

## MP1 Report

### Design

For our design, we chose a client/server model where clients direct their queries to multiple servers parallelly. The servers run grep on their local files and send back their results. Servers can also handle queries from multiple clients parallelly.

We assessed that fetching entire files to the querier and then performing grep at the client side would involve a waste of network bandwidth and the client side can become a bottleneck as the number of servers grows. By streaming only matching lines, we instead made our design more scalable and effectively minimized network overhead, especially in the case of grepping infrequent patterns. We can observe this when grepping for frequent patterns as well, to a lesser extent (which is still better than downloading entire files). A client loads a list of servers and starts a thread for each server. The thread handles both transmitting the query and receiving the results. After each server performs grep locally and streams the results back to the client, the client thread counts the number of matching lines and adds it to a running total. The thread then outputs the results and counts to the console in a thread-safe way.

### Testing

We have created several unit tests for basic gadgets such as generating logfile names and creating sockets. Additionally, we have tested querying for patterns that are considered rare/somewhat infrequent/frequent and regular expressions on log files generated on each machine.

### Plots

We run the grep GET (which is one of the most frequent patterns) on 4 machines with 10 trials. The average query latency is 381.90 ms. We used the same files across our machines between trials. The ordering of results differed across trials which is expected in a parallel design, but the correctness of output in terms of the result contents and line counts remained the same. The latencies across trials remained within a narrow standard deviation of 18.58 ms, which shows fairly consistent performance.

