CS 425 – MP1 Report

Alex Huynh (ah10), Kaimeng Zhu (kaimeng2)

Design Description

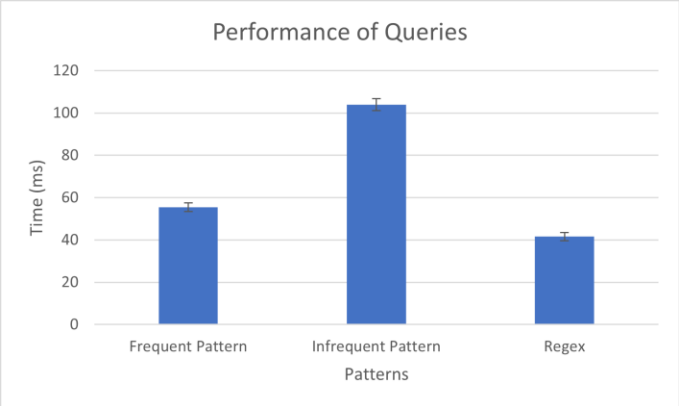
The high-level idea behind the design was like a chat server. Any machine could send messages (grep commands) into a message stream, and the connected machines will run the message when connected. They will then either return their output or timeout if they took too long.

On a low-level: client side handles parsing command from user terminal input for grep and transmit it to server, while server handles calling system's grep function using Golang's exec() function. The main function first establishes the server as a goroutine, which listens on a specified port. Then the main function will attempt to establish connection with all servers, including the server located on same machine as main(). Then the main will enter the request loop, which will keep parse in user input, send it to a helper server_handler object, and server_handler will call its corresponding server's grep function. This is also done concurrently. The server will transmit back result from grep to a shared response channel shared by all servers. main() will display results from the response channel and begin the next iteration of request loop.

Unit Tests

The unit tests written were as follows: a basic test to see if the server was properly started. This was tested by sending a message onto the port. Another test was to see if the `grep` properly ran queries. This was done by generating a random batch of log files with known and random text and seeing if the API call on our grep function returned the proper result.

Performance



Three queries were done five times on the main function, and the average latency with standard deviation of the results are as shown. There is a decrease in performance for infrequent patterns. For frequent patterns, the program does not need to finish searching a line after it finds a pattern (which it frequently will, by definition). However, infrequent patterns will have to fully parse more lines. Regex, interestingly had the best performance despite also being an infrequent patterns. One hypothesis is that the regex pattern in testing was longer than the other patterns, so `grep` did not have to look as extensively i.e. it could stop `n - length` from the end.