CS425 MP1 Report Wenhao Su (wenhaos3) Yichen Yang (yy18)

Distributed Log File System Design

The distributed log file system is written in C++. We use socket for communication.

Once the server initiates, it calls *run_server* function, which runs an infinity loop to listen to the 8000 port. Upon receiving a socket connection from a client, the server creates a new thread of *handle_connection* function. This new thread runs the received query using the *grep* function and sends the result back to the client.

Once the client is launched, it calls *run_client* function, which runs an infinity loop to display the command-line user interface and captures the query input from the user. Also, it starts a new thread *process_queries*, running an infinity loop monitoring the data structure *queries*.

Once user inputs the grep command, it will be pushed into *queries*. If *queries* is not empty, *process_queries* will obtain the string in the front of the query, pop it and create multiple *send_grep_query* threads to send queries concurrently to cloud servers.

Upon receiving the result from server, *send_grep_query* will create a temp file to store the result in local file system, and denote the related query_status to be true. If the server is down and it meets a connection failure, it will denote the related ip_status to be false.

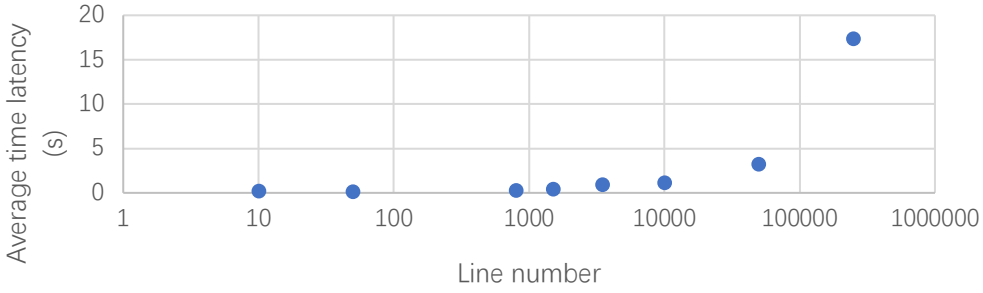
When all the results from alive serves are received, all temp files are merged into "result.out" and the line counts are printed in the command-line user interface.

Unit Test Design

Generate some pre-specific logs with random number and calculate the expected result to get the corresponding test result and test request for verification. It will contain rare, frequent pattern of request. Then we can run the test shell to check whether we can get right results.

Average query latency (4 machines each store 60 MB log files):

Grep Request	Line	Average Time (s)	Standard Deviation
grep '178.165.128.5' vm.log	~10	0.1526236	0.044361698
grep '12/Dec/2015' vm.log	~50	0.12627964	0.03644078
grep 'apple' vm.log	~800	0.250053625	0.082478654
grep 'Dec/2015' vm.log	~1500	0.3632555	0.137514784
grep -E '^[0-9]*[a-z]{5}' vm.log	~3500	0.9020441	0.389278789
grep '211' vm.log	~10000	1.146007778	0.15286987
grep '2017' vm.log	~50000	3.187451	0.179522016
grep -E '*' vm.log	~250000	17.32679	0.226240008



From the above table and plot, we can see that the standard deviation is very small, which shows the time cost of the system is stable to the requests.

For the trend, we can see that the time cost is proportional to the returned line number. With more lines sent back, it will cost longer time. It makes sense since more lines will cost longer time to send and read it. In addition, store the received data and read the line number will also cost time. Therefore, we think it meets our expectation.