Marc Soda
HW2

- Summary:
  - Overall

    Communication is done via Tokio TcpStreams because I wanted to be able to host the server on a different network for reasons that will be explained later. I changed the established protocol because it contained unnecessary characters, which bothered me. Each request is sent with the header (GET/PUT/DEL), followed by the length of the header, followed by the key. There are no newlines or extra characters. The PUT request also has the val length followed by the val. The server stores the keys and values as u8 vectors to allow for maximum robustness.
  - Server

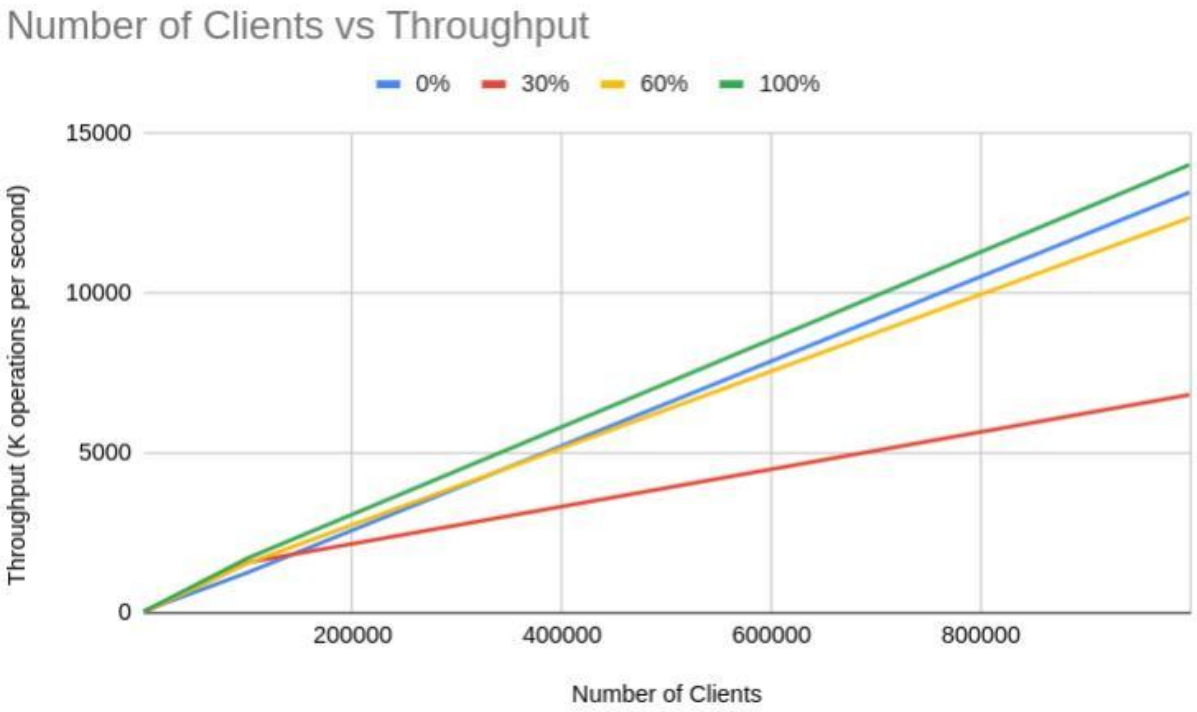
    The server is backed by a DashMap. I wanted to implement my own concurrent map, but I did not have the time. I have previously written a concurrent map in C++, so I did not feel it was urgent for me to implement this from an educational perspective. The main thread binds the TcpListener on the specified port. It then waits for incoming client connections. Each client gets its own thread for the duration of its life. As illustrated by my benchmarks, the server is highly concurrent.
  - Client

    The client simply connects to the server via a TcpStream. It exposes a neat API to the developer for GET/PUT/DEL operations. After sending a request, the client will wait for a response from the server. The client sends
- Testing

  The unit tests I made serve to test the general functionality of the system. Each one spawns a server and a client then tests an interaction between them. To be clear, these unit tests would not be used in production because they require a connection, but they are fine for this project.
- Benchmark Setup

  The primary purpose of the benchmark was to determine the effectiveness of the concurrency in the server. Rather than using Rust's benchmarking suite, I had the main function take the number of clients, the number of operations to be run on each client, the max key value, and the read-only ratio as parameters. The output of main is the throughput. In operations / second.
- Plot and Conclusions

  The following plot describes the number of clients vs throughput for different read only ratios. The max key value is fixed at 999999 and the number of operations per client is fixed at 1000. The server is hosted on a machine outside the client network. I chose to do this because if clients were running on independent threads as the server was trying to spawn threads to handle their requests, the results would have been inconsistent.



This plot shows two things. Firstly, throughput scales nicely as the number of clients increases. Secondly, throughput generally increases as the readonly ratio increases.