



University Of Science And
Technology
Of Hanoi

ICT Department

LAB REPORT

Title: Practical Work 5 – The Longest Path

Subject:	Distributed Systems (DS2026)
Student:	Le Nam Anh
StudentID:	23BI14029
Major:	Cyber Security

1 Introduction

The objective of this practical work is to apply the MapReduce programming model to a different problem domain: finding the longest file path within a large dataset.

I used the custom C++ MapReduce framework developed in the previous practical work. The system processes a list of file paths (picked randomly from the file system) to identify the path with the maximum character length. This demonstrates the reusability and flexibility of the MapReduce paradigm.

2 Implementation Choice

Instead of adopting a new library, I reused the **C++ Multi-threaded MapReduce Framework** from the previous lab. This is possible because the MapReduce model separates the control flow (Splitting, Shuffle, Reduce) from the application logic (Map and Reduce functions).

By simply swapping the `map()` and `reduce()` logic, the same infrastructure can solve a completely different problem (Max Finding vs. Counting).

3 System Design

The core architecture remains a **Shared-Nothing** parallel processing system, but the logic within the workers has changed to solve the "Maximum" problem.

3.1 How the Mapper works

The **Mapper** processes a chunk of text containing multiple file paths (one per line).

- **Input:** A raw text chunk representing a subset of the file list.
- **Process:**
 - Iterates through the chunk line by line.
 - Measures the length of each path string (`string.length()`).
 - Maintains a local variable `local_max_path` that stores the longest path seen *so far* within that specific chunk.
- **Output:** A single string representing the "champion" (longest path) of that chunk.

This significantly reduces the data transfer to the Reducer (sending 1 string instead of all paths).

3.2 How the Reducer works

The **Reducer** receives the local champions from all worker threads.

- **Input:** A vector of strings (local maximums from all threads).
- **Process:**

- Iterates through the vector.
 - Compares the length of each local maximum against a global maximum variable.
 - Updates the global maximum if a longer path is found.
- **Output:** The single longest path found across the entire dataset.

4 Implementation Details

The modifications for Max Finding are implemented in `longestpath.cpp`.

Listing 1: C++ Implementation for Finding Longest Path

```

1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 #include <vector>
5 #include <thread>
6 #include <sstream>
7 #include <algorithm>
8 #include <mutex>
9
10 using namespace std;
11
12 const int NUM_THREADS = 4;
13
14 void map_function(const string& text_chunk, string& local_max) {
15     stringstream ss(text_chunk);
16     string line;
17     local_max = "";
18
19     while (getline(ss, line)) {
20         if (line.length() > local_max.length()) {
21             local_max = line;
22         }
23     }
24 }
25
26 void reduce_function(const vector<string>& all_local_maxes, string&
    global_max) {
27     global_max = "";
28     for (const auto& path : all_local_maxes) {
29         if (path.length() > global_max.length()) {
30             global_max = path;
31         }
32     }
33 }
34
35 int main(int argc, char* argv[]) {
36     if (argc < 2) {
37         cout << "Usage: ./longestpath <filename>" << endl;
38         return 1;
39     }
40
41     string filename = argv[1];
42     ifstream file(filename);
43     if (!file.is_open()) {

```

```

44     cerr << "Error: Could not open file " << filename << endl;
45     return 1;
46 }
47
48     cout << "[Master] Reading file: " << filename << "..." << endl;
49     string content((istreambuf_iterator<char>(file)),
istreambuf_iterator<char>());
50     file.close();
51
52     long filesize = content.length();
53     if (filesize == 0) {
54         cout << "[Master] Warning: File is empty." << endl;
55         return 0;
56     }
57     cout << "[Master] File size: " << filesize << " bytes." << endl;
58
59     vector<string> chunks(NUM_THREADS);
60     size_t chunk_size = filesize / NUM_THREADS;
61
62     for (int i = 0; i < NUM_THREADS; ++i) {
63         if (i == NUM_THREADS - 1) {
64             chunks[i] = content.substr(i * chunk_size);
65         } else {
66             chunks[i] = content.substr(i * chunk_size, chunk_size);
67         }
68     }
69
70     cout << "[Master] Launching " << NUM_THREADS << " threads for Map
phase..." << endl;
71     vector<thread> threads;
72     vector<string> intermediate_results(NUM_THREADS);
73
74     for (int i = 0; i < NUM_THREADS; ++i) {
75         threads.push_back(thread(map_function, ref(chunks[i]), ref(
intermediate_results[i])));
76     }
77
78     for (auto& t : threads) {
79         t.join();
80     }
81     cout << "[Master] Map phase complete." << endl;
82
83     cout << "[Master] Starting Reduce phase..." << endl;
84     string final_result;
85     reduce_function(intermediate_results, final_result);
86
87     cout << "\n[Master] === RESULT ===" << endl;
88     cout << "Longest Path found: " << final_result << endl;
89     cout << "Length: " << final_result.length() << " characters." <<
endl;
90
91     return 0;
92 }

```

Listing 1: Longest Path Logic (longestpath.cpp)

5 Testing and Results

The testing was conducted on a **Kali Linux** virtual machine.

5.1 Data Preparation

A dataset of real file paths was generated using the Linux `find` command, scanning the `/usr` directory:

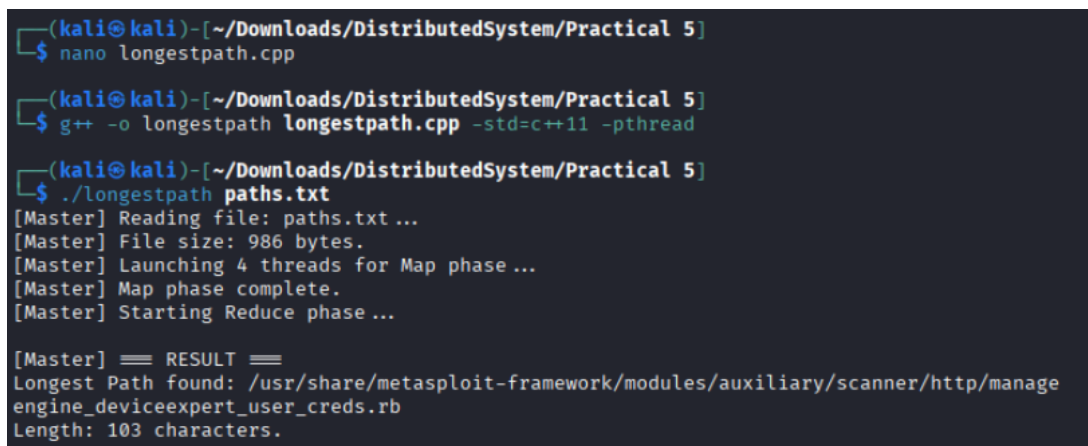
```
find /usr -print > paths.txt
```

The resulting file contained thousands of paths, serving as a realistic workload.

5.2 Execution

The program was compiled and executed with 4 worker threads:

```
g++ -o longestpath longestpath.cpp -std=c++11 -pthread
./longestpath paths.txt
```

A terminal window screenshot showing the execution of the longestpath program. The prompt is (kali@kali)~[~/Downloads/DistributedSystem/Practical 5]. The user enters 'nano longestpath.cpp', then 'g++ -o longestpath longestpath.cpp -std=c++11 -pthread', and finally './longestpath paths.txt'. The program output shows: [Master] Reading file: paths.txt..., [Master] File size: 986 bytes., [Master] Launching 4 threads for Map phase..., [Master] Map phase complete., [Master] Starting Reduce phase..., [Master] == RESULT ==, Longest Path found: /usr/share/metasploit-framework/modules/auxiliary/scanner/http/manageengine_deviceexpert_user_creds.rb, Length: 103 characters.

```
(kali@kali)~[~/Downloads/DistributedSystem/Practical 5]
$ nano longestpath.cpp
(kali@kali)~[~/Downloads/DistributedSystem/Practical 5]
$ g++ -o longestpath longestpath.cpp -std=c++11 -pthread
(kali@kali)~[~/Downloads/DistributedSystem/Practical 5]
$ ./longestpath paths.txt
[Master] Reading file: paths.txt...
[Master] File size: 986 bytes.
[Master] Launching 4 threads for Map phase...
[Master] Map phase complete.
[Master] Starting Reduce phase...
[Master] == RESULT ==
Longest Path found: /usr/share/metasploit-framework/modules/auxiliary/scanner/http/manageengine_deviceexpert_user_creds.rb
Length: 103 characters.
```

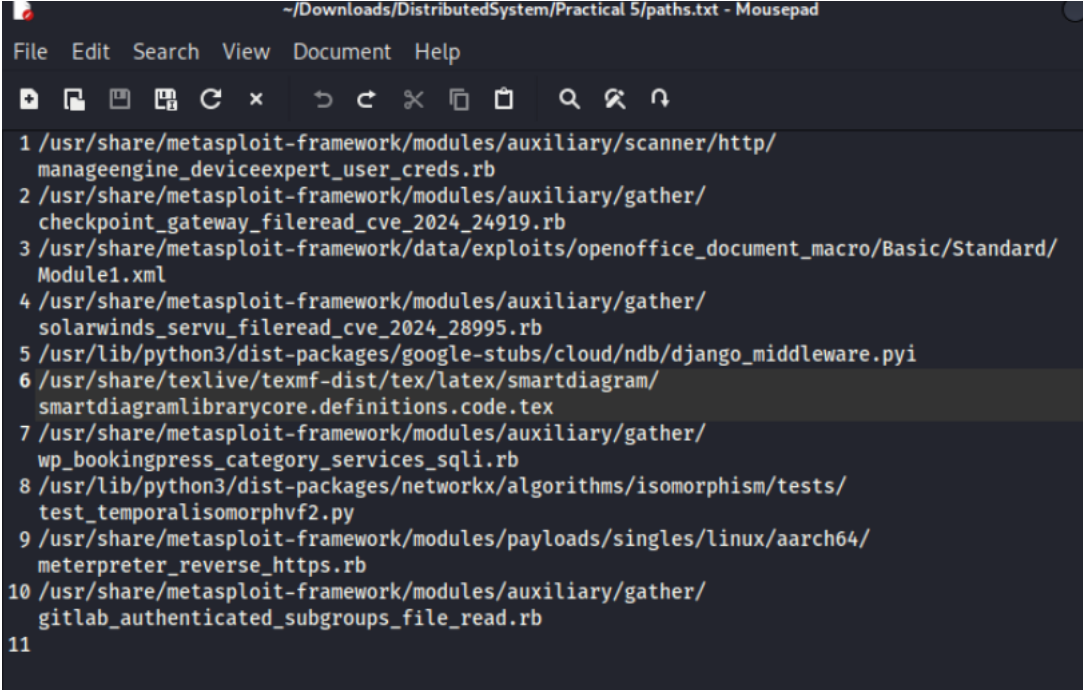
Figure 1: Terminal Execution Command

5.3 Result Analysis

The system successfully identified the longest path in the dataset.

- **File Size:** 986 bytes (Test sample).
- **Longest Path Found:**
/usr/share/metasploit-framework/modules/auxiliary/scanner/http/manageengine_deviceexpert_user_creds.rb
- **Length:** 103 characters.

The result was manually verified by inspecting the `paths.txt` file, confirming the accuracy of the MapReduce logic.

A screenshot of a text editor window titled "-/Downloads/DistributedSystem/Practical 5/paths.txt - Mousepad". The window has a menu bar with "File", "Edit", "Search", "View", "Document", and "Help". Below the menu bar is a toolbar with various icons for file operations. The main text area contains a list of 11 file paths, each preceded by a number from 1 to 11. The paths are: 1 /usr/share/metasploit-framework/modules/auxiliary/scanner/http/manageengine_deviceexpert_user_creds.rb, 2 /usr/share/metasploit-framework/modules/auxiliary/gather/checkpoint_gateway_fileread_cve_2024_24919.rb, 3 /usr/share/metasploit-framework/data/exploits/openoffice_document_macro/Basic/Standard/Module1.xml, 4 /usr/share/metasploit-framework/modules/auxiliary/gather/solarwinds_servu_fileread_cve_2024_28995.rb, 5 /usr/lib/python3/dist-packages/google-stubs/cloud/ndb/django_middleware.pyi, 6 /usr/share/texlive/texmf-dist/tex/latex/smartdiagram/smartdiagramlibrarycore.definitions.code.tex, 7 /usr/share/metasploit-framework/modules/auxiliary/gather/wp_bookingpress_category_services_sqli.rb, 8 /usr/lib/python3/dist-packages/networkx/algorithms/isomorphism/tests/test_temporalisomorphvf2.py, 9 /usr/share/metasploit-framework/modules/payloads/singles/linux/aarch64/meterpreter_reverse_https.rb, 10 /usr/share/metasploit-framework/modules/auxiliary/gather/gitlab_authenticated_subgroups_file_read.rb, and 11. The text is in a monospaced font on a dark background.

```
1 /usr/share/metasploit-framework/modules/auxiliary/scanner/http/
  manageengine_deviceexpert_user_creds.rb
2 /usr/share/metasploit-framework/modules/auxiliary/gather/
  checkpoint_gateway_fileread_cve_2024_24919.rb
3 /usr/share/metasploit-framework/data/exploits/openoffice_document_macro/Basic/Standard/
  Module1.xml
4 /usr/share/metasploit-framework/modules/auxiliary/gather/
  solarwinds_servu_fileread_cve_2024_28995.rb
5 /usr/lib/python3/dist-packages/google-stubs/cloud/ndb/django_middleware.pyi
6 /usr/share/texlive/texmf-dist/tex/latex/smartdiagram/
  smartdiagramlibrarycore.definitions.code.tex
7 /usr/share/metasploit-framework/modules/auxiliary/gather/
  wp_bookingpress_category_services_sqli.rb
8 /usr/lib/python3/dist-packages/networkx/algorithms/isomorphism/tests/
  test_temporalisomorphvf2.py
9 /usr/share/metasploit-framework/modules/payloads/singles/linux/aarch64/
  meterpreter_reverse_https.rb
10 /usr/share/metasploit-framework/modules/auxiliary/gather/
  gitlab_authenticated_subgroups_file_read.rb
11
```

Figure 2: Data Verification