Practical Work 4: Word Count

Nguyen The Xuan

December 5, 2025

1 Introduction

The objective of this practical work is to implement the classic "Word Count" algorithm using the MapReduce programming model. The goal is to process a text file and count the frequency of each word by distributing the workload across parallel processes.

2 Implementation Choice

We chose to implement a **custom MapReduce framework using Python**.

- **Reasoning:** As the lecture notes suggested "Invent yourself" for C/C++, we applied the same logic to Python to deeply understand the mechanics of MapReduce rather than relying on a high-level abstraction like Hadoop Streaming or MRJob initially.
- **Technology:** We used Python's `multiprocessing` library. This allows us to spawn true OS-level processes to simulate the distributed nature of Mappers and Reducers running on separate nodes (or cores), overcoming Python's Global Interpreter Lock (GIL).

3 Mapper and Reducer Design

Our system follows the standard MapReduce data flow:

3.1 Mapper Logic

The input file is split into chunks. Each Mapper process:

1. Receives a text chunk.
2. Cleans the text (removes punctuation, converts to lowercase).
3. Splits the text into words.
4. Emits a key-value pair: `(word, 1)` for every occurrence.

3.2 Reducer Logic

Between Map and Reduce, a "Shuffle and Sort" phase groups all values belonging to the same key. The Reducer process:

1. Receives a pair: `(word, [1, 1, 1, ...])`.
2. Sums the list of ones.
3. Returns the final count: `(word, total_count)`.

4 Implementation Snippets

4.1 The Mapper

```python
def mapper(text_chunk):
    # ... cleaning logic ...
    words = clean_text.split()
    mapped_data = []
    for word in words:
        if word:
            mapped_data.append((word, 1))
    return mapped_data
```

Listing 1: Mapper Function

4.2 The Reducer

```python
def reducer(item):
    word, counts = item
    return (word, sum(counts))
```

Listing 2: Reducer Function

4.3 Parallel Execution Driver

```python
# Parallel Map
with multiprocessing.Pool(processes=NUM_MAPPERS) as pool:
    map_results = pool.map(mapper, chunks)

# ... Shuffling ...

# Parallel Reduce
with multiprocessing.Pool(processes=NUM_REDUCERS) as pool:
    reduced_results = pool.map(reducer, grouped_data)
```

Listing 3: Using Multiprocessing

5 Experimental Results

We tested the implementation with a sample text file.

- **Input:** "Hello world mapreduce hello map reduce python python python"
- **Output:**

```
python: 3
hello: 2
mapreduce: 1
world: 1
map: 1
reduce: 1
```

The system successfully parallelized the task across 4 simulated worker processes.