Implementing a Scalable Multithreaded Chat Room Service

Fernandez, Alan; Gray, Jackson; Hill, Colton

CPSC 4333 Group Project

Dr. Bo Sun, Lamar University

26 November 2023

Using Java, we implemented a chat service where a server can run multiple chat rooms concurrently. This was implemented at a small scale, but could provide insight as to how global communication systems operate at a much larger scale.

One of the modern miracles of internet communication is the instant messaging service. With a few taps and a couple clicks, people across the world can seamlessly communicate via text with anyone anywhere, as long as they have an internet connection. Communication services such as Telegram, Whatsapp, Messenger, WeChat, and many more are partially responsible for holding our global society together, so it is important to understand how to implement a chat system at scale. The aim of this project was to develop a chat server that could run several chat rooms concurrently. What we achieved is a basic implementation of how a large-scale chat service might operate.

To start off, we created an outline for what we wanted our project to accomplish. We made two distinct classes for the client and server. The client had three primary functions: it had to establish a connection to the server, send the server whatever information it received from the user, and display to the user whatever information it received from the server. The server, on the other hand, was much more intricate. The server had to handle all of the logistics of the chatroom service, such as clients joining and leaving, spinning off new chat rooms, sending and receiving information to and from the client, and more, all while allowing for concurrency and not bloating the system.

After some experimentation in C, we collectively agreed to write the programs in Java, as we were all most comfortable using the latter. While this choice potentially compromised some runtime efficiency, our familiarity with Java and the scope of the project made it the correct choice. We consciously elected not to use a GUI for simplicity. While this gave us the advantage of portability, it complicated matters on the client side in regard to sending and receiving messages concurrently.

The ChatroomClient class has only a main method and a nested ReadMessage class. The main method is simple: it establishes a connection with the server, launches a new thread for ReadMessage, prompts the user to enter a username, sends it to the server, and then enters a while loop. This loop waits for the user to enter text and sends it to the server. This loop only ends if the user inputs text that starts with **LEAVE** (written in all caps) whereupon it closes the connection with the server and gracefully shuts down. We discovered early on that a Scanner would block a single-threaded operation until it received text. This was deemed unacceptable for a program where receiving the latest possible information is of the utmost importance. Having already started work on the server, we realized that using threads would be the most elegant way to overcome this limitation. We elected to put reading and writing messages on separate threads client-side to eliminate blocking. That way, one thread can handle only user input while the other thread handles only server output, letting the user see the latest messages in real time.

The ChatroomServer class is decidedly more complicated, including a main method, a ClientHandler class, and a ChatRoom class. The main method opens the server on port 12345 and enters a while loop that waits for clients to connect and sends them to a ClientHandler on a new thread. The ClientHandler handles receiving messages from the client, including a few commands, namely **JOIN**, **LEAVE**, and **NAME**. If it receives an input in the format of **JOIN** *servername*, where servername can be replaced with any given string of characters, it will join the user to a chatroom of the given name. If a chatroom already exists with the given name, it adds the client to the existing room. If a room with that name doesn't exist, it will start a new thread with the ChatRoom class. If it receives an input starting with **LEAVE**, it will disconnect the client from the socket and delete the chatroom if no one is left. The last command is **NAME** *username*, where username can be replaced with any given string of characters. This allows for

the user to change their display name at any time. If a command is not specified, the ClientHandler will simply use the relevant ChatRoom to broadcast whatever text the client inputted. The ChatRoom class, aside from a run method which checks if the room is still in use, works simply as an object that stores relevant variables and contains methods that the ClientHandler invokes for functionality. These methods include addClient, which adds clients to the server; removeClient, which removes clients from the server; stopRoom, which closes the room; broadcastMessage, which broadcasts a given message to every client in the server; and isEmpty, which returns true if there are no clients connected.

The project succeeded in its goal of creating a functional chatroom application capable of small-scale operation. While we met the intended objectives of the project, there are still some hurdles that would need to be overcome for implementing this solution at scale. For instance, the use of individual threads for each ClientHandler and ChatRoom instance will strain the CPU resources of the server, demanding optimization for larger deployments. Additionally, incorporating encryption and more robust messaging protocols would be necessary for any competent chat service in broad use. Given the scope of the project, however, it is reasonable to neglect these concerns in favor of a program that functions as intended. Despite the barriers that would keep this particular implementation from operating at scale, the project demonstrates the foundational workings of a distributed chat system, providing valuable insights into the complexities and possibilities of a scalable distributed system.