Practical Work 5 - MapReduce Longest Path

Tran Hai Nam

December 2025

1 Introduction

This practical work implements the "Longest Path" assignment using a MapReduce-style programming model. The goal is to process one or more text files containing absolute file paths (one path per line) and to find the path or paths having the maximal length. Instead of deploying a full Hadoop cluster, we reuse the lightweight MapReduce-style framework written in Java for Practical Work 4 and adapt it to the longest-path problem.

2 Objectives

The main objectives of this lab are:

- Define a clear notion of "path length" and state it explicitly in the report.
- Design appropriate **Mapper** and **Reducer** functions for the Longest Path problem.
- Implement a MapReduce-style engine that can read multiple input files and compute the longest path(s).
- Provide a small test dataset and report the observed results.

3 Problem Definition and Design

3.1 Input and Output Specification

The input dataset follows the slides:

- Each input text file contains one absolute or full file path per line.
- The union of all lines across all input files forms the global set of candidate paths.

In this implementation, the **length** of a path is defined as the *number of characters* in the full string, including directory separators and file extension. The output contains all paths whose length is equal to the maximal length observed in the input, together with that length.

3.2 MapReduce Design

The design is intentionally simple, in line with the assignment description.

Mapper The mapper reads each line (a candidate path), computes its length, and emits:

- key = the full path string;
- value = an integer equal to the number of characters in the path.

We keep the full path as the key so that the reducer can directly identify the path associated with a given length.

Reducer The reducer receives a path and the list of length values associated with it (in practice, one value per path). For robustness it scans the list and keeps the maximal length for that path. Afterwards, a separate pass computes the global maximum across all paths and outputs only those entries whose length equals this global maximum.

4 Implementation

4.1 Language and Environment

The implementation is written in Java (JDK 8+), reusing the interfaces from the Word Count lab:

- `Mapper` and `Reducer` (from `WordCount/`).
- `Emitter` as a functional interface for emitting key–value pairs.

No external MapReduce framework is used; the job runs on a single machine with a multi-threaded map phase.

4.2 Code Structure

The Longest Path code is located in the `LongestPath/` directory:

- `LongestPathMapper.java`:
  - Implements `Mapper`.
  - For each non-empty line `value`, trims it to obtain `path`, computes `path.length()`, and emits `(path, length)`.
- `LongestPathReducer.java`:
  - Implements `Reducer`.
  - Iterates over all integer values for a given path and keeps the maximum (defensive programming, even if there is usually only one value).
  - Emits `(path, maxLength)`.
- `LongestPathJob.java`:
  - Plays the same role as `WordCountJob` but for paths:
    - ∗ Creates a concurrent map `Map<String, List<Integer>>` for intermediate results.
    - ∗ Spawns a fixed-size thread pool to run the mapper on each input file.
    - ∗ Waits for all map tasks to complete.
    - ∗ Runs the reducer for each path, producing a `Map<String, Integer>` of path lengths.
    - ∗ Computes the global maximum length and writes only the longest path(s) to the output file.
- `LongestPathMain.java`:
  - Acts as the CLI entry point.
  - Parses command-line arguments:
    - ∗ `args[0]`: output file.
    - ∗ `args[1..]`: list of input files.
  - Filters out non-regular files with a log message.

  - Chooses the number of worker threads via `Runtime.getRuntime().availableProcessors()`.
  - Creates `LongestPathJob` and runs it.
  - Logs progress and the location of the output file.

- `sample/sample_paths.txt`:
  - Small test file containing a few Unix-style and Windows-style absolute paths of different lengths.

5 Build and Run

5.1 Compilation

From the repository root, the following command compiles both the shared interfaces and the Longest Path implementation:

```
javac WordCount/*.java LongestPath/*.java
```

5.2 Execution

On Windows, the classpath uses a semicolon (`;`) separator. An example run on the sample dataset is:

```
java -cp "WordCount;LongestPath" LongestPathMain ^
  LongestPath/sample/longest_paths_output.txt ^
  LongestPath/sample/sample_paths.txt
```

The program logs messages such as:

```
[longest-path] Running with 8 worker thread(s)...
[longest-path] Done. Results written to D:\Repo\He_phan_tan\LongestPath
   \longest_paths_output.txt
```

6 Experiments and Results

6.1 Test Dataset

For this report, a synthetic test file `LongestPath/sample/sample_paths.txt` is used. It contains five absolute paths, for example:

```
/home/user/documents/report.txt
/home/user/documents/projects/distributed-systems/mapreduce/assignment5
   /longest_path_example.txt
/home/user/downloads/music/song.mp3
/var/log/system/system.log
C:\Users\student\Documents\university\distributed_systems\assignments\
   practical_work_5\longest_path_windows_example.txt
```

The last path is significantly longer than the others and is expected to be the longest according to the “number of characters” definition.

6.2 Output

After running the program on this sample file, the output file `LongestPath/sample/longest_paths_output.txt` contains:

```
C:\Users\student\Documents\university\distributed_systems\assignments\
    practical_work_5\longest_path_windows_example.txt     119
```

This confirms that:

- the program correctly computes the character length of each path;
- it identifies the longest path according to this metric;
- it outputs both the path and its length, as required by the assignment.

If several paths shared the same maximal length, they would all appear in the output file, one per line.

7 Discussion

7.1 Strengths

- The implementation reuses the MapReduce-style abstraction from Practical Work 4, which keeps the structure (map, shuffle, reduce) clear.
- The code is modular: changing the definition of "length" (e.g., number of directory components instead of characters) only requires modifying the mapper or reducer.
- The multi-threaded map phase can exploit multiple cores, even though the job runs on a single machine.

7.2 Limitations

- The implementation is not distributed: all data is processed in one JVM and intermediate results are stored in memory.
- There is no fault tolerance; crashes lead to job failure and require restarting the run.
- The test dataset is small; large-scale behaviour (millions of paths) was not evaluated in this lab.

7.3 Possible Extensions

Several extensions could be explored in future work:

- Redefine path length as the number of directory components and compare results.
- Read multiple input files generated on different machines and combine them in one run.
- Persist intermediate data to disk or use batching to reduce memory usage for very large datasets.
- Generalise the mini MapReduce engine to support other aggregation tasks beyond Longest Path and Word Count.

8 Conclusion

This practical work applies a MapReduce-style programming model to the Longest Path problem. By reusing the Java-based mini MapReduce framework from the Word Count lab, it demonstrates how different tasks can be expressed using the same abstraction: a mapper that emits key–value pairs and a reducer that aggregates them. The experiments on a small synthetic dataset confirm that the implementation correctly identifies the longest path according to the chosen length definition and illustrate how MapReduce-style thinking can be used even without a full Hadoop installation.