ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 1

Đề tài: Distributed Chat System (gRPC-based)

Môn học: Chuyên đề Hệ thống phân tán

Sinh viên thực hiện:

Võ Hữu Tuấn - 22127439

Giáo viên hướng dẫn:

Thầy Trần Trung Dũng

Mục lục

1 Giới thiệu

Mục tiêu của đồ án là xây dựng một hệ thống chat thời gian thực phân tán sử dụng công nghệ **gRPC**. Hệ thống cho phép nhiều người dùng giao tiếp đồng thời, cho phé gửi tin nhắn cá nhân hoặc nhóm, lưu trữ lịch sử trò chuyện và duy trì kết nối ổn định giữa các client.

2 Phân Tích và Thiết Kế Hệ Thống

2.1 Tổng quan

- **Ngôn ngữ lập trình:** Python
- **Thư viện sử dụng:** gRPC, protobuf, threading, colorama
- **Định dạng lưu trữ dữ liệu:** JSON
- **Ghi lại các sự kiện bảo mật quan trọng (logs):** JSONL

3 Kiến trúc hệ thống

3.1 Tổng quan

Hệ thống gồm hai thành phần chính:

- **Server gRPC:** chịu trách nhiệm xử lý logic nghiệp vụ, điều phối kết nối và quản lý dữ liệu.
- **Client:** giao diện dòng lệnh (CLI) cho phép người dùng nhập lệnh, gửi và nhận tin nhắn thời gian thực.

Hình 1: Kiến trúc tổng quan hệ thống Client–Server gRPC

3.2 Luồng dữ liệu

1. Client gửi lệnh đến server qua phương thức `SendCommand()`.
2. Server xử lý, ghi log và phản hồi kết quả.
3. Các tin nhắn đến được truyền ngược lại cho client thông qua `StreamMessages()` (bidirectional streaming).

4 Giao tiếp gRPC

4.1 Kiến trúc giao tiếp gRPC

Hệ thống sử dụng mô hình **Client–Server hai chiều (Bidirectional Streaming)** với gRPC làm nền tảng truyền thông. Mỗi client duy trì hai kênh song song:

- **Unary RPC**: Dùng cho việc gửi lệnh điều khiển, cấu trúc hoặc truy vấn dữ liệu (ví dụ: tạo nhóm, thêm thành viên, xem danh sách người dùng, đăng xuất,...).
- **Streaming RPC**: Dùng để nhận tin nhắn và sự kiện thời gian thực từ server.

Cả hai kênh đều hoạt động độc lập, giúp hệ thống có thể vừa nhận vừa gửi mà không bị chặn luồng.

4.2 Định nghĩa service

File `chat.proto` định nghĩa hai RPC chính:

```
service ChatService {
    rpc SendCommand (CommandRequest) returns (CommandResponse);
    rpc StreamMessages (ConnectRequest) returns (stream ChatMessage);
}
```

Các phương thức được sinh tự động bằng công cụ `protoc`, tạo ra hai module Python: `chat_pb2.py` (chứa message classes) và `chat_pb2_grpc.py` (chứa stub và server class).

4.3 Mô tả các message

- **ConnectRequest**: Gửi từ client đến server khi người dùng đăng nhập.
- **ChatMessage**: Đại diện cho một tin nhắn (riêng, nhóm, hoặc hệ thống).
- **CommandRequest / CommandResponse**: Gửi lệnh từ client và nhận kết quả từ server.

```
message ChatMessage {
  string type = 1;        // "private", "group", hoặc "system"
  string sender = 2;
  string to = 3;           // người nhận (nếu là private)
  string group = 4;        // nhóm (nếu là group)
  string msg = 5;
  string timestamp = 6;
}
```

4.4 Luồng hoạt động

1. Client khởi tạo kết nối và gửi `ConnectRequest(username)`.
2. Server phản hồi bằng một luồng `ChatMessage` liên tục để cập nhật tin mới.
3. Khi người dùng nhập lệnh (như `msg user ...` hoặc `create group ...`), client gọi RPC `SendCommand()` để gửi yêu cầu.

4. Server xử lý, ghi log và phát lại tin nhắn cho các client liên quan thông qua kênh streaming.

4.5 Ưu điểm của gRPC trong hệ thống

- **Hiệu năng cao:** Dữ liệu được tuần tự hóa bằng Protocol Buffers giúp giảm kích thước gói tin.
- **Truyền song song (HTTP/2):** Cho phép nhiều luồng RPC chạy đồng thời trên cùng một kết nối TCP.
- **Tự sinh mã nguồn:** Không cần định nghĩa lại hàm hoặc kiểu dữ liệu ở hai phía client–server.
- **Đa nền tảng:** gRPC hỗ trợ hơn 10 ngôn ngữ khác nhau, dễ mở rộng sang Java, C++, hoặc Go.

4.6 Xử lý lỗi và duy trì kết nối

Hệ thống client sử dụng cấu trúc:

```
try:
    for msg in stub.StreamMessages(...):
        ...
except grpc.RpcError:
    pass
```

Nhờ đó, khi server bị ngắt hoặc client mất mạng, chương trình vẫn hoạt động ổn định và không hiển thị lỗi cảnh báo ra terminal (chỉ ghi vào file log riêng).

5 Cơ chế lưu trữ và ghi log

5.1 Tổng quan

Hệ thống chat gRPC được thiết kế với hai loại log riêng biệt:

1. **Chat log**: ghi lại toàn bộ tin nhắn (riêng, nhóm, hệ thống) phục vụ cho việc truy xuất lịch sử, hiển thị hộp thư đến (và hộp thư đã gửi) và thống kê hoạt động.

2. **Server log**: ghi lại các sự kiện hoạt động của máy chủ như tạo nhóm, thêm thành viên, người dùng đăng nhập, lỗi RPC, v.v.

Tất cả log được lưu dưới dạng **dòng JSON độc lập (JSONL)** để dễ dàng truy xuất, đọc, phân tích và xử lý tự động. Mỗi dòng tương ứng với một sự kiện, có cấu trúc và timestamp rõ ràng.

5.2 Cấu trúc thư mục dữ liệu

Các tệp log và dữ liệu người dùng được lưu trong hai thư mục riêng biệt:

```
data/
   users.json
   groups.json
log/
    chatlog.jsonl
    serverlog.jsonl
```

5.3 Cấu trúc log chi tiết

1. Chat log (`chatlog.jsonl`)

Mỗi dòng là một đối tượng JSON biểu diễn một tin nhắn hoặc sự kiện liên quan đến truyền thông giữa các người dùng:

```
{
  "timestamp": "2025-10-31 19:53:26",
  "type": "group",
  "group": "Alpha",
  "from": "user01",
  "msg": "hello"
}
```

Các trường chính:

- `timestamp`: thời điểm gửi tin (định dạng `YYYY-MM-DD HH:MM:SS`).

- `type`: loại tin (`"private"`, `"group"`, hoặc `"system"`).
- `from`: tên người gửi.
- `to`: (chỉ có khi là tin nhắn riêng).
- `group`: tên nhóm (chỉ có khi là tin nhắn nhóm).
- `msg`: nội dung tin nhắn.

Chức năng:

- Dùng để hiển thị lịch sử hội thoại (`history user/group`).
- Dùng để thống kê số lượng tin nhắn gửi/nhận.
- Dễ dàng parse bằng Python để lọc hoặc phân tích (vì mỗi dòng là JSON độc lập).

2. Server log (`serverlog.jsonl`)

Mỗi dòng phản ánh một sự kiện hoạt động nội bộ của máy chủ:

```
{
  "timestamp": "2025-10-31 20:00:22",
  "event": "add_member",
  "group": "alpha",
  "user": "user02",
  "by": "user01"
}
```

Các sự kiện phổ biến:

- `"create"` – khi tạo nhóm mới.
- `"add_member"` – khi thêm thành viên.
- `"remove_member"` – khi xoá thành viên.
- `"delete"` – khi nhóm bị xoá.
- `"error"` – khi có lỗi RPC hoặc dữ liệu không hợp lệ.

5.4 Cơ chế ghi log trong mã nguồn

Cả hai loại log đều được ghi thông qua các hàm trong module `storage.py`:

- `append_chat_log(entry)`
  Nhận vào một từ điển (dictionary) chứa các trường như `type`, `from`, `to`, `msg`, `group`... Hàm này tự động thêm `timestamp` nếu chưa có, sau đó ghi nối tiếp vào file `chatlog.jsonl`.
- `append_server_log(entry)`
  Tương tự, dùng cho các sự kiện nội bộ của server. Cấu trúc tương thích với JSONL và được lưu trong `serverlog.jsonl`.

5.5 Cơ chế truy xuất log

- Lệnh `inbox [n]` đọc từ `chatlog.jsonl` và lọc ra các tin nhắn có `to = username` hoặc nhóm mà user là thành viên.
- Lệnh `sent [n]` lọc theo `from = username`.
- Lệnh `history user/group` lọc theo bộ tiêu chí kết hợp (gửi và nhận giữa 2 người hoặc 1 nhóm).

Các hàm xử lý này đảm bảo:

- Chỉ người dùng liên quan mới xem được nội dung.
- Log đọc theo chiều thời gian tăng dần.
- Giới hạn `n` dòng gần nhất để tránh quá tải bộ nhớ.

5.6 Lợi ích của định dạng JSONL

- Dễ dàng **append** mà không cần load toàn bộ file.
- Dễ phân tích, lọc dữ liệu bằng các công cụ như `jq`, `grep`, hoặc `pandas`.
- Có thể chuyển đổi trực tiếp sang định dạng CSV hoặc Database khi mở rộng quy mô hệ thống.

6 Các lệnh trong chương trình

Phần này trình bày toàn bộ các lệnh có thể sử dụng trên giao diện dòng lệnh (CLI) của hệ thống chat gRPC. Mỗi lệnh được thiết kế để tương tác với máy chủ thông qua RPC call, đảm bảo truyền thông tin theo thời gian thực và cập nhật đồng bộ giữa các client.

6.1 Lệnh nhắn tin

- **msg user <username> <message> /end/**
  Gửi tin nhắn riêng (private message) đến người dùng cụ thể. Người gửi không thể gửi tin cho chính mình. Ví dụ:

```
>> msg user user02 Xin chào bạn /end/
Sent to user02.
```

- **msg group <group> <message> /end/**
  Gửi tin nhắn đến tất cả thành viên của một nhóm. Người gửi phải là thành viên trong nhóm. Ví dụ:

```
>> msg group alpha Chào mọi người /end/
Sent to group alpha.
```

6.2 Lệnh quản lý nhóm

- **create group <name>**
  Tạo nhóm mới, người tạo mặc định là quản trị viên (admin) của nhóm đó.

```
>> create group alpha
Group 'alpha' created.
```

- **add member <group> <user>**
  Thêm người dùng vào nhóm (mọi thành viên đều có thể thêm người khác).

```
>> add member alpha user02
Added user02 to alpha.
```

- **remove member <group> <user>**

  Loại thành viên khỏi nhóm. Lệnh này chỉ được thực hiện bởi admin.

```
>> remove member alpha user03
Removed user03 from alpha.
```

- **leave group <group>**

  Rời khỏi nhóm. - Nếu người dùng là admin và nhóm vẫn còn thành viên khác: hệ thống từ chối yêu cầu. - Nếu nhóm chỉ còn admin: thao tác này đồng thời xoá nhóm.

- **delete group <group>**

  Xoá nhóm (chỉ admin có quyền).

```
>> delete group beta
Group 'beta' deleted.
```

6.3 Lệnh truy xuất thông tin

- **list users**

  Hiển thị danh sách người dùng đang trực tuyến.

```
Online users:
- user01
- user02
```

- **list groups**

  Hiển thị các nhóm mà người dùng hiện tại đã tham gia.

```
- alpha <admin: you>
- beta <member>
```

- **history user <name> [n]**

  Hiển thị lịch sử tin nhắn giữa người dùng hiện tại và `name`, giới hạn `n` dòng gần nhất (mặc định 10).

- **history group <name> [n]**

  Hiển thị lịch sử trò chuyện của nhóm. Chỉ thành viên trong nhóm mới được phép xem.

- **inbox [n]**

  Hiển thị `n` tin nhắn đến gần nhất (chỉ nhận, không bao gồm tin đã gửi).

- **sent [n]**

  Hiển thị `n` tin nhắn đã gửi gần nhất.

6.4 Lệnh hệ thống

- **logout**

  Ngắt kết nối với máy chủ và thoát khỏi hệ thống chat.

```
>> logout
Logged out successfully.
```

6.5 Quy tắc hiển thị

- Mọi dòng tin nhắn đều bắt đầu bằng **timestamp** dạng `YYYY-MM-DD HH:MM:SS`.
- Lịch sử `inbox`, `sent` và `history` đều đọc từ file log định dạng JSONL:

```
{"timestamp": "2025-10-31 20:53:26",
 "type": "group",
 "group": "alpha",
```

```
"from": "user01",
"msg": "hello"}
```

- Các thông báo lỗi, hệ thống, và tin nhóm được đồng bộ thời gian thực giữa các client.

6.6 Một số quy tắc quyền hạn về nhóm chat

- Bất kỳ người dùng nào đều có thể tạo nhóm mới.
- Bất kỳ thành viên nhóm đều có thể thêm thành viên mới vào nhóm.
- Quản trị viên mới có quyền xóa thành viên ra khỏi nhóm.
- Quản trị viên không thể rời nhóm khi còn thành viên khác.
- Quản trị viên mới có quyền xoá nhóm.
- Nếu quản trị viên là người cuối cùng rời nhóm thì đồng nghĩa với việc xóa nhóm đó.

7 Hướng dẫn cài đặt và sử dụng

Xem hướng dẫn cài đặt chi tiết tại README.md

8 Kết Luận

Đồ án đã đáp ứng tất cả các yêu cầu đặt ra và mô phỏng gần như đầy đủ các khía cạnh bảo mật thực tế: từ định danh, mã hóa, xác thực, đến phân quyền quản trị. Hệ thống lưu trữ thông tin dữ liệu bằng JSON giúp đơn giản hóa và dễ triển khai.

Phụ Lục

A. Liên kết mã nguồn

- Link GitHub Repository

B. Video demo

- Link Youtube

Tài liệu Tham Khảo

1. gRPC Guides
2. Python gRPC Tutorial - Create a gRPC Client and Server in Python with Various Types of gRPC Calls - MissCoding