MP1 Report

chinhao2 Chin-Hao Lo
hcma2 Ho-Chih Ma

Overall Design

We implemented a single client to multi-servers design using socket and multi-threading. The process starts with collecting and pickling user input. We utilize ***Pool.apply_async()*** to establish multi-threading functionality and connect to numerous servers. We then check if the connection is successfully set up and utilize sockets to send pickled queries to each server. The client would wait for the response from the server via the ***get()*** function. On the server side, each server, running on its own thread, starts listening once connection is set up. Once data is received, the server runs subprocess, "***grep data_received vm*.log***," (Note that there is only one vmX.log file residing on the same directory as a server, i.e. vm1.log on server 1) The server sends back the result of the subprocess and closes the particular thread. The client then collects all responses from working servers, outputting the result through stdout.

Unit Test - Pytest

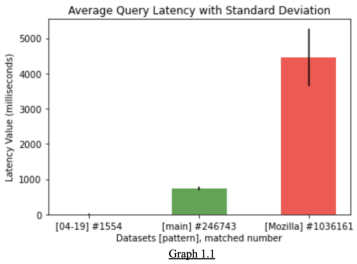
Multiple unit tests were written to make sure we cover rare, somehow frequent, and frequent patterns, and patterns that occur in one/some/all logs. The unit test runs grep locally to collect matched patterns from all vm.logs. Then, we set up a client and request all servers to run grep remotely on its own VM, fetching matched patterns from their own vmX.log file. We first make sure both lists of matched patterns have the same size via ***assert***. If the sizes are identical, we sort both lists, then call assert on the sorted lists.

Testing Results

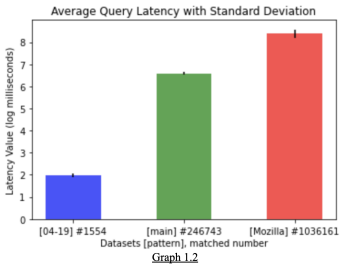
We collected the numbers of matched patterns and process time of grepping "04-19", "main," and "Mozilla" on 4 servers (grep from vm1.log to vm4.log). Table 1 contains mean, standard deviation of our testing results (tested five times for each pattern). We plotted the means and standard deviations in millisecond, as shown in Graph 1.1; graph 1.2 displays the log of our data to provide a more visually helpful representation. We believe that our system roughly performs at O(n) time complexity as the average time spent for each matched pattern is almost the same for all three inputs.

	04-19	main	Mozilla
# of Matched Pattern	1554	246743	1036161
Mean (ms)	7.25911	727.94188	4459.45506
Standard Dev. (ms)	0.71676	47.00884	816.79811
Time per pattern (ms)	0.0046712	0.0029502	0.0043038

Table 1



Graph 1.1



Graph 1.2