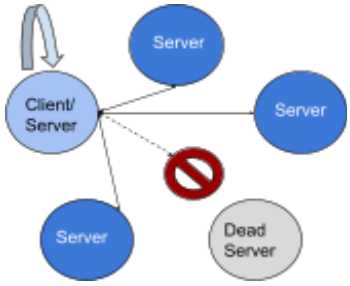
MP1 Report - CS425

Carson Sprague (cs104) & Rohan Kumar (rohandk2)

Design:

We implemented our distributed log querier using a simple broadcast/multicast protocol. A user logs into any of the ten machines, runs the client, and sends parallel requests to all of the servers. Each server handles the request locally and in parallel, then responds when finished, with the client logging the output to a file per server. This approach leverages server-side parallelism for all grep patterns without much overhead, and avoids unnecessarily large data transfers. Our system is also fault-tolerant. We used a fail-stop model, that is if a machine fails, we report the error, remove the connection, and ignore it in future queries.

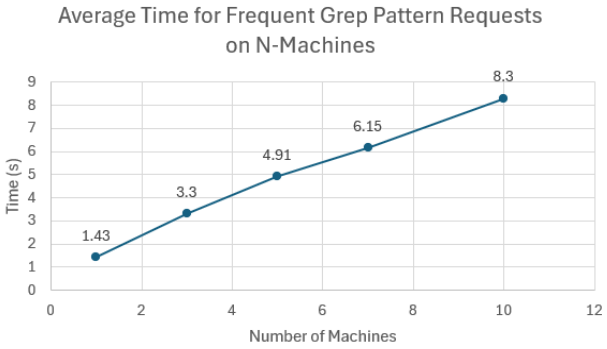


Unit Tests:

We have a total of four unit tests. The first emulates a client trying to connect to a network of dead servers, at which point we obviously expect to exit or return an error. The second tests for a successful client-server connection by starting a server and simulating a client connecting to that server. The third emulates a basic grep functionality on a distributed system. Again, a client-server connection is made, and then grep command is run on one of the given files and checked for output. The final test case runs grep on all 10 machines for multiple patterns and checks that the output matches our expected output, for common, uncommon, and unique patterns.

Performance:

We ran a performance test on how our system scales while handling a frequent grep pattern. While we did leverage parallel requests to the servers, our system has O(n) performance. This is likely due to the single client still being responsible for writing each individual output to a file and overall data-aggregation, which is a major bottleneck.



We also tested the average latency and standard deviation for varying frequencies on four machines. On low and medium frequency counts, we find that our average latency and standard deviation remain relatively similar, while on high frequency counts we find a very significant increase in average latency. We mostly attribute this to the overhead of the file I/O operations taking place on the client, as we did with the performance scaling analysis earlier.