CS425 / ECE 428: Distributed System, Fall 2022, MP 1

Name: Hang Yu (hangy6), Tian Luan (tian23)

Design Outline

We utilized the Client/Server architecture to implement the whole distributed log query system so that we can query log files on all machines from any one of these machines. To execute the log query, every machine needs to run a server. Then we run a client to send the request to these servers (including its server) concurrently by RPC to execute the 'grep' command. Every server will 'grep' the result from logs and send the result back to our client; then, the result can be output appropriately. We chose Go to implement the system. We applied Go's channel and sync package to guarantee output success. Go has features that allow us to implement concurrency elegantly. We used a goroutine for each RPC the client calls, ensuring concurrency while providing considerable fault tolerance since the program does not break when an error occurs on the server side. The error message can be sent to the channel and output appropriately so we can troubleshoot the error efficiently.

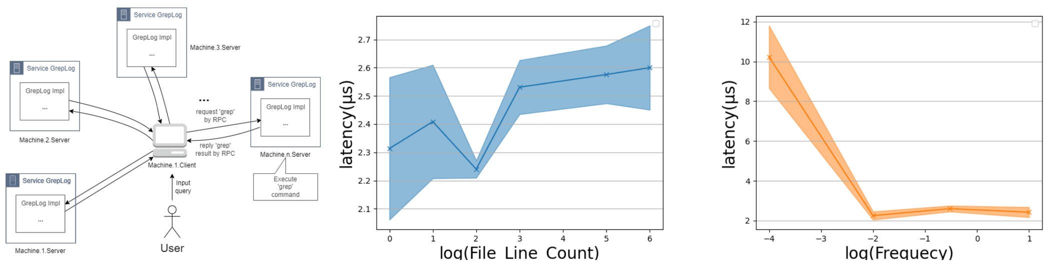

Figure 1: Framework of querying log (L);

Unit Test

We use standard go test to do the unit tests. All the query tests locate in ***local_test.go*** and ***distributed_test.go*** which contains: query to one local/remote server and query to multiple servers and collect log from very small to large size. We also change the frequency of the query and test. The querier will always collect all the logs so when not all VM are used, we regard it as an automatic test to the situation when some servers are down. In this case, the output log will tell that the unreachable server "connection refused" which is as expected. **Empty query is regarded as invalid input and not handled.** Also, when we increase the line counts in the log files, we keep the frequency of the content to be queried fixed; When we modify the frequency of the content to be queried, we fix the line counts.

Conclusion

End-to-end output latency of averaged five runs are shown in the plot. Our "end-to-end latency" refers to the period during which querier client get the query, process and send to desired servers, fetch the results from the servers, and write to the output. X-axis is log-scaled as we increase the test logs' line count exponentially and y-axis for the latency(μs). We find that:

1. With exponential grow of line numbers at a fixed frequency of content to be queried, the latency grows linearly.

2. Giving fixed line counts, the lower frequency cause higher latency.

We speculate that the above phenomenon is mainly due to the bottleneck caused by the network latency, which reasonably explains why the output latency does not increase rapidly with the exponential growth of the number of log lines when the frequency is the same. Of course, the pattern frequency is still a primary influence of latency. The sudden drop in the figure could be due to the error by chance, because we only test four times for the logs of same lines.