CS 425 MP1 Report

**Team Members:** Kuan-Yu Tseng, Ksheerasagar Balaraju  
**NetID:** ktseng2, balaraj2

# System Design

Distributed Log Query System comprise of client and server components on each node. Any node in the distributed system can query the logs from all nodes, and display the result. The implementation is done using Java, and sockets are used for Client-Server communication.

## Log Query Client

The client component comprise of LogQueryClient and LogQueryClientThread classes. The client adds the list of nodes (including local host) on which the query needs to be performed, and assign a worker thread for each node. The client then waits for each worker to finish. The result from each node is maintained in different temporary files. This is to ensure that each worker can run asynchronously, and also have less memory footprint in maintaining the results from different nodes. The final result is then built reading from these temporary files, which is ordered based on node identifiers.

## Log Query Server

The server component comprises of LogQueryServer and LogQueryServerThread classes. The server can accept connections from multiple clients. Each client request is served using a separate worker thread that is asynchronous with each other. Each worker thread runs a grep process. The results of grep are read and sent to client in fixed length buffers. The worker threads are run with minimal memory footprint to ensure better scalability.

# Unit Test Design

We write LogGenerator in Java and LogQueryUnitTest in Shell Script to unit test our query system. The coverage of our unit tests includes testing the queries of logs with the (frequent/somewhat frequent/rare) pattern that exists in (one/some/all) machines. In other words, it tests 9 combination of (LogPattern, MachineCoverage). For example, (Frequent, some machines) means that the test of frequent patterns that only exist in some machines. The LogQueryUnitTest will first generate the logs of size 5MB for each machine (in our case, 4 machines). Since the log directory is shared across different EWS machines, LogQueryUnitTest does not need to copy those files to the corresponding machines. Then LogQueryUnitTest will test queries with those 9 combinations by making request to the LogQueryServer using the LogQueryClient and also running the same ‘grep’ command by SSH remote execution. After it gets both the results, it compares whether these two results are the same. The generated log files are randomly generated with such distribution that all 9 combinations of (LogPattern, MachineCoverage) can be tested without regenerating the log file. For each log file, it contains 70% of INFO and WARNING logs, 25% of FINE and FINEST logs, and 5% of SEVERE logs, which represent frequent/somewhat frequent/rare patterns, respectively. So for testing all the LogPatterns in all machines, we only need to query logs for INFO or WARNING, FINE or FINEST, and SEVERE, respectively. For each line of the logs generated for machine i, we append the “MachineID: i” so that if we query for “MachineID: i”, only machine i will hit the query, others will not return anything. Similarly, we append the “SomeMachineIDs: i,j” such that i < j and only machine i or j contains these message. So if we query “SomeMachineIDs: i, j”, only machine i and j will hit the query, others will not return anything. This enables us to test for logs that only exist in some machines.

# Performance Evaluation

## Query Latency

We measure the query latency for 100MB query for 5 times. The measured time is 26.64s, 28.03s, 29.83s, 21.62s, and 31.21s respectively. The average query latency is 27.47s.

Query latency is determined by network latency and process execution time on different server nodes. The implementation allows all the nodes to run to completion. A node with high network latency and high process execution time will increase the overall Query latency of the distributed system.

## Performance over Different Log Sizes

Tests are run with different query patterns that occurs frequently, somewhat frequently and rarely on one/some/all nodes. The performance is also measured for different log file sizes in these cases. Figure 1 illustrates the response time for each query patterns with varying log file sizes. Logs with Frequent/Somewhat frequent/rare patterns take up 75%/20%/5% of the total log size, respectively.

From Figure 1, we have the following observations:

1. **The response time is almost proportional to the log size in each machine.** We think what affects the response time the most is the network latency and the disk I/O time because for larger log files, it takes longer to read the log file from the disk in LogQueryServer and to transmit the grepped result via the network. These two operations are expensive and highly related to the size of the log file. In addition, our LogQueryClient stores the result from each server into the temporary file in the client disk, which is also expensive. Since most of the execution time of the query is spent in operations highly related to the size of the log file, it is reasonable that the response time is also proportional to the log size.

**Figure 1: The Response Time for Each Query Patterns with Varying Log File Size**

1. **The small standard deviation time (marked as vertical bar around each data point) shows that our experiments are consistent and repeatable**. At first, we think in such a varying environment as EWS machines, the result should be very inconsistent. But it turns out that the result is quite consistent. I think it is because we run the experiment at night when not many users are running heavy tasks.
2. **The time for each server to grep the result is relatively small compared with the transmission time of network and the time the client spends to write buffer data to temporary file** because the time spent to grep log file with rare pattern and frequent pattern should be very similar since they all need to read the whole log file, but grepping logs with frequent patterns takes much more time than grepping logs with rare patterns. That means the time spent to grep log file is small compared with the total execution time. Furthermore, the grepping time for different nodes is done in parallel, so it takes shorter time than those done in serial. Therefore, most of the execution time are spent on expensive network latency and disk I/O.