G64 MP1 Overview (bofanc2, yiweiw4)

This distributed, fault-tolerant log query program is written in Go using net/rpc for socket connections and data transfer. It follows the server-client model.

Server

The server listens on a port and runs system grep commands on request. It resolves the target log file (/home/shared/logs/vm#.log), executes grep, and returns the result via RPC.

Client

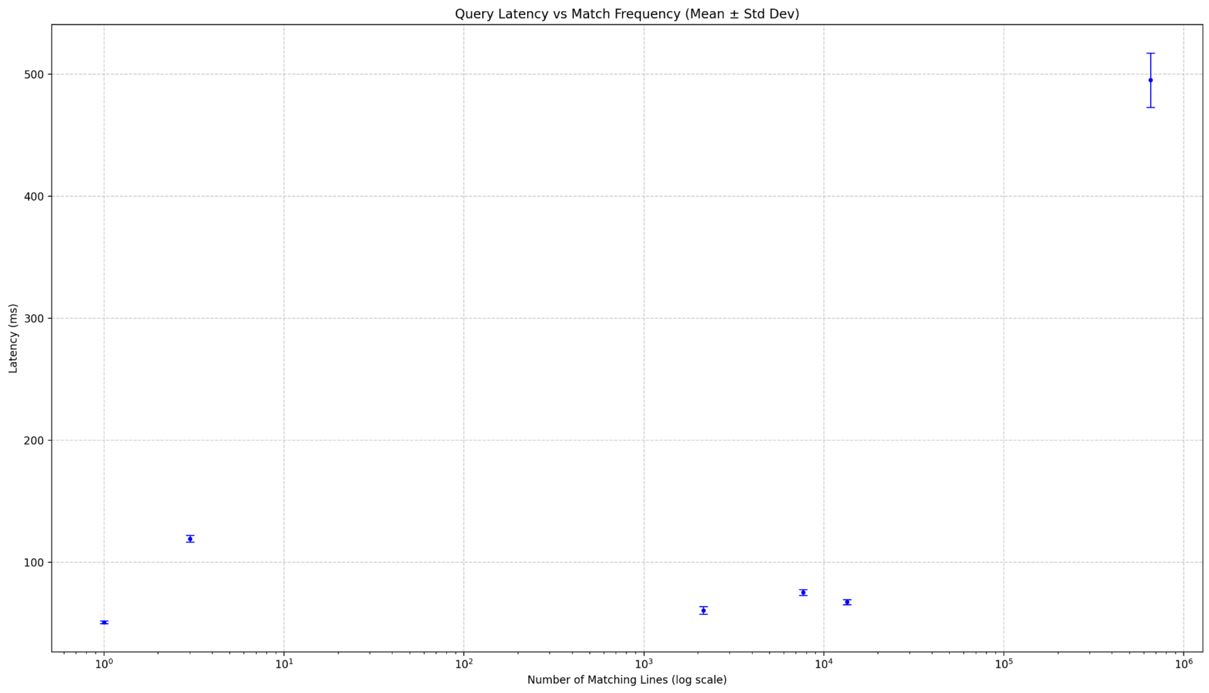
The client reads /home/shared/sources.json for the list of servers. For each server, it spawns a worker goroutine that connects via TCP, calls the RPC function, and stores the result. All results are collected in an array for writing output and printing summaries.

Unit Tests

We implemented unit tests in Go that test through the mock local server with generated random logs with controlled content. We test the client function getLogAll and validate results by re-reading saved `.txt` output files and count line numbers, including frequent, infrequent, regex pattern, crash simulation, and no match.

Performance Evaluation

We run the queries on 4 VMs (6401, 6402, 6403, 6404) each with ~60MB of data with total lines of 1,091,209.



GET — 654,694 matches, 495.047 ms avg, 22.120 ms std

DELETE /explore — 13,481 matches, 67.290 ms avg, 2.067 ms std

200 4995 — 7,659 matches, 75.098 ms avg, 2.356 ms std

washington — 2,142 matches, 60.446 ms avg, 3.107 ms std

141.[0-9]{2}.[0-9]{3}.86 — 3 matches, 119.198 ms avg, 2.626 ms std

17/Aug/2022:19:12:34 — 1 match, 50.765 ms avg, 1.050 ms std

Discussion:

The trend observed in the plot is consistent with our expectation. As the frequency of matched patterns increases, the latency goes up due to large result transfer. The latency is much higher when a regular expression is passed in even as an infrequent pattern, compared to plain string.