# CPEN 533 – Assignment 1 – Scott Greene and Oksana Hlushko May 20, 2019

System Design

This section will outline the design of our system. We have chosen just to create our programs in single java files to make it easier to run and test locally as well as deploy to the EC2 instances. We chose to write two different programs, a Server and a Client program. Both programs connect using TCP/IP to communicate the data between themselves. Once again, for simplicities sake we have hardcoded the port, private IP addresses of instances, and log locations. We would expand this functionality in the future to have self addressing log files, and external config files for network information.

The Server program will be constantly running on all the instances. The server creates a thread pool that will allow it to handle multiple requests at the same time. Each thread will individually handle a grep command from the user so that if multiple clients are all requesting at the same time then that is fine. The finePhrase function takes in the regex expression that the user wants (using java.regex formatting) and then it will search the log file and add any resulting lines to a response vector. This vector is then responded back to the Client. The Server will then close the socket and wait for an new request.

The Client program is called to search for any regex required by the user. There are no arguments, and currently the network info is hardcoded with the port and private IPs. When the application begins, it asks the user for the regex phrase they would like to search for. It then evaluates the number of instances that it will be searching by getting the size of the address vector. It will create a new thread for each instance (including itself) and connect to the server on that machine. It will currently just pass the regex that it is looking for and await a response. Each thread will print the responses to the screen as they come in.

If any of the instances are not running then the response will include an exception error line that has the IP address and what the exception is. One possible case that could cause an issue is if the server crashes mid request. We have put a socket timeout on the client so that if nothing comes in for 10 seconds then it will return the thread. The biggest improvements we could make would be to better organize and structure out code. We think the approach should be fast due to using each instances resources in parallel and then just dumping that to the screen.

Testing

Unfortunately to do time constraints we were unable to implement the testing in the most ideal way. We have done manual testing to make sure that the processes are working and have run the system with 5 instances open. Each search request will print out the IP searched and total number of responses found at the end of its thread. Once all threads are complete, the main program will print out the elapsed time in milliseconds to complete.

Due to a lack of time we implemented just some basic automatic testing. We loaded 5 different static log files taken from different points in time on a machine. We then figured out the number of instances for certain test strings so that we could compare the returned numbers with the expected numbers.

\*\*\*\*\*All test log files were of size ~60KB instead of 60MB because we misread the requirement. We will redo this part and submit the testing once available\*\*\*\*\*

Test 1:

This test was done to find all lines that had the text NetworkManager as an exact match anywhere in the string.

File1: 22 instances

File2: 31 instances

File3: 36 instances

File4: 28 instances

File5: 30 instances

This whole test took (154, 90, 60, 87, 72) milliseconds. This gave us an average of ~92 milliseconds to complete this task.

Test 2:

Search for a string that does not exist in any of the files.

This test took (27, 21, 16, 14, 26) milliseconds. This gave us an average of ~20.8 milliseconds.

Test 3:

Every line of the test files contains the computer name virtualubuntu. We also know the number of lines in the files so we can match the number of returned lines to these lines. \*\*\* The client is written so that you can search over and over until done with the program. This test was done without closing the application.\*\*

This test took (260, 143, 87, 92, 81) milliseconds. The average time for these five tests was ~132.6 milliseconds.

Test 3 – 2:

Repeat with closing the application and starting a new one between each request.

(227, 227, 235, 256, 259) average of 240.8 milliseconds per request.

Next Steps

We are currently unhappy with the layout and structure of our code and testing. We plan to properly implement a logger with automatic testing functionality built in. We would also be able to use this logging class to add to all of our future project classes.