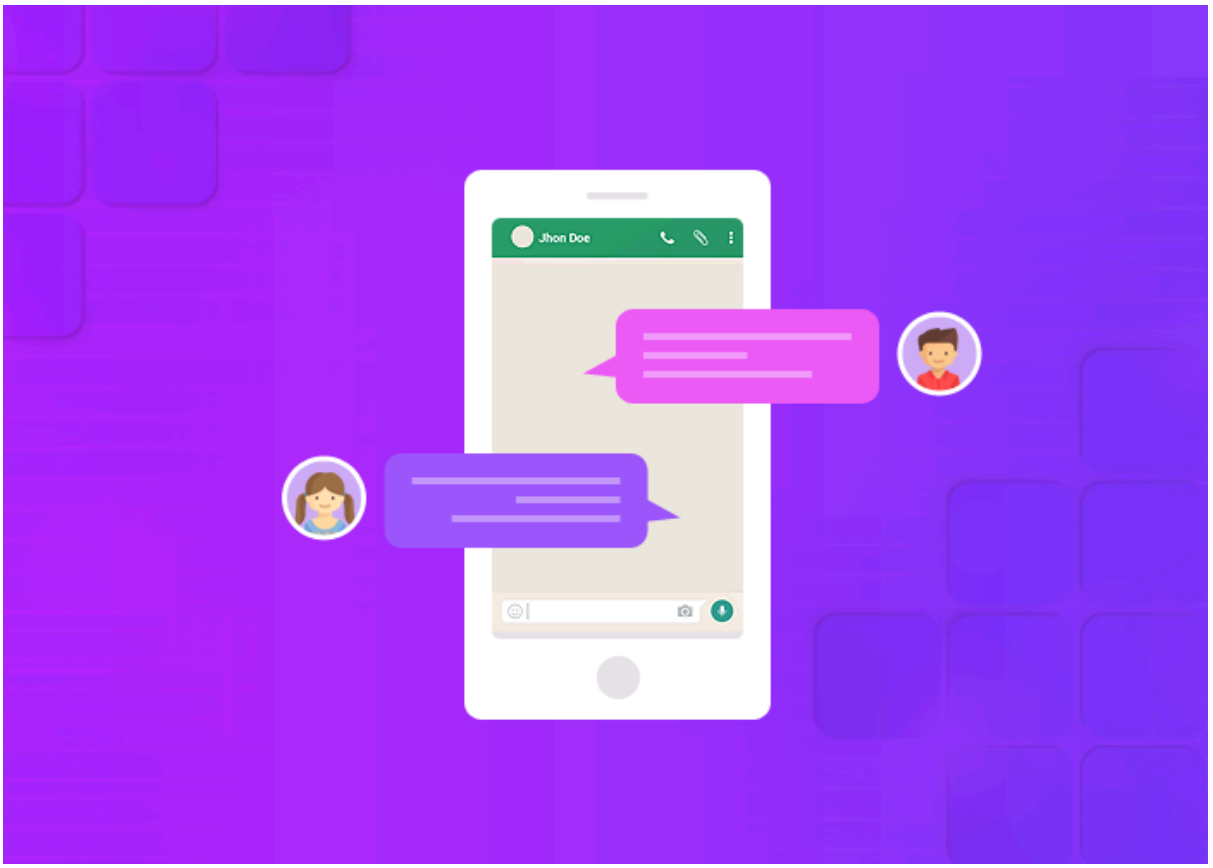
Pràctica 3 – RPC-based Chat amb gRPC



Assignatura: Arquitectura Distribuïda
Grau: Enginyeria Informàtica
Autors: Aniol Vergés Herrera, Aleix Batchelli
Curs acadèmic: 2025–2026

Índex

1. Introducció

Aquesta pràctica se centra en la implementació d'un sistema distribuït basat en Remote Procedure Calls (RPC) utilitzant gRPC i Protocol Buffers. L'objectiu principal és entendre com es pot estructurar una aplicació client-servidor on la comunicació es realitza mitjançant crides remotes, abstraient els detalls de xarxa i serialització de dades.

El sistema desenvolupat és una aplicació de xat distribuït, on diversos clients poden enviar i consultar missatges a través d'un servidor central. Aquesta pràctica permet analitzar aspectes clau dels sistemes distribuïts com ara la concurrència, l'exclusió mútua, la consistència de dades i el comportament del sistema quan el nombre de clients augmenta.

2. Objectius de la pràctica

Els principals objectius d'aquesta pràctica són:

- Implementar un servei RPC funcional utilitzant gRPC.
- Definir interfícies de comunicació mitjançant Protocol Buffers.
- Desenvolupar un servidor capaç de gestionar múltiples clients concurrents.
- Analitzar i justificar els mecanismes d'exclusió mútua i consistència de dades.
- Avaluar el rendiment i les limitacions del sistema.

3. Enunciat i requisits del sistema

El sistema es basa en una arquitectura client-servidor amb les següents restriccions:

- Tant el client com el servidor estan implementats en un llenguatge compatible amb gRPC.
- El servidor exposa dues crides RPC principals:
    - sendMessage: permet inserir un nou missatge al xat.
    - getMessages: retorna els missatges emmagatzemats al servidor.
- El servidor manté un fitxer local amb l'historial del xat.
- Els clients no poden mantenir dades persistents i han de consultar periòdicament el servidor.
- Cada client s'executa indicant el servidor i un sobrenom (nickname).

4. Arquitectura general del sistema

El sistema implementat segueix una arquitectura client-servidor centralitzada utilitzant gRPC com a mecanisme de comunicació RPC. El servidor exposa un servei anomenat ChatService, mentre que múltiples clients s'hi connecten de manera concurrent.

Components principals

- Servidor gRPC (server.py): gestiona les peticions RPC, manté l'estat persistent del xat en un fitxer local (messages.txt) i garanteix la concurrència segura.
- Clients (client.py): aplicacions de terminal (TUI) que permeten a l'usuari enviar i rebre missatges. Els clients no mantenen cap estat persistent.
- Protocol Buffers (chat.proto): defineix les estructures de dades i els mètodes RPC (SendMessage i GetMessages).

El servidor escolta al port 50051 i utilitza un ThreadPoolExecutor amb fins a 10 fils per atendre múltiples clients simultàniament. A la Figura 1 es mostra el diagrama d'arquitectura del sistema, on es pot observar la comunicació RPC entre clients i servidor, així com el mecanisme d'exclusió mútua en l'accés al fitxer persistent.

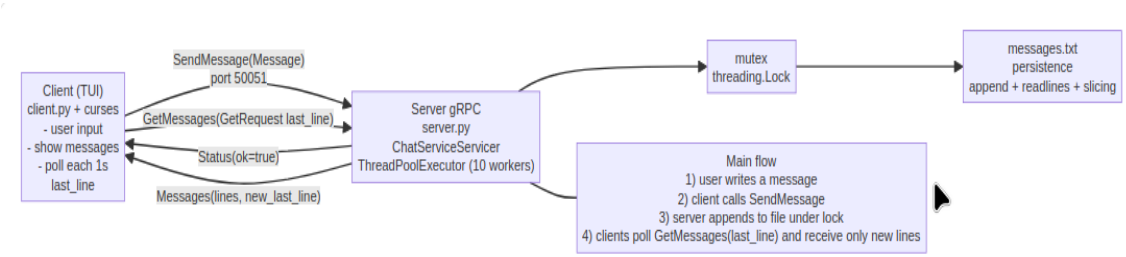


Figura 1: Arquitectura general del sistema RPC-based Chat.

Durant la fase d'ideació es van considerar diverses alternatives abans de fixar l'arquitectura final. En primer lloc, es va valorar mantenir l'historial del xat en memòria al servidor (sense persistència a disc), però es va descartar perquè qualsevol reinici del servidor implicaria perdre tots els missatges. Per aquest motiu, es va optar per persistir els missatges en un fitxer (messages.txt), assegurant la durabilitat de la informació.

També es va considerar la possibilitat que els clients mantinguessin una còpia local dels missatges per reduir consultes al servidor, però l'enunciat ho restringeix explícitament, i a més podria introduir inconsistències si es produïssin desconnexions o desfasaments entre clients. Per això, el client només manté un estat mínim no persistent (last_line) per demanar al servidor únicament les línies noves.

5. Disseny del servei RPC

El servei RPC està definit al fitxer chat.proto. Aquest defineix les estructures de dades i les operacions remotes següents:

- **Message**: conté el sobrenom de l'usuari (nickname) i el text del missatge.
- **Status**: retorna un booleà (ok) per indicar si l'operació s'ha completat correctament.
- **GetRequest**: inclou el camp last_line, que indica fins a quina línia ha llegit el client.
- **Messages**: conté una llista de noves línies i el nou índex new_last_line.

Els mètodes RPC implementats són:

- SendMessage(Message) -> Status: el client envia un missatge al servidor, que l'afegeix al fitxer.
- GetMessages(GetRequest) -> Messages: el client sol·licita només els missatges nous des de l'última línia llegida.

Aquest disseny evita reenviar tot l'historial del xat a cada petició i millora l'eficiència de la comunicació. A la Figura 2 es mostra el diagrama de seqüència que descriu el funcionament del servei RPC implementat. El diagrama reflecteix la interacció entre l'usuari, el client, el servidor gRPC i el fitxer de persistència durant l'ús del sistema de xat.

En primer lloc, quan l'usuari escriu un missatge al client, aquest invoca la crida remota SendMessage, enviant al servidor el sobrenom de l'usuari i el contingut del missatge. El servidor gestiona aquesta petició adquirint un mecanisme d'exclusió mútua (mutex) abans d'escriure el missatge al fitxer messages.txt, garantint així un accés segur al recurs compartit. Un cop completada l'operació, el servidor retorna una resposta d'estat al client indicant que l'operació s'ha realitzat correctament.

D'altra banda, el diagrama també mostra el mecanisme de recepció de missatges mitjançant polling. Cada client realitza periòdicament la crida GetMessages, indicant l'última línia llegida. El servidor retorna únicament els missatges nous, juntament amb el nou índex, permetent al client actualitzar el seu estat i mostrar els missatges sense duplicacions.

Aquest diagrama permet entendre de manera visual el flux complet de comunicació del sistema, així com el paper del servidor com a punt central de coordinació i garant de la consistència de dades.

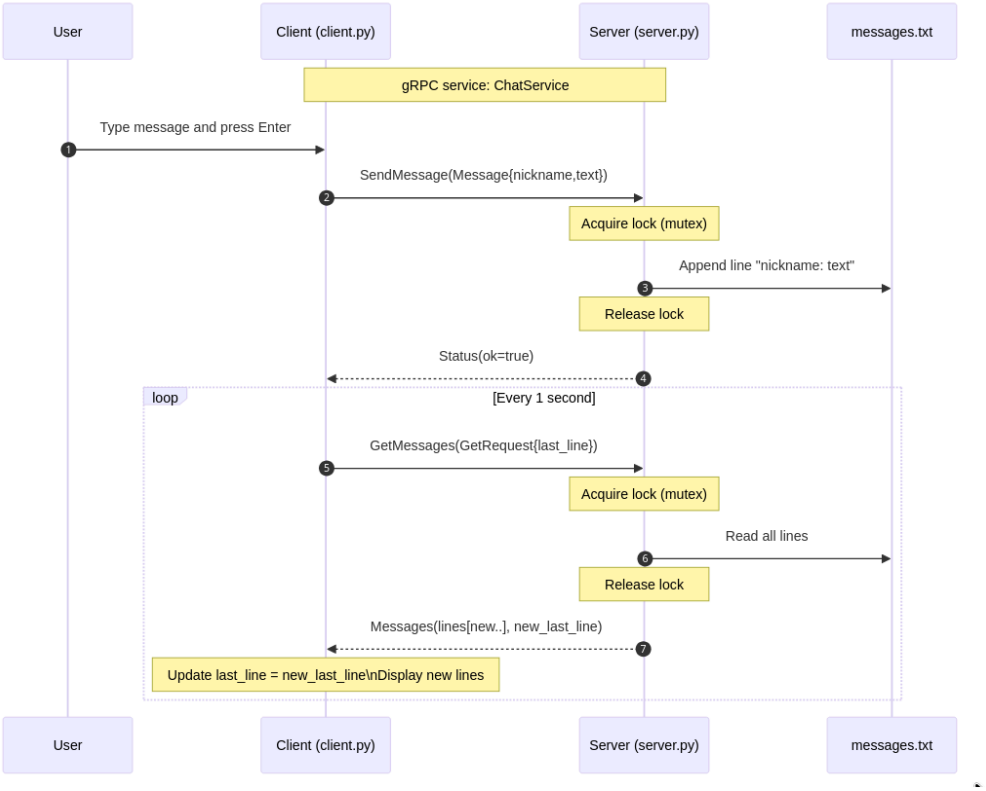


Figura 2: Diagrama de seqüència del servei RPC del sistema de xat, mostrant el flux de comunicació entre client i servidor mitjançant les crides SendMessage i GetMessages, així com l'accés sincronitzat al fitxer de persistència.

6. Exclusió mútua i concurrència

El servidor pot gestionar múltiples clients de manera concurrent gràcies al ThreadPoolExecutor de gRPC. Tot i això, tots els fils comparteixen un recurs crític: el fitxer messages.txt.

Per garantir l'exclusió mútua, s'utilitza un mutex global (threading.Lock) que protegeix tant les operacions d'escriptura com de lectura del fitxer:

- En SendMessage, el lock garanteix que només un fil escriu al fitxer alhora.
- En GetMessages, el mateix lock assegura que la lectura no coincideixi amb una escriptura.

Aquesta solució evita condicions de cursa i corrupció de dades, mantenint l'ordre correcte dels missatges.

7. Consistència de dades

El sistema garanteix una consistència forta, ja que:

- El servidor és l'única font de veritat.
- Tots els missatges s'escriuen seqüencialment en un únic fitxer.
- Els clients sempre consulten l'estat actual mitjançant RPC.

L'ús del paràmetre last_line permet a cada client recuperar únicament els missatges nous, assegurant que no hi ha pèrdua ni duplicació d'informació.

8. Rendiment i escalabilitat

El rendiment del sistema es veu afectat directament pel nombre de clients:

- A mesura que augmenten els clients, augmenta el nombre de crides RPC.
- El servidor pot esdevenir un coll d'ampolla, especialment en operacions d'I/O sobre el fitxer.

Per tant, el sistema presenta un rendiment adequat per a un nombre moderat de clients concurrents, complint correctament els requisits funcionals de la pràctica. No obstant això, en escenaris amb una càrrega elevada, el servidor central pot convertir-se en un coll d'ampolla, especialment a causa de les operacions d'entrada/sortida sobre el fitxer i del mecanisme de polling periòdic dels clients.

En conseqüència, aquesta solució no està pensada per escalar horitzontalment sense modificacions estructurals. Per millorar l'escalabilitat seria necessari introduir canvis com l'ús de mecanismes de comunicació basats en streaming, sistemes d'emmagatzematge més eficients o arquitectures distribuïdes amb coordinació entre servidors. Tot i això, aquestes millores s'han considerat fora de l'abast de la pràctica actual.

9. Limitacions i possibles millores

Algunes limitacions identificades són:

- Punt únic de fallada al servidor.
- Absència de mecanismes de tolerància a fallades.
- Polling constant dels clients, que pot generar càrrega innecessària.

Com a millores futures es podrien considerar:

- Implementar streaming gRPC per evitar polling.
- Afegir rèpliques del servidor amb protocols de consens.
- Introduir persistència més eficient mitjançant una base de dades.

10. Testing i validació

El sistema s'ha validat mitjançant proves funcionals i de concurrència, orientades a verificar tant el correcte funcionament de les crides RPC com la gestió de l'accés concurrent als recursos compartits.

En primer lloc, s'han executat múltiples instàncies del client de manera simultània, comprovant que tots els clients poden enviar i rebre missatges correctament. S'ha verificat que els missatges apareixen en el mateix ordre per a tots els clients, independentment del moment en què s'hi connecten, garantint així la consistència del sistema.

En segon lloc, s'han realitzat proves de concurrència enviant missatges de forma quasi simultània des de diferents clients. Aquestes proves permeten validar el mecanisme d'exclusió mútua implementat mitjançant un threading.Lock, assegurant que no es produeixen condicions de cursa ni corrupció del fitxer messages.txt.

Finalment, s'han simulat desconnexions de clients durant l'execució per comprovar que el servidor continua funcionant correctament i que la resta de clients no es veuen afectats. No s'han utilitzat eines automàtiques de testing, ja que l'objectiu principal era validar el comportament funcional i concurrent del sistema en escenaris d'ús realista.

11. Conclusions

En aquesta pràctica s'ha implementat un sistema distribuït senzill basat en RPC utilitzant gRPC i Protocol Buffers, seguint una arquitectura client-servidor. El desenvolupament d'una aplicació de xat ha permès entendre de manera pràctica com funcionen les crides remotes i com es pot estructurar la comunicació entre processos distribuïts.

Al llarg de la pràctica s'han treballat conceptes importants com la concurrència, l'exclusió mútua i la consistència de dades. L'ús d'un servidor centralitzat i d'un mecanisme de sincronització mitjançant un mutex ha permès garantir que l'accés al fitxer compartit es realitza de forma segura, fins i tot quan diversos clients interactuen simultàniament amb el sistema.

També s'han analitzat les limitacions de la solució implementada, especialment pel que fa al rendiment i l'escalabilitat. Tot i que el sistema funciona correctament amb un nombre moderat de clients, s'ha vist que pot aparèixer un coll d'ampolla al servidor quan augmenta la càrrega, principalment a causa del polling periòdic i de les operacions d'entrada/sortida sobre el fitxer.

En conclusió, la pràctica ha complert els objectius plantejats i ha servit per consolidar els coneixements teòrics de l'assignatura mitjançant una implementació real. El sistema desenvolupat constitueix una bona base per entendre els principals reptes dels sistemes distribuïts i com aquests es poden abordar en projectes de major complexitat.