MP1 Report Amaan Khan (amaanmk2) & Samaah Khan (skhan330)

Design & Algorithm

Every machine acts as a server and a client by having both run in parallel. Every machine is connected to every other machine using a TCP socket.

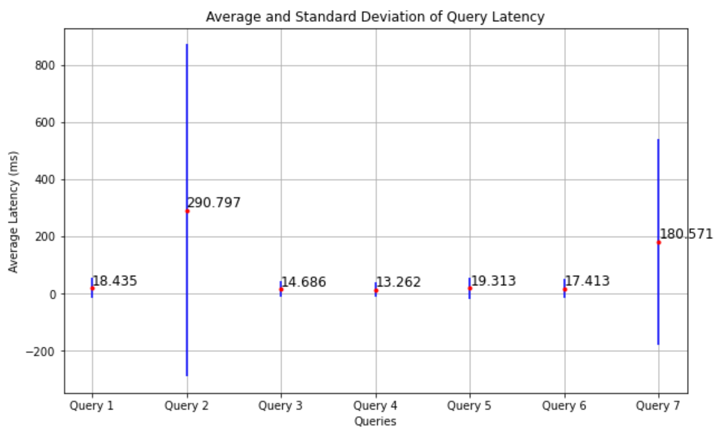
When a grep query was written by the user, the client on that machine sends the query to all actively connected peer machines (i.e., it is not sent to machines who's connection was disconnected). The server on each peer machine would read the query, check if the query is present in its local cache, retrieve the output from the cache if its present otherwise it executes the grep query locally, and then it sends back the grep output as a serialized struct containing the filename, output, number of lines, and execution latency. In the meantime, the client waits to receive the outputs from its connected peer machines, and once it does receive everything it prints it all out, including the execution latency of the time between the query was written to the time all the outputs were printed out to stdout.

Since frequent grep queries are slow and take time to execute, every machine caches the outputs of a grep query it executes to save time for subsequent calls to the same query. Every machine has an in-memory LRU cache where the key is the grep query and the value is the grep output struct. Also, we executed grep queries by having it execute the existing grep shell command.

Unit Tests (brief)

We wrote unit tests to ensure the program is able to parse the user input and return the correct output from the grep command (without connecting to other servers). We also created unit tests to ensure the program correctly serializes and deserializes both the Grep Query and Grep output structs. Additionally, we created tests to test the overall functionality of the program of if it correctly returned the grep outputs from all the machines. We did this by running `main.go` inside our testing program and piping the grep inputs into stdin (assuming the other VMs are booted up and connected). The program then outputs its grep outputs to a JSON file, from which the test suite opens it up and evaluates if the values are correct.

Plot



This is the plot of 7 queries that we ran using 4 VMs and on 60mb data log files (the log files given to us for demoing). Query 2 and Query 7 were complex regular expressions (exact queries can be seen under `scripts/plot_data.ipynb`), which is why we believe the standard deviation was so high. The first time it was ran the latency was quite large, but in subsequent runs the latency was consistently small since it was pulling the output from the caches, explaining why the standard deviation was so high on those. The other queries, however, didn't have large standard deviations which we believe is because despite the result being cached for subsequent runs, the bottleneck for the latency was more so for sending the data across the network rather than the grep run or the cache retrieval itself. We know this because we displayed the latency for individual grep queries from each machine.