BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

-----□□□□□-----

BÁO CÁO

MÔN: CÁC HỆ THỐNG PHÂN TÁN

ĐỀ TÀI:

XÂY DỰNG HỆ THỐNG BIÊN TẬP VĂN BẢN CỘNG TÁC THỜI GIAN THỰC

Giảng viên: TS. Kim Ngọc Bách

Lớp: M25CQHT01-B

Học viên thực hiện: Nguyễn Duy Khương - B25CHHT032

Hồ Viết Sơn Tùng - B25CHHT066

Phạm Công Trường - B25CHHT058

Hà Nội - Năm 2025

LỜI NÓI ĐẦU

Xây dựng một ứng dụng Web ngày nay không còn dừng lại ở việc hiển thị thông tin tĩnh. Thách thức lớn nhất đối với các nhà phát triển hiện đại là tạo ra các ứng dụng giàu tính tương tác, nơi người dùng có thể giao tiếp và thay đổi dữ liệu đồng thời với độ trễ thấp nhất. Hệ thống biên tập văn bản cộng tác thời gian thực là ví dụ điển hình nhất cho bài toán khó về tính nhất quán dữ liệu (Data Consistency) trong hệ phân tán.

Đề tài **"Xây dựng hệ thống biên tập văn bản cộng tác thời gian thực"** được thực hiện nhằm mục đích nghiên cứu và áp dụng các thuật toán giải quyết xung đột dữ liệu (Dựa trên mô hình Client-Server & Kiến trúc hướng dữ liệu) cùng các công nghệ truyền tải dữ liệu thời gian thực (như WebSocket).

Thông qua bài tập chúng em mong muốn làm rõ cơ chế "phía sau màn hình" của các ứng dụng như Google Docs, đồng thời xây dựng một mô hình thực nghiệm có khả năng chịu tải và đảm bảo tính toàn vẹn của văn bản khi có nhiều luồng chỉnh sửa diễn ra song song.

Chúng em xin gửi lời cảm ơn chân thành đến **TS Kim Ngọc Bách** đã tận tình chỉ bảo và định hướng kỹ thuật giúp chúng em hoàn thành đề tài này.

MỤC LỤC

CHƯƠNG I: ĐẶT VẤN ĐỀ VÀ CƠ SỞ LÝ THUYẾT

1.1. Thách thức trong Hệ thống Phân tán

Sự chuyển dịch mô hình làm việc từ văn phòng truyền thống sang không gian số hóa phi tập trung đã đặt ra những yêu cầu chưa từng có đối với hạ tầng công nghệ phần mềm. Nhu cầu về khả năng cộng tác thời gian thực (real-time collaboration), nơi nhiều người dùng có thể đồng thời chỉnh sửa một tài liệu, bảng tính hoặc bản thiết kế đồ họa với độ trễ tối thiểu, đã trở thành tiêu chuẩn mặc định cho các ứng dụng năng suất hiện đại như Google Docs, Figma, hay Overleaf. Tuy nhiên, sự đơn giản và mượt mà trên giao diện người dùng lại che giấu một hệ thống kỹ thuật cực kỳ phức tạp "phía sau màn hình", nơi các kỹ sư phải giải quyết những bài toán kinh điển của khoa học máy tính về đồng thuận phân tán, xử lý xung đột và tối ưu hóa độ trễ.

Báo cáo này được xây dựng nhằm mục đích phân tích toàn diện cơ chế hoạt động của các hệ thống soạn thảo cộng tác, tiếp cận vấn đề từ nền tảng của mô hình Client-Server và Kiến trúc hướng dữ liệu (Data-Centric Architecture). Thay vì đi sâu vào các thuật toán vi mô, chúng tôi tập trung phân tích cách thức tổ chức và quản lý luồng dữ liệu giữa các máy trạm (Clients) và máy chủ trung tâm (Server). Thông qua việc đánh giá các mô hình thực nghiệm và kịch bản chịu tải cao, báo cáo sẽ làm rõ các thách thức về nút thắt cổ chai tại máy chủ, cơ chế khóa và đồng bộ dữ liệu tập trung, cũng như cách đảm bảo tính toàn vẹn dữ liệu khi tài nguyên lưu trữ là trọng tâm của hệ thống.

Việc xây dựng một hệ thống cho phép nhiều tác nhân (agents) cùng thay đổi một trạng thái chia sẻ (shared state) mà không gây ra xung đột dữ liệu là một trong những vấn đề khó khăn nhất của hệ thống phân tán. Khác với các ứng dụng web truyền thống dựa trên mô hình Yêu cầu Phản hồi và phi trạng thái, các ứng dụng cộng tác trong kiến trúc hướng dữ liệu đòi hỏi sự quản lý trạng thái chặt chẽ tại phía Server và cơ chế kiểm soát tranh chấp (concurrency control) nghiêm ngặt để duy trì tính nhất quán (consistency) của dữ liệu gốc.

1.1.1. Sự trong suốt phân tán (Distribution Transparency)

Sự trong suốt (Transparency) là thuộc tính quan trọng nhằm che giấu sự

phân tán của hệ thống đối với người dùng và lập trình viên ứng dụng. Mô hình tham chiếu ISO RM-ODP xác định 8 dạng trong suốt, mỗi dạng giải quyết một khía cạnh cụ thể của sự phân tán.

Phân tích 8 Dạng của Sự Trong suốt

1. Access Transparency (Trong suốt về Truy cập)
   Che giấu sự khác biệt trong biểu diễn dữ liệu và cơ chế gọi tài nguyên. Người dùng dùng cùng một lệnh để truy cập tệp tin cục bộ và tệp tin trên mạng (ví dụ: Network File System - NFS). Điều này đòi hỏi middleware phải xử lý triệt để các vấn đề marshalling và endianness đã nêu ở trên.18
2. Location Transparency (Trong suốt về Vị trí)
   Cho phép truy cập tài nguyên mà không cần biết vị trí vật lý hoặc mạng của nó. Ví dụ: Sử dụng URL (tên miền) thay vì địa chỉ IP cụ thể. Hệ thống cần các dịch vụ định danh (Naming Services) như DNS để phân giải tên thành địa chỉ động.
3. Migration Transparency (Trong suốt về Di trú)
   Cho phép di chuyển tài nguyên (ví dụ: một trang web) từ máy chủ này sang máy chủ khác mà không cần thay đổi tên định danh của nó. Điều này hỗ trợ việc cân bằng tải và bảo trì hệ thống mà không làm gián đoạn dịch vụ.
4. Relocation Transparency (Trong suốt về Tái định vị)
   Là cấp độ cao hơn của Migration Transparency, cho phép tài nguyên được di chuyển ngay trong khi nó đang được sử dụng (ví dụ: di chuyển máy ảo - Live Migration, hoặc người dùng di động chuyển vùng trạm phát sóng) mà không làm ngắt kết nối.
5. Replication Transparency (Trong suốt về Nhân bản):
   Che giấu việc có nhiều bản sao của cùng một tài nguyên tồn tại để tăng độ tin cậy và hiệu năng. Người dùng không cần biết họ đang đọc dữ liệu từ bản sao nào, và hệ thống phải đảm bảo các bản sao này nhất quán với nhau.
6. Concurrency Transparency (Trong suốt về Đồng thời):
   Cho phép nhiều người dùng cùng truy cập và chia sẻ tài nguyên đồng thời mà không gây xung đột. Hệ thống phải cài đặt các cơ chế kiểm soát đồng thời phức tạp như khóa (locking), timestamp ordering, hoặc optimistic concurrency control để đảm bảo tính toàn vẹn dữ liệu.

7. Failure Transparency (Trong suốt về Lỗi):
   Che giấu các lỗi và quá trình phục hồi của hệ thống. Người dùng có thể tiếp tục công việc của mình dù một số thành phần phần cứng hoặc phần mềm bị hỏng. Đây là dạng khó đạt được nhất, yêu cầu các cơ chế dự phòng (redundancy), checkpointing và chuyển đổi dự phòng (failover) tự động.
8. Persistence Transparency (Trong suốt về Bền vững):
   Che giấu việc tài nguyên đang nằm trong bộ nhớ tạm thời (RAM) hay lưu trữ vĩnh viễn (Disk). Các hệ thống cơ sở dữ liệu đối tượng thường cung cấp tính năng này, tự động lưu trữ trạng thái đối tượng xuống đĩa khi cần thiết.

1.1.2. Đồng bộ hóa (Synchronization) & Giải quyết xung đột

Đây là bài toán khó nhất khi "Nhiều Client cùng gửi dữ liệu". Vấn đề không chỉ là xử lý tuần tự, mà là xử lý sự đồng thời (Concurrency).

Vấn đề Race Condition (Điều kiện đua): Giả sử văn bản là "ABC".

- User A muốn chèn "X" vào sau "A".
- User B muốn xóa "C".
- Nếu Server không xử lý đúng thứ tự hoặc ngữ cảnh, kết quả cuối cùng có thể bị sai lệch (dữ liệu không nhất quán) giữa các máy trạm.

Thách thức về thứ tự (Ordering): Gói tin trên mạng có thể đến không theo thứ tự gửi. Server cần cơ chế để sắp xếp lại (Re-ordering) hoặc gộp các thay đổi (Merge) một cách thông minh mà không cần khóa (Lock) toàn bộ văn bản (vì lock sẽ làm treo hệ thống).

1.1.3. Thách thức về tính Không đồng nhất (Heterogeneity) và Biểu diễn dữ liệu

Tính không đồng nhất là đặc tính cố hữu của hệ thống phân tán, thể hiện qua sự đa dạng của phần cứng, hệ điều hành, mạng lưới, ngôn ngữ lập trình và các cấu trúc dữ liệu. Sự đa dạng này đặt ra thách thức về khả năng tương tác (interoperability): làm sao để các thành phần khác biệt này có thể hiểu và làm việc với nhau.

Các Tầng của Sự Không đồng nhất:

1. Phần cứng và Hệ điều hành: Các máy tính trong hệ thống có thể sử dụng các kiến trúc vi xử lý khác nhau (x86, ARM, SPARC) và hệ điều hành khác nhau (Linux, Windows, macOS). Điều này dẫn đến sự khác biệt trong cách quản lý bộ nhớ, luồng và biểu diễn dữ liệu cơ bản.
2. Mạng lưới: Sự khác biệt về giao thức truyền thông, băng thông, độ trễ và độ tin cậy giữa các loại mạng (Ethernet, Wi-Fi, mạng di động) đòi hỏi ứng dụng phải có khả năng thích ứng linh hoạt.
3. Ngôn ngữ lập trình: Các thành phần phần mềm có thể được phát triển bằng Java, C++, Python, Go, v.v. Mỗi ngôn ngữ có hệ thống kiểu dữ liệu và cách quản lý bộ nhớ riêng, gây khó khăn cho việc trao đổi đối tượng dữ liệu phức tạp.

1.2. Lựa chọn Mô hình Kiến trúc

Thay vì dùng mô hình P2P phức tạp, nhóm lựa chọn Mô hình Khách - Chủ (Client-Server) (Chương 2 Giáo trình PTIT):

1.2.1. Tại sao chọn Client-Server thay vì P2P?

Trong bối cảnh công nghệ thông tin đương đại, sự chuyển dịch từ các hệ thống tập trung đơn lẻ sang các hệ thống phân tán (distributed systems) đã trở thành một xu hướng tất yếu nhằm đáp ứng nhu cầu về khả năng mở rộng, hiệu năng và độ tin cậy. Một hệ thống phân tán được định nghĩa là một tập hợp các máy tính độc lập, không đồng nhất về phần cứng và hệ điều hành, được liên kết với nhau thông qua mạng lưới truyền thông để cùng phối hợp làm việc hướng tới một mục tiêu chung. Mục tiêu tối thượng của thiết kế này là tạo ra sự trong suốt về phân tán (distribution transparency), đảm bảo rằng người dùng cuối cảm nhận hệ thống như một thực thể thống nhất, che giấu sự phức tạp của các thành phần rời rạc bên dưới.

Để đạt được sự phối hợp hiệu quả giữa các thành phần phân tán, các kỹ sư và kiến trúc sư phần mềm phải đối mặt với quyết định cốt lõi: lựa chọn phong cách kiến trúc (Architectural Style). Phong cách kiến trúc không chỉ định nghĩa

cấu trúc tĩnh của hệ thống mà còn quy định hành vi, cơ chế giao tiếp và cách thức tổ chức các thành phần. Trước khi đi sâu vào phân tích lý do lựa chọn mô hình Khách - Chủ (Client-Server), việc khảo sát tổng quan các phong cách kiến trúc phân tán khác là cần thiết để thiết lập bối cảnh so sánh và đánh giá.

Trong giáo trình Hệ phân tán (và thực tế triển khai Google Docs hay Microsoft Office 365), mô hình Client-Server giải quyết được bài toán khó nhất là "Source of Truth".

- **Client-Server:** Server dễ dàng kết nối với Cơ sở dữ liệu (MySQL, MongoDB...) để lưu lại nội dung văn bản. Nếu tất cả người dùng thoát ra (offline), văn bản vẫn còn đó trên Server.
- Nếu dùng P2P: Mọi Client đều bình đẳng. Khi A sửa dòng 1, B sửa dòng 1, việc quyết định ai đúng ai sai cực kỳ phức tạp (cần thuật toán đồng thuận phức tạp như trong Blockchain hoặc CRDTs).
- Dùng Client-Server: Server là trọng tài. Ai đến trước (hoặc ai được Server xử lý trước) là đúng. Server nắm giữ "Master Copy" nên việc giải quyết xung đột dễ dàng hơn nhiều.

1.2.2. Chi tiết vai trò các thành phần

a) Server (Máy chủ)

- Lưu trữ tập trung (Centralized Storage): Nơi duy nhất lưu trữ phiên bản chuẩn xác nhất của văn bản vào Cơ sở dữ liệu (Database).
- Sắp xếp thứ tự (Ordering): Server gán nhãn thời gian (timestamp) hoặc số thứ tự (sequence number) cho từng thao tác nhận được từ các Client. Điều này đảm bảo mọi người đều nhìn thấy các thay đổi theo cùng một trình tự.
- Phát tán (Broadcasting): Sử dụng mẫu thiết kế Publish/Subscribe. Khi Client A gửi một ký tự mới, Server phải ngay lập tức "bắn" (push) cập nhật này tới Client B, Client C... đang mở cùng văn bản đó.

b) Client (Máy khách)

- Gửi yêu cầu (Request): Đóng gói thao tác người dùng thành bản tin JSON và gửi lên Server.
- Hiển thị & Cập nhật (Rendering & Patching):
    - Nhận dữ liệu từ Server và vẽ lại văn bản.
    - Quan trọng: Client không thụ động chờ Server. Nó cần hiện chữ ngay khi người dùng gõ (cơ chế Optimistic UI đã nhắc ở phần 1.1) để đảm bảo độ mượt, sau đó mới âm thầm đồng bộ với Server.

1.2.3. Ưu nhược điểm của mô hình đã chọn

Đặc điểm	Phân tích cho bài toán Soạn thảo văn bản
Ưu điểm	- Dễ quản lý: Dễ dàng kiểm soát quyền truy cập (ai được xem, ai được sửa). - Nhất quán: Do có một bản Master Copy, dữ liệu ít bị xung đột (conflict) hơn P2P. - Phù hợp với Web: Trình duyệt web (Browser) sinh ra là để làm Client.
Nhược điểm	- Điểm nghẽn cổ chai (Bottleneck): Nếu quá nhiều người cùng sửa một lúc, Server có thể bị quá tải. - Điểm chết duy nhất (Single Point of Failure): Nếu Server "sập", không ai soạn thảo được nữa.

1.3. Giải thuật quản lý tương tranh: Centralized Broadcasting

Để giải quyết xung đột cập nhật, nhóm sử dụng cơ chế Quảng bá tập trung (Centralized Broadcasting) kết hợp Last-Writer-Wins (Ghi sau thắng):

1.3.1. Nguyên lý hoạt động chi tiết (Flow of Execution)

Nguyên lý: Mọi thay đổi không được áp dụng cục bộ ngay mà phải gửi lên Server. Server nhận gói tin nào trước thì xử lý trước.

Mô tả quy trình theo từng bước :

1. Gửi (Send): Khi người dùng gõ một ký tự, Client không hiển thị ngay lập tức. Client đóng gói thao tác (ví dụ: Insert 'A' at index 5) và gửi lên Server. Lúc này giao diện người dùng có thể bị "khóa" hoặc hiện trạng thái "đang xử lý" (loading).

2. Hàng đợi & Tuần tự hóa: Server nhận các yêu cầu từ nhiều Client. Vì Server xử lý đơn luồng (hoặc dùng cơ chế khóa/lock), các yêu cầu này được đưa vào một Hàng đợi (Queue).

Ví dụ: Client A gửi lúc 10:00:01, Client B gửi lúc 10:00:02. Server sẽ xử lý yêu cầu của A trước, sau đó đến B.

3. Xử lý & Chốt trạng thái : Server áp dụng thay đổi vào "Master Copy". Tại đây, nguyên tắc Last-Writer-Wins được áp dụng ở mức độ: "Ai đến sau (trong hàng đợi xử lý) sẽ ghi đè lên trạng thái trước đó nếu có xung đột tại cùng một vị trí".

4. Quảng bá: Sau khi cập nhật thành công, Server gửi bản cập nhật mới nhất (hoặc chỉ thị thay đổi) tới tất cả các Client đang kết nối (bao gồm cả người gửi).

5. Cập nhật giao diện (Render): Các Client nhận dữ liệu từ Server và vẽ lại văn bản. Lúc này người dùng mới nhìn thấy ký tự mình vừa gõ.

1.3.2. Phân tích sâu về Ưu/Nhược điểm

- **Ưu điểm:**

Mô hình Client-Server vẫn là "xương sống" của hầu hết các hệ thống doanh nghiệp và web hiện đại nhờ vào khả năng kiểm soát và tính ổn định cao.

- **Quản lý và Kiểm soát Tập trung (Centralization):**

  Đây là lợi thế lớn nhất. Mọi dữ liệu và logic nghiệp vụ cốt lõi đều nằm tại Server. Điều này giúp việc sao lưu dữ liệu (backup), bảo trì và nâng cấp trở nên đơn giản hơn rất nhiều so với mô hình phi tập trung (P2P). Khi có bản và lỗi (patch) hoặc tính năng mới, quản trị viên chỉ cần cập nhật tại Server thay vì phải triển khai trên hàng ngàn máy trạm của người dùng.

- **Bảo mật Vượt trội (Enhanced Security):**

  Do dữ liệu được lưu trữ tập trung, doanh nghiệp có thể tập trung nguồn lực để xây dựng các hàng rào bảo mật "kiên cố" (như tường lửa, mã hóa, kiểm soát truy cập) xung quanh Server. Client không bao giờ tiếp xúc trực tiếp với cơ sở dữ liệu gốc mà phải thông qua các lớp trung gian được xác thực, giảm thiểu rủi ro rò rỉ dữ liệu từ phía người dùng cuối.

- **Khả năng Mở rộng (Scalability):** Mô hình này hỗ trợ tốt cả hai phương thức mở rộng:
  - *Vertical Scaling (Scale-up):* Dễ dàng nâng cấp phần cứng (RAM, CPU) cho Server để xử lý nhiều tải hơn.
  - *Horizontal Scaling (Scale-out):* Có thể thêm nhiều Server xử lý song song kết hợp với Load Balancer để phân tải, đặc biệt hiệu quả trong kiến trúc 3-Tier hoặc Stateless.
- **Tính Nhất quán Dữ liệu (Data Consistency):**

  Vì dữ liệu được quản lý tại một nguồn duy nhất (hoặc một cụm database đồng bộ), việc duy trì tính toàn vẹn và nhất quán của dữ liệu (ACID properties) dễ dàng hơn nhiều so với việc phải đồng bộ hóa dữ liệu giữa hàng ngàn peer trong mạng ngang hàng.

- **Đơn giản hóa logic Client:** Client cực kỳ nhẹ (Thin Client), không cần logic phức tạp để xử lý xung đột, chỉ cần nhiệm vụ hiển thị (View).
- **Tính nhất quán tuyệt đối (Strong Consistency):** Vì mọi thứ đều đi qua "nút cổ chai" là Server, nên không bao giờ có chuyện Client A thấy một kiểu, Client B thấy một kiểu khác sau khi đồng bộ xong.

Tuy nhiên, kiến trúc này mang trong mình những điểm yếu mang tính cấu trúc mà các kỹ sư hệ thống buộc phải có giải pháp giảm thiểu (mitigation).

- **Điểm Lỗi Đơn (Single Point of Failure - SPoF):** Đây là "gót chân Achilles" của mô hình. Nếu Server trung tâm gặp sự cố (phần cứng hỏng, mất điện, lỗi phần mềm), toàn bộ hệ thống sẽ tê liệt. Tất cả các Client đều phụ thuộc hoàn toàn vào sự sống còn của Server.
    - ***Giải pháp:*** Buộc phải đầu tư vào các hệ thống dự phòng (Redundancy) và Failover, làm tăng chi phí.

- **Nghẽn Cổ chai Hiệu năng (Performance Bottlenecks):**

Khi lượng truy cập tăng đột biến (ví dụ: tấn công DDoS hoặc sự kiện Flash Sale), Server có thể bị quá tải do không kịp xử lý hàng loạt yêu cầu đồng thời. Điều này dẫn đến độ trễ cao hoặc từ chối dịch vụ.

- **Chi phí Đầu tư và Vận hành (Cost):** Khác với mô hình P2P tận dụng tài nguyên của người dùng, mô hình Client-Server đòi hỏi chi phí lớn để mua sắm máy chủ chuyên dụng, thiết bị mạng băng thông rộng và chi phí điện năng, làm mát để duy trì hoạt động 24/7. Chi phí bảo trì đội ngũ kỹ thuật (IT Admin) cũng là một gánh nặng tài chính.
- **Phụ thuộc vào Mạng (Network Dependency):** Mô hình này tách biệt nơi xử lý (Server) và nơi hiển thị (Client), do đó nó phụ thuộc hoàn toàn vào kết nối mạng. Nếu mạng bị gián đoạn hoặc băng thông thấp, Client (đặc biệt là Thin Client) trở nên vô dụng.

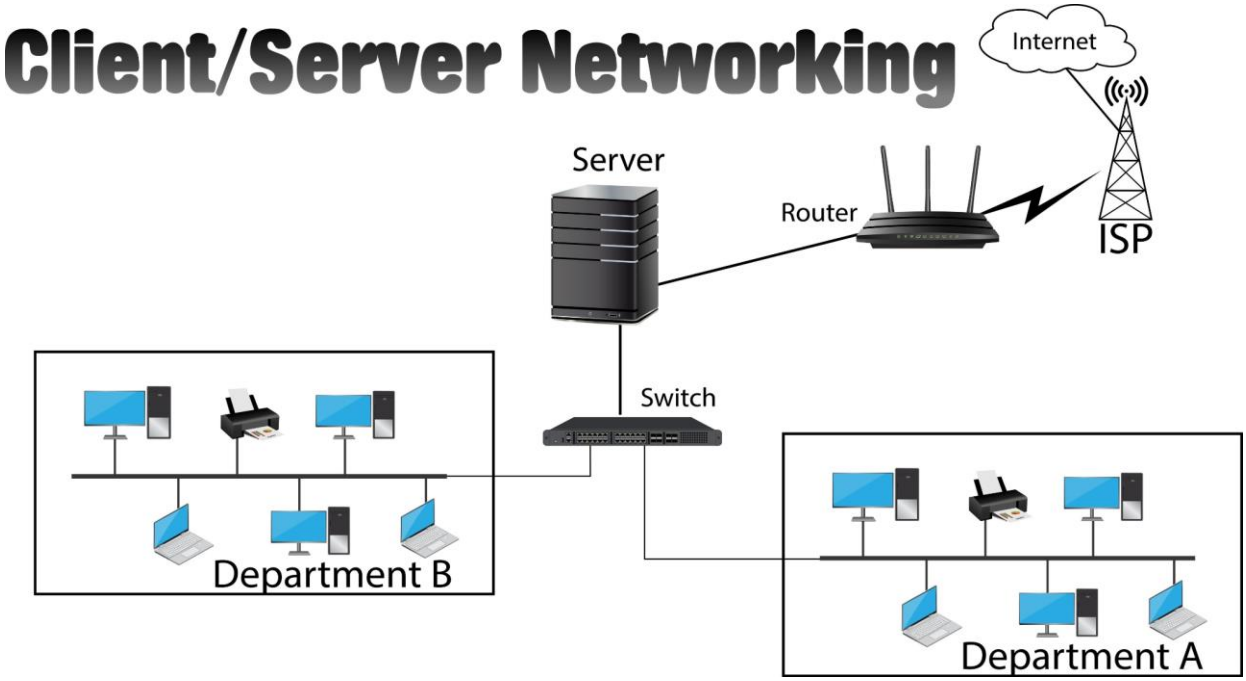
- **Độ trễ trải nghiệm**: Đây là vấn đề lớn nhất.
    - Nhưng với giải thuật này, người dùng phải chờ tín hiệu đi vòng qua Server rồi mới thấy chữ hiện lên. Nếu mạng chậm (ping cao), cảm giác gõ sẽ bị giật (lag).
    - *Cách gọi kỹ thuật:* Đây là cơ chế Non-optimistic UI (Giao diện không lạc quan).
- **Vấn đề ghi đè:** Với *Last-Writer-Wins*:
    - Nếu A sửa dòng 1.
    - B cũng sửa dòng 1 nhưng yêu cầu đến Server sau A một chút.

- Yêu cầu của B sẽ được thực thi trên kết quả của A. Nếu không cẩn thận, nội dung của A có thể bị biến đổi ý nghĩa hoặc mất mát mà B không hề hay biết.

CHƯƠNG II: THIẾT KẾ GIẢI THUẬT

2.1. Thiết kế Giao thức truyền thông (Communication Protocol)

2.1.1. Lựa chọn mô hình Client – Server



Hệ thống được xây dựng theo mô hình Client - Server, trong đó:

- Server: đóng vai trò trung tâm quản lý dữ liệu văn bản, tiếp nhận các thay đổi từ Client và phân phối lại cho các Client khác
- Client: là các ứng dụng soạn thảo văn bản, cho phép người dùng nhập liệu và nhận dữ liệu cập nhật theo thời gian thực.

Mô hình này phù hợp với bài toán soạn thảo văn bản cộng tác, giúp đảm bảo:

- Tính nhất quán dữ liệu
- Dễ dàng mở rộng số lượng người dùng
- Đơn giản hóa việc đồng bộ dữ liệu

2.1.2. Giao thức tầng vận tải

Hệ thống sử dụng TCP/IP (Transmission Control Protocol) để truyền dữ liệu giữa Client và Server.

Lý do lựa chọn TCP:

- TCP đảm bảo độ tin cậy cao, dữ liệu không bị mất gói
- Các gói tin được sắp xếp theo đúng thứ tự, rất quan trọng cho việc cập nhật văn bản
- Có cơ chế kiểm soát lỗi và tái truyền khi cần thiết

So với UDP, TCP phù hợp hơn cho các ứng dụng yêu cầu độ chính xác 100% như soạn thảo văn bản.

2.1.3. Định dạng dữ liệu truyền (Encoding)

Để đáp ứng việc hỗ trợ đầy đủ tiếng việt có dấu và các ký tự Unicode, dữ liệu văn bản được mã hóa theo chuẩn UTF-8, đảm bảo tương thích trên nhiều nền tảng và hệ điều hành khác nhau.

2.1.4. Vấn đề TCP Stickiness (Dính gói tin)

TCP là giao thức dạng luồng (stream based), do đó không phân biệt ranh giới giữa các thông điệp tại các lần gửi. Vì vậy có thể xảy ra hiện tượng dính gói tin: nhiều thông điệp gửi liên tiếp bị nhập gộp lại.

Ví dụ: Nếu Client gửi liên tiếp 2 lần "Hello", Server có thể nhận được "HelloHello" dính liền.

Do đó, nếu không có cơ chế đóng gói, phía nhận sẽ không thể xác định chính xác được một thông điệp bắt đầu và kết thúc từ đâu.

2.1.5. Giải pháp Message Framing (Đóng gói thông điệp)

Để giải quyết vấn đề TCP Stickiness - Dính gói tin ở trên, trong phạm vi ứng dụng soạn thảo văn bản đồng thời, hai phương pháp được đề xuất sử dụng đó là: Độ dài cố định (Length-based Framing) và Ký tự phân tách đặc biệt (Delimiter-based Framing).

a. Phương pháp Độ dài cố định (Length-based Framing)

Với phương pháp này, mỗi thông điệp được gửi sẽ kèm theo độ dài dữ liệu ở phần đầu. Cấu trúc gói tin bao gồm:

[**Header: Độ dài (k bytes)**] + [Payload: Dữ liệu]

Với cấu trúc gói tin như trên, để phía bên nhận ghép dữ liệu thành một thông điệp hàn chỉnh, phía nhận cần dựa vào phần header để xác định chính xác N byte dữ liệu cần đọc.

b. Phương pháp sử dụng Ký tự phân tách đặc biệt (Delimiter-based Framing).

Với phương pháp này, mỗi thông điệp kết thúc bằng một chuỗi ký tự đặc biệt, đóng vai trò như một dấu phân cách giữa các thông điệp (ví dụ: \n, \r\n, …).

Với cấu trúc gói tin như trên, để phía bên nhận ghép dữ liệu thành một thông điệp hàn chỉnh, phía nhận phải đọc dữ liệu liên tục từ socket và ghép dữ liệu liên tục vào buffer, sau đó xác định ký tự phân cách trước khi thực hiện tách và lấy thông điệp hoàn chỉnh.

2.2. Giải thuật phía Server (Đa luồng - Multithreading)

Server được thiết kế theo mô hình Multithreading Server, trong đó mỗi Client kết nối sẽ được xử lý bởi một luồng riêng biệt. Mô hình này giúp hệ thống có khả năng phục vụ đồng thời nhiều người dùng, phù hợp với các ứng dụng phân tán và cộng tác thời gian thực. Để đáp ứng yêu cầu xử lý trên, xác định rõ vai trò của Main Thread, các Worker Threads và cơ chế Broadcast như sau:

a. Main Thread (Luồng chính)

Main Thread đảm nhận vai trò điều phối, có các nhiệm vụ chính:

- Khởi tạo socket Server.
- Lắng nghe kết nối từ Client thông qua hàm listen().
- Chấp nhận kết nối mới bằng accept().
- Với mỗi kết nối mới, tạo một Worker Thread (ClientHandler) tương ứng.

Main Thread không xử lý dữ liệu ứng dụng, điều này giúp tránh bị block khi một Client gửi dữ liệu lớn đồng thời đảm bảo Server luôn sẵn sàng tiếp nhận các kết nối mới.

b) Worker Thread (ClientHandler)

Mỗi Worker Thread gắn với **một Client duy nhất**, có các nhiệm vụ:

- Nhận dữ liệu từ Client thông qua recv()
- Giải mã dữ liệu theo giao thức Message Framing đã định nghĩa.
- Xử lý nội dung văn bản nhận được.
- Gửi dữ liệu đó tới các Client khác (Broadcast).

Việc tách riêng mỗi Client vào một luồng giúp các Client hoạt động độc lập với nhau. Khi một Client bị treo hoặc gửi dữ liệu chậm không ảnh hưởng tới Client khác.

c. Cơ chế Broadcast dữ liệu

Broadcast là cơ chế gửi dữ liệu từ một Client tới tất cả các Client khác thông qua Server, khi một luồng nhận được văn bản mới, nó sẽ duyệt qua danh sách tất cả các kết nối đang mở (clients list) và gửi dữ liệu đó đi. Broadcast dữ liệu bao gồm các quá trình sau:

- Worker Thread nhận dữ liệu từ Client A.
- Server cập nhật trạng thái văn bản hiện tại.
- Server gửi dữ liệu này tới toàn bộ Client còn lại.

Broadcast trong ứng dụng đóng vai trò như một bộ điều phối trung tâm đảm bảo dữ liệu được phân phối theo đúng thứ tự, tránh việc client gửi trực tiếp cho nhau, giúp hệ thống dễ kiểm soát và mở rộng.

2.3. Giải thuật phía Client (Bất đồng bộ)

Để đảm bảo các client hoạt động và phản hồi mượt mà, đồng thời không gặp tình trạng bị treo khi chờ dữ liệu từ Server, client cần thực hiện 2 tác vụ sau một cách song song:

- Gửi dữ liệu người dùng nhập: Client bắt sự kiện thay đổi nội dung từ bàn phím. Sau mỗi lần thay đổi nội dung, lấy toàn bộ nội dung Text Area, đóng gói theo giao thức định nghĩa và gửi lên Server.
- Nhận dữ liệu: liên tục lắng nghe và nhận dữ liệu từ Server (recv()) nhờ một background thread chạy song song với giao diện. Khi nhận được dữ liệu cần thực hiện giải mã UTF-8 và cập nhật nội dung Text Area.

CHƯƠNG III: HIỆN THỰC HÓA (SOURCE CODE CHI TIẾT)

3.1. Phân tích server (Máy chủ)

Máy chủ đóng vai trò là bộ não trung tâm. Nhiệm vụ của nó là “Lắng nghe kết nối, Lưu trữ trạng thái văn bản và phân phối (Broadcast) dữ liệu.

3.1.1. Khai báo cấu hình và kho dữ liệu

```
import socket

import threading

clients = []

current_text = ""

lock = threading.Lock()
```

- clients = []: Đây là danh sách quản lý thành viên. Khi một Client kết nối, họ được thêm vào đây. Khi Client gửi tin, Server sẽ duyệt danh sách này để gửi tin lại cho những người khác.
- current_text: Đây là biểu hiện của Trạng thái toàn cục (Global State). Vì chúng ta dùng mô hình tập trung (Centralized), biến này là "chân lý" duy nhất.
- lock (Khóa): Tưởng tượng biến current_text là một nhà vệ sinh công cộng. lock là cái chìa khóa. Khi một luồng (Thread) đang vào sửa dữ liệu, nó khóa cửa lại. Các luồng khác phải đứng chờ bên ngoài. Điều này giúp Server không bị "loạn" khi nhiều người gửi tin cùng lúc.

3.1.2. Hàm send_json (sock_data)

```
json_str = json.dumps(data_dict) + "\n"

sock.sendall(json_str.encode('utf-8'))
```

Thêm \n? TCP giống như một dòng nước chảy. Tương tự như 2 con thuyền giấy (2 gói tin) liên tiếp, chúng có thể dính vào nhau. Dấu \n giống như cái vách ngăn để người nhận biết đâu là đuôi của con thuyền trước.

3.1.3. Hàm handle_client (vòng lặp xử lý chính)

```
buffer = ""
while True:
    data = client_socket.recv(4096)
    if not data: break
    buffer += data.decode('utf-8')
    while "\n" in buffer:
    message, buffer = buffer.split("\n", 1)
```

Cơ chế Bộ đệm (Buffer):

1. Nhận dữ liệu thô đổ vào buffer.
2. Dùng vòng lặp while để cắt buffer dựa trên dấu \n.
3. Phần cắt ra được (message) đem đi xử lý.
4. Phần thừa còn lại giữ trong buffer để chờ nối với dữ liệu lần sau.

3.1.4. Hàm start_server()

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))
server.listen()
```

- AF_INET: Sử dụng địa chỉ IPv4.
- SOCK_STREAM: Sử dụng giao thức TCP. Đây là giao thức hướng kết nối và tin cậy, đảm bảo văn bản không bị mất chữ giữa đường.

3.1.5. Phần Lưu trữ (Persistence)

save_to_file & load_from_file: Đơn giản là ghi nội dung biến current_text vào file.

shared_doc.txt trên ổ cứng. Giúp Server có tắt đi bật lại thì văn bản vẫn còn.

3.2. Phân tích client (Máy khách)

Máy khách có nhiệm vụ: Hiển thị giao diện, Gửi những gì client gõ, và Nhận những gì từ client khác gõ.

3.2.1. Giao diện (GUI)

```
self.text_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, width=50, height=20)

self.text_area.bind('<KeyRelease>', self.send_update)
```

- Chúng ta dùng thư viện tkinter để vẽ cửa sổ.
- Sự kiện <KeyRelease>: Đây là mấu chốt của tính năng Real-time. Ngay khi nhấc tay khỏi một phím bất kỳ, hàm send_update sẽ được kích hoạt để gửi dữ liệu đi ngay lập tức.

3.2.2. Hàm send_update (gửi dữ liệu)

```
def send_update(self, event):
    packet = {"type": "update_text", "content": content}
    json_str = json.dumps(packet) + "\n"
    self.client_socket.sendall(json_str.encode('utf-8'))
```

- Client lấy toàn bộ văn bản trong khung, đóng gói vào phong bì JSON, dán tem \n và gửi đi.

3.2.3. Hàm receive_updates() - Luồng nghe ngầm

Chạy trên một Luồng riêng (Background Thread) để không làm đơ giao diện.

- Logic xử lý buffer và \n y hệt như phía Server.
- Xử lý con trỏ chuột:

```
def receive_updates(self):
    buffer = "" # Bộ đệm lưu trữ dữ liệu chưa hoàn chỉnh
```

```
while True:
    try:
        data = self.client_socket.recv(4096)
        if not data: break
        buffer += data.decode('utf-8')
        while "\n" in buffer:
            message, buffer = buffer.split("\n", 1)
            if not message.strip(): continue
            try:
                self.process_packet(json.loads(message))
            except json.JSONDecodeError:
                print(f"Lỗi JSON: {message}")
    except Exception as e:
        print(f"Ngắt kết nối: {e}")
        Break
```

Tại sao phải chạy trong Thread riêng?

Nếu không có đoạn này sẽ xảy ra trường hợp: Client A hiện tại đang gõ dòng 10, Client B sửa dòng 1, màn hình cập nhật xong con chuột của Client A sẽ bị trở về về dòng 1. Đoạn code này giúp giữ con chuột nằm yên chỗ Client hiện tại đang gõ.

3.3. Tóm tắt Luồng hoạt động (Workflow)

Quy trình khi Client A gõ chữ "A":

1. Tại Client A:
    - Sự kiện KeyRelease bắt được chữ "A".
    - Hàm send_update tạo gói tin: {"content": "...A"} + \n.

- Gửi qua đường truyền internet.

2. Tại Server:
    - Hứng được dữ liệu vào buffer.
    - Thấy dấu \n, cắt gói tin ra.
    - Cập nhật biến current_text.
    - Ghi xuống file shared_doc.txt.
    - Broadcast: Gửi gói tin đó + \n cho tất cả mọi người khác.
3. Tại Client Client B:
    - Hứng dữ liệu vào buffer.
    - Thấy dấu \n, cắt gói tin ra.
    - Xóa màn hình cũ, điền văn bản mới (có chữ "A") vào.
    - Đặt lại con trỏ chuột để họ tiếp tục làm việc.

CHƯƠNG IV: ĐÁNH GIÁ HỆ THỐNG VÀ KẾT LUẬN

4.1. Kết quả đạt được

Hệ thống soạn thảo văn bản theo thời gian thực đã được xây dựng thành công và đáp ứng được các yêu cầu cơ bản của bài toán đặt ra:

- Cho phép từ hai người dùng trở lên kết nối đồng thời đến Server và cùng thao tác trên một tài liệu chung. Việc sử dụng mô hình Client – Server kết hợp với xử lý đa luồng giúp Server có thể phục vụ nhiều Client cùng lúc mà không xảy ra hiện tượng treo hoặc gián đoạn.
- Khi một Client thực hiện thao tác nhập liệu, nội dung văn bản được gửi lên Server và phát tán đến các Client còn lại gần như ngay lập tức. Nhờ đó, người dùng ở các Client khác có thể quan sát được sự thay đổi của văn bản theo thời gian thực.
- Đáp ứng được các yêu cầu nền tảng của một hệ thống phân tán, bao gồm khả năng chia sẻ tài nguyên (tài liệu văn bản) và truyền thông dữ liệu giữa các tiến trình chạy trên các máy khác nhau thông qua mạng

4.2. Hạn chế (Dựa trên lý thuyết)

Một số hạn chế của hệ thống:

- Server trở thành một điểm nghẽn tập trung (Centralized Bottleneck): trong trường hợp Server gặp sự cố hoặc ngừng hoạt động, toàn bộ hệ thống sẽ không thể tiếp tục vận hành (Single Point of Failure).
- Tiêu tốn băng thông lớn nếu văn bản dài: hệ thống hiện tại áp dụng phương pháp gửi toàn bộ nội dung văn bản (Full Text Transmission) mỗi Client khi thực hiện gõ phím. Việc này sẽ gây ra tiêu tốn nhiều băng thông, đặc biệt khi văn bản có độ dài lớn hoặc số lượng Client tăng lên.
- Chưa có cơ chế xử lý xung đột chỉnh sửa hiệu quả: Khi 2 Client cùng nhập liệu gần như đồng thời, các gói tin gửi về Server có thể đến không đúng thứ tự thao tác thực tế. Trong trường hợp này, hệ thống áp dụng cơ chế “Last-Writer-Wins”, dữ liệu đến sau có thể ghi đè dữ liệu đến trước, dẫn đến khả năng mất một số ký tự do các Client khác nhập.

4.3. Hướng phát triển

Một số hướng cải tiến hệ thống:

- Hay đổi giải thuật truyền dữ liệu từ việc gửi toàn bộ văn bản (Full Text Transmission) sang gửi các sự kiện chỉnh sửa (operation-based), như thao tác chèn hoặc xóa ký tự. Giải thuật mới này giúp giảm lượng dữ liệu truyền qua mạng và nâng cao hiệu năng hệ thống.
- Để tối ưu thêm xung đột khi nhiều Client cùng chỉnh sửa, hệ thống có thể bổ sung các cơ chế đồng bộ khóa (locking) hoặc mô hình Token Ring, trong đó chỉ một Client được phép chỉnh sửa tại một thời điểm. Các cơ chế này phù hợp trong trường hợp hệ thống yêu cầu độ chính xác tuyệt đối và dữ liệu không được phép bị ghi đè.

TÀI LIỆU THAM KHẢO

[1] P. V. Cường, N. X. Anh, Giáo trình các hệ thống phân tán, Hà Nội: Học Viện Công Nghệ Bưu Chính Viễn Thông, 2024.

[2] Maarten van Steen Andrew S. Tanenbaum, Distributed Systems Third edition Version 3.03: Pearson Education, 2020.

[3] " 8 Thách Thức Của Việc Triển Khai Trong Khai Thác Dữ Liệu". [Online]. Available: https://bluecore.vn/blogs/news/8-thach-thuc-cua-viec-trien-khai-trong-khai-thac-du-lieu. [Accessed 15 December 2025].