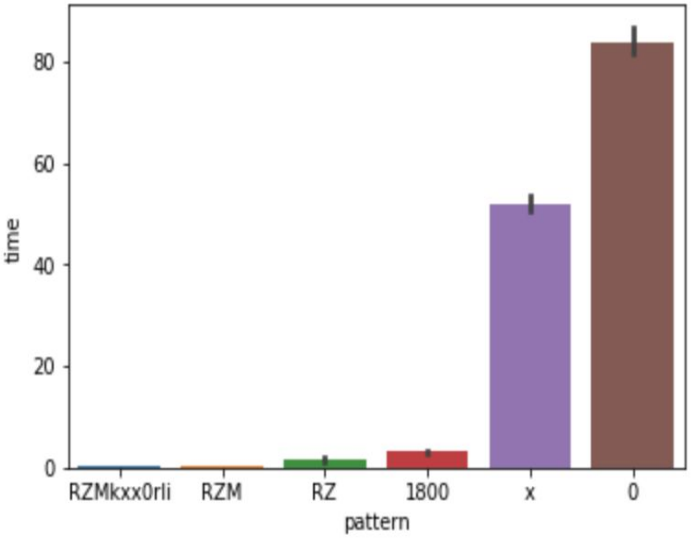
CS425 Distributed System MP1 Report

Tanqiu Liu (tanqiul2), Dehua Chen (dchen51)

Design - The distributed query system has a client-server architecture over TCP connection. Each VM maintains a dlq_server.py daemon running on it and the daemon binds to port 12345, listening to income connection from the clients. When calling dlq_client.py, the client process firstly read conf.json to get the information about the nodes in the cluster and connect to each of them. The sockets for connections, node status, query completeness, message buffers are handled in a dictionary. The client sends query messages to each of the server. The servers receive and parse the message, and do query by raising a subprocess which calls the system "grep". After the system grep gives an output, the servers send back the results to client. Therefore, the servers do query and send back results simultaneously. The client receives results from different servers by looping over all the nodes and receiving and printing a chunk of message of fixed length from each server each time. A node is marked as "complete" when there is no more message from that node and when all nodes are "complete" or "crashed", the client stops looping.

Unit Test - The unit test (dlq_test.py) has 5 hard coded tests that test the line count received from all the vm logs that were given to use for the demo with the line counts received from servers during a query from the client. There is also a 6th option that runs all patterns in a file, which includes even more test cases and more can be added to the file that the unit test can run.

pattern: RZMkxx0rli, frequency: 7,7,7,7 time:0.1342, 0.1301, 0.1267, 0.1260, 0.1257 average: 0.12854 std-dev: 0.00361
pattern: RZM, frequency: 251, 248, 245, 246 time: 0.1674, 0.1489, 0.1494, 0.1601, 0.1628 average: 0.15772 std-dev: 0.00824
pattern: RZ, frequency: 15508, 15483, 15449, 15449 time: 1.242, 2.476, 1.642, 1.111, 1.022 average: 1.4986 std-dev:0.595
pattern: 1800, frequency: 142880, 891, 587, 605 time: 3.206, 2.971, 3.091, 3.094, 3.343 average: 3.141 std-dev: 0.140
pattern: x frequency: 637772, 637882, 638104, 638097 time:51.129, 51.608, 55.248, 50.585, 50.375 average: 51.789 std-dev: 1.9926147394818
pattern: 0, frequency: 1M for each vm time: 89.664, 83.337, 81.305, 82.504, 82.720 average: 83.906 std-dev: 3.3021101586713



The trends we see above are based off running the vm logs we were given for the demo (around 70-90 MBs each log) for vms 1-4. This graph is also based off the query latency from the start of when the query runs and when it ends with all lines printed from the vms printed out on the client terminal. The times are averaged and the standard deviation is the black tick at the top of the bars. The trend is to be expected as we print every single line from large files and rare or infrequent patterns take less than a second while the more frequent ones that show up in almost every vm log take much more time.