UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

Practical Work 5
The Longest Path using MapReduce

Author:
Nguyen Tuan Dung
Student ID: 23BI14113

December 6, 2025

Contents

1 Introduction

This report details the implementation of the Longest Path problem using the MapReduce paradigm. The objective is to efficiently find the path *(file or directory name)* with the maximum character length across a distributed dataset (collected from the `find /` command on multiple machines). This exercise demonstrates how MapReduce can be adapted to find global maximum values, which contrasts with the common aggregation (summation) tasks like Word Count.

2 MapReduce Implementation and Strategy

2.1 Mapper and Reducer Strategy

The key strategy is to ensure all intermediate values are processed by a single Reducer (or a controlled small set of Reducers) which can then perform the final comparison.

- **Mapper:** Calculates the local metric (path length) for each item.
- **Shuffle/Sort:** Collects all calculated lengths together.
- **Reducer:** Compares all lengths and outputs the maximum length found globally.

2.2 Mapper Logic

The Mapper's primary task is localized calculation.

Table 1: Mapper Input/Output Specification for Longest Path

Input Key (Implicit)	Mapper Task	Intermediate Output
Text Line (Full Path)	Calculate the string length of the path.	`<"1", path_length>`

The use of a constant key (`"1"`) is crucial. Since the MapReduce framework partitions data based on the key, using the same key for all emissions ensures that all the calculated lengths are routed to the same Reducer.

2.3 Reducer Logic

The Reducer receives a list of all path lengths and performs the final comparison.

Table 2: Reducer Input/Output Specification for Longest Path

Intermediate Input	Reducer Task	Final Output
`<"1", list(L1, L2, ...)>`	Iterate through all values (L_i). Find the maximum value $L_{\max}$.	`<Max_Length, 1>`

The Reducer maintains a running maximum length, updating it every time a new, longer path length is encountered.

3 System Organization and Workflow

The system organization follows the same simulated streaming MapReduce architecture established in Practical 4

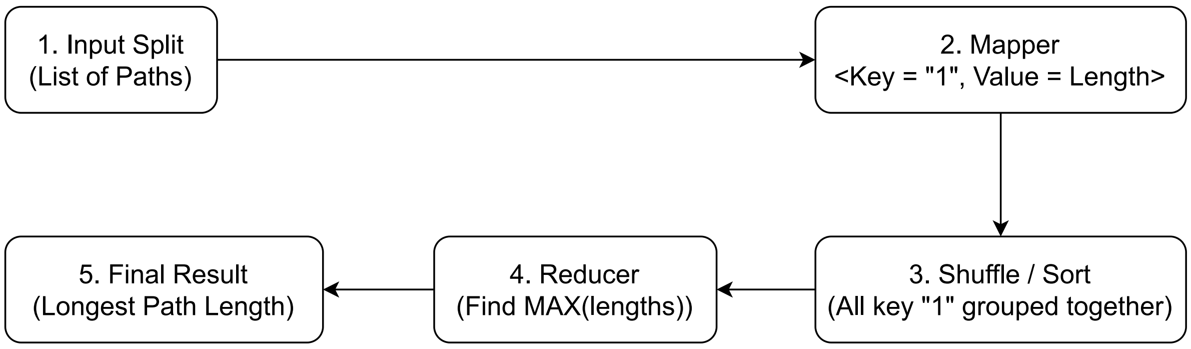


Figure 1: Longest Path MapReduce Workflow. The centralized sorting/grouping phase is leveraged to collect all data necessary for the global maximum computation.

3.1 Who Does What

- **The C++ Executable (`longest_path`):**
  - **Mapper Role:** Reads an input path string, determines its length, and outputs `<"1", length>`. This task is perfectly parallelizable.
  - **Reducer Role:** Receives a sorted input stream (all lengths associated with key "1") and performs a serial comparison to identify the largest integer value among them.
- **The Operating System/Shell (Simulated Master/Framework):**
  - **Splitting:** Divides the input collection of path files into splits for each Mapper instance.
  - **Shuffling & Sorting:** Gathers all intermediate output files. Since the key is always "1," the shuffle step effectively routes all data to a single Reducer (or a single input file for the Reducer).

4 Implementation Snippets (C++ Core Logic)

The efficiency comes from the simplicity and parallelism of the Mapper.

4.1 Mapper Snippet

Listing 1: C++ Mapper Core Logic (Length Calculation)

```
void run_mapper() {
    string path;
    while (getline(cin, path)) {
        int length = path.length();
        // Emit constant key '1' to group all lengths together
        cout << "1" << "\t" << length << endl;
    }
}
```

4.2 Reducer Snippet

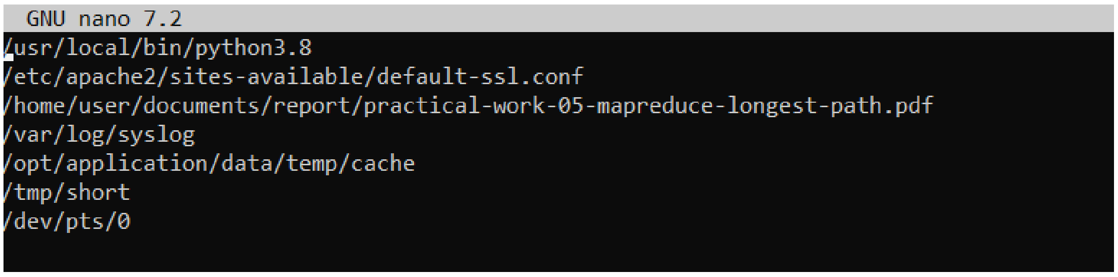
Listing 2: C++ Reducer Core Logic (Finding Global Maximum)

```
void run_reducer() {
    string line;
    int max_length = 0;
    string key;
    int length;

    // Input is guaranteed to contain all lengths with key '1'
    while (getline(cin, line)) {
        stringstream ss(line);
        ss >> key >> length;

        if (length > max_length) {
            max_length = length;
        }
    }

    // Output the final maximum length
    cout << max_length << "\t" << 1 << endl;
}
```

5 Results

5.1 Input Example

Figure 2 shows a sample of the file paths used as input for the MapReduce Longest Path experiment.

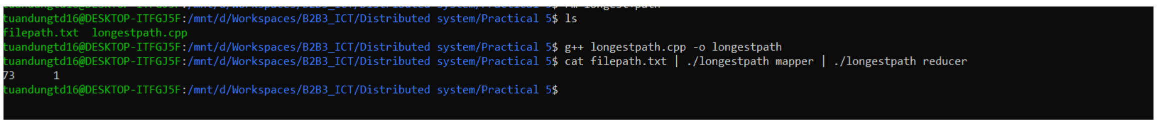


```
GNU nano 7.2
/usr/local/bin/python3.8
/etc/apache2/sites-available/default-ssl.conf
/home/user/documents/report/practical-work-05-mapreduce-longest-path.pdf
/var/log/syslog
/opt/application/data/temp/cache
/tmp/short
/dev/pts/0
```

Figure 2: Example input file paths collected from the `find /` command.

5.2 Mapper and Reducer Output

The output produced after running the Mapper and Reducer programs is shown in Figure 3. The mapper computes the length of each file path and the reducer returns the global maximum.



```
tuandungtd16@DESKTOP-ITFGJ5F:/mnt/d/Workspaces/B2B3_ICT/Distributed system/Practical 5$ ls
filepath.txt  longestpath.cpp
tuandungtd16@DESKTOP-ITFGJ5F:/mnt/d/Workspaces/B2B3_ICT/Distributed system/Practical 5$ g++ longestpath.cpp -o longestpath
tuandungtd16@DESKTOP-ITFGJ5F:/mnt/d/Workspaces/B2B3_ICT/Distributed system/Practical 5$ cat filepath.txt | ./longestpath mapper | ./longestpath reducer
73      1
tuandungtd16@DESKTOP-ITFGJ5F:/mnt/d/Workspaces/B2B3_ICT/Distributed system/Practical 5$
```

Figure 3: Mapper and Reducer output showing the computed maximum path length.

6 Conclusion

The Longest Path problem was successfully solved using MapReduce by implementing a parallel length calculation in the Mapper and utilizing a constant grouping key ("1") to force a single Reducer to compute the global maximum. This technique is a standard pattern for finding global aggregates (Min, Max, or Top N) in the MapReduce model, demonstrating its versatility beyond simple summation.