Performance Comparison Analysis of gRPC and zeroMQ for Efficient Library Selection in Chat Service Development

Jincheol Jung○1, Sungwon Lee*1
1Department of Software Convergence, KyungHee University
bik1111@khu.ac.kr, drsungwon@khu.ac.kr*

Abstract

In the current chat service market, real-time chat services based on open chat formats centered on specific topics and interests are gaining popularity, with the number of open chat users steadily increasing. Consequently, domestic IT companies are prioritizing the expansion of open chat functionalities as a core business strategy to invigorate community engagement. However, since the performance of chat services heavily depends on the messaging system used in development, companies face challenges in selecting a library that satisfies both service performance optimization and cost efficiency. This paper compares and analyzes the performance of gRPC and ZeroMQ, which are applicable in distributed system environments, from the perspective of chat service implementation. Through this paper, we aim to contribute to the design and implementation of more efficient chat services, ultimately enhancing user satisfaction

Ⅰ. INTRODUCTION

In the contemporary chat service industry, the scope of online chat services is progressively expanding to include interest-based communities for sharing personal tastes, as well as one-on-one chat services like those found on fandom platforms. This evolution in chat services is significantly impacting the content industry, which is one of the most popular areas of interest [1][2]. For instance, Naver's 'OpenTalk' service saw approximately 2.78 million users during the Qatar World Cup period [3]. In response to the growing chat service market, IT companies like Naver and Kakao are increasingly considering the expansion of open chat functionalities as part of their strategies to enhance community engagement. The expansion of chat services requires complex system designs, highlighting the importance of efficient data processing and communication, particularly in distributed system environments. Despite the availability of various messaging systems, there is a lack of specific research elucidating their performance differences.

This paper addresses this gap by comparing and analyzing the performance of gRPC, an HTTP/2 protocol-based Remote Procedure Call (RPC) framework, and ZeroMQ, which operates over pure TCP and offers various messaging patterns.

Through this comparison, we aim to provide insights that aid in the design and implementation of more efficient chat services, ultimately contributing to increased user satisfaction.

Ⅱ. BACKGROUND

• zeroMQ

ZeroMQ is an asynchronous messaging library used in distributed or parallel application environments. Unlike traditional network communication methods that operate within the kernel, ZeroMQ operates at the application layer. It supports various socket transmission methods, including In-Process, Inter-Process, TCP, and Multicast. To implement these socket transmission methods, ZeroMQ supports various messaging patterns such as fan-out, pub-sub, and request-reply [4].

• gRPC

gRPC is a high-performance open-source RPC framework developed by Google. In a gRPC environment, client applications can directly call methods on a server application on another machine

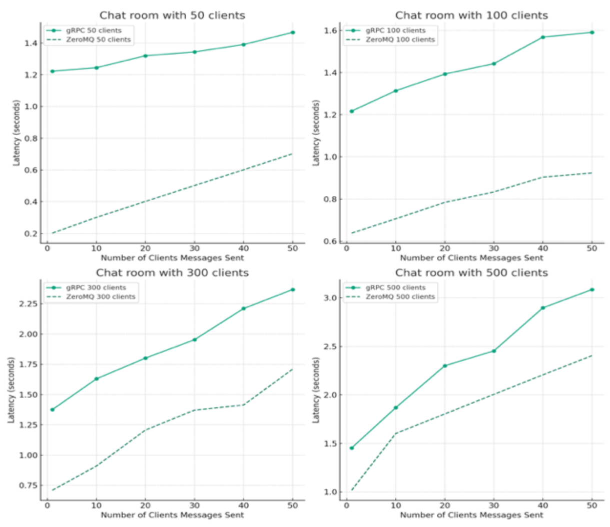
Figure 1. Latency with Increasing Number of Clients

as if they were local objects. gRPC offers four communication methods: Unary, Server Streaming, Client Streaming, and Bidirectional Streaming. For this study, Bidirectional Streaming was chosen to implement bidirectional communication.

Ⅲ. CHAT SYSTEM ARCHITECTURE

This paper combines zeroMQ's Push-Pull and Pub-Sub patterns in a zeroMQ chat system, as illustrated in Figure 2. The roles of sending and receiving messages between clients and the server are managed by the Push-Pull pattern. When a user sends a message (Push), the server receives it (Pull). The Pub-Sub pattern functions by having the server distribute the received message to its subscribers. Specifically, when the server receives a message from a user, it publishes the message to all users who are subscribed to it, thereby delivering the message to other users in the chat room.

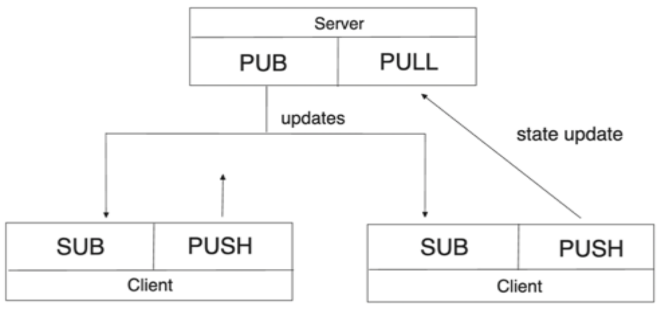

Figure 2. zeroMQ Chat System

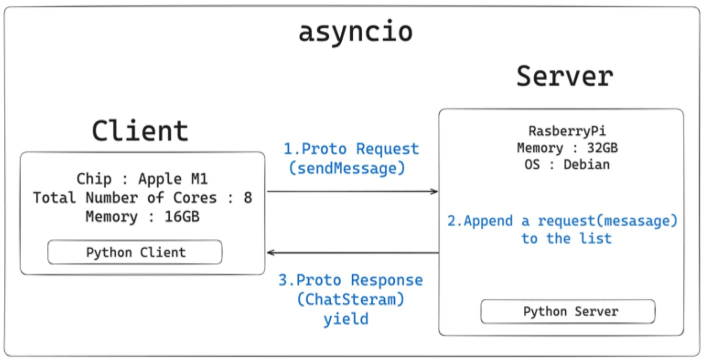

Figure 3. gRPC Chat System

The gRPC chat system is implemented as shown in Figure 3. Since gRPC does not natively support asynchronous communication, we used Python's asyncio module to implement this functionality. The messaging system operates as follows: first, a client uses the SendMessage function to send a message to the server. The server adds the received message to a list. Once a message is added to the message management list, the server uses the ChatStream function to broadcast the message to all clients except the sender. In the transmission process, the yield keyword is used. Unlike a simple return statement, which returns a value and terminates the function, yield allows the function to return a generator, enabling the function to be executed repeatedly. Due to the nature of chat services, where the server must continuously deliver received messages to clients, yield is used to facilitate the real-time transmission of messages.

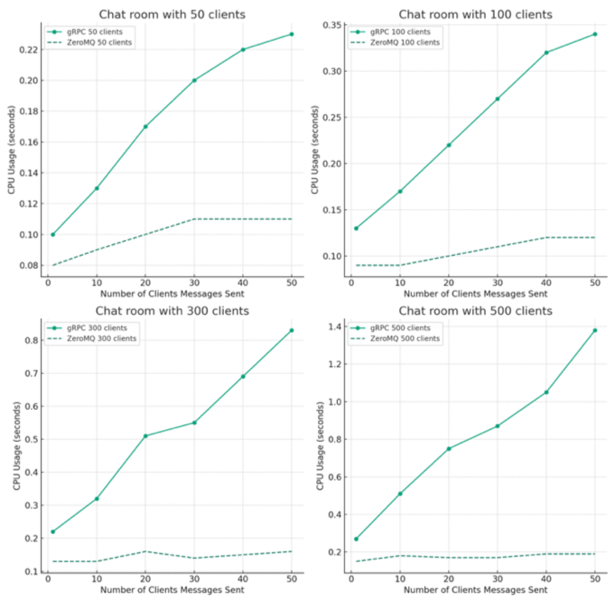
Figure 4. CPU Usage with Increasing Number of Clients

IV. EXPERIMENTAL ENVIRONMENT

In this study, the client operations were conducted in a Darwin OS environment based on ARM64 architecture, equipped with an Apple M1 chip (8 CORE) and 16GB of memory. The server operations were performed on Raspberry Pi hardware running Debian OS with a 32GB memory configuration. Python was used for implementation, and a multi-threaded environment was established to facilitate multi-user communication. Therefore, each client was mapped to a single thread, and the experiment was conducted in a manner where messages sent by a specific client were relayed to other clients through the server.

V. EXPERIMENTAL RESULTS

- Latency

The latency measurement method is as follows. For example, in a chat room with 100 clients, we measure the time it takes for 10 users to send messages simultaneously until all 10 messages are successfully received by the remaining 99 clients.

The number of clients was assumed to be 50, 100, 300, and 500, representing the total number of clients in the chat room. The number of clients sending messages in the chat room was gradually increased in the order of 1, 10, 20, 30, 40, and 50, and the time taken was measured accordingly.

In Figure 1, zeroMQ recorded average latencies of 0.4518, 0.7980, 1.220, and 1.8395 seconds (for 50, 100, 300, and 500 clients respectively), while gRPC recorded average latencies of 1.3309, 1.4209, 1.8889, and 2.3426 seconds. This indicates that zeroMQ is approximately 2.9 times, 1.75 times, 1.54 times, and 1.27 times faster than gRPC.

- CPU Usage

Figure 4 presents the results of CPU usage time measured during the latency analysis. zeroMQ recorded average CPU usage times of 0.1, 0.105, 0.145, and 0.175 seconds, whereas gRPC showed usage times of 0.175, 0.2417, 0.52, and 0.805 seconds. This indicates that zeroMQ's CPU utilization is up to 4.6 times, and at least 1.75 times more efficient compared to gRPC.

VI. CONCLUSION

In this paper, we built a chat service system using zeroMQ and gRPC, and conducted a performance analysis. The results showed that zeroMQ demonstrated up to approximately three times better performance in terms of latency. This can be attributed to zeroMQ operating directly over the pure TCP protocol, whereas gRPC involves additional overhead due to its use of features like header compression and flow control based on the HTTP/2 protocol. Additionally, the low latency characteristic of zeroMQ translated into more efficient CPU usage time.

For future research, we plan to build a multi-distributed system environment to expand the range of clients. Additionally, we aim to measure transmission time and computing resource usage based on message size and conduct an analysis by increasing the number of comparison groups for the messaging systems.

ACKNOWLEDGMENT

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the National Program for Excellence in SW)(2023-0-00042) supervised by the IITP(Institute of Information & Communications Technology Planning&Evaluation) in 2024.

REFERENCES

[1] 테크 M 트렌드, “서로 취향 공유하는 채팅 서비스 ‘인기’ 콘텐츠 깊이 더해준다 https://www.techm.kr/news/articleView.html?idxno=106416, 2023

[2] 서울 경제, “위버스도 ‘1대1 채팅 서비스, 달아오르는 팬덤 플랫폼 시장”, https://www.sedaily.com/NewsView/29OFG5G6XW, 2023

[3] 조선 일보, “테크 기업들, 오픈 채팅, 커뮤니티 확대하는 이유는” https://www.chosun.com/economy/economy_general/2023/03/02/FSUYOZZCP5EGRL23LODI6QAYMY, 2023

[4] zeroMQ, https://zeromq.org/

[5] gRPC, https://grpc.io/