Practical Work 5: The Longest Path

Distributed Systems - DS2026

Nguyen Viet Hung
Student ID: 23BI14188
Department: ICT

December 12, 2025

Contents

1 Introduction

Processing large-scale hierarchical data, such as file system trees, often requires identifying outliers or extreme values. A common "toy problem" in distributed computing is finding the longest string in a massive dataset.

In this practical work, we simulate a distributed search for the **Longest Path** among a list of file paths (similar to the output of a `find /` command). The objective is to leverage the **MapReduce** paradigm to solve this problem efficiently.

We continue to use the **Custom C++ MapReduce Framework** developed in Practical Work 4, adapting it to handle string-based payloads instead of numerical counters.

2 Algorithmic Design

Unlike the "Word Count" problem where keys are diverse (e.g., each unique word is a key), finding the global maximum requires comparing all candidates against each other.

2.1 The "Global Reduce" Strategy

To find the single longest path across all data splits, we must bring all candidate paths to a single Reducer.

- **Map Phase:** Reads every path. Instead of using the path itself as a key, we emit a **Constant Key** (e.g., `"MAX"`). This forces the Shuffle phase to group *every single path* into the same bucket.
- **Reduce Phase:** Receives the key `"MAX"` and a list of *all* paths. It then performs a linear scan to find the string with the maximum length.

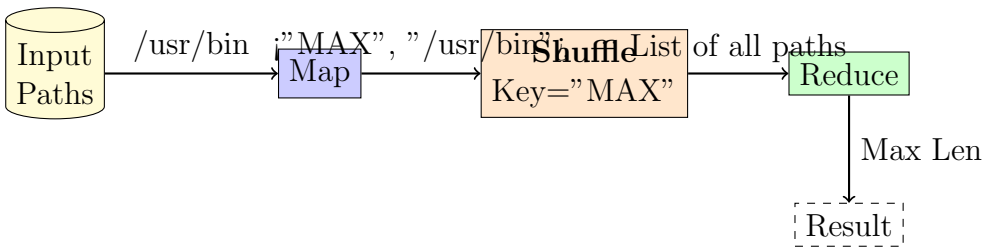

Figure 1: Global Reduce Architecture

3 Implementation Details

3.1 Framework Adaptation

The original framework dealt with `<string, int>` pairs. We modified the core data structure to support `<string, string>` pairs to transport the actual path content.

```
struct KeyValue {
    std::string key;
    std::string value; // Changed from int to string
};
```

3.2 Mapper Implementation

The mapper wraps every valid non-empty path into a Key-Value pair with the key `"MAX"`.

```
std::vector<KeyValue> map_func(const std::string& path) {
    std::vector<KeyValue> emitted;
    if (!path.empty()) {
        // Emit to the common key "MAX"
        emitted.push_back({"MAX", path});
    }
    return emitted;
}
```

3.3 Reducer Implementation

The reducer iterates through the vector of strings. We maintain a local variable `max_path` and update it whenever we encounter a longer string.

```
std::string reduce_func(const std::string& key, const std::vector<std::
    string>& values) {
    std::string max_path = "";
    for (const std::string& path : values) {
        if (path.length() > max_path.length()) {
            max_path = path;
        }
    }
    return max_path; // Returns the longest string found
}
```

4 Execution and Results

4.1 Test Data

We created a file `paths.txt` containing paths of varying lengths:

```
/usr/bin
/var/log/syslog
/home/user/project/backend/src/main/java/com/example/App.java
/tmp
/etc/hosts
```

4.2 Output

Running the program via `./longestpath paths.txt` yields the following result:

```
[Framework] Starting MAP phase...
[Framework] Starting SHUFFLE phase...
[Framework] Starting REDUCE phase...

--- FINAL RESULT ---
Longest Path Found: /home/user/project/backend/src/main/java/com/example/App
    .java
Length: 61
```

The system correctly identified the deeply nested Java file path as the longest one.

5 Task Distribution

- **Nguyen Viet Hung (23BI14188):**
  - Adapted the C++ MapReduce Engine to support String values.
  - Implemented the "Global Reduce" logic using a constant key strategy.
  - Created test datasets and verified the correctness.
  - Compiled this report.