CPEN 522 PA1 Report

Kexin Wen (76020940)
Cecilia CAO (62040092)

Algorithm design

Design rationale

SSH provides an intuitive way to connect to remote Linux servers. Due to its convenience, we choose this method to establish communications between servers for our project. The SSH connection is implemented using a client-server model so we have no needs to build traditional client-server ends which can save us time to focus on realizing functionalities. However, this also means we need to manually config ssh key pairs and set permissions for each server.

We use subprocesses in our python code to send queries asynchronously to each server and each server can implement simultaneously without blocking.

To ensure the fault tolerance ability of our code, we implement a timer when sending queries to servers. When attempts have been made to access the unavailable server, it will timeout and then return the exit code. Only connections with return code 0 can continue to record output and stored as a value in our resMap which has keys with corresponding ip addresses. By implementing this design, the program can run and terminate successfully with unreachable ip addresses.

Configuration:

In this assignment, we create six AWS EC2 instances as servers. All the machines store their individual log files. The configuration file contains a list of server ipv4 addresses, a list of each server corresponding log file name, which is all located in the root directory.

In the beginning, a chosen text content has been stored in every server for the first part of its lof file. To generate random contents of log files, we have written a script to generate random strings for a length of fewer than 9 characters with a total of 500 characters. These randomly generated text files are then be pasted after the same chosen content which then accomplishes each log file. In this way, more possibilities can appear when using grep commands.

Procedure:

The user login one of the servers, which would take the grep command from the user and runs the distributed grep program. The program firstly checks its own server IP address. Then construct each Linux command based on each server IP address and log file name, and pass the command into each server through SSH.

Each subprocess is conducted asynchronously. The subprocess will log in to the designated server and run the grep locally, then return the query lines back to the main server. Once all the subprocesses are completed, the program would store each subprocess result as value in a dictionary with its corresponding IP as key.

Testing approach

For unit testing, we used the python unit test library in Python. We created 10 unit test cases. The unit tests include some common grep options (simple pattern, “-i” Case-insensitive, “-c”

Output count of matching lines only, “-w” Checking for full words, ‘-o” To see only the match). It then tests four regular expressions (use the "^" anchor to match the pattern at the very beginning of the line, use “$” anchor to match patterns that occur at the end of the line, use the period character “.” to match any single character that can exist at the specified location, use bracket expressions to match the variations found within the bracket group). The last unit test checks when one of the servers is down, the program should be fault-tolerant and grep answers from all other machines that are still available.

The unit tests verify automatically that all results are what we expect as shown below.

```
----------------------------

Ran 10 tests in 2.394s


OK
```

Query latency

The latency time for distributed grep mainly depends on the query output process from servers through SSH. To calculate the query latency, we take our program and run 12 representative unique distributed grep queries on four different servers then record the average time to receive the query result. Each of the four servers stores the identical 60 MB log file. The graph below shows the results of those 12 grep queries. We conclude that the more query output, the higher latency will be but the overall time consumption is tolerable. By observation of terminal during runtime, the whole process ran smoothly.