USTH Master Spoiler Alert

05.word.count

Nguyen Quynh Trang - 22BA13303

1 Introduction

This report documents the implementation of the Longest Path problem using the MapReduce programming paradigm. The objective is to efficiently find the maximum length of a file path within a given list of paths. This exercise demonstrates the MapReduce Maximum pattern.

2 Implementation Choice

2.1 Language Selection and Justification

The implementation was developed in C language. This choice aligns with the preference stated in the practical work instructions to use C/C++ or ”Invent Yourself” due to the lack of a standard distributed framework for C.

2.2 The Single-Process Simulation

The program is a single-process application that simulates the MapReduce logic:

- **Map Phase:** Implemented by the `mapper` function.
- **Shuffle & Sort Phase:** Simulated using the standard C library function `qsort`() to ensure all paths are grouped under a single key.
- **Reduce Phase:** Implemented by the `reducer` function for finding the Maximum value.

3 MapReduce Logic and Execution Flow

3.1 Data Flow Diagram

Figure 1 illustrates the complete process, from input data (path list) to the final maximum length.

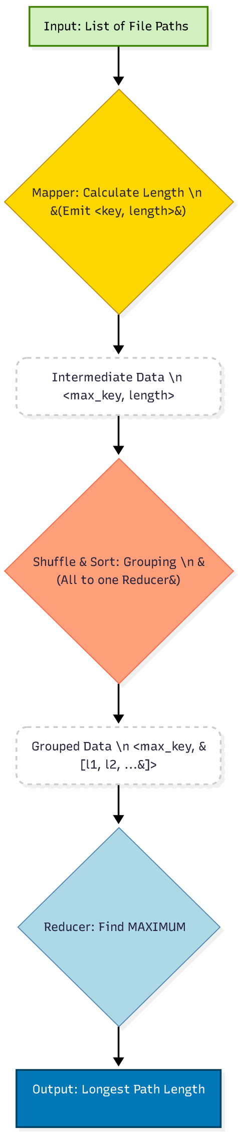

Input: List of File Paths
Mapper: Calculate Length \n &(Emit <key, length>&)
Intermediate Data \n <max_key, length>
Shuffle & Sort: Grouping \n & (All to one Reducer&)
Grouped Data \n <max_key, & [l1, l2, ...&]>
Reducer: Find MAXIMUM
Output: Longest Path Length

Figure 1: The MapReduce Data Flow for the Longest Path problem (Maximum Pattern).

3.2 Mapper Functionality

The `mapper` function takes a single path (line of text) as input. Its primary goal is to emit the length of the path associated with a universal, fixed key.

1. **Length Calculation:** The length of the input string (`path`) is calculated using `strlen`, excluding the newline character (`\n`).

2. **Emit Intermediate Pairs:** For every path, the Mapper emits a Key-Value pair of the format ⟨**"max_len_key"**, **path_length**⟩.

Using a fixed key ensures that all intermediate length values are routed to the same Reducer instance for the final calculation.

3.3 Reducer Functionality

The `reducer` function is responsible for determining the maximum length among all emitted values.

1. **Shuffle & Sort:** The simulated phase groups all `path_length` values under the fixed key.
2. **Aggregation (Maximum):** The Reducer iterates through the list of received path lengths and finds the single largest value (Maximum) among them.
3. **Final Output:** It prints the final result as ⟨**Longest_Path_Length**, **max_value**⟩.

4 Source Code Listings

The C code implementation combines the Map, Shuffle/Sort, and Reduce logic into a single program file (`longest_path.c`).

```c
typedef struct {
    char key[20];
    int value;
} KeyValue;

// --- Mapper Function ---
void mapper(char *line) {
    size_t length = strlen(line);

    if (length > 0 && line[length - 1] == '\n') {
        length--;
    }

    if (length > 0) {
        // ... (ensure_capacity call removed for brevity)
        strcpy(intermediate_data[intermediate_count].key, "max_len_key");
        intermediate_data[intermediate_count].value = (int)length;
        intermediate_count++;
    }
}

// --- Reducer Function ---
void reducer() {
    // qsort(intermediate_data, intermediate_count, sizeof(KeyValue),
        compare_keys);

    int max_length = 0;

    for (int i = 0; i < intermediate_count; i++) {
        int current_length = intermediate_data[i].value;
        if (current_length > max_length) {
            max_length = current_length;
        }
    }

    printf("<Longest_Path_Length, %d>\n", max_length);
}
```

Listing 1: Core Code Snippet (longest_path.c)

5 Results and Verification

This section presents the results obtained by executing the C implementation on an input derived from a literary work, verifying the correctness of the maximum-finding logic.

5.1 Input Data Source

The input file used for testing was generated by extracting several lines from the book: **The curious incident of the dog in the night-time** by Mark Haddon. The input lines simulate file paths of varying lengths.

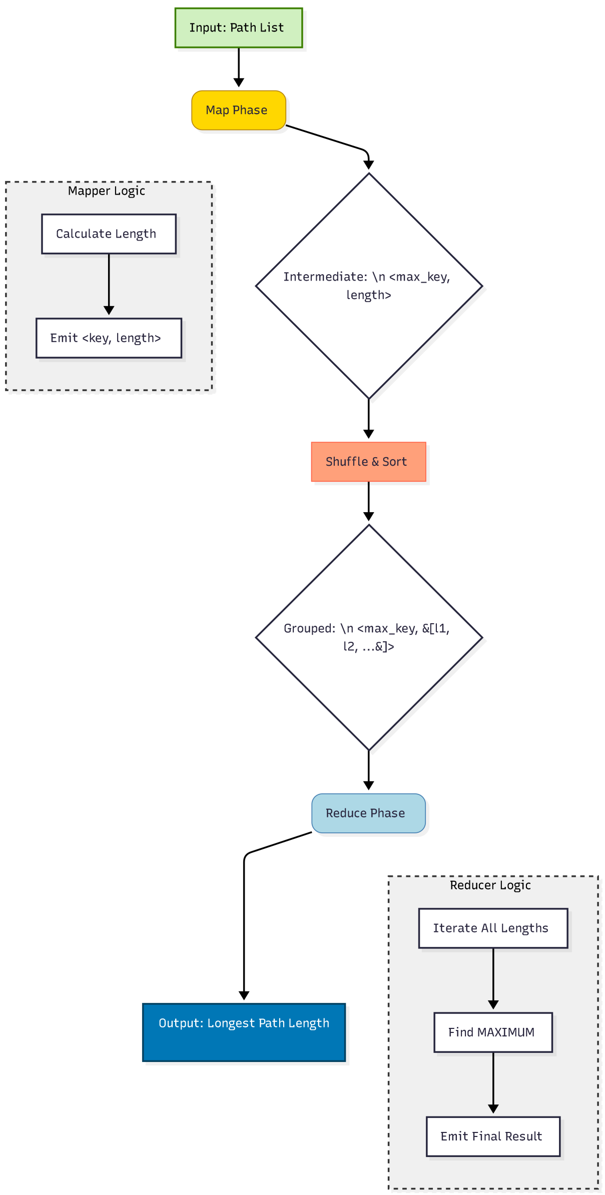
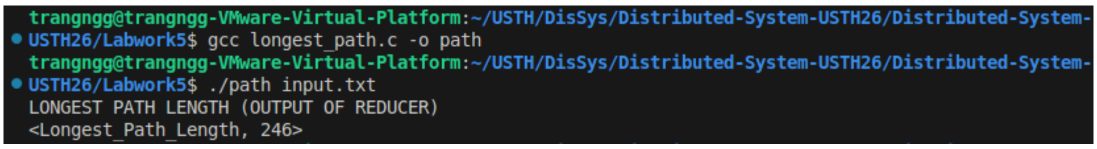


Figure 2: Combined view of the literary-sourced Input Data and the Program Output, showing the final maximum path length found.

The output confirms that the Reducer correctly processed all intermediate values (path lengths) and successfully identified the single largest value, demonstrating that the MapReduce Maximum pattern was accurately implemented.



```
trangngg@trangngg-VMware-Virtual-Platform:~/USTH/DisSys/Distributed-System-USTH26/Distributed-System-
USTH26/Labwork5$ gcc longest_path.c -o path
trangngg@trangngg-VMware-Virtual-Platform:~/USTH/DisSys/Distributed-System-USTH26/Distributed-System-
USTH26/Labwork5$ ./path input.txt
LONGEST PATH LENGTH (OUTPUT OF REDUCER)
<Longest_Path_Length, 246>
```

Figure 3: Output of the word count system

6 Conclusion

The MapReduce paradigm was successfully simulated using a C program to solve the Longest Path problem. The implementation demonstrates the core principles of data distribution, grouping (under a fixed key), and aggregation (finding the Maximum), proving the model's effectiveness in processing data tasks that require identifying extreme values.