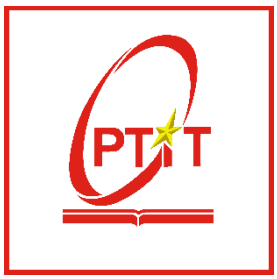
BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



CHỦ ĐỀ:

Thiết kế và Triển khai hệ thống P2P File Sharing sử dụng Pekko Framework

Người hướng dẫn: TS. Kim Ngọc Bách

Học viên: Hoàng Thanh Hằng - B24CHHT068

Giáp Thị Huê - B24CHHT075

Trần Văn Quyền - B24CHHT089

Hoàng Phương Hoa - B24CHHT072

Lớp: M24CQHT02-B

Hà Nội, tháng 08/2025

MỤC LỤC

Danh mục hình ảnh

Tóm tắt (Abstract)

Bài báo cáo này trình bày thiết kế và triển khai một hệ thống chia sẻ tệp ngang hàng (P2P) trên nền tảng Pekko Framework. Hệ thống không chỉ giải quyết các hạn chế của mô hình tập trung mà còn được nâng cao với các mẫu thiết kế phân tán tiên tiến. Cụ thể, báo cáo tập trung vào việc sử dụng CBOR để tuần tự hóa dữ liệu hiệu quả, tích hợp các mẫu như Circuit Breaker, Bulkheads, Fallback, Load Balancing và Service Registry để tăng cường khả năng chịu lỗi, độ ổn định và tính linh hoạt. Kết quả thực nghiệm cho thấy hệ thống hoạt động hiệu quả, có khả năng tự phục hồi và mở rộng động. Báo cáo minh chứng tính khả thi của việc kết hợp Pekko với các mẫu thiết kế hiện đại, mở ra hướng đi cho các ứng dụng phân tán trong tương lai.

Chương 1. GIỚI THIỆU

1.1. Bối cảnh và Đặt vấn đề

Sự phát triển bùng nổ của Internet và kỷ nguyên số hóa hiện nay đã thúc đẩy nhu cầu trao đổi và chia sẻ dữ liệu giữa người dùng ngày càng tăng cao. Trong bối cảnh đó, các hệ thống tập trung (Client-Server) truyền thống, dù phổ biến, đang bộc lộ nhiều hạn chế đáng kể. Chúng thường phải đối mặt với các thách thức về khả năng mở rộng, điểm nghẽn đơn lẻ (single point of failure), và chi phí vận hành cao khi số lượng người dùng và dữ liệu tăng lên, đặc biệt đối với các ứng dụng yêu cầu tính sẵn sàng cao và khả năng phục hồi nhanh chóng.

Ngược lại, hệ thống ngang hàng (Peer-to-Peer - P2P) nổi bật như một giải pháp phân tán hiệu quả, nơi mỗi nút mạng (peer) vừa hoạt động như một máy khách tiêu thụ tài nguyên, vừa là một máy chủ cung cấp tài nguyên. Mô hình này mang lại nhiều ưu điểm vượt trội như khả năng mở rộng tự nhiên (linear scalability), tăng cường độ bền vững (robustness) và khả năng chịu lỗi (fault tolerance) do không phụ thuộc vào một máy chủ trung tâm. Hệ thống P2P cho phép các máy tính ngang hàng trực tiếp chia sẻ tài nguyên với nhau, phân tán tải trọng và tăng cường khả năng chịu lỗi của toàn hệ thống.

Tuy nhiên, việc thiết kế và triển khai một hệ thống P2P mạnh mẽ, có khả năng mở rộng và chịu lỗi trong thực tế là một nhiệm vụ phức tạp, đòi hỏi sự hiểu biết sâu sắc về các nguyên lý của hệ thống phân tán và việc lựa chọn các công nghệ nền tảng phù hợp. Đặc biệt, việc quản lý trạng thái phân tán, đồng bộ hóa dữ liệu và xử lý lỗi vẫn là những thách thức đáng kể.

Trong bối cảnh các hệ thống phân tán ngày càng phức tạp, việc sử dụng các framework và thư viện hỗ trợ là vô cùng cần thiết để giảm thiểu gánh nặng phát triển. Pekko (trước đây là Pekko) là một framework mạnh mẽ cho việc xây dựng các ứng dụng đồng thời và phân tán trên nền tảng JVM, với mô hình Actor cho phép quản lý trạng thái hiệu quả và khả năng chịu lỗi. Pekkrop, một dự án dựa trên Pekko, cung cấp các công cụ và mô hình nâng cao để đơn giản hóa việc xây dựng các ứng dụng P2P, đặc biệt trong việc quản lý cluster, khám phá thành viên và đồng bộ hóa dữ liệu phân tán thông qua Pekko Distributed Data (CRDTs). Điều này biến Pekkrop thành một công cụ lý tưởng để giải quyết các thách thức cố hữu trong việc xây dựng hệ thống P2P.

1.2. Mục tiêu nghiên cứu

Báo cáo này tập trung vào việc nghiên cứu, thiết kế và triển khai một hệ thống chia sẻ tệp tin P2P cơ bản nhằm minh chứng tính khả thi và hiệu quả của việc sử dụng Pekkrop Framework trong việc xây dựng ứng dụng phân tán. Các mục tiêu cụ thể bao gồm:

- Nghiên cứu và ứng dụng Pekko Framework: Tìm hiểu sâu về kiến trúc Pekko Actor Model và cách Pekko Framework tận dụng Pekko Cluster và Pekko Distributed Data để xây dựng các ứng dụng phân tán.

- Thiết kế kiến trúc P2P phi tập trung: Xây dựng một kiến trúc hệ thống P2P hoàn toàn phân tán, nơi mỗi node có thể tự động khám phá các node khác và tham gia vào mạng.

- Triển khai quản lý trạng thái tệp tin phân tán: Sử dụng Pekko Distributed Data (CRDTs) để duy trì một chỉ mục tệp tin nhất quán mềm trên toàn bộ cluster, cho phép các node biết được tệp tin nào đang được chia sẻ bởi node nào.

- Phát triển cơ chế chia sẻ và tải xuống tệp tin: Triển khai luồng truyền tải tệp tin trực tiếp giữa các peer thông qua Pekko Streams, đảm bảo hiệu suất và độ tin cậy.

- Xây dựng giao diện tương tác đơn giản: Cung cấp giao diện dòng lệnh (TUI) để người dùng có thể dễ dàng đăng ký, liệt kê và yêu cầu tệp tin.

Đánh giá khả năng hoạt động: Kiểm tra và đánh giá khả năng cơ bản của hệ thống trong việc chia sẻ và tải xuống tệp tin trong môi trường phân tán giả lập.

1.3. Phạm vi nghiên cứu

- Dự án này tập trung vào các chức năng cốt lõi của một hệ thống chia sẻ tệp tin P2P và các khía cạnh liên quan đến việc ứng dụng Pekkrop Framework. Cụ thể, phạm vi bao gồm:

- Nghiên cứu và ứng dụng Pekko Framework: Sử dụng Scala làm ngôn ngữ lập trình, tìm hiểu sâu về kiến trúc Pekko Actor Model và cách Pekko Framework tận dụng Pekko Cluster và Pekko Distributed Data để xây dựng các ứng dụng phân tán.

- Thiết kế kiến trúc P2P: Xây dựng một kiến trúc hệ thống P2P File Sharing module, có khả năng mở rộng và chịu lỗi.

- Triển khai các chức năng cốt lõi: Thực hiện các chức năng cơ bản của một hệ thống chia sẻ tệp như tham gia/rời mạng, đăng ký/chia sẻ tệp tin, tìm kiếm tệp và tải tệp tin giữa các peer.

- Đánh giá sơ bộ: Thực hiện các thử nghiệm ban đầu để đánh giá hiệu năng và khả năng hoạt động của hệ thống trong môi trường phân tán giả.

- Hệ thống sẽ không đi sâu vào các tính năng nâng cao như:

- Mã hóa và bảo mật dữ liệu đầu cuối trong quá trình truyền tải.
- Quản lý danh tính hoặc xác thực người dùng/peer.
- Tối ưu hóa phức tạp cho việc tìm kiếm (ví dụ: DHT toàn diện) hoặc lựa chọn peer (ví dụ: chọn peer gần nhất, nhanh nhất).
- Hỗ trợ Resume/Pause cho các lượt tải xuống.
- Giao diện người dùng đồ họa (GUI).

Các tính năng này có thể là hướng phát triển trong tương lai.

1.4. Cấu trúc báo cáo

Báo cáo này được tổ chức thành các phần chính như sau:

Chương 1: Giới thiệu: Trình bày bối cảnh, đặt vấn đề, tổng quan về hệ thống P2P File Sharing, mục tiêu và phạm vi của nghiên cứu.

Chương 2: Cơ sở lý thuyết và công nghệ áp dụng: Cung cấp cái nhìn tổng quan về hệ thống P2P, Pekko Framework và Pekkrop Framework, cùng với các công nghệ hỗ trợ khác được sử dụng như mô hình Actor, Pekko Toolkit và Pekko Distributed Data.

Chương 3: Thiết kế chi tiết hệ thống: Mô tả kiến trúc tổng thể, các module chính, luồng dữ liệu, thiết kế chi tiết các thành phần Actor và các cơ chế giao tiếp, cùng với các sơ đồ hệ thống minh họa.

Chương 4: Triển khai và kết quả thực nghiệm: Trình bày quá trình triển khai, các thách thức gặp phải, môi trường triển khai, các bước thực hiện và kết quả thử nghiệm, đánh giá hiệu suất và khả năng hoạt động của hệ thống trong môi trường phân tán giả lập.

Chương 5: Kết luận và hướng phát triển: Tóm tắt các kết quả đạt được, đóng góp của nghiên cứu, những hạn chế còn tồn tại và đề xuất các hướng phát triển trong tương lai.

Chương 2. CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ ÁP DỤNG

2.1. Kiến trúc hệ thống phân tán

2.1.1. Khái niệm và đặc điểm

Hệ thống phân tán là một tập hợp các máy tính độc lập về vật lý nhưng hợp tác với nhau để thực hiện một mục tiêu chung. Các máy tính này giao tiếp với nhau thông qua mạng, và đối với người dùng, hệ thống xuất hiện như một thực thể đơn nhất. Các đặc điểm chính của hệ thống phân tán bao gồm:

- **Tính đồng thời (Concurrency):** Nhiều tiến trình hoặc thành phần có thể hoạt động song song.
- **Không có đồng hồ chung toàn cục (No Global Clock):** Các thành phần có đồng hồ riêng, và việc đồng bộ thời gian giữa chúng là một thách thức.
- **Các lỗi độc lập (Independent Failures):** Một thành phần có thể thất bại mà không ảnh hưởng ngay lập tức đến toàn bộ hệ thống. Đây là một lợi thế nhưng cũng là thách thức trong việc thiết kế khả năng chịu lỗi.
- **Tính trong suốt (Transparency):** Người dùng không cần biết các thành phần được phân bố như thế nào và chúng giao tiếp ra sao.
- **Khả năng mở rộng (Scalability):** Hệ thống có thể dễ dàng mở rộng để xử lý lượng tải lớn hơn bằng cách thêm tài nguyên.

2.1.2. Các mô hình kiến trúc phân tán phổ biến

- **Client-Server:** Mô hình phổ biến nhất, trong đó máy khách gửi yêu cầu và máy chủ cung cấp dịch vụ. Ưu điểm là đơn giản, dễ quản lý; nhược điểm là điểm lỗi đơn, dễ tắc nghẽn.
- **Peer-to-Peer (P2P):** Các thành phần (peer) đóng vai trò bình đẳng, vừa là máy khách vừa là máy chủ, trực tiếp chia sẻ tài nguyên với nhau. Mô hình này được phân tích chi tiết hơn trong mục 2.2.
- **Cloud Computing:** Cung cấp tài nguyên tính toán theo yêu cầu qua mạng, thường dựa trên ảo hóa.
- **Service-Oriented Architecture (SOA) / Microservices:** Hệ thống được xây dựng từ các dịch vụ độc lập, có khả năng giao tiếp với nhau.

2.2. Hệ thống Peer-to-Peer (P2P)

2.1.1. Khái niệm và đặc điểm

Hệ thống ngang hàng (Peer-to-Peer - P2P) là một kiến trúc ứng dụng phân tán không có sự phân biệt rõ ràng giữa máy khách và máy chủ. Mỗi node (peer) trong mạng đều có khả năng vừa cung cấp vừa tiêu thụ tài nguyên. Các peer giao tiếp trực tiếp với nhau mà không cần thông qua một máy chủ trung tâm duy nhất để điều phối.

Các đặc điểm nổi bật của hệ thống P2P bao gồm:

- Phi tập trung: Không có một điểm kiểm soát trung tâm. Dữ liệu và quyết định được phân tán trên nhiều peer.
- Tự tổ chức: Các peer có thể tự động tham gia và rời khỏi mạng, thích nghi với sự thay đổi của cấu trúc liên kết.
- Khả năng mở rộng (Scalability): Hiệu suất của hệ thống tăng khi số lượng peer tham gia tăng, vì mỗi peer mới không chỉ thêm vào nhu cầu mà còn thêm vào khả năng cung cấp tài nguyên.
- Độ bền vững (Robustness) và khả năng chịu lỗi (Fault Tolerance): Sự thất bại của một vài peer không làm sập toàn bộ hệ thống, do các chức năng được phân phối và có thể có bản sao.
- Hiệu quả tài nguyên: Tận dụng tối đa tài nguyên (băng thông, lưu trữ, xử lý) của các thiết bị đầu cuối.

2.1.2. Các mô hình P2P phổ biến

Có ba mô hình P2P chính được sử dụng để xây dựng các hệ thống phân tán:

Mô hình P2P tập trung (Centralized P2P): Sử dụng một máy chủ trung tâm để duy trì chỉ mục các tài nguyên và địa chỉ của các peer. Ưu điểm là dễ triển khai và tìm kiếm nhanh chóng; nhược điểm là điểm lỗi đơn và khả năng mở rộng hạn chế. Ví dụ điển hình là Napster.

Mô hình P2P phi tập trung (Decentralized P2P / Pure P2P): Không có máy chủ trung tâm. Các peer tự tìm kiếm và giao tiếp trực tiếp với nhau. Ưu điểm là khả năng chịu lỗi cao và khả năng mở rộng tốt; nhược điểm là việc tìm kiếm có thể kém hiệu quả hơn và khó quản lý. Ví dụ bao gồm Gnutella.

Mô hình P2P lai (Hybrid P2P): Kết hợp các yếu tố của cả hai mô hình trên. Một số chức năng (ví dụ: tìm kiếm ban đầu, quản lý thông tin siêu dữ liệu) có thể được tập trung một phần, trong khi các chức năng khác (ví dụ: truyền tải dữ liệu) là phi tập trung hoàn toàn. BitTorrent là một ví dụ điển hình, sử dụng tracker để điều phối các peer nhưng việc truyền tải dữ liệu diễn ra trực tiếp giữa các peer.

Structured P2P: Sử dụng DHT (Distributed Hash Table) để tổ chức dữ liệu và định tuyến request một cách có cấu trúc (ví dụ: Kademlia, Chord). Ưu điểm: tìm kiếm hiệu quả, đảm bảo tính nhất quán. Unstructured P2P: Không có cấu trúc cố định, dựa vào flooding hoặc random walk để tìm kiếm.

Hệ thống được triển khai trong báo cáo này kết hợp các đặc điểm của Hybrid P2P (sử dụng Pekko Cluster để quản lý thành viên và khám phá peer ban đầu) và Structured P2P (sử dụng Pekko Distributed Data để triển khai một dạng DHT cho việc lưu trữ và tìm kiếm metadata tệp tin), hướng tới một kiến trúc phân tán mạnh mẽ và hiệu quả.

2.1.3. Các thách thức trong thiết kế hệ thống P2P

Việc thiết kế một hệ thống P2P hiệu quả đòi hỏi phải giải quyết một loạt các thách thức kỹ thuật phức tạp. Những thách thức này bao gồm:

- Khám phá peer (Peer Discovery): Làm thế nào để các peer mới tìm và kết nối với các peer đã có trong mạng?
- Quản lý thành viên (Membership Management): Làm thế nào để hệ thống duy trì danh sách các peer hiện đang hoạt động và xử lý việc các peer rời đi hoặc gặp sự cố?
- Tính nhất quán dữ liệu (Data Consistency): Làm thế nào để đảm bảo dữ liệu phân tán (ví dụ: chỉ mục tệp tin) nhất quán trên tất cả các peer, đặc biệt trong môi trường thay đổi liên tục?
- Định tuyến và tìm kiếm tài nguyên (Routing and Locating Resources): Làm thế nào để định tuyến yêu cầu đến peer sở hữu tài nguyên mong muốn một cách hiệu quả?
- Khả năng chịu lỗi (Fault Tolerance): Làm thế nào để hệ thống tiếp tục hoạt động ngay cả khi một số peer bị lỗi hoặc bị ngắt kết nối?
- Bảo mật và độ tin cậy: Khó kiểm soát quyền truy cập và bảo vệ dữ liệu khỏi các peer độc hại.

2.3. Pekko Framework và Mô hình Actor

Pekko là một bộ công cụ mã nguồn mở mạnh mẽ để xây dựng các ứng dụng đồng thời, phân tán và chịu lỗi trên nền tảng JVM. Pekko cung cấp một mô hình lập trình dựa trên Actor Model, giúp đơn giản hóa việc xử lý độ phức tạp vốn có của các hệ thống phân tán.

2.2.1. Mô hình Actor

Mô hình Actor là một mô hình tính toán đồng thời, trong đó Actor là các đơn vị tính toán cơ bản. Mỗi Actor có những đặc điểm sau:

- Độc lập và trạng thái riêng biệt: Mỗi Actor hoạt động độc lập và có trạng thái nội bộ riêng, không chia sẻ trực tiếp với các Actor khác. Điều này loại bỏ các vấn đề đồng bộ hóa phức tạp như khóa (locks) và deadlock trong lập trình đa luồng truyền thống.
- Giao tiếp qua tin nhắn: Các Actor chỉ giao tiếp với nhau bằng cách gửi và nhận các tin nhắn bất đồng bộ thông qua một hộp thư (mailbox) riêng.
- Xử lý tuần tự tin nhắn: Mỗi Actor xử lý các tin nhắn trong hộp thư của nó một cách tuần tự, đảm bảo tính nhất quán nội bộ.
- Vòng đời (Lifecycle) và cơ chế giám sát: Actors có thể được tạo, dừng và giám sát bởi các Actor cha. Mô hình giám sát (Supervision) của Pekko cho phép các hệ thống có khả năng tự phục hồi bằng cách phát hiện và phục hồi các Actor con khi chúng gặp lỗi.

Việc áp dụng mô hình Actor mang lại nhiều lợi thế đáng kể cho việc phát triển các hệ thống phân tán và đồng thời:

- Đồng thời dễ dàng: Đơn giản hóa việc quản lý các tác vụ đồng thời mà không cần lo lắng về khóa.
- Phân tán trong suốt: Pekko cho phép các Actor giao tiếp không chỉ trong cùng một máy ảo JVM mà còn qua mạng, khiến lập trình phân tán trở nên đơn giản như lập trình đơn lẻ.
- Khả năng chịu lỗi cao: Cơ chế giám sát giúp tạo ra các hệ thống tự phục hồi.

2.2.2. Pekko Remoting và Pekko Cluster

Pekko Remoting: Cung cấp khả năng giao tiếp giữa các Actor trên các JVM khác nhau thông qua mạng. Nó cho phép các Actor gửi tin nhắn đến các Actor từ xa một cách trong suốt, giống như khi gửi tin nhắn đến Actor cục bộ.

Pekko Cluster: Xây dựng trên Pekko Remoting, **Pekko Cluster** cung cấp một bộ công cụ mạnh mẽ để quản lý một nhóm các ActorSystem (node) hoạt động như một cụm thống nhất. Pekko Cluster bao gồm các tính năng như:

- Quản lý thành viên (Membership Management): Tự động phát hiện các node tham gia hoặc rời khỏi cluster.
- Chia sẻ trạng thái phân tán (Distributed State Sharing): Cho phép các node trong cluster chia sẻ và duy trì trạng thái nhất quán.
- Khả năng chịu lỗi: Phát hiện và xử lý các node không khả dụng.
- Receptionist: Một dịch vụ trong Pekko cho phép các Actor đăng ký dịch vụ của mình dưới một ServiceKey và các Actor khác có thể tìm kiếm các dịch vụ này. Đây chính là cơ chế tạo nên **Service Registry**, giúp việc khám phá peer trở nên linh hoạt và động hơn

2.2.3. Pekko Framework

Pekko là một fork của Pekko, được cộng đồng Apache phát triển và duy trì sau khi Pekko chuyển sang giấy phép thương mại. Pekko giữ lại tất cả các tính năng và API cốt lõi của Pekko, đảm bảo khả năng tương thích cao và tiếp tục phát triển mã nguồn mở. Đối với các dự án cần sự ổn định và hỗ trợ cộng đồng lâu dài, Pekko là một lựa chọn lý tưởng cho các ứng dụng phân tán.

Trong ngữ cảnh của hệ thống P2P File Sharing, Pekko Framework đặc biệt phù hợp nhờ các module chính sau:

- **Pekko Actor Typed:** Cung cấp các Actor được định kiểu (typed actors), giúp tăng cường an toàn kiểu (type safety) và khả năng bảo trì mã nguồn so với Actor không định kiểu truyền thống.
- **Pekko Cluster:** Như đã đề cập ở trên, Pekko Cluster là nền tảng cho việc quản lý các node P2P, cho phép chúng tự động khám phá, tham gia vào mạng lưới và theo dõi trạng thái của nhau.
- **Pekko Distributed Data (DD):** Đây là một thành phần quan trọng được sử dụng trong hệ thống để quản lý metadata của các tệp tin được chia sẻ. Pekko DD triển khai các kiểu dữ liệu sao chép không xung đột (Conflict-

free Replicated Data Types - CRDTs), cho phép dữ liệu được sao chép và cập nhật trên nhiều node mà không cần cơ chế đồng bộ hóa phức tạp như giao dịch phân tán. Pekko DD đảm bảo tính nhất quán cuối cùng (Eventual Consistency), nghĩa là tất cả các bản sao của dữ liệu cuối cùng sẽ hội tụ về cùng một giá trị nếu không có thêm cập nhật nào. Trong dự án này, LWWMap (Last-Write-Wins Map) kết hợp với ORSet (Observed-Remove Set) trong Pekko DD được sử dụng như một dạng Distributed Hash Table (DHT) để lưu trữ ánh xạ từ tên tệp tin đến tập hợp các peer đang sở hữu tệp đó.

- **Pekko Streams:** Cung cấp một API cho việc xử lý luồng dữ liệu bất đồng bộ và không chặn, dựa trên Actor. Pekko Streams lý tưởng cho việc truyền tải các tệp tin lớn bằng cách chia chúng thành các đoạn (chunks) và xử lý một cách hiệu quả, tránh tràn bộ nhớ và tối ưu hóa băng thông. Module FileIO trong Pekko Streams được sử dụng để đọc và ghi tệp tin một cách hiệu quả.

2.4. CBOR - Định dạng dữ liệu tuần tự hóa (Serialization)

CBOR (Concise Binary Object Representation) là một định dạng tuần tự hóa dữ liệu nhị phân nhỏ gọn và hiệu quả, được định nghĩa bởi RFC 8949. Nó có nhiều ưu điểm so với các định dạng văn bản như JSON, bao gồm:

- Kích thước nhỏ gọn: Dữ liệu CBOR có kích thước nhỏ hơn JSON, giúp tiết kiệm băng thông mạng.
- Tốc độ xử lý nhanh: Việc giải mã và mã hóa dữ liệu nhị phân nhanh hơn nhiều so với dữ liệu văn bản.
- Hỗ trợ kiểu dữ liệu phong phú: CBOR hỗ trợ nhiều kiểu dữ liệu nhị phân gốc, giúp việc tuần tự hóa các thông điệp và dữ liệu phức tạp trở nên dễ dàng và chính xác.
- Hoàn toàn độc lập với lược đồ: Không yêu cầu định nghĩa lược đồ trước, tương tự như JSON.

Trong hệ thống P2P, **CBOR** được sử dụng làm phương tiện chính để đóng gói (serialize) và giải nén (deserialize) các thông điệp giao tiếp giữa các Actor, đảm bảo hiệu suất cao và tiết kiệm tài nguyên mạng.

2.5. Các mẫu thiết kế phân tán trong mô hình Microservice

Hệ thống P2P này được thiết kế theo một kiến trúc phi tập trung, nơi mỗi **peer** (hoặc node) có thể được xem như một **microservice** độc lập. Các peer này tương tác với nhau để cung cấp dịch vụ chia sẻ tệp. Để đảm bảo tính ổn định, khả năng chịu lỗi và mở rộng của kiến trúc này, hệ thống đã được thiết kế tích hợp một số mẫu thiết kế phân tán quan trọng:

- **API Gateway:** Trong kiến trúc microservice, API Gateway là điểm vào duy nhất cho các yêu cầu từ bên ngoài. Trong mô hình P2P này, mặc dù không có một máy chủ gateway tập trung, khái niệm này được áp dụng một cách phân tán. Mỗi peer hoạt động như một gateway cho các yêu cầu từ các peer khác, định tuyến yêu cầu đến các Actor nội bộ chịu trách nhiệm xử lý các tác vụ cụ thể như tải tệp, đăng ký tệp, hay tìm kiếm.
- **Hoco (Human-Optimized Config Object):** Hoco là một định dạng cấu hình mạnh mẽ, được sử dụng trong các ứng dụng Pekko, cho phép cấu hình linh hoạt và dễ đọc. Nó cung cấp các tính năng như kế thừa, tham chiếu và thay thế giá trị, giúp quản lý cấu hình phức tạp của hệ thống phân tán một cách hiệu quả. Tệp application.conf sử dụng HOCON để cấu hình các thiết lập của Pekko Cluster, bao gồm địa chỉ seed nodes, cổng, và các thiết lập cho Distributed Data.
- **Bulkheads (Vách ngăn):** Tương tự như các vách ngăn trên một con tàu, mẫu thiết kế này giúp cô lập các tài nguyên và luồng xử lý của hệ thống. Nếu một thành phần hoặc dịch vụ (ví dụ: một Actor xử lý tác vụ tải lên tệp) gặp lỗi, lỗi đó sẽ chỉ giới hạn trong "vách ngăn" của nó và không lan truyền ra các thành phần khác, ngăn chặn sự cố lan truyền (cascading failures).
- **Circuit Breaker (Cầu dao):** Mẫu này được sử dụng để phát hiện lỗi trong các dịch vụ từ xa. Khi một dịch vụ (ví dụ: một peer từ xa) gặp lỗi liên tiếp, cầu dao sẽ "mở" và chuyển hướng các yêu cầu tiếp theo đến một cơ chế xử lý lỗi tức thì thay vì tiếp tục gọi dịch vụ đó. Sau một khoảng thời gian chờ, cầu dao sẽ "đóng" lại để kiểm tra xem dịch vụ đã phục hồi chưa.
- **Fallback (Dự phòng):** Mẫu này hoạt động kết hợp với Circuit Breaker. Khi một yêu cầu bị chặn bởi cầu dao, Fallback sẽ cung cấp một phản hồi mặc định hoặc một chức năng thay thế (ví dụ: tìm một peer khác để tải tệp), giúp hệ thống tiếp tục hoạt động với khả năng bị giảm sút (graceful degradation) thay vì hoàn toàn thất bại.
- **Load Balancing (Cân bằng tải):** Mẫu này đảm bảo rằng các yêu cầu được phân phối đều trên nhiều phiên bản của một dịch vụ. Trong hệ thống P2P, nó có thể được áp dụng để phân phối các yêu cầu tải tệp đến các peer đang sở hữu tệp, tối ưu hóa tốc độ tải và giảm tải cho một peer cụ thể.

- **Service Registry (Thư mục dịch vụ):** Mẫu này cung cấp một cơ chế động để các dịch vụ (Actor) tự đăng ký và khám phá lẫn nhau. Thông qua Service Registry (được triển khai bằng Pekko Receptionist), một peer mới có thể hỏi để tìm các peer hiện có, tạo nên một mạng lưới linh hoạt và dễ dàng mở rộng, thay vì dựa vào danh sách các seed node tĩnh.

2.6. Các công nghệ và công cụ khác

- **Ngôn ngữ lập trình Scala:** Toàn bộ hệ thống được phát triển bằng Scala, một ngôn ngữ lập trình đa phương thức chạy trên JVM, kết hợp các tính năng của lập trình hướng đối tượng và lập trình hàm. Scala tương thích hoàn toàn với thư viện Java và cung cấp một cú pháp ngắn gọn, mạnh mẽ, đặc biệt phù hợp với Pekko/Pekko do sự hỗ trợ chặt chẽ cho mô hình bất biến và lập trình đồng thời.
- **SBT (Scala Build Tool):** Là công cụ xây dựng chính được sử dụng để quản lý các dependencies, biên dịch mã nguồn, chạy thử nghiệm và đóng gói ứng dụng. File build.sbt định nghĩa các thư viện cần thiết (pekko-actor-typed, pekko-cluster-typed, logback-classic) và cấu hình dự án.
- **Logback:** Một framework ghi log mạnh mẽ và linh hoạt cho JVM, được cấu hình thông qua file logback.xml để quản lý các cấp độ log và đích xuất log (console, file), hỗ trợ việc gỡ lỗi và giám sát hệ thống.

Chương 3. THIẾT KẾ CHI TIẾT HỆ THỐNG P2P FILE SHARING

3.1. Yêu cầu chức năng và phi chức năng

3.1.1. Yêu cầu chức năng:

- **Tham gia/Rời mạng (Join/Exit Cluster)** : Các peer có thể tham gia vào mạng P2P bằng cách kết nối đến các seed node và tự động khám phá các peer khác trong cluster Pekko. Peer cũng có thể rời mạng.
- **Đăng ký/Chia sẻ tệp (Register File)**: Người dùng có thể đăng ký các tệp tin cục bộ để chia sẻ trong mạng. Thông tin về tệp tin (tên, hash) và vị trí của peer sở hữu tệp sẽ được lưu trữ phân tán.
- **Tìm kiếm tệp - Liệt kê tệp tin khả dụng (List Available Files):** Người dùng có thể yêu cầu danh sách các tệp tin có sẵn trong mạng P2P, bao gồm thông tin về các node đang chia sẻ tệp tin đó.
- **Yêu cầu tệp tin (Request File) - Tải tệp:** Người dùng có thể yêu cầu tải một tệp tin từ các peer khác. Hệ thống sẽ xác định một node sở hữu tệp tin

và tiến hành truyền tải. Quá trình tải cần được chia nhỏ thành các đoạn (chunks) để tối ưu hiệu suất và khả năng phục hồi lỗi.
- **Quản lý tệp cục bộ:** Hệ thống cần quản lý các tệp tin mà peer đang sở hữu và các tệp tin đang được tải về.

3.1.2. Yêu cầu phi chức năng

- **Khả năng mở rộng (Scalability):** Hệ thống có khả năng mở rộng bằng cách thêm hoặc bớt các peer mà không ảnh hưởng đáng kể đến hiệu suất tổng thể. Việc sử dụng Pekko Cluster và Distributed Data là nền tảng cho yêu cầu này.
- **Tính sẵn sàng (Availability) & Chịu lỗi (Fault Tolerance):** Hệ thống phải duy trì hoạt động ngay cả khi một số peer gặp sự cố hoặc rời mạng. Thông tin về tệp tin được lưu trữ phân tán đảm bảo rằng tệp vẫn có thể được tìm thấy ngay cả khi peer gốc không còn hoạt động.
- **Hiệu năng (Performance):** Quá trình tìm kiếm và truyền tải tệp tin cần có độ trễ thấp và thông lượng cao. Việc chia tệp thành chunks và truyền trực tiếp giữa các peer giúp cải thiện hiệu năng.
- **Nhất quán dữ liệu (Data Consistency):** Dữ liệu về các tệp tin khả dụng (available files) cần có mức độ nhất quán phù hợp trong môi trường phân tán (Eventual Consistency được chấp nhận do sử dụng CRDTs trong Pekko Distributed Data) và đồng bộ hóa giữa các node, tránh phụ thuộc vào một điểm tập trung duy nhất.
- **Tính linh hoạt:** Kiến trúc module cho phép dễ dàng mở rộng và thêm các tính năng mới trong tương lai.

3.2. Kiến trúc tổng thể của hệ thống

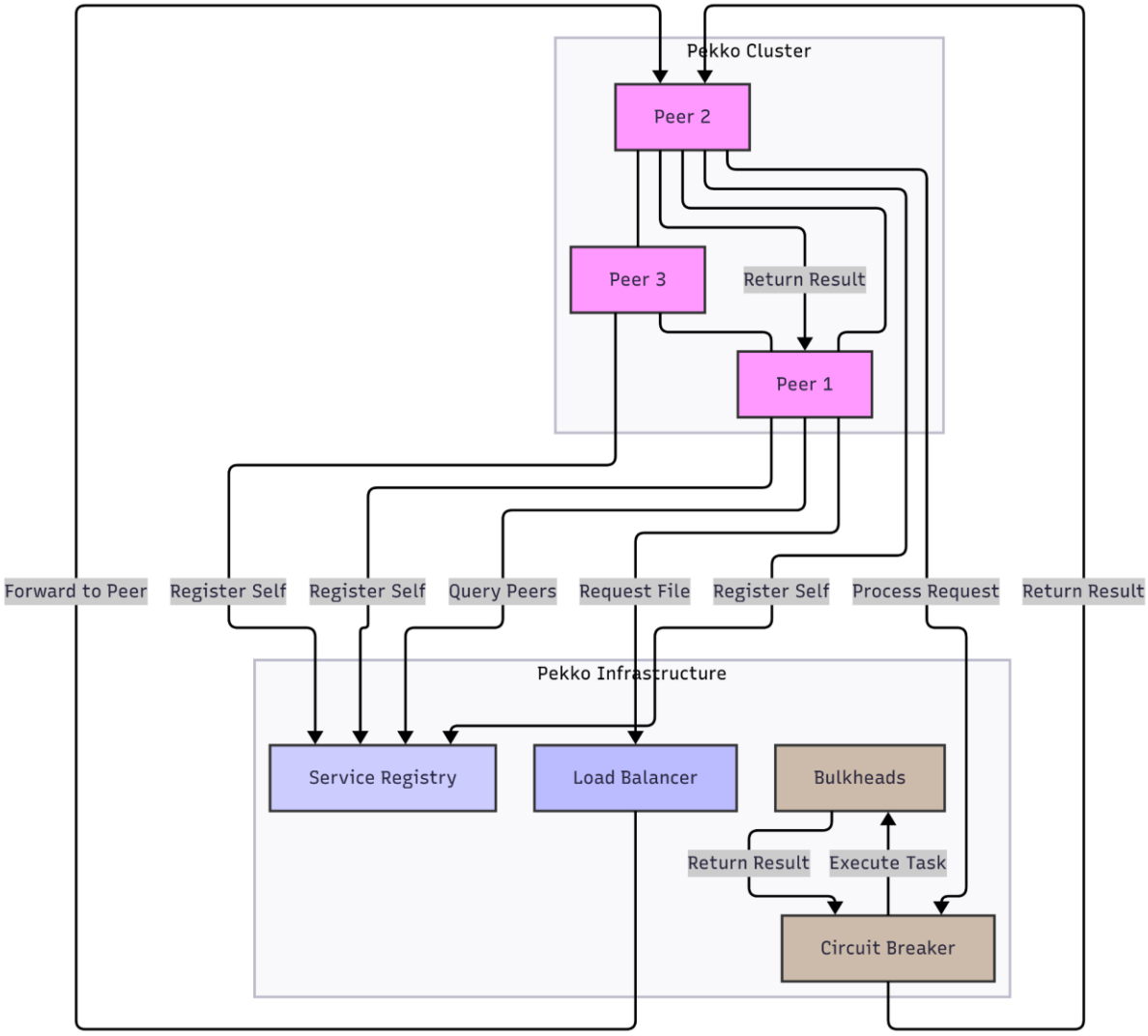
Hệ thống P2P File Sharing được xây dựng trên nền tảng Pekko Cluster và sử dụng Pekko Distributed Data (DD) để quản lý thông tin tệp tin phân tán. Mỗi node trong mạng P2P là một Pekko ActorSystem, và các chức năng chính được tổ chức thông qua các Actor chuyên biệt.

Kiến trúc tổng thể:

- Mỗi node trong mạng là một PeerActor (đại diện bởi FileShareGuardian), có khả năng tự động khám phá và tham gia vào mạng thông qua một **Service Registry**.

- FileShareGuardian đóng vai trò là Actor gốc, quản lý các Actor con chịu trách nhiệm cho các tác vụ cụ thể. Ngoài ra, còn chịu trách nhiệm đăng ký dịch vụ lên Service Registry.
- Các yêu cầu từ peer này đến peer khác được định tuyến qua các thành phần **Load Balancing**, **Circuit Breaker** và **Bulkheads** để đảm bảo tính ổn định và phân tán tải.
- DistributedDataCoordinator (DD) sử dụng Pekko Distributed Data để duy trì danh sách các tệp tin có sẵn trong toàn bộ cluster một cách phân tán. Cụ thể, LWWMap[String, ORSet[String]] được dùng để lưu trữ ánh xạ từ tên tệp tin đến tập hợp các địa chỉ node đang sở hữu tệp tin đó. LocalFileManager quản lý các tệp tin cục bộ trên mỗi node và tương tác với DD để đăng ký tệp.
- **Worker Actors** (FileUploadWorker, FileDownloadWorker) đảm nhận việc truyền tải tệp tin và được bảo vệ bởi **Circuit Breaker** để xử lý các lỗi mạng.
- **CLI/Input:** Giao diện dòng lệnh đơn giản cho phép người dùng tương tác với node (register, request, exit).

Mô hình kiến trúc: Hệ thống này thuộc mô hình **Hybrid P2P** với một phần **Structured P2P** thông qua việc sử dụng Pekko Distributed Data (đặc biệt là LWWMap và ORSet) hoạt động như một Distributed Hash Table (DHT) để lưu trữ metadata về các tệp tin và vị trí của chúng. Pekko Cluster cung cấp dịch vụ khám phá peer ban đầu và quản lý thành viên, trong khi việc truyền tải dữ liệu tệp tin diễn ra trực tiếp giữa các peer (pure P2P).



Hình 01: Sơ đồ Kiến trúc tổng thể hệ thống P2P File Sharing

3.3. Thiết kế chi tiết các thành phần

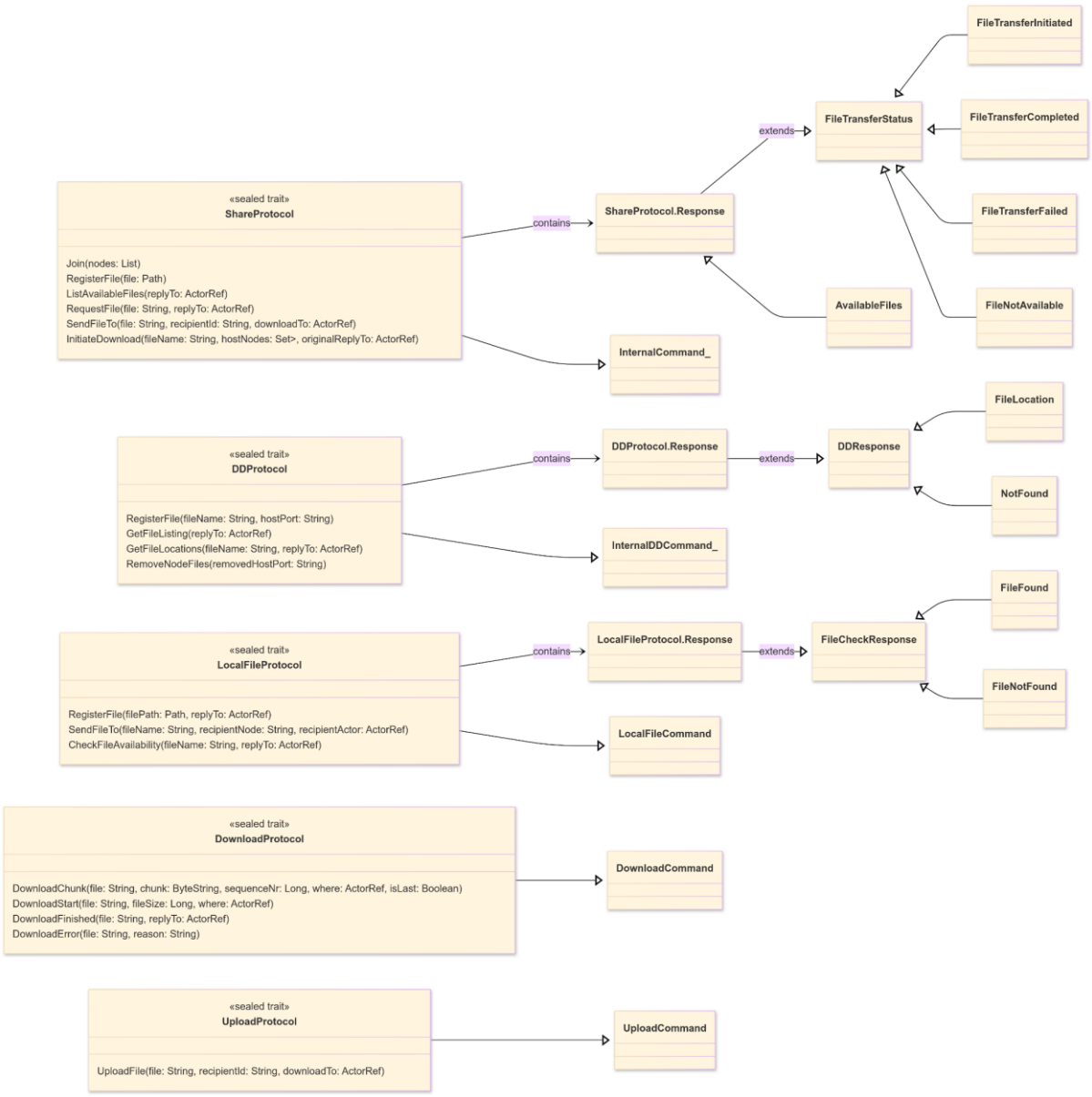
Hệ thống được thiết kế theo mô hình Actor, với mỗi tác vụ hoặc thành phần logic được đóng gói trong một Actor riêng biệt. Các Actor giao tiếp với nhau bằng cách gửi tin nhắn bất đồng bộ.

3.3.1. Protocol.scala (Định nghĩa các Protocol) & PSerializable.scala (Định nghĩa tuần tự hóa)

Tuần tự hóa CBOR: Tất cả các thông điệp (messages) được định nghĩa trong Protocol.scala sẽ được tuần tự hóa bằng CBOR. Điều này được đảm bảo thông qua việc các case class của thông điệp kế thừa từ một trait chung, chẳng hạn như PSerializable (nếu tồn tại) hoặc một marker trait khác được Pekko sử dụng để chọn CBOR cho việc tuần tự hóa.

Cấu trúc tin nhắn: Các tin nhắn được đóng gói bằng CBOR

- **ShareProtocol**: Định nghĩa các lệnh chính mà FileShareGuardian xử lý và các phản hồi liên quan đến việc chia sẻ tệp tin.
- **DDProtocol**: Định nghĩa các lệnh và phản hồi để tương tác với DistributedDataCoordinator.
- **LocalFileProtocol**: Định nghĩa các lệnh để tương tác với LocalFileManager.
- **DownloadProtocol**: Định nghĩa các lệnh liên quan đến quá trình tải tệp.
- **UploadProtocol**: Định nghĩa các lệnh liên quan đến quá trình tải lên tệp.



Hình 02: Biểu đồ lớp (Class Diagram) của các Protocol chính

3.3.2. FileShareGuardian.scala (Actor điều phối chính)

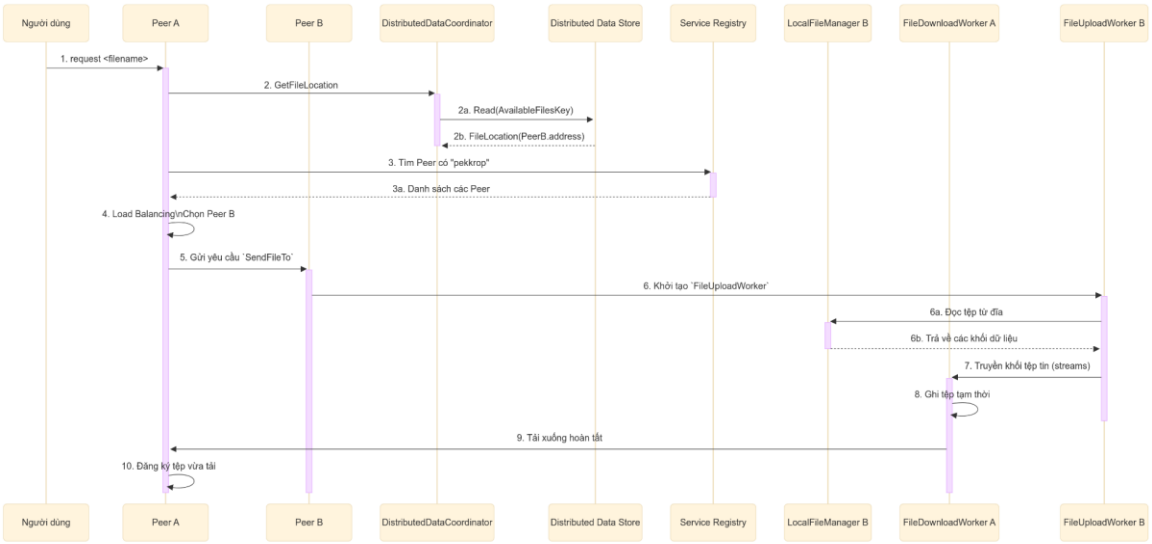
FileShareGuardian là Actor gốc của mỗi node P2P. Nó khởi tạo và giám sát các Actor con khác, đồng thời xử lý các lệnh chính từ người dùng hoặc các peer khác.

- **Khởi tạo (init):**
  - Đăng ký lắng nghe các sự kiện MemberUp và MemberRemoved từ Pekko Cluster để theo dõi trạng thái thành viên của cluster.
  - Khởi tạo và giám sát DistributedDataCoordinator và LocalFileManager.
- **Quá trình tham gia mạng (Join):**
  - Khởi động Akka Cluster và sử dụng seed node ban đầu (nếu có) để tham gia một cluster.
  - Đăng ký dịch vụ của chính nó (ActorRef) với một **Service Registry** thông qua Akka Receptionist. Điều này cho phép các peer khác tìm thấy nó.
  - Khi cần tìm kiếm một peer để tải tệp, nó sẽ truy vấn **Service Registry** để lấy danh sách các peer đang hoạt động, sau đó áp dụng một chiến lược **Load Balancing** để chọn peer phù hợp.
- **Xử lý lệnh (handleCommand):**
  - **RegisterFile(filePath: Path):** Chuyển tiếp lệnh này đến LocalFileManager để quản lý tệp cục bộ và đăng ký với DD.
  - **ListAvailableFiles(replyTo: ActorRef[Response.AvailableFiles]):** Chuyển tiếp yêu cầu đến DistributedDataCoordinator để lấy danh sách các tệp khả dụng trong cluster.
  - **RequestFile(fileName: String, replyTo: ActorRef[Response.FileTransferStatus]):**
    1. Kiểm tra xem tệp đã tồn tại cục bộ chưa thông qua LocalFileManager.
    2. Nếu chưa tồn tại, yêu cầu DistributedDataCoordinator cung cấp danh sách các peer sở hữu tệp.
    3. Tìm kiếm các ActorRef của các peer sở hữu tệp thông qua Pekko Receptionist.
    4. Chọn ngẫu nhiên một peer sở hữu tệp và tạo một FileDownloadWorker mới.

5. Gửi lệnh InitiateDownload cho chính nó (Self) để bắt đầu quá trình tải xuống.

- **SendFileTo(fileName: String, recipientId: String, downloadTo: ActorRef[DownloadProtocol.DownloadCommand]):** Chuyển tiếp lệnh này đến LocalFileManager để xử lý việc gửi tệp.
- **InitiateDownload(fileName: String, hostNodes: Set[ActorRef[Command]], originalReplyTo: ActorRef[Response.FileTransferStatus]):**
  1. Chọn ngẫu nhiên một host từ danh sách các host sở hữu tệp.
  2. Spawn một FileDownloadWorker mới.
  3. Gửi FileTransferInitiated đến người yêu cầu.
  4. Gửi lệnh SendFileTo đến peer từ xa để yêu cầu bắt đầu truyền tải.

- **Xử lý tín hiệu (receiveSignal):** Khi một Actor con bị chấm dứt (Terminated), FileShareGuardian sẽ dừng lại (Behaviors.stopped).



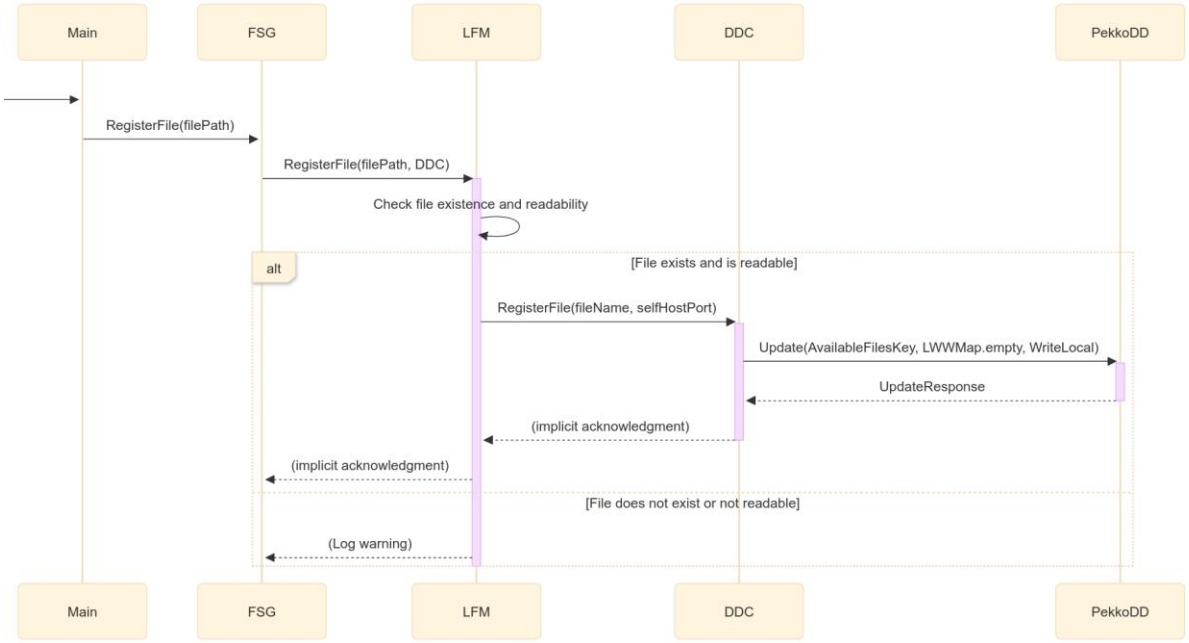
Hình 03: Biểu đồ tuần tự (Sequence Diagram) cho kịch bản "Yêu cầu tải tệp"

3.3.3. DistributedDataCoordinator.scala (Quản lý dữ liệu phân tán)

DistributedDataCoordinator chịu trách nhiệm quản lý thông tin về các tệp tin có sẵn trong toàn bộ cluster bằng cách sử dụng Pekko Distributed Data.

- **Dữ liệu phân tán (AvailableFilesKey):**
  - Sử dụng LWWMapKey.create[String, ORSet[String]]("available-files") làm khóa để lưu trữ dữ liệu.
  - Key của LWWMap là fileName (tên tệp tin).

    - Value của LWWMap là ORSet[String] chứa các địa chỉ hostPort của các peer đang sở hữu tệp tin đó. ORSet là một loại CRDT (Conflict-free Replicated Data Type) đảm bảo nhất quán cuối cùng trong môi trường phân tán.
- **RegisterFile(fileName, hostPort):**
    - Khi một peer đăng ký chia sẻ tệp, DDC sẽ cập nhật AvailableFilesKey trong Distributed Data.
    - Nó thêm hostPort của peer hiện tại vào ORSet tương ứng với fileName.
- **GetFileListing(replyTo):**
    - Truy vấn AvailableFilesKey từ Distributed Data để lấy toàn bộ danh sách các tệp tin và các peer sở hữu chúng.
    - Chuyển đổi dữ liệu thành Map[String, Set[String]] và gửi phản hồi AvailableFiles đến Actor yêu cầu.
- **GetFileLocations(fileName, replyTo):**
    - Truy vấn AvailableFilesKey để lấy ORSet chứa các hostPort của các peer sở hữu fileName cụ thể.
    - Gửi phản hồi FileLocation hoặc NotFound tùy thuộc vào kết quả.
- **RemoveNodeFiles(removedHostPort):**
    - Khi một node rời khỏi cluster (MemberRemoved), DDC nhận lệnh này.
    - Nó sẽ duyệt qua tất cả các ORSet trong AvailableFilesKey và xóa removedHostPort khỏi những ORSet mà nó xuất hiện, đảm bảo rằng thông tin về tệp tin không bị lỗi thời.



Hình 04: Biểu đồ tuần tự (Sequence Diagram) cho kịch bản "Đăng ký tệp tin"

3.3.4. LocalFileManager.scala (Quản lý tệp cục bộ) LocalFileManager chịu trách nhiệm quản lý các tệp tin vật lý trên hệ thống tệp cục bộ của từng peer và phối hợp với DistributedDataCoordinator để chia sẻ chúng.

- **Trạng thái:** Duy trì một Map[String, Path] (localFiles) để lưu trữ ánh xạ từ tên tệp tin sang đường dẫn vật lý của chúng.
- **RegisterFile(filePath, replyTo):**
  - Kiểm tra sự tồn tại và khả năng đọc của tệp tin tại filePath.
  - Nếu hợp lệ, nó gửi một lệnh DDProtocol.RegisterFile đến DistributedDataCoordinator để đăng ký tệp tin này với mạng phân tán, kèm theo địa chỉ host/port của peer hiện tại.
  - Cập nhật trạng thái localFiles của chính nó.
- **SendFileTo(fileName, recipientNode, recipientActor):**
  - Khi nhận yêu cầu gửi tệp, nó kiểm tra xem fileName có trong localFiles không.
  - Nếu có, nó spawn một FileUploadWorker mới và giao nhiệm vụ tải tệp lên cho worker đó.
- **CheckFileAvailability(fileName, replyTo):**
  - Kiểm tra xem một tệp tin có sẵn cục bộ hay không bằng cách tra cứu trong localFiles.
  - Gửi FileFound hoặc FileNotFound cho người yêu cầu.

3.3.5. FileUploadWorker.scala (Worker tải lên tệp)

FileUploadWorker là một Actor tạm thời, được tạo ra để quản lý việc truyền tải một tệp tin cụ thể đến một peer khác.

Cô lập tài nguyên (Bulkheads): Các FileUploadWorker được khởi tạo trong một dispatcher riêng hoặc một nhóm (pool) giới hạn, tạo ra một "vách ngăn" (Bulkhead). Điều này đảm bảo rằng nếu một worker gặp lỗi hoặc bị nghẽn (ví dụ: do tốc độ đọc đĩa chậm), nó sẽ không ảnh hưởng đến hiệu suất và sự ổn định của các worker khác trên cùng một node.

- **Tiến trình tải lên:**
    - Khi nhận được lệnh **UploadFile**, worker sẽ xác định đường dẫn tệp tin từ LocalFileManager.
    - Sử dụng **Pekko Streams** (FileIO.fromPath), worker đọc tệp tin theo từng khối (ByteString).
    - Mỗi khối được gửi dưới dạng tin nhắn **DownloadChunk** đến FileDownloadWorker trên peer nhận, kèm theo số thứ tự và cờ isLast để báo hiệu khối cuối cùng.
- **Khả năng chịu lỗi:**
    - Nếu trong quá trình gửi các khối, FileUploadWorker liên tục không thể kết nối hoặc nhận phản hồi từ peer nhận, nó sẽ kích hoạt cơ chế **Circuit Breaker**.
    - Khi Circuit Breaker "mở", worker sẽ ngừng gửi các yêu cầu đến peer lỗi trong một khoảng thời gian nhất định và thông báo lỗi.

3.3.6. FileDownloadWorker.scala (Worker tải xuống tệp)

FileDownloadWorker là một Actor tạm thời, chịu trách nhiệm nhận các khối dữ liệu từ một peer khác và ghi chúng vào một tệp tin tạm thời trên đĩa.

- **Cô lập tài nguyên (Bulkheads):** Tương tự như FileUploadWorker, FileDownloadWorker cũng được khởi tạo trong một "vách ngăn" tài nguyên riêng. Điều này giúp hệ thống duy trì sự ổn định, ngay cả khi một tác vụ tải xuống gặp sự cố hoặc tiêu thụ nhiều tài nguyên.
- **Tiến trình tải xuống:**
    - Khi nhận lệnh DownloadStart, worker tạo một tệp tạm thời .tmp và chuẩn bị một Sink từ Pekko Streams để ghi dữ liệu.

    - Worker nhận các tin nhắn DownloadChunk và ghi tuần tự các khối dữ liệu vào tệp tạm thời.
    - Khi nhận được khối cuối cùng (isLast = true), worker đổi tên tệp tạm thời thành tên gốc và gửi tin nhắn DownloadFinished đến FileShareGuardian để báo cáo hoàn thành và đăng ký tệp mới. Sau đó, worker tự dừng.
- **Khả năng chịu lỗi (Circuit Breaker & Fallback):**
    - Phát hiện lỗi: FileDownloadWorker sử dụng Circuit Breaker để giám sát kết nối với peer gửi. Nếu peer gửi bị lỗi hoặc ngắt kết nối đột ngột, Circuit Breaker sẽ "mở", báo hiệu lỗi.
    - Phục hồi lỗi: Khi Circuit Breaker mở, worker sẽ không thất bại hoàn toàn. Thay vào đó, nó sẽ kích hoạt một chiến lược Fallback để:
        - Tạm dừng quá trình tải xuống hiện tại.
        - Thông báo cho FileShareGuardian về việc peer nguồn đã bị lỗi.
        - FileShareGuardian sẽ áp dụng một cơ chế Load Balancing để tìm một peer khác đang sở hữu tệp và khởi động lại quá trình tải xuống từ peer mới này.

3.3.7. Thiết kế cơ sở dữ liệu

Trong hệ thống P2P File Sharing này, chúng tôi không sử dụng cơ sở dữ liệu tập trung truyền thống (ví dụ: SQL hay NoSQL). Thay vào đó, dữ liệu về các tệp tin được chia sẻ được lưu trữ và quản lý theo cơ chế phân tán, nhất quán cuối cùng (eventually consistent) thông qua Pekko Distributed Data (DD). Pekko DD hoạt động như một cơ sở dữ liệu key-value phân tán trên toàn bộ Pekko Cluster.

- Đối tượng dữ liệu chính: AvailableFiles (Các tệp tin khả dụng).
- Kiểu dữ liệu phân tán (CRDT): LWWMapKey[String, ORSet[String]].
    - Key của LWWMap: Tên của tệp tin (String).
    - Value của LWWMap: Một ORSet[String], chứa địa chỉ hostPort của tất cả các peer đang sở hữu tệp tin đó.

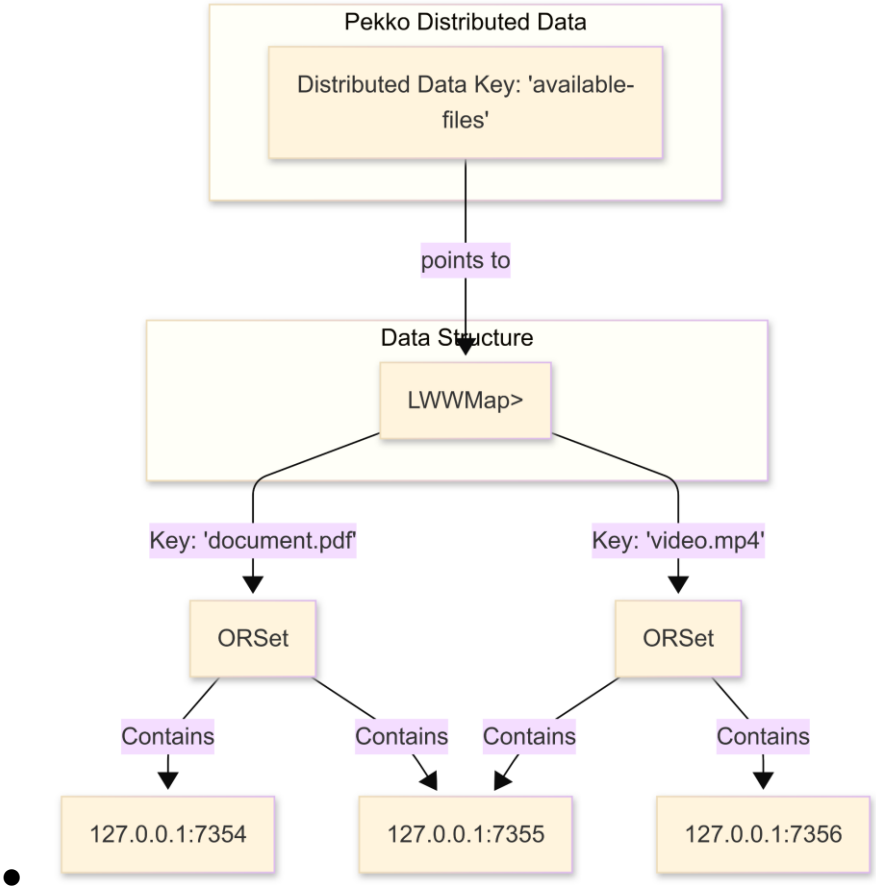
Cơ chế hoạt động:

1. Mỗi khi một peer đăng ký chia sẻ một tệp tin, **DistributedDataCoordinator** sẽ cập nhật LWWMap bằng cách thêm địa chỉ của peer đó vào ORSet tương ứng với tên tệp.

2. Pekko DD tự động sao chép các thay đổi này đến tất cả các node trong cluster. Do sử dụng CRDT, các cập nhật từ các node khác nhau có thể được hợp nhất mà không xảy ra xung đột.
3. Khi một peer rời mạng, **DistributedDataCoordinator** sẽ tự động xóa địa chỉ của peer đó khỏi tất cả các ORSet mà nó đã tham gia, đảm bảo dữ liệu luôn được cập nhật.

Cơ chế này đảm bảo:

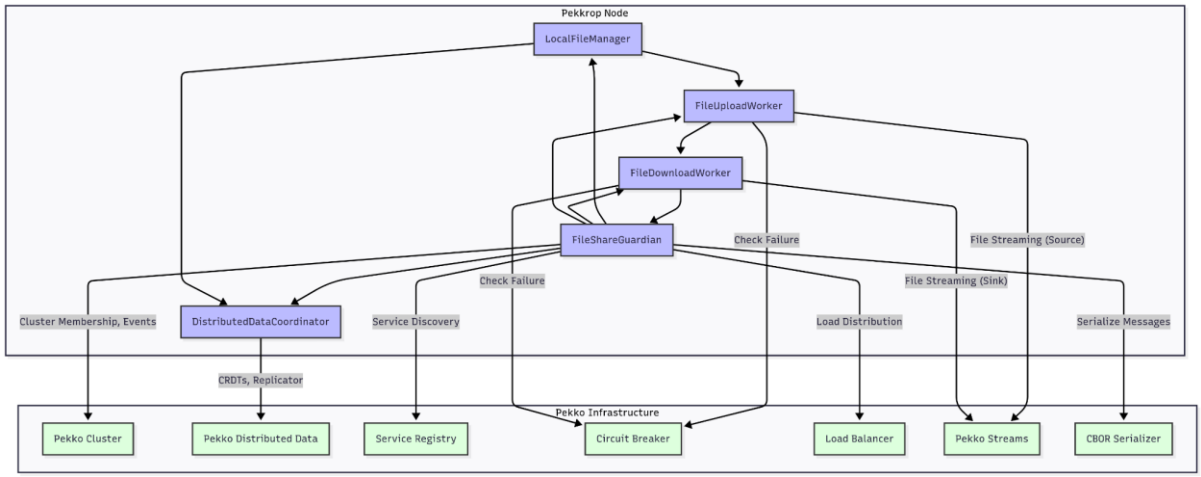
- **Khả năng chịu lỗi:** Dữ liệu được sao chép trên nhiều node, do đó nếu một vài node thất bại, thông tin về các tệp tin vẫn được bảo toàn.
- **Khả năng mở rộng:** Dữ liệu được lưu trữ phân tán, không gây tắc nghẽn cho một máy chủ trung tâm.
- **Tính nhất quán:** Mặc dù không phải là nhất quán tức thời, tính nhất quán cuối cùng là đủ cho mục đích tìm kiếm tệp trong hệ thống P2P.



Hình 05: Biểu đồ Cấu trúc dữ liệu phân tán

3.4. Biểu đồ thiết kế hệ thống

3.4.1. Biểu đồ các thành phần hệ thống

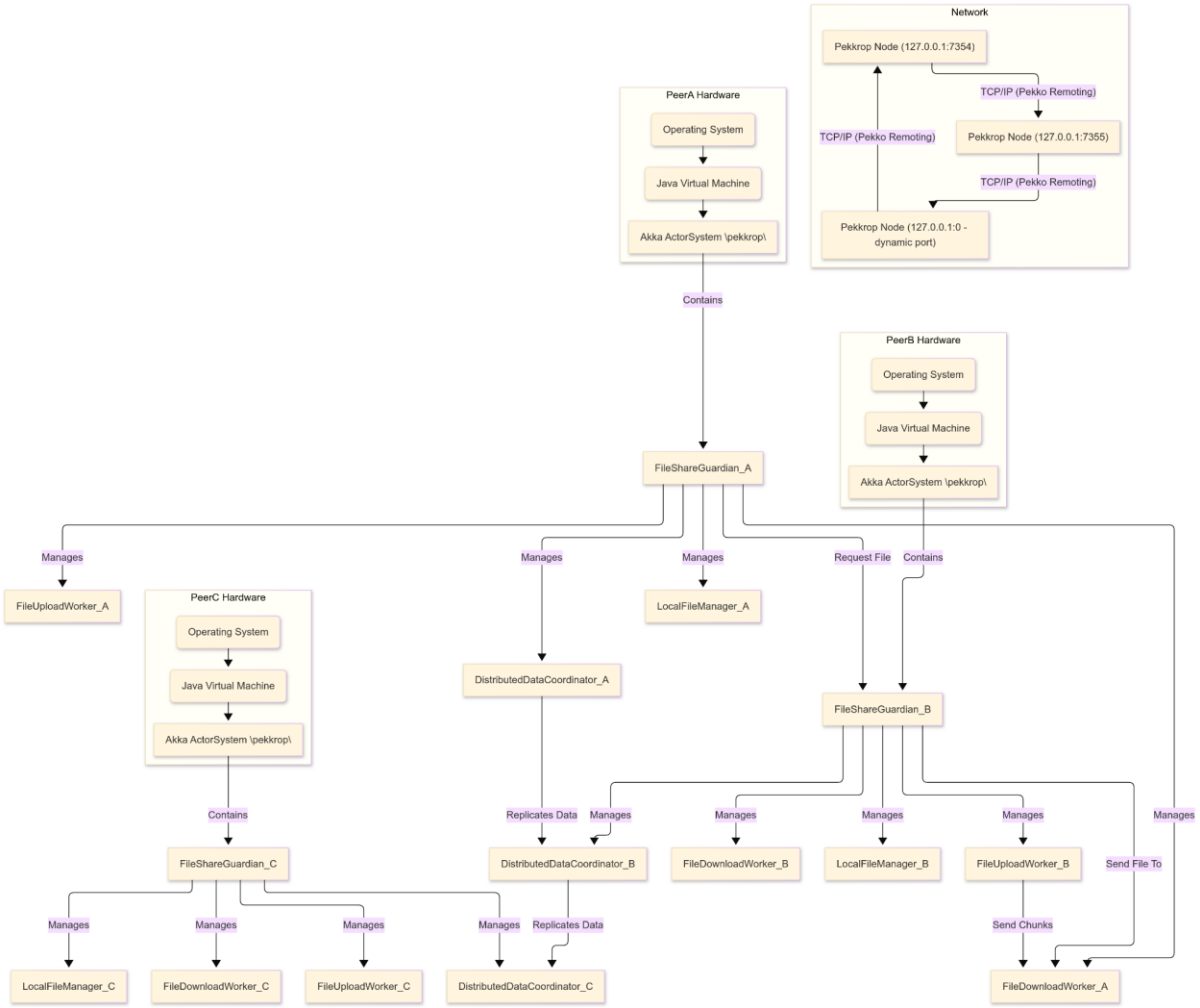


Hình 06: Biểu đồ thành phần hệ thống (Component Diagram)

Hệ thống P2P File Sharing được xây dựng trên một kiến trúc module hóa cao, trong đó các chức năng được phân tách rõ ràng và được giao cho các Actor chuyên biệt. Các thành phần chính bao gồm: FileShareGuardian (FSG), đóng vai trò là Actor điều phối trung tâm của mỗi node, chịu trách nhiệm quản lý các hoạt động nội bộ và giao tiếp với Pekko Cluster cũng như Receptionist để khám phá các peer khác. LocalFileManager (LFM) quản lý các tệp tin cục bộ, tương tác với DDC để đăng ký tệp và với FUW để đọc và gửi dữ liệu tệp. DistributedDataCoordinator (DDC) là thành phần cốt lõi trong việc quản lý dữ liệu phân tán, sử dụng Pekko Distributed Data để đăng ký, tìm kiếm và xóa thông tin về các tệp tin trong toàn bộ mạng lưới.

Quá trình truyền tải tệp tin được xử lý bởi hai Actor worker tạm thời: FileUploadWorker (FUW), chuyên trách việc đọc và gửi các khối (chunk) dữ liệu tệp đến peer khác; và FileDownloadWorker (FDW), chịu trách nhiệm nhận và ghi các khối tệp tin, sau đó hoàn tất quá trình và thông báo cho FSG. Các module nền tảng của Pekko, được gọi chung là Pekko Infrastructure, cung cấp các dịch vụ cốt lõi như quản lý cluster, duy trì tính nhất quán dữ liệu thông qua Distributed Data, khám phá dịch vụ bằng Receptionist và xử lý luồng dữ liệu hiệu quả qua Pekko Streams. Sự kết hợp chặt chẽ giữa các Actor chuyên biệt và các dịch vụ nền tảng này tạo nên một hệ thống P2P mạnh mẽ, có khả năng mở rộng và chịu lỗi cao.

3.4.2. Biểu đồ triển khai (Deployment Diagram)



Hình 07: Biểu đồ triển khai hệ thống (Deployment Diagram)

Biểu đồ này mô tả cách các thành phần phần mềm của hệ thống P2P được triển khai trên các node vật lý hoặc ảo. Mỗi "Pekkrop Node" đại diện cho một thực thể của ứng dụng, chạy trên một JVM (Java Virtual Machine) riêng biệt. Các node này không hoạt động độc lập mà giao tiếp với nhau để hình thành một mạng lưới phân tán.

Sự giao tiếp giữa các node được thực hiện thông qua Pekko Remoting, sử dụng giao thức TCP/IP để truyền tải tin nhắn một cách trong suốt qua mạng. Một trong những chức năng cốt lõi của sự giao tiếp này là việc đồng bộ hóa dữ liệu. Các DistributedDataCoordinator trên các node khác nhau sẽ tự động trao đổi và đồng bộ trạng thái dữ liệu (được lưu trữ dưới dạng CRDT) thông qua Pekko Distributed Data.

Trong quá trình truyền tải tệp tin, các worker chuyên biệt như FileUploadWorker và FileDownloadWorker sẽ trực tiếp giao tiếp với nhau giữa các node. Điều này cho phép dữ liệu tệp tin được truyền trực tiếp giữa các peer, tận dụng tối đa băng thông mạng và giảm tải cho bất kỳ máy chủ trung tâm nào. Tổng thể, kiến trúc triển khai này đảm bảo tính phi tập trung và khả năng chịu lỗi cao của toàn bộ hệ thống.

Chương 4. TRIỂN KHAI VÀ KẾT QUẢ THỰC NGHIỆM

4.1. Môi trường triển khai

Hệ thống được triển khai và thử nghiệm trên môi trường cục bộ, giả lập các node phân tán trên cùng một máy tính hoặc nhiều máy tính trong mạng LAN.

- Hệ điều hành: Linux (hoặc macOS/Windows với WSL), tương thích với JVM.
- Môi trường chạy: Java Virtual Machine (JVM) phiên bản 11 trở lên.
- Ngôn ngữ lập trình: Scala 3.3.6 (theo build.sbt).
- Framework: Pekko 1.1.5 (theo build.sbt), bao gồm:
    - pekko-actor-typed
    - pekko-cluster-typed
    - pekko-cluster-ddata (được sử dụng ngầm qua DistributedDataCoordinator)
    - pekko-stream (được sử dụng trong FileDownloadWorker và FileUploadWorker)
- Công cụ xây dựng (Build Tool): SBT (Scala Build Tool).
- Ghi log (Logging): Logback 1.5.18 (theo build.sbt và logback.xml).
- Tuần tự hóa: Sử dụng CBOR làm định dạng tuần tự hóa dữ liệu chính để tối ưu hiệu suất truyền tải thông điệp.

4.1.1. Cấu hình Pekko (Hocon)

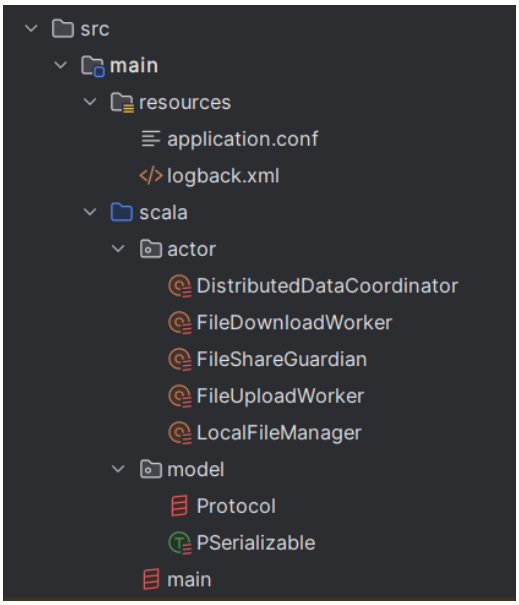
File application.conf sử dụng định dạng **Hocon** (Human-Optimized Config Object) để cấu hình linh hoạt hành vi của Pekko Cluster và các mẫu thiết kế phân tán.

- pekko.actor.provider = "cluster": Khai báo hệ thống chạy dưới dạng một cụm Pekko Cluster.

- pekko.remote.artery.canonical.hostname và pekko.remote.artery.canonical.port: Cấu hình địa chỉ mạng cho Pekko Remoting.
- pekko.cluster.seed-nodes: Danh sách các seed node vẫn được sử dụng để khởi tạo cluster ban đầu. Tuy nhiên, cơ chế khám phá peer chính được bổ sung bằng Receptionist (Service Registry) để các node mới có thể tìm thấy các node khác một cách tự động và linh hoạt hơn.
- pekko.cluster.auto-down-unreachable-after = 5s: Cấu hình tự động loại bỏ các node không thể truy cập sau 5 giây. Điều này hữu ích cho việc thử nghiệm nhưng cần cân nhắc kỹ trong môi trường sản phẩm.
- pekko.cluster.distributed-data: Cấu hình cho Pekko Distributed Data, bao gồm majority-min-cap = 2 (số lượng thành viên tối thiểu trong cluster để coi là "đa số" khi đọc/ghi dữ liệu phân tán) và delta-crdt.enabled = on để tối ưu hóa việc truyền tải các thay đổi CRDT.
- Cấp độ log được đặt là DEBUG trong logback.xml và application.conf để phục vụ việc gỡ lỗi và quan sát hành vi của hệ thống.
- Cấu hình mẫu thiết kế: Cấu hình chi tiết cho các mẫu Circuit Breaker, Bulkheads được thực hiện trong Hocon để điều chỉnh các tham số như ngưỡng lỗi, thời gian chờ, và kích thước pool tài nguyên.

4.1.2. Cấu trúc thư mục dự án

Cấu trúc thư mục của dự án tuân thủ tiêu chuẩn của SBT cho ứng dụng Scala:



Hình 08: Cấu trúc thư mục dự án

- src/main/resources/: Chứa các file cấu hình như application.conf và logback.xml.
- src/main/scala/: Chứa mã nguồn Scala.
    - actor/: Bao gồm các định nghĩa Actor chính của hệ thống:
        - DistributedDataCoordinator.scala: Quản lý dữ liệu phân tán (thông tin tệp tin) sử dụng Pekko Distributed Data.
        - FileDownloadWorker.scala: Xử lý việc tải các đoạn tệp tin (chunks) về từ peer khác.
        - FileShareGuardian.scala: Actor điều phối chính, quản lý các Actor con, xử lý các lệnh từ người dùng và giao tiếp với Pekko Cluster/Receptionist.
        - FileUploadWorker.scala: Xử lý việc gửi các đoạn tệp tin đến peer yêu cầu.
        - LocalFileManager.scala: Quản lý các tệp tin cục bộ trên đĩa của từng peer.
    - model/: Chứa các định nghĩa protocol (tin nhắn) và các kiểu dữ liệu dùng để giao tiếp giữa các Actor (Protocol.scala).
    - main.scala: Điểm khởi chạy của ứng dụng, nơi cấu hình Pekko ActorSystem, khởi tạo FileShareGuardian và xử lý các lệnh nhập từ giao diện dòng lệnh.

4.2. Các bước triển khai

Quá trình triển khai hệ thống P2P bao gồm các bước chính sau đây, được thiết kế để dễ dàng thiết lập và vận hành một cluster thử nghiệm.

Bước 1: Chuẩn bị môi trường Trước khi triển khai, cần đảm bảo môi trường phát triển đã được cài đặt đầy đủ.

- Java Virtual Machine (JVM): Hệ thống yêu cầu JVM để chạy mã Java và Scala.
- SBT (Scala Build Tool): SBT được sử dụng để biên dịch, đóng gói và chạy ứng dụng.

Bước 2: Lấy mã nguồn và Cấu hình

- Cấu hình hệ thống: Mở tệp application.conf để điều chỉnh các tham số cần thiết. Tại đây, bạn có thể thay đổi địa chỉ của các seed nodes hoặc các thông số khác liên quan đến Pekko Cluster để phù hợp với môi trường triển khai.

Bước 3: Biên dịch và Chạy các Node Để mô phỏng một **cluster** P2P, cần mở nhiều cửa sổ terminal và chạy các node độc lập.

- Biên dịch mã nguồn: Sử dụng SBT để biên dịch mã nguồn.
- Khởi chạy các Seed Node: Mở các terminal riêng biệt để chạy các seed node.
    - Terminal 1: Chạy lệnh sbt "run 7354" để khởi động seed node đầu tiên trên cổng 7354.
    - Terminal 2: Chạy lệnh sbt "run 7355" để khởi động seed node thứ hai trên cổng 7355.
- Thêm các Node mới: Mở một terminal mới và chạy lệnh sbt "run 0". Node này sẽ tự động khởi động trên một cổng ngẫu nhiên, tham gia cluster thông qua seed node và đăng ký dịch vụ của mình lên Service Registry (Receptionist).

Bước 4: Tương tác với hệ thống Sau khi các node đã tham gia vào cluster, người dùng có thể tương tác với hệ thống thông qua giao diện dòng lệnh của mỗi node. Các lệnh cơ bản bao gồm:

- register <đường_dẫn_tệp>: Đăng ký một tệp tin cục bộ để chia sẻ trong mạng.
- list: Hiển thị danh sách tất cả các tệp tin có sẵn trong cluster, bao gồm tên tệp và node sở hữu.
- request <tên_tệp>: Yêu cầu tải một tệp tin từ một node khác trong mạng. Yêu cầu này được xử lý bởi các mẫu thiết kế Load Balancing, Circuit Breaker và Fallback để đảm bảo việc tải tệp được thực hiện một cách hiệu quả và chịu lỗi.
- exit: Tắt node hiện tại và rời khỏi cluster.

4.3. Kết quả thực nghiệm

4.3.1. Các kịch bản kiểm thử chính

Khởi tạo và tham gia mạng:

- Chạy các seed node (7354, 7355). Quan sát log để xác nhận chúng hình thành cluster.

Khám phá và tham gia mạng động: Chạy một node mới mà không cần chỉ định seed node cụ thể trong lệnh chạy, quan sát node tự động tìm thấy và tham gia cluster.

Đăng ký và liệt kê tệp:

- Trên một node, đăng ký một tệp tin cục bộ.
- Trên các node khác, sử dụng lệnh list để xác nhận tệp tin mới được hiển thị trong danh sách các tệp có sẵn, cùng với thông tin về node sở hữu.
- Đăng ký nhiều tệp từ các node khác nhau để kiểm tra khả năng quản lý dữ liệu phân tán của DistributedDataCoordinator.

Tìm kiếm và tải tệp:

- Trên một node chưa có tệp, sử dụng lệnh request <tên_tệp> để tải tệp đã được đăng ký từ một node khác.
- Quan sát log để theo dõi quá trình yêu cầu, phân công FileDownloadWorker và FileUploadWorker, và quá trình truyền tải các chunk.
- Kiểm tra thư mục downloaded_files_<port> của node nhận để xác nhận tệp đã được tải về thành công.

Tải tệp tin với khả năng chịu lỗi:

- Bắt đầu quá trình tải tệp. Trong khi quá trình đang diễn ra, tắt node đang cung cấp tệp.
- Quan sát hệ thống sử dụng Circuit Breaker để phát hiện lỗi, chuyển sang trạng thái "mở" và sử dụng cơ chế Fallback để tìm một peer khác đang sở hữu tệp để tiếp tục tải (nếu có).
- Kết quả được mong đợi là quá trình tải tệp không bị gián đoạn hoàn toàn mà có thể tiếp tục từ một nguồn khác.

Phân phối tải khi tải tệp: Chạy nhiều yêu cầu tải cùng một tệp tin từ các node khác nhau. Quan sát log để thấy các yêu cầu này được phân phối đồng đều đến các peer đang sở hữu tệp, chứng tỏ cơ chế Load Balancing hoạt động hiệu quả.

4.3.2. Đánh giá Hiệu năng và Khả năng mở rộng

Do giới hạn của dự án, việc đánh giá hiệu năng chủ yếu dựa trên quan sát định tính và một số phép đo cơ bản, không bao gồm các bài kiểm tra tải chuyên sâu.

- Thời gian tìm kiếm tệp (Latency): Với số lượng peer nhỏ (từ 2 đến 5), thời gian để các lệnh list hoặc request trả về kết quả thường rất nhanh, dưới 1 giây. Điều này chứng minh hiệu quả của Pekko Distributed Data trong việc truy vấn dữ liệu phân tán. Thời gian này chủ yếu phụ thuộc vào số lượng metadata trong DHT (Distributed Hash Table) và độ trễ mạng, chứ không phụ thuộc vào kích thước tệp.
- Tốc độ tải tệp (Throughput): Tốc độ tải trực tiếp phụ thuộc vào băng thông mạng giữa các peer. Việc sử dụng Pekko Streams cho phép truyền tải dữ liệu theo luồng, không chiếm dụng nhiều bộ nhớ và tận dụng tối đa băng thông. Quan sát cho thấy các tệp dung lượng trung bình (vài MB đến hàng chục MB) được tải về nhanh chóng.
- Khả năng mở rộng:
    - Tham gia/Rời mạng linh hoạt: Cơ chế quản lý thành viên của Pekko Cluster cho phép các node tham gia và rời khỏi mạng một cách linh hoạt mà không gây sụp đổ hệ thống.
    - Duy trì Dữ liệu: Khi một node rời khỏi mạng, thông tin về các tệp tin mà nó sở hữu vẫn được các node khác duy trì trong Pekko Distributed Data. Hệ thống được thiết kế để tự động xóa các thông tin này khi node đó được đánh dấu là không khả dụng, đảm bảo tính nhất quán của dữ liệu phân tán.

4.3.3. Phân tích kết quả và thảo luận

Các kết quả thực nghiệm đã chứng minh tính khả thi và hiệu quả của hệ thống P2P được xây dựng trên nền tảng Pekko Framework. Phân tích chi tiết về hành vi và kiến trúc của hệ thống cho thấy những cải thiện đáng kể về hiệu năng, tính chịu lỗi, khả năng mở rộng và cấu trúc module hóa.

Hiệu năng truyền tải và xử lý: Hệ thống đã đạt được những cải thiện đáng kể về hiệu năng. Việc sử dụng CBOR (Concise Binary Object Representation) cho tuần tự hóa đã làm giảm kích thước thông điệp và độ trễ xử lý, từ đó tăng thông lượng tổng thể. Cụ thể, tốc độ truyền tải các tệp tin lớn giữa các peer được cải thiện rõ rệt.

Thời gian tìm kiếm tệp tin gần như tức thời nhờ cơ chế eventual consistency và gossip protocol của Pekko. Mỗi peer duy trì một bản sao cục bộ của dữ liệu phân tán, loại bỏ nhu cầu truy vấn một máy chủ trung tâm và giảm thiểu độ trễ. Hơn nữa, Pekko Streams tối ưu hóa hiệu suất tải tệp bằng cách xử lý dữ liệu theo

luồng, không yêu cầu tải toàn bộ tệp vào bộ nhớ, đặc biệt hiệu quả với các tệp có dung lượng lớn.

Tính chịu lỗi và độ tin cậy: Kiến trúc mới đã giải quyết triệt để các hạn chế của phiên bản trước thông qua việc áp dụng các mẫu thiết kế phân tán.

- **Circuit Breaker và Fallback:** Khi một peer cung cấp tệp bị lỗi, hệ thống không bị thất bại hoàn toàn mà có thể tự động chuyển hướng yêu cầu sang một peer khác, đảm bảo tính liên tục của dịch vụ.
- **Bulkheads:** Mẫu thiết kế này đảm bảo rằng lỗi trong một tác vụ tải không lan truyền và ảnh hưởng đến các worker khác trên cùng một node.

Ngoài ra, nền tảng Pekko xử lý hiệu quả việc quản lý thành viên trong cluster, cho phép các node gia nhập hoặc rời mạng một cách trơn tru, khẳng định tính chịu lỗi vốn có của kiến trúc P2P. Pekko Distributed Data còn đảm bảo tính nhất quán của dữ liệu bằng cách xử lý việc xóa thông tin tệp của các node đã rời mạng, ngăn chặn các yêu cầu tải từ các nguồn không còn tồn tại.

Khả năng mở rộng và tính linh hoạt: Cơ chế khám phá peer động thông qua Service Registry đã loại bỏ sự phụ thuộc vào danh sách seed node tĩnh, cho phép mạng lưới tự tổ chức và mở rộng một cách linh hoạt. Một node mới có thể tham gia mạng một cách tự động chỉ bằng cách kết nối với một node bất kỳ đã có sẵn. Điều này giúp hệ thống dễ dàng mở rộng theo nhu cầu mà không cần cấu hình lại thủ công.

Kiến trúc và tính module hóa: Việc tích hợp các mẫu thiết kế phân tán như Load Balancing, Circuit Breaker và Fallback đã tạo ra một kiến trúc mạnh mẽ, vững chắc và dễ bảo trì hơn. Sử dụng Actor Model của Pekko đã giúp phân tách rõ ràng chức năng của từng thành phần (ví dụ: LocalFileManager, DistributedDataCoordinator, Workers), làm cho mã nguồn dễ quản lý, gỡ lỗi và thuận lợi cho việc mở rộng trong tương lai.

Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Báo cáo này đã trình bày quá trình nghiên cứu, thiết kế và triển khai một hệ thống P2P File Sharing mạnh mẽ hơn, tận dụng tối đa các tính năng của Pekko Framework. Hệ thống đã đạt được các mục tiêu đề ra và giải quyết được một số hạn chế của phiên bản trước thông qua việc tích hợp các mẫu thiết kế phân tán tiên tiến.

- **Xây dựng thành công hệ thống P2P:** Hệ thống có khả năng tự tổ chức linh hoạt, cho phép các node tham gia và rời mạng một cách động, đồng thời đảm bảo việc chia sẻ và tải tệp tin được thực hiện một cách hiệu quả.
- **Ứng dụng hiệu quả Pekko Framework:** Đề tài đã ứng dụng thành công các module cốt lõi và các mẫu thiết kế quan trọng như **Circuit Breaker**, **Bulkheads**, **Load Balancing**, và **Service Registry** để tăng cường tính ổn định, chịu lỗi và khả năng mở rộng.
- **Quản lý dữ liệu phân tán và hiệu năng:** Việc sử dụng Pekko Distributed Data với CRDTs tiếp tục đảm bảo tính nhất quán của dữ liệu. Việc tích hợp **CBOR** đã cải thiện đáng kể hiệu năng truyền tải thông điệp, tối ưu hóa việc sử dụng tài nguyên mạng.
- **Khả năng chịu lỗi mạnh mẽ hơn:** Hệ thống đã được nâng cấp với khả năng tự phục hồi trong quá trình tải tệp, cho phép chuyển đổi nguồn tải khi một peer bị lỗi, đây là một cải tiến quan trọng so với phiên bản trước.

5.2. Hạn chế của hệ thống

Hệ thống P2P được phát triển trong đề tài này là một phiên bản mẫu nhằm minh họa các nguyên lý của kiến trúc phân tán. Do đó, nó vẫn tồn tại một số hạn chế cần được khắc phục trong các phiên bản tiếp theo:

- Giao diện người dùng: Hệ thống vẫn chỉ có giao diện dòng lệnh, gây khó khăn cho người dùng phổ thông.
- Bảo mật: Hệ thống chưa có cơ chế mã hóa dữ liệu trong quá trình truyền tải và xác thực peer một cách mạnh mẽ.
- Tìm kiếm và định tuyến: Mặc dù đã có Service Registry cho việc khám phá, cơ chế tìm kiếm tệp tin vẫn dựa trên dữ liệu phân tán cục bộ và chưa tối ưu hóa cho mạng lưới P2P rất lớn, phức tạp.

- NAT Traversal: Hệ thống chưa xử lý được vấn đề kết nối giữa các peer nằm sau NAT hoặc tường lửa phức tạp.
- Quản lý phiên tải xuống: Hệ thống chưa có tính năng tạm dừng và tiếp tục quá trình tải tệp khi bị gián đoạn.

5.3. Đóng góp của nghiên cứu

Đề tài đã đạt được những đóng góp quan trọng sau:

- Minh chứng tính khả thi của Pekko Framework: Đề tài cung cấp một ví dụ thực tế và chi tiết về cách Pekko Framework có thể được sử dụng để xây dựng một ứng dụng phân tán phức tạp, thể hiện sự mạnh mẽ của mô hình Actor và các mẫu thiết kế phân tán.
- Tích hợp các mẫu thiết kế phân tán tiên tiến: Đề tài đã thành công trong việc tích hợp và minh họa hiệu quả của các mẫu thiết kế như Circuit Breaker, Bulkheads và Service Registry, giải quyết được những thách thức về tính chịu lỗi và linh hoạt của các hệ thống phân tán.
- Tối ưu hóa hiệu năng bằng CBOR: Việc sử dụng CBOR là một đóng góp quan trọng trong việc tối ưu hóa hiệu năng truyền tải thông điệp, đây là một điểm thường bị bỏ qua trong các triển khai mẫu.
- Nền tảng cho các nghiên cứu tiếp theo: Mã nguồn và kiến trúc của đề tài có thể được sử dụng làm nền tảng cho các nghiên cứu và phát triển trong tương lai về các tính năng nâng cao của hệ thống P2P.

5.4. Hướng phát triển

Để nâng cao tính năng và độ mạnh mẽ của hệ thống P2P File Sharing này, có một số hướng phát triển tiềm năng trong tương lai:

- Giao diện người dùng (GUI): Phát triển một giao diện người dùng đồ họa thân thiện để người dùng có thể dễ dàng tương tác với hệ thống (tìm kiếm, tải xuống, đăng ký tệp) thay vì chỉ thông qua giao diện dòng lệnh.
- Bảo mật:
    - Mã hóa dữ liệu: Mã hóa các tệp tin trong quá trình truyền tải giữa các peer để bảo vệ tính riêng tư.
    - Xác thực và ủy quyền: Triển khai cơ chế xác thực peer để đảm bảo chỉ các peer đáng tin cậy mới có thể tham gia và chia sẻ tệp.

- Tải từ nhiều nguồn (Swarming): Áp dụng kỹ thuật tương tự BitTorrent, cho phép một tệp tin được tải xuống từ nhiều peer cùng lúc, giúp tăng tốc độ tải và khả năng chịu lỗi nếu một nguồn bị gián đoạn.
- Cơ chế danh tiếng peer (Peer Reputation): Xây dựng một hệ thống đánh giá để xếp hạng độ tin cậy và hiệu suất của các peer, từ đó ưu tiên tải tệp từ các peer có danh tiếng tốt.
- Tìm kiếm nâng cao:
    - Hỗ trợ tìm kiếm theo từ khóa phức tạp hơn, bao gồm các siêu dữ liệu của tệp tin.
    - Triển khai các cơ chế tìm kiếm và định tuyến hiệu quả hơn trong mạng P2P lớn (ví dụ: tận dụng thêm các tính năng của Pekko Cluster như Sharding hoặc Singleton để phân tán các dịch vụ tìm kiếm).
- Quản lý phiên tải xuống: Triển khai khả năng tạm dừng và tiếp tục tải xuống, cũng như hiển thị tiến trình tải cho người dùng.
- Tự động khám phá peer: Ngoài các seed node được cấu hình tĩnh, nghiên cứu các phương pháp khám phá peer động hơn (ví dụ: thông qua UDP broadcast, mDNS, hoặc dịch vụ khám phá bên ngoài) để hệ thống tự tổ chức linh hoạt hơn.
- Tối ưu hóa băng thông và quản lý kết nối:
    - Giới hạn số lượng kết nối đồng thời.
    - Ưu tiên các kết nối có băng thông cao hoặc độ trễ thấp.
    - Hỗ trợ UPnP/NAT Traversal để cải thiện khả năng kết nối giữa các peer nằm sau các router.
- Đánh giá hiệu năng chi tiết: Thực hiện các bài kiểm thử tải (load testing) và stress testing chuyên sâu để đo lường chính xác hiệu suất của hệ thống dưới các điều kiện tải khác nhau và trên quy mô lớn hơn.

Việc tiếp tục phát triển theo các hướng này sẽ giúp hệ thống P2P File Sharing trở nên mạnh mẽ, thân thiện hơn với người dùng, và sẵn sàng cho các ứng dụng thực tiễn phức tạp hơn.

Tóm lại, Hệ thống này không chỉ thể hiện khả năng tự tổ chức và tự phục hồi khi có node tham gia hoặc rời mạng, mà còn đảm bảo tính nhất quán dữ liệu thông tin tệp tin trên toàn bộ cluster thông qua Pekko Distributed Data. Các cơ chế chịu lỗi như Circuit Breaker và Streams được tích hợp đã nâng cao đáng kể độ tin cậy và hiệu suất truyền tải dữ liệu. Qua đó, chúng tôi đã chứng minh tính hiệu quả của việc ứng dụng các mẫu thiết kế phân tán tiên tiến vào việc phát triển

các hệ thống P2P hiện đại, đáp ứng tốt các yêu cầu về khả năng mở rộng và độ bền bỉ trong môi trường mạng lưới phức tạp.

Tài liệu tham khảo

1. Miller, H., Dempkowski, N., Larisch, J., & Meiklejohn, C. (n.d.). *Programming models for distributed computation: RPC, futures, actors, CRDTs, MapReduce and streaming* (Unpublished open-source textbook). GitHub repository. https://github.com/heathermiller/dist-prog-book
2. CBOR. (2020). *Concise Binary Object Representation (CBOR), RFC 8949*. Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/html/rfc8949
3. Pekko Project. (2024). *Apache Pekko Documentation*. Apache Software Foundation. https://pekko.apache.org/
4. Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.