CS425 MP1 Report

Fangwei Gao, Xiaolan Ke
fangwei2@illinois.edu,

Design

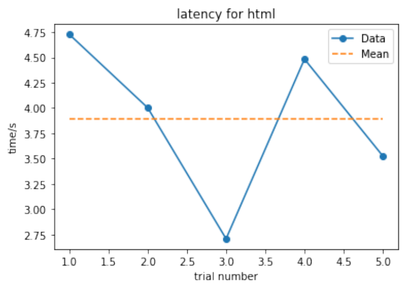
Since the requirement of this project is to perform grep command on multiple logs on different machines, all the task can be split into two major parts. One is to set up reliable communication channel to pass information from one master node to N server nodes (N = 10 in our implementation), the other is to perform grep command on each individual node. In the communication part, we built up a model with one server and 10 clients, which use sockets as their only way of communicating. 10 Server nodes are listened on the same specific port and the client will launch multiple threads to dial all the servers after it receives the proper grep commands from the console. Once the socket connection is established, the client will write the parsed grep command to all channels and wait for the server sides to get the command, perform the command locally and then send the result back. Line number of match result for every VMs also get sent to the central client.
Notice that the operation of 10 servers and their communications with the client are handled simultaneously using Go routine and Go channel. We did not send all the whole log files to the client for the sake of performance when infrequent pattern is used.

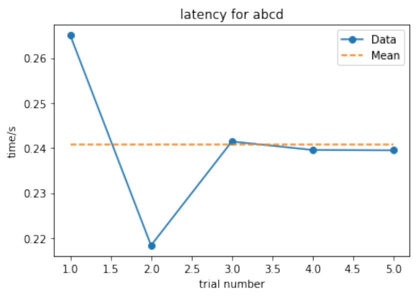
Unit Test

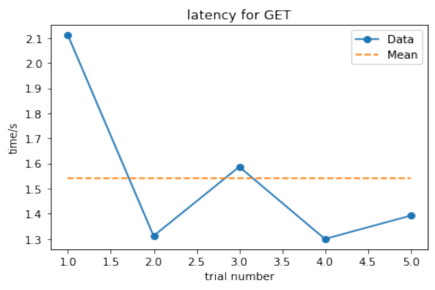
The test part is mainly written in Python code, which includes a random log generator, a deployment unit and a test unit. We firstly used the random log generator to create 10 random log files of more than 60MB each. Then the deployment unit would deploy the client/server code, log files and setup shell scripts to all the VMs using ssh. Then the test unit will run the script remotely and dump the result from the client back to the test environment. Finally, the test unit would run the same grep command directly on ten VMs and compare the result with the outcome of previous distributed grep that was performed on the client node line by line. We tested our program with words of different frequency such as “html” (very frequent), “GET”(frequent)” “abcd(rare)” and many other words using regular expression.

Performance

We performed three measurements with different workloads. Their runtimes for 5 trials and average runtime and STD are plotted below.
“html”: 6550489 Lines matched in total; 0.7207964806867989
“GET”: 3276483 Lines matched in total; 0.30331799084862626
“abcd”:138 Lines matched in total; 0.014799741145800538
The increase of the latency is almost the same as our expected. As the matches for query increases, our program would require more run time to handle the socket io and file read/write io etc, which result in overall more runtime need for a complete query.


latency for html
Data
Mean
time/s
trial number


latency for abcd
Data
Mean
time/s
trial number


latency for GET
Data
Mean
time/s
trial number