UNIVERSITY OF SCIENCE & TECHNOLOGY OF HANOI
TRƯỜNG ĐẠI HỌC KHOA HỌC & CÔNG NGHỆ HÀ NỘI

Practical Work 3: Word Count
Vũ Đức Hiếu - BI12-162

I) System architecture

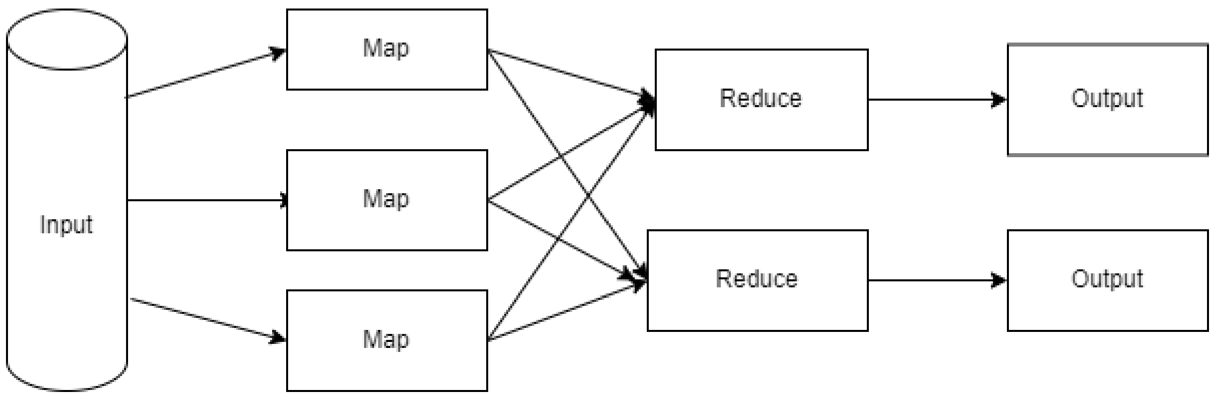


Figure 1. Workflow of MapReduce

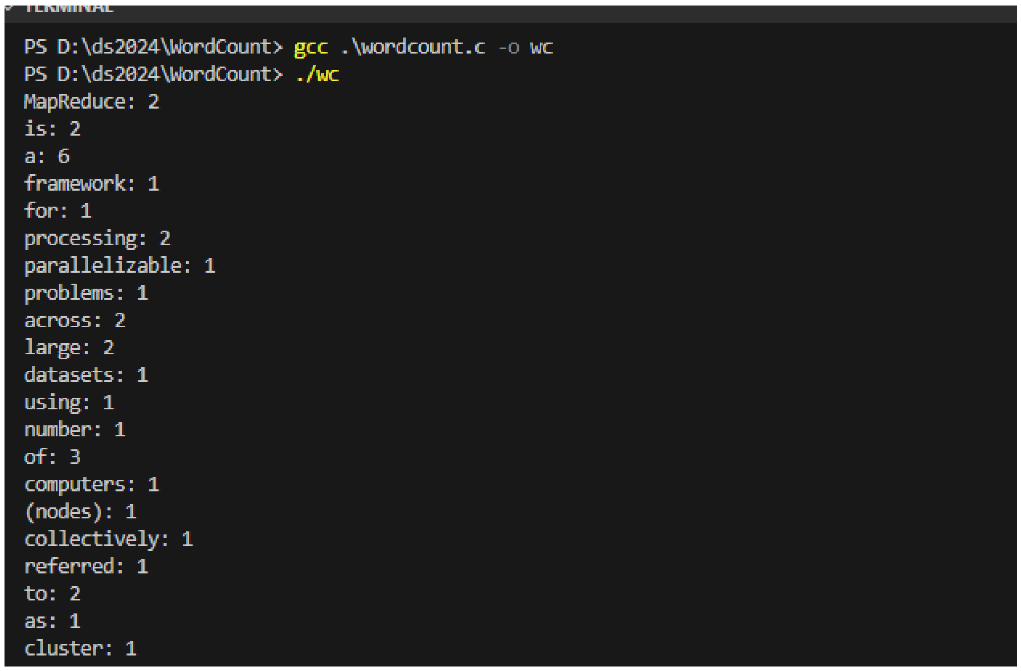
II) Implementation

- Map function:
  +) Map function takes a line of text as input, tokenizes it into individual words using common delimiters, and stores each word along with an initial count of 1in a KeyValue array. This process allows for counting the occurrences of each word later during the reduction phase.
- Reduce function:
  +) The reduce function aggregates the key-value pairs from the input array, counting the occurrences of each word and storing the result in the result array. Finally, it prints out the aggregated word counts.
- The source code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_WORD_LENGTH 100


typedef struct {
    char key[MAX_WORD_LENGTH];
    int value;
} KeyValue;

// Function to tokenize a line of text and count occurrences of each word
void map(char *line, KeyValue *kv, int *kv_count) {
    // Tokenize the line using common delimiters
    char *token = strtok(line, " .,\t\n\r");
    // Loop through each token (word) until NULL is returned by strtok
    while (token != NULL) {
        // Copy the token (word) into the key field of the KeyValue struct
        strcpy(kv[*kv_count].key, token);
        // Set the initial count for the word to 1
        kv[*kv_count].value = 1;
        // Increment the key-value pair count
        (*kv_count)++;
        // Move to the next token
        token = strtok(NULL, " .,\t\n\r");
    }
}

// Function to aggregate word counts and print the result
void reduce(KeyValue *input, int input_size) {
    // Array to hold the aggregated key-value pairs
    KeyValue result[input_size];
    // Variable to track the number of unique words in the result array
    int result_size = 0;

    // Loop through each key-value pair in the input array
    for (int i = 0; i < input_size; i++) {
        int j;
        // Check if the current word already exists in the result array
        for (j = 0; j < result_size; j++) {
            // If the word exists, increment its count
            if (strcmp(input[i].key, result[j].key) == 0) {
                result[j].value++;
                break;
            }
        }
        // If the word doesn't exist in the result array, add it as a new key-value pair
        if (j == result_size) {
            strcpy(result[result_size].key, input[i].key);
            result[result_size].value = 1;
            result_size++;
        }
    }

    // Print the aggregated word counts
    for (int i = 0; i < result_size; i++) {
        printf("%s: %d\n", result[i].key, result[i].value);
    }
```

```
        }

        int main() {
            FILE *file;
            char line[1000];
            KeyValue kv[1000];
            int kv_count = 0;

            // Open the file for reading
            file = fopen("test.txt", "r");
            // Check if the file opened successfully
            if (file == NULL) {
                perror("Error opening file");
                return 1;
            }

            // Read each line from the file, tokenize it, and count word occurrences
            while (fgets(line, sizeof(line), file)) {
                map(line, kv, &kv_count);
            }

            // Close the file
            fclose(file);

            // Aggregate word counts and print the result
            reduce(kv, kv_count);

            return 0;
}
```

- Result of the code



```
PS D:\ds2024\WordCount> gcc .\wordcount.c -o wc
PS D:\ds2024\WordCount> ./wc
MapReduce: 2
is: 2
a: 6
framework: 1
for: 1
processing: 2
parallelizable: 1
problems: 1
across: 2
large: 2
datasets: 1
using: 1
number: 1
of: 3
computers: 1
(nodes): 1
collectively: 1
referred: 1
to: 2
as: 1
cluster: 1
```