**Distributed File System**

**Báo cáo Giữa kỳ**

# Thành viên

* Nguyễn Nhật Khang – 21522193
* Nguyễn Hoàng Minh - 21522343

# I. Cơ sở lý thuyết

## 1. Giới thiệu tổng quan

Hệ thống lưu trữ tệp phân tán (Distributed File System) mà nhóm em đang xây dựng là một hệ thống cho phép người dùng có thể lưu trữ tệp tin trên hệ thống phân tán, với dữ liệu được chia nhỏ thành các chunks và lưu trữ trên nhiều node khác nhau. Mục tiêu là giúp tối ưu hoá việc phân phối dữ liệu, tăng tính sẵn sàng và khả năng chống lỗi khi có mất mát xảy ra với dữ liệu.

## 2. Công nghệ sử dụng

* Ngôn ngữ lập trình Golang
* Microservice và giao tiếp thông qua RESTFul API.
* MongoDB
* MessageQueue

# II. Yêu cầu chức năng

**- Upload file:**

* Người dùng có thể chọn file mà mình muốn upload lên hệ thống thông qua UI đơn giản. File này sẽ được FileService xử lý, phân chia nó thành các chunks và lưu trữ trên nhiều node khác nhau. Điều này sẽ tăng khả năng chống lỗi của hệ thống vì nếu dữ liệu trên 1 node bị mất, ta vẫn có thể lấy từ các node khác.
* Các thông tin về file (tên, kích thước, loại tệp,,…) và các chunks (id, start-byte, end-byte) sẽ được lưu trữ trong File Metadata Service.

**- Download file:**

* Người dùng có thể tải về các file đã tải lên trước đó, hệ thống sẽ tìm các chunks tương ứng để ghép lại thành tệp hoàn chỉnh và gửi về cho người dùng.

**- View file list:**

* Người dùng có thể xem danh sách các tệp mà họ đã tải lên, bao gồm các thông tin như tên tệp, kích thước, loại tệp, ngày tải lên.

**- Delete file:**

* Người dùng có thể xóa tệp tin mà họ đã tải lên, khi xoá thì tệp sẽ được xoá trên toàn bộ node chứa dữ liệu của tệp đó.

\*\*\* Các chức năng dự kiến (dựa trên tiến độ của nhóm và feedback từ thầy): Xác thực người dùng, chia sẻ tập tin cho người khác,…

# III. Mô tả một vài luồng hoạt động

## 1. Upload file

Khi người dùng muốn upload một file nào đó lên hệ thống, họ sẽ lựa chọn file mình muốn thông qua UI đơn giản. Sau đó gửi request đến File Service, File Service sẽ kiểm tra xem file này đã từng được upload chưa:

* Nếu file đã tồn tại thì gửi trả về thông báo cho người dùng để tránh lưu trữ trùng lặp.
* Nếu chưa thì sẽ gửi các thông tin của file này đến File Metadata Service để xử lý và lưu vào database.

## 2. Xử lý file khi upload (quản lý các chunks)

Khi người dùng tải tệp lên hệ thống, tệp sẽ được chia thành các chunks nhỏ để dễ dàng quản lý và giảm thiểu rủi ro mất mát dữ liệu. Khi tải lên phiên bản mới của một tệp đã có, hệ thống sẽ so sánh các chunks hiện tại với phiên bản trước đó: những chunks nào đã thay đổi, những chunks nào có thể tái sử dụng (so sánh dựa trên hash). Sau đó sẽ upload các chunks đã thay đổi lên.

Ví dụ:

Ta có version 1: [C1, C2, C3, C4, C5]

Sau đó người dùng tải lên phiên bản mới sau khi chỉnh sửa: version 2 [C1, C2, C3, C6].

Khi đó hệ thống sẽ so sánh và chỉ upload các chunk đã thay đổi.

(Tuy nhiên logic khá phức tạp, nhóm em sẽ nêu rõ hơn ở phần khó khăn)

# IV. Khó khăn

Vì đây là một chủ đề khá mới với nhóm em, kể cả công nghệ sử dụng (mới tìm hiểu về golang) nên thời gian nghiên cứu khá lâu.

Hiện tại, chúng em đang nghiên cứu logic để phân chia file thành các chunks và quản lý chúng. Như đã nói ở trên, hệ thống dự kiến sẽ so sánh sự khác biệt giữa các chunks của phiên bản cũ và mới (dựa trên hash nội dung của chunk) và chỉ upload các chunks bị thay đổi để tối ưu hoá việc xử lý và lưu trữ.

Tuy nhiên việc phát hiện chính xác các phần thay đổi trong tệp mới khá phức tạp. Giả sử tệp cũ là 1 file docs 10MB với 1000 ngàn dòng, file có thể được chia thành 10 chunks với mỗi chunks gồm 100 dòng, kích thước 1MB.

Nếu phiên bản mới được upload chỉ thay đổi đoạn cuối thì sẽ dễ dàng hơn cho việc so sánh (giống ví dụ ở mục III.2). Nhưng nếu phiên bản mới có sự chỉnh sửa ở đầu file (ví dụ thêm n dòng), việc đó sẽ khiến toàn bộ dòng bên dưới bị đẩy đi 1 đoạn => toàn bộ chunks đều thay đổi dù chỉ thêm 1 đoạn nhỏ ở đầu file => Không mang lại nhiều ưu điểm so với việc xoá toàn bộ file cũ và upload file mới.