CS 425 MP1 Report by Aarya Bhatia and William Zheng

Server Architecture

Our server is written in C/C++. The main thread is responsible for listening for TCP connections on a given port and creates a worker thread for each connection. The worker thread will read the request from the client, run the requested shell command in a child process, and send the output back to the client.

Client Architecture

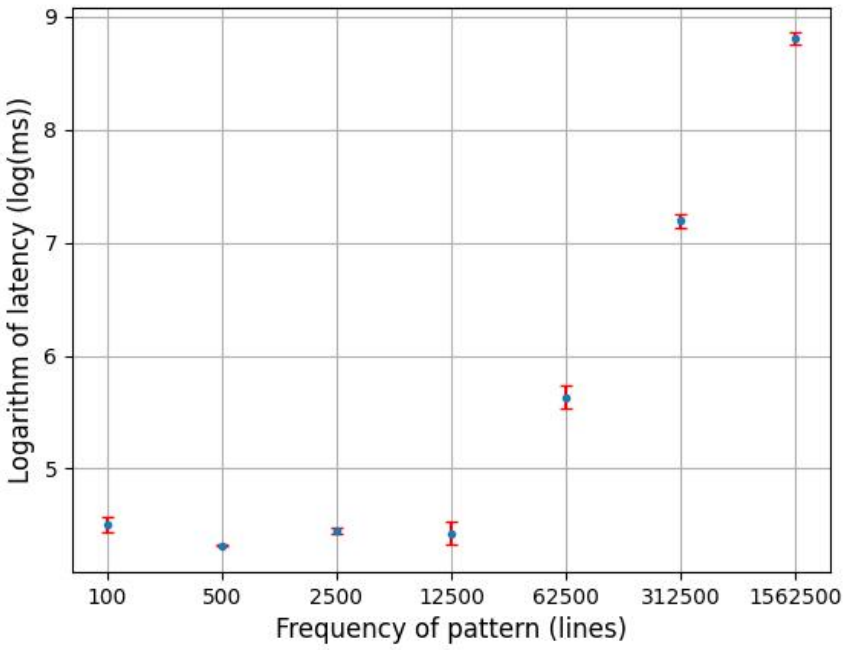
The client is written in Go and provides a CLI to run commands, silence the output or configure input and output directories. The main function parses the CLI arguments into a struct and calls the RunClient function, which sends the request to the server and collects the output. The client reads the server address from a file and creates a thread for each connection. The server output is synchronized with a queue. Each session collects data about the total lines, bytes, latency, and successful hosts.

Testing

Our test performs grep on a known generated log file, deployed to each VM. The test case is considered as a pass if the number of lines matched equals the expected number of lines.

Results

We measured the latency for 5 trials for each query with 4 randomly chosen hosts. The following plot describes the frequency of the query pattern on the X-axis and the logarithm of the average latency on the Y-axis with a standard deviation error bar. We notice a linear increase in latency across the different frequency patterns. This result is expected as the frequency patterns scale exponentially.



Conclusion

In our design, the servers can simultaneously run the grep command in the backend and send the output to the client, where the lines are combined into a single data stream. The network I/O becomes a bottleneck for large query sizes. Overall our architecture is scalable and fault-tolerant.