University Of Science And Technology Of Hanoi
ICT Department

LAB REPORT

Title: Practical Work 4 – Word Count

Subject: Distributed Systems (DS2026)

Student: Nguyen Phuong Anh

StudentID: 22BA13020

Major: CS

Introduction

The goal of this practical work is to implement a simplified Word Count program based on the MapReduce model. The assignment requires students to choose a MapReduce framework or create one themselves, preferably using the C programming language. Since no MapReduce framework currently exists for C/C++, our group decided to design and implement a lightweight MapReduce workflow from scratch.

The program follows the traditional MapReduce pipeline:

- **Map**: process the input text and emit key-value pairs `<word,1>`
- **Shuffle/Sort**: group identical keys together
- **Reduce**: aggregate the counts of each word

Why We Chose C for MapReduce

We chose the C programming language for several reasons:

- It aligns with the course objectives on understanding low-level systems.
- C provides high performance and full control over memory management.
- Implementing MapReduce manually in C helps us understand internal mechanisms such as key-value storage, dynamic arrays, sorting, and data aggregation.
- Unlike Java or Python, we do not rely on any external MapReduce libraries.

MapReduce Design

Our implementation is divided into three main phases that mimic the classic MapReduce architecture.

1. Mapper

The Mapper reads the input file word by word. Each word is normalized (lowercased, cleaned of non-alphabetic characters) and then emitted as a key-value pair:

```
emit(word, 1)
```

Each emitted pair is stored in a dynamically growing array until all input has been processed.

2. Shuffle and Sort

The intermediate key-value pairs are sorted using `qsort()` in C. Sorting ensures that identical keys appear consecutively, enabling efficient reduction.

3. Reducer

The Reducer iterates through the sorted list and aggregates all identical keys:

$$\texttt{reduce(word)} = \sum \texttt{value for that word}$$

The output is written to an output file in the format:

```
word   count
```

Example Output

After running our Word Count program on the sample input file, the output contains the list of unique words along with their corresponding frequencies. Below is an excerpt of the generated `output.txt`:

```
ai      2
ang     1
bao     2
bào     1
bch     1
bi      1
bn      3
biên    1
bin     1
```

MapReduce Workflow Diagram

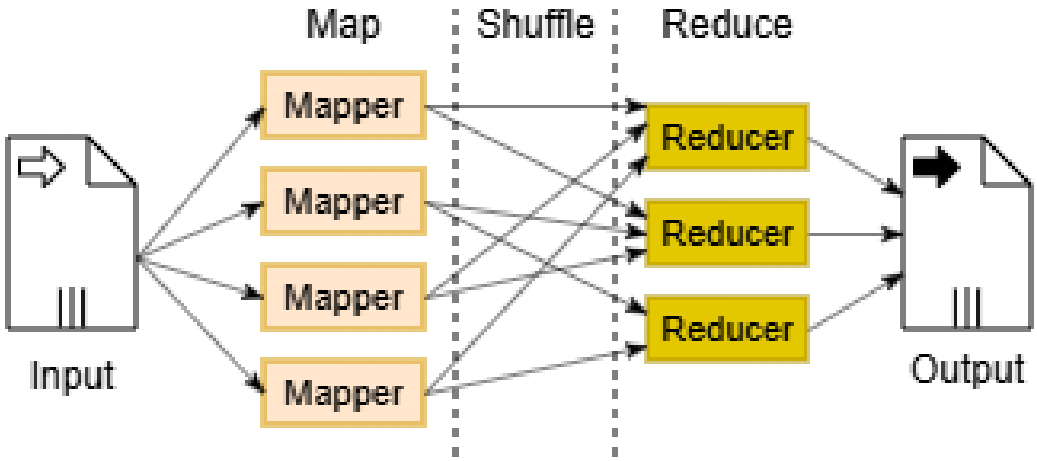

Figure 1: MapReduce workflow used in our Word Count implementation

Implementation Details

The core program is implemented in a single file:

- `wordcount.c`

Key components include:

- A custom dynamic array implementation for storing key-value pairs.
- A custom `my_strdup()` function to duplicate strings (since `strdup()` is not standard C).
- A normalization routine to clean and lowercase words.
- Sorting using the C standard library function `qsort()`.
- A reduction loop to aggregate word frequencies.

The program is compiled using:

```
gcc -std=c11 -Wall -Wextra -O2 wordcount.c -o wordcount
```

Execution command:

```
./wordcount input.txt output.txt
```

Conclusion

This practical work allowed us to better understand how MapReduce works internally. By implementing the Word Count algorithm manually in C, we gained a deeper appreciation for distributed processing concepts such as mapping, shuffling, sorting, and reducing. The assignment also strengthened our skills in memory management, string processing, and modular system design.