MP1 Report: Group 12 (Sharayu Janga - sjanga2, Aarushi Aggrwal - aggrwal3)

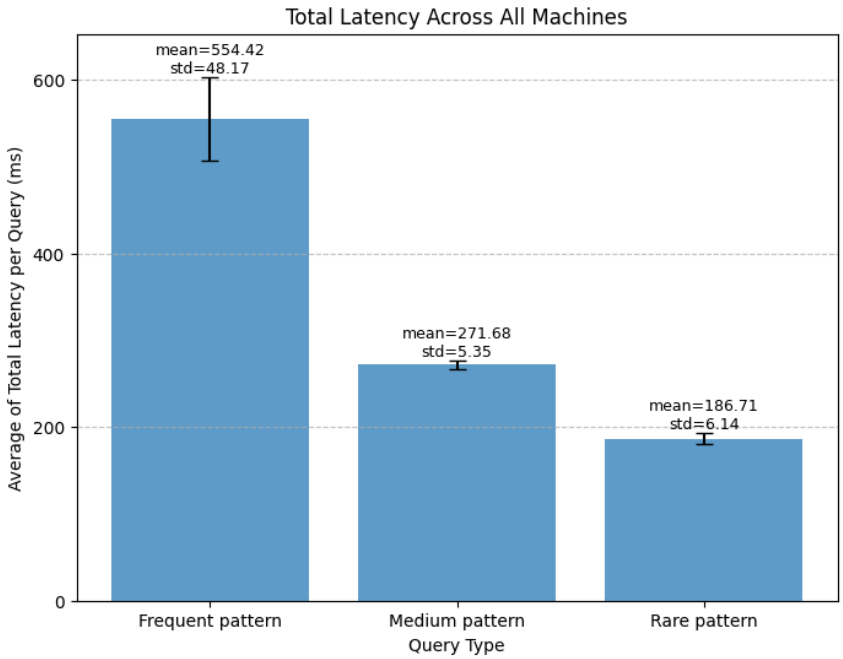
Design of the Program

Our program uses Go's built-in RPC framework to search logs across multiple machines at the same time. Each server is started by running server.go, which registers a CommandExecutor struct as the handler for any client requests. The Run method for CommandExecutor receives shell commands with their arguments from the client, determines the log file name based on the machine number, and executes the command locally using exec.Command. The output or error is returned to the client using a reference to a string, reply, on the client. The server listens for TCP connections on port 1234. For every incoming connection, it runs rpc.ServeConn. This allows the server to handle multiple clients without needing to restart. On the client side, the user connects to all active servers using their IP addresses and port 1234. The client provides a terminal-like interface to the user for entering grep commands with any of the regular grep flags and the search pattern. Each command is sent to all connected servers using the RPC method CommandExecutor.Run. The client aggregates and displays the results from all servers. At any point, if any of the servers fail or disconnect, the client will print out an error message indicating that the machine has disconnected. It will still continue querying the active servers and displaying their results.

Unit Tests

Our unit tests generate sample logs and verify the results of different grep queries. We first generate logs with random characters but some known patterns injected with specific frequencies. The unit test checks that the log files are created successfully on each VM before searching. We then run various grep commands using the client, checking for correct pattern counts across all machines for rare, medium, frequent, etc. patterns. We also test for nonexistent patterns and regex queries to confirm correct matching and aggregation. For each test, the client sends a grep command to all servers, and we check the output from each VM to ensure the number of matches aligns with the expected values for that pattern and machine.

Timing Analysis



We analyzed the performance of our program by measuring the time it takes for a single grep query from the client to execute completely (on all 4 VMs). Each VM has 60 MB log files. For each pattern, the average total latency per query over 5 trials varied significantly. Frequent pattern searches take the longest, while rare pattern searches are quicker. We know that grep is O(n), which indicates that the amount of data transferred from server to client is the bottleneck of this program. We also noticed that initial queries took much longer than subsequent ones. This may be due to caching benefits at the servers when querying the same file multiple times. After the first couple of queries, the query times for a specific pattern stabilized.