CS425 MP1 REPORT

Yixuan Li, Netid: yixuan19; Tong Wei, Netid: twei11

September 16, 2024

1. Design Overview

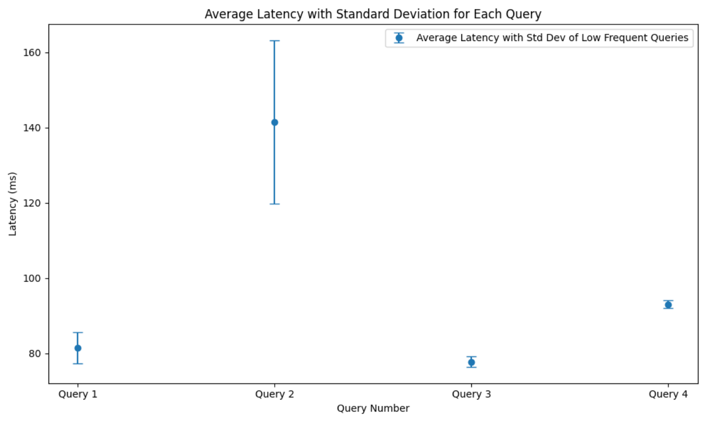
The system is implemented in Python using socket communication, following a client-server architecture. Each machine is both a server and a client, using asynchronous programming for parallel task handling.
Server Task: Continuously listens for incoming grep queries. Upon receiving a request, it searches its local log file and returns the result via socket.
Client Task: Listens for user input. When a grep query is issued, the machine sends it to all other machines in the hostlist (10 machines in total) and asynchronously processes the responses. If a machine is unresponsive, it reports the issue and closes the connection.
This asynchronous design allows all machines to handle incoming queries and send their own queries in parallel, improving efficiency.

2. Unit Tests

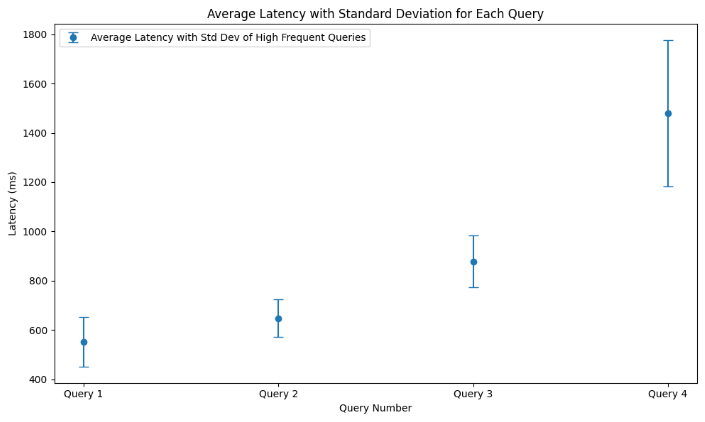
We wrote several unit tests running on logs with 60MB in size.:
Accuracy Test: We tested performance using 10 query patterns, each repeated 5 times. Queries were sent to a 4-server cluster (one is down to cope with edge cases, and results from each server were compared against local results where the test machine had access to the global log.
We also write some function unit test and using mock to simulate the server connection.
Client Task Testing: 1. Tested matching grep patterns: The function correctly identifies and sends back matching lines from the logs. 2. Tested non-matching grep patterns: The function returns an appropriate message indicating no matches were found. 3. Tested Invalid commands: The function handles invalid client requests without crashing or sending incorrect responses. 4. Tested successful log retrieval: Verified that logs were correctly fetched from active servers and combined with local results. 5. Tested timeout handling: Simulated server timeouts to ensure the system logged errors and continued querying other servers.

3. Query Latency Results

Two sets of experiments were conducted, each with 4 queries. In the first set, query frequencies in the logs were in single digits, while in the second set they were around 100,000. The average latency of the second set is noticeably higher due to the larger data transmission. The low standard deviation across both sets indicates stable system performance. Since our implementation performs local grep first before sending results, the delay caused by higher data frequencies is expected. As shown, complex queries result in higher latency, as expected. Error bars represent standard deviations, indicating that most latencies fall within an acceptable range.



(a) Latency Plot For Low Frequency Queries



(b) Latency Plot For High Frequency Queries