Practical Work 5: The Longest Path

MapReduce Implementation with Multithreading

1 Introduction

This report presents a detailed implementation of the Longest Path algorithm using the MapReduce model combined with multithreading in Python. The program processes multiple input files in parallel to find the file path with the greatest length.

1.1 Objectives

- Implement MapReduce framework for distributed data processing
- Utilize multithreading to optimize processing performance
- Find the longest file path from multiple data sources

2 MapReduce Architecture

2.1 Overview

MapReduce is a programming model designed to process and generate large datasets with a parallel, distributed algorithm on a cluster of computers.

2.2 Main Phases

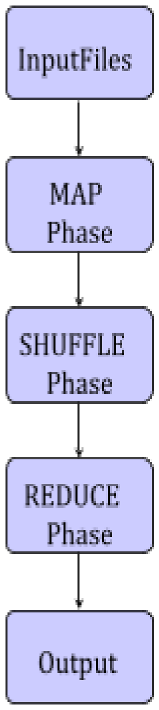

Figure 1: Overall MapReduce Architecture

3 Implementation Details

3.1 Mapper Function

Purpose: Process each input line and generate key-value pairs.

```
def     mapper(line):     path      =
        line.strip() if not path:
                return None
        return ("longest", (len(path), path))
```

Operation:

1. Receives input: a text line containing a file path
2. Removes extra whitespace using strip()
3. Calculates the path length
4. Returns tuple: ("longest", (length, path)) **Input/Output Example:**

- **Input:** "/home/user/documents/file.txt"
- **Output:** ("longest", (29, "/home/user/documents/file.txt"))

3.2 Reducer Function

Purpose: Aggregate values and find the longest path.

```
def reducer(key, values):
    # values: list[(length, path)]
    return max(values)
```

Operation:

1. Receives a list of tuples (length, path)
2. Uses max() function to find the tuple with the largest length
3. Returns (max length, longest path)

Example:

Input values = [(10,”*/usr/bin*”),(25,”*/home/user/documents*”),(15,”*/var/log/sys*”)]
Output = (25,”*/home/user/documents*”)

3.3 Map Task (Multithreading)

Purpose: Process one input file in a separate thread.

```
def map_task(filename):
    results = [] try:
        with open(filename, ’r’, encoding=’utf-8’) as f:
            for line_num, line in enumerate(f, 1):
                output = mapper(line) if output:
                    results.append(output)  key,  (length,  path)
                    = output with Lock:
                        print(f"[{filename}]  Line  {line_num:3d}  |  Length={length:3d}  |
                            {path}")
    except FileNotFoundError:
        with Lock: print(f"File not found: {filename}")
    return results
```

Execution Steps:

1. Open file and read line by line
2. Call mapper() for each line
3. Store results in results list
4. Use Lock to ensure thread-safe printing
5. Return all results from the file

4 Main Processing Flow

4.1 MAP Phase (Parallel)

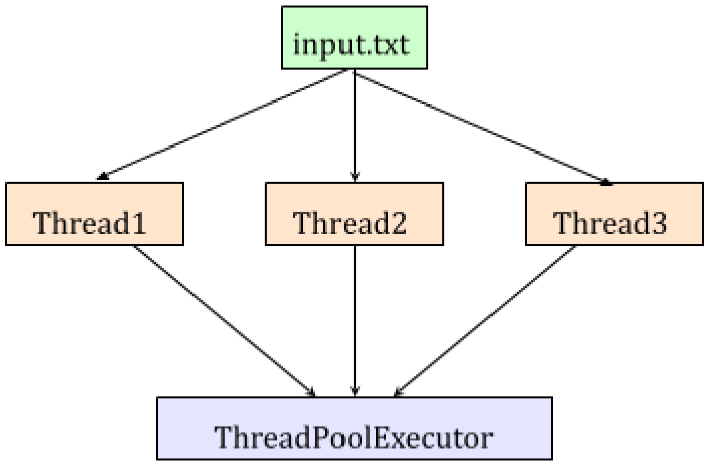

Figure 2: MAP Phase with Multithreading

ThreadPoolExecutor manages up to 8 threads concurrently, with each thread processing one input file.

```
with ThreadPoolExecutor(max_workers=min(8, len(input_files))) as executor:
        futures = [executor.submit(map_task, f) for f in input_files] for future in
        as_completed(futures): map_output.extend(future.result())
```

4.2 SHUFFLE Phase

Purpose: Group all values by key.

```
shuffled = defaultdict(list) for key, value in
map_output: shuffled[key].append(value)
```

Result:

- Key: "longest"
- Values: [(10, path1), (25, path2), (15, path3), ...]

4.3 REDUCE Phase

```
for key, values in shuffled.items():
        max_length, longest_path = reducer(key, values) print(f"Maximum length:
        {max_length} characters") print(f"Longest path: {longest_path}")
```

5 Conclusion

The program successfully implements the MapReduce model with multithreading to solve the longest path problem. Using ThreadPoolExecutor optimizes the processing of multiple files concurrently, while Lock ensures thread safety of the program.