splitting, Mapping, Shuffling, Reducing) without the overhead of configuring a complex Hadoop cluster.
- Python's dictionary data structure makes the "Shuffle and Sort" phase intuitive and easy to implement.

2 System Design

My implementation follows the standard MapReduce data flow pipeline:

Input Text $\rightarrow$ **Map Phase** $\rightarrow$ **Shuffle Phase** $\rightarrow$ **Reduce Phase** $\rightarrow$ **Output**

Figure 1: Data Flow Pipeline

1. **Input:** The system reads the raw text file.
2. **Mapper:** It splits the text into words, cleans punctuation, and emits key-value pairs in the format: `(word, 1)`.
3. **Shuffle & Sort:** It aggregates the emitted pairs by key. It converts a list of pairs into a dictionary where each word maps to a list of occurrences: `word` $\rightarrow$ `[1, 1, 1...]`.
4. **Reducer:** It takes the list of counts for each word and sums them up to get the final frequency.

3 Implementation Code

3.1 The Mapper

The Mapper function cleans the word and emits a count of 1.

```python
def mapper(text_chunk):
    results = []
    # Split text and remove punctuation
    words = text_chunk.strip().lower().split()
    for word in words:
        clean_word = word.strip(string.punctuation)
        if clean_word:
            results.append((clean_word, 1))
    return results
```

Listing 1: Mapper Implementation

3.2 The Reducer

The Reducer sums up the list of ones.

```python
def reducer(word, counts_list):
    # Aggregates the list [1, 1, 1] into a total sum
    return word, sum(counts_list)
```

Listing 2: Reducer Implementation

4 Who does what

Member	**Task**
Nghiêm Xuân Son	Implemented Custom MapReduce Logic & Report