FastChat

Kadivar Lisan Kumar
Ameya Vikrama Singh
Krishna Narasimhan Agaram

IIT Bombay

November 2022

Table of Contents

Project Goals

The AIM of the project is to

- Build network of Clients interacting with each other using some **Servers as Mediators**
- Focus on obtaining high throughput and ensuring low latency of individual messages
- Providing End-to-End Encryption of messages and Password authentication along with features like Creating Groups and sending images

Introduction

Main `python` libraries used include:

- `socket`:- for networking and remote communication.
- `psycopg2`:- PostgreSql for Database Management.
- `select`:- for handling concurrent I/O in sockets.
- `ast` and `json`:- to convert objects to string and back.
- `cryptography`:- for secure end-to-end encryption of messages while sending, receiving and storing features.
- `pwnlib`:- for scripting and obtaining results

Project Architecture

The code relies on sockets - read, write from sockets - select.select to get which have something to receive and then receive it - msg protocol in msghandler.
The design is based mainly on the concept of sockets. Sockets connect clients to their servers. The exact connection setup is as follows.
- Each client is connected (after login)
-
- The file system is resonsible for creating db, starting the servers and load balancer remotely.

Message Protocol

Message Protocol is the agreement between sender and receiver which defines the format in which the messages will be sent and received across the connection. The following is an overview of the protocol that we used.

- Three block formatting - header, jsonheader and message_block
- header is a fixed size (in bytes) data which stores the size (in bytes) of the jsonheader
- Jsonheader is a json string which defines the size of the message_block along with other features of that meassage like type of encoding
- message_block is the actual message that we want to send
- Assuming this Message formatting, the receiver uses the particular number of bytes to get the whole message
- This ensures that the data sent through the connection is not lost
- To implement the message protocol, we use a class called MessageHandler which stores the connection object, receive buffer and send buffer to name some of the most important
- This class provides `write(input)` and `read()` methods which allows us to send the message and receive a message

Critique - Load Balancing

- We first tried out a round-robin strategy - The load balanceer chooses each server in turn. This proves to be good in some cases - for example, when clients rarely disconnect or they disconnect uniformly across the servers. In this case, the servers have nearly the same load, and so the system is balanced well.
- This does have problems, though - consider a scenario when we do a round-robin balancing but all clients connected to one particular server logout. Then essentially one less server is handling all the traffic, and the free server will only get its first client when its turn comes again in the round-robin.

Critique - Load Balancing

- Hence we tried another balancing strategy that addresses the concern above - while the free server has less clients than any other, we add new clients exclusively to it. This ensures (can prove) that **the difference between the number of clients connected to any two servers is atmost 1** under the (reasonable) assumption that logins happen at least as often as logouts.
- This performs much faster, and indeed is a good strategy. Hence, our main algorithms are 'round-robin' and 'least-load'. Which one to choose can be specified to the load-balancer.

Critique - Load Balancer

- **Further Thoughts:** There could be further optimizations to our number-of-clients minimization idea above, for example: Instead of number of clients, we could minimize the total number of bytes sent by each server.
- Our code is modular and can easily accomodate this different load balancer.