CS 425 Distributed System MP1 Report

Group 43: Yash Saboo (ysaboo2), Tzu-Bin Yan (tbyan2)

I. Design

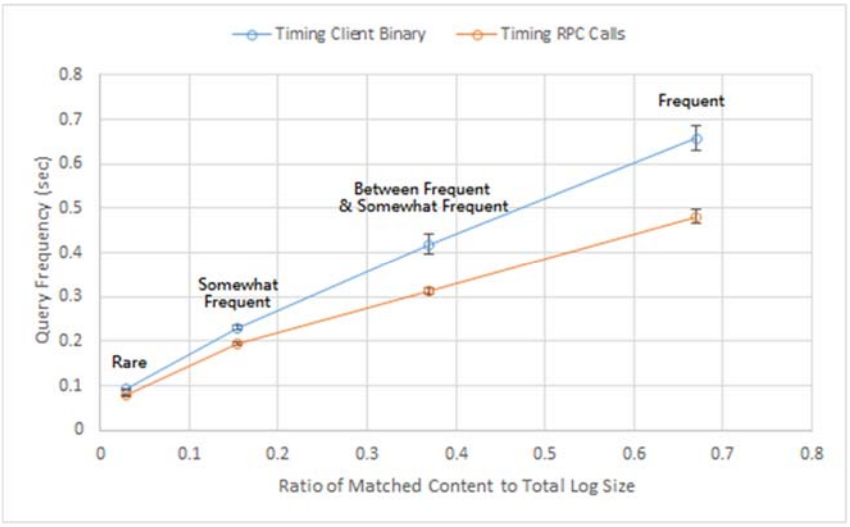
Our architecture design of this MP mainly constitute of two programs, one being the client (client.go) and the other one being the server (server.go). The client is a program, which runs locally on any given machine with Internet access. The main capability of client is that it sends the given regex pattern to the remote servers (via golang RPC), retrieves the regex-matching entries from all available servers, and combine them for display. The server is a program that runs on the target VM and serves the log file of the particular VM. The server stays on and waits for client to call its RPC and based on the supplied pattern, the server calls the grep command and, after some alterations to each log entry, return the grep results back to client for display.

II. Unit Tests

The unit testing for this architecture is done by first generating the logs that differs based on the VM number and then distribute them based on the corresponding VM, with copies of all of them being stored locally on the machine performing unit testing. Based on the given logs, the unit testing program calls the client.go program (multiple times) with regex patterns (which in turn queries the remote servers serving the previously distributed logs and returns the results) and then run GNU grep locally to see if the two corresponding results matches or not.

III. Average Query Latency

For the average query latency, we ran the distributed greps on four VMs, each storing 60MB of generated log files using the genlog.go utility we wrote in this MP. The client program is always run on the first VM of the four VMs used. There are two different timing schemes, namely "Timing Client Binary" and "Timing RPC Calls". The "Timing Client Binary" times the whole time needed for the compiled client.go binary to execute based on a given regex pattern. The "Timing RPC Calls" times the time used to call RPCs on the server and get the results. A total of 4 query patterns are used, each returns a different portion and amount of logs. The results of the two schemes are shown in Graph I (each circle represents the mean of the five trials per pattern, the sample STDs are plotted next to them as black lines). We can see that there is an overall tendency that when the returned log size increases, the query time for both schemes also increases. The increasing tendency is roughly linear to the ratio of matched content to total log size, with the rate of increase of "Timing Client Binary" being larger than "Timing RPC Calls". We believe that the reason for the linear tendency is that there are some parts of the client program running in linear time of the matched content (e.g. the retrieving matched entries and displaying part) and that there isn't any other part that has higher complexity. As for the higher increase rate, we believe it is due to the "Timing Client Binary" also timing the displaying part (runs in linear), which is not considered in the "Timing RPC Calls" scheme.



Graph I. Query Testing Results