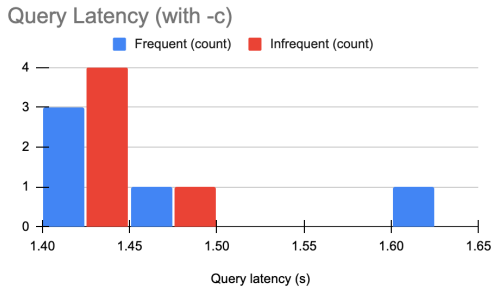
Distributed Log Analyzer Report
CS 425 - MP1 - Group 70 (@sushmam3 and @jadonts2)

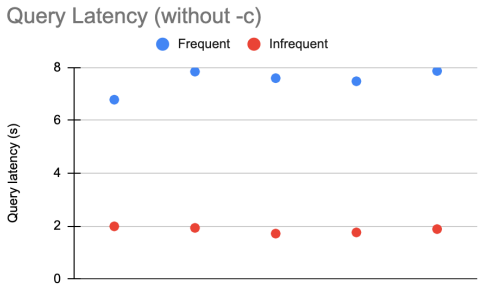
Design: We used the client-server architecture with multithreading. All machines are treated as servers, waiting for grep requests. Any machine can act as a client and send grep requests to all servers. A client employs multithreading to connect to each server, send grep queries, receive responses, and print them locally. In order to ensure printed results are not intermixed, each thread acquires a lock to print. If any machine is offline/crashed, it does not affect the response from other machines. The server executes the grep query locally and can handle multiple client grep requests at once through multithreading, so there is no delay in the response to a grep request, even when there are multiple clients querying at the same time. Thus, our system is fault tolerant, scalable, and quick.

a) **Code:** For the grep implementation we used the unix4j java library. We have also implemented grep options. Unix4j library does not have the -E option, but it does query on regex. Client and Server can be run using client.sh and server.sh script respectively. Details of the servers that the client can connect to are provided in a json file.

b) **Unit Testing:** For unit testing, we used Junit. We tested all functional units of code, including load testing for ~700,000 matches, and also had distributed unit tests to make sure results were what we expected across all servers.

Performance Metrics: We tested with 4 servers running on 60 MB files, using frequent (~25,000 matches per log) and infrequent (~5,000 on a single log) patterns both with the -c option and without. Since the search algorithm grep uses somewhat depends on pattern length, we fixed the patterns to 8 characters long each.





On the left we see that when using the -c option, frequent patterns are more likely to be slower while infrequent patterns are more likely to be quicker. However, performance is similar in the context of what the user perceives. For this reason, we also tested without -c, since we expected the bottleneck to be the data transfer and printing. As expected, the frequent pattern was again slower, but to a greater degree. The below table summarizes the results:

		Mean	Std
With -c	Frequent	1.479 s	0.095 s
	Infrequent	1.426 s	0.021 s
Without -c	Frequent	7.518 s	0.441 s
	Infrequent	1.857 s	0.115 s