Thiết kế hệ thống Scalable Cloud Storage

1. Giới thiệu

Cùng với sự phát triển của công nghệ điện toán đám mây và các công nghệ lưu trữ, một phương thức lưu trữ mới ra đời, đó là phương thức **Object Storage Cloud**. Phương thức này cho phép người dùng lưu trữ các nội dung không cấu trúc - các Object data lên các Cloud server với các ưu điểm so với phương thức lưu trữ dữ liệu truyền thống: Khả năng sao lưu bản sao- replication, tốc độ truy cập dữ liệu, tính chịu lỗi, khả năng scale up - scale down hệ thống lưu trữ… Những ưu điểm trên giúp cho các nền tảng block storage phát triển mạnh mẽ thời gian gần đây như: Amazon S3, Swift, Google Cloud Storage, Azure Blob storage (object storage)…

Sự phát triển của các hệ thống Cloud Object Storage đem đến cho người dùng nhiều lựa chọn hơn, tuy nhiên các như cầu sử dụng đa dạng của người dùng đặt ra một loạt vấn đề mới cho Object Storage Cloud:

- Các Object Storage Cloud không có tính đồng nhất(các API cung cấp cho người dùng khác nhau, định dạng dữ liệu khác nhau, vv…).
- Nhu cầu lưu trữ dữ liệu trên nhiều nền tảng khác nhau của những người dùng có nhiều hệ thống Object Storage Cloud.
- Nhu cầu tương tác, đồng bộ dữ liệu giữa 2 Cloud Object Storage.
- Implement phương thức lưu trữ Object Storage trên nền tảng các phương thức lưu trữ khác (như Block Storage, File System Storage, …) và tương tác/ đồng bộ các phương thức lưu trữ này với các phương thức lưu trữ Object Storage.

Xuất phát từ nhu cầu thực tiễn của người dùng, nhóm phát triển quyết định xây dựng một hệ thống cho phép giải quyết các vấn đề đã nêu trên. Hệ thống được xây dựng có tên là: Scalable Cloud Storage (SCS). SCS cho phép kết hợp tất cả các cơ sở lưu trữ mà người dùng đang có thành một kho lưu trữ thống nhất. Kho lưu trữ thống nhất sau khi được xây dựng sẽ cung cấp cho người dùng năng lực lưu trữ của tất cả các đám mây mà người dùng đang có, đồng thời có những tính năng nổi bật sau:

- High-availability
- Scalable
- Fault-tolerance
- Load-balancing

- Redundancy storage

Bên cạnh những tính năng trên, hệ thống SCS đảm bảo các tương tác với dữ liệu của người dùng như lưu trữ, truy cập thay đổi dữ liệu… được thực hiện một cách tối ưu - optimize nhất.

1.1 SCS Case Study

Hệ thống SCS phù hợp để giải quyết các trường hợp thực tế sau:

2. Scenario - System Environment

Trong phần này, chúng ta sẽ trình bày một kịch bản thực tế được sử dụng để làm cơ sở xuất phát cho việc xây dựng hệ thống SCS:

Một tập đoàn lớn có nhiều công ty con, mỗi một công ty con sở hữu hàng loạt các cơ sở lưu trữ dữ liệu sử dụng nhiều công nghệ lưu trữ khác nhau như như swift, amazon S3, Ceph, Google Cloud Storage, vv… Dữ liệu và các cơ sở lưu trữ của các công ty con là riêng biệt và độc lập với nhau.

Tập đoàn sẽ triển khai hệ thống SCS để thực hiện nhiệm vụ chính của SCS, đó là tích hợp tất cả các cơ sở lưu trữ dữ liệu mà một công ty con đang có thành một cơ sở lưu trữ dữ liệu thống nhất cho công ty con đó. Các yêu cầu khác của tập đoàn đối với hệ thống SCS là:

- Hệ thống được xây dựng phải phục vụ cho cả tập đoàn, tuy nhiên phải đảm các công ty độc lập với nhau.
- Dữ liệu do các công ty đưa lên được phân phối đều trên các cloud của công ty đó.
- Đảm bảo hiệu suất hoạt động của hệ thống là tối ưu.
- Đảm bảo hệ thống được thiết kế theo mô hình high-availability, đáp ứng được một lượng tải lớn.
- Đảm bảo sự an toàn của dữ liệu, kể cả trong trường hợp một số các cơ sở lưu trữ bị hỏng hóc - ngừng hoạt động.

3. Thiết kế hệ thống

3.1 Xây dựng mô hình kiến trúc hệ thống

Xuất phát từ mục tiêu đầu tiên của hệ thống SCS, là tích hợp nhiều hệ thống lưu trữ dữ liệu của một người dùng thành một hệ thống lưu trữ thống nhất, chúng ta tiến hành thiết kế mô hình kiến trúc và xác định các thành phần cơ bản của hệ thống lưu trữ thống nhất multi-cloud:

Một cách tổng quát nhất, hệ thống có kiến trúc như sau:

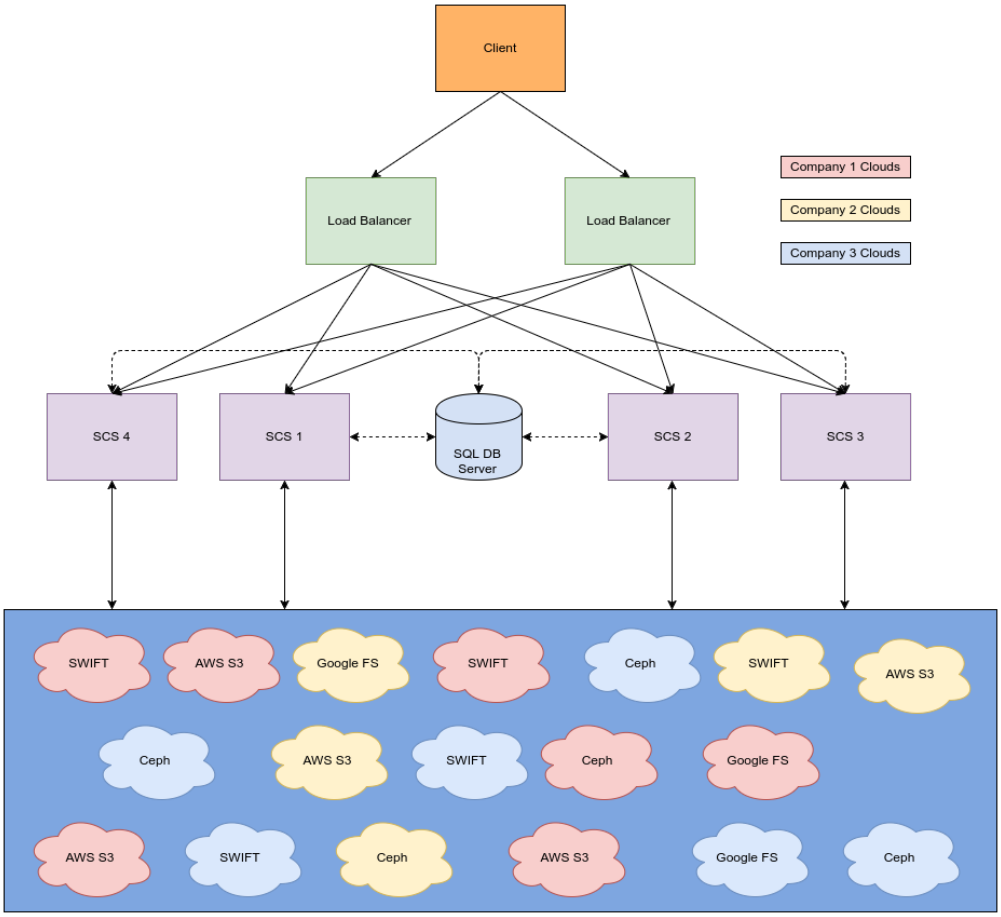

Figure 1: System Architect

Mô hình kiến trúc phía trên là mô hình kiến trúc đầy đủ của hệ thống, với đầy đủ các thành phần, cùng với đó là hệ thống SCS được scaling để thể hiện đầy đủ các tính chất của một hệ thống phân tán, và phục vụ cho nhiều người dùng cùng một lúc, với mỗi người dùng có tập các hệ thống lưu trữ độc lập với nhau. Tuy nhiên, sử dụng kiến trúc đầy đủ này để thiết kế hệ thống là không phù hợp, do kiến trúc này quá phức tạp và không tập trung vào nhiệm vụ quan trọng nhất của hệ thống, đó là: Tích hợp các hệ thống lưu trữ của một người dùng thành một cơ sở lưu trữ thống nhất cho người đó. Đặt trọng tâm vào giải quyết nhiệm vụ này, chúng ta sẽ sử dụng mô hình kiến trúc sau để thiết kế hệ thống:

Dưới góc nhìn này, hệ thống bao gồm các thành phần chính sau:

- SCS: Hệ thống chính mà chúng ta xây dựng, bao gồm WSGI Server để nhận và xử lý request của người dùng, và các Deamon Process thực hiện các chức năng của hệ thống.
- SQL Database Server: Lưu trữ dữ liệu của hệ thống SCS
- Các cloud: Tập hợp các cơ sở lưu trữ

Để bắt đầu việc thiết kế hệ thống SCS, chúng ta đi vào phân tích nhiệm vụ chính của SCS: kết hợp tất cả các Cloud Storage Server của một User thành một kho lưu trữ thống nhất cho user đó.

3.2 Storage Cloud, Data Object và Chord Protocol

Mục đích của việc tạo ra kho lưu trữ thống nhất, đó là cho phép người dùng hệ thống có thể sử dụng hệ thống để lưu trữ các data object hiệu quả mà không cần quan tâm tới việc object được lưu trữ như thế nảo ở hạ tầng lưu trữ bên dưới. Đó là cái nhìn ở góc độ người dùng. Còn ở góc độ người thiết kế hệ thống SCS, chúng ta hiểu rằng, bản chất của việc lưu trữ một Data Object vào hệ thống là việc lưu trữ các bản sao Data Object đó lên các cơ sở dữ liệu mà người dùng sở hữu, và nhiệm vụ của chúng ta là xây dựng các cơ chế để thực hiện công việc lưu trữ này một cách hiệu quả nhất. Các cơ chế được xây dựng để giải quyết các vấn đề sau:

- Khi người dùng tài khoản này muốn lưu trữ một Data Object mới trên hệ thống, hệ thống sẽ sao lưu data Object trên thành bao nhiêu bản, và mỗi bản sao của Data Object trên sẽ được lưu trong Cloud nào trong số các Cloud mà tài khoản sở hữu?

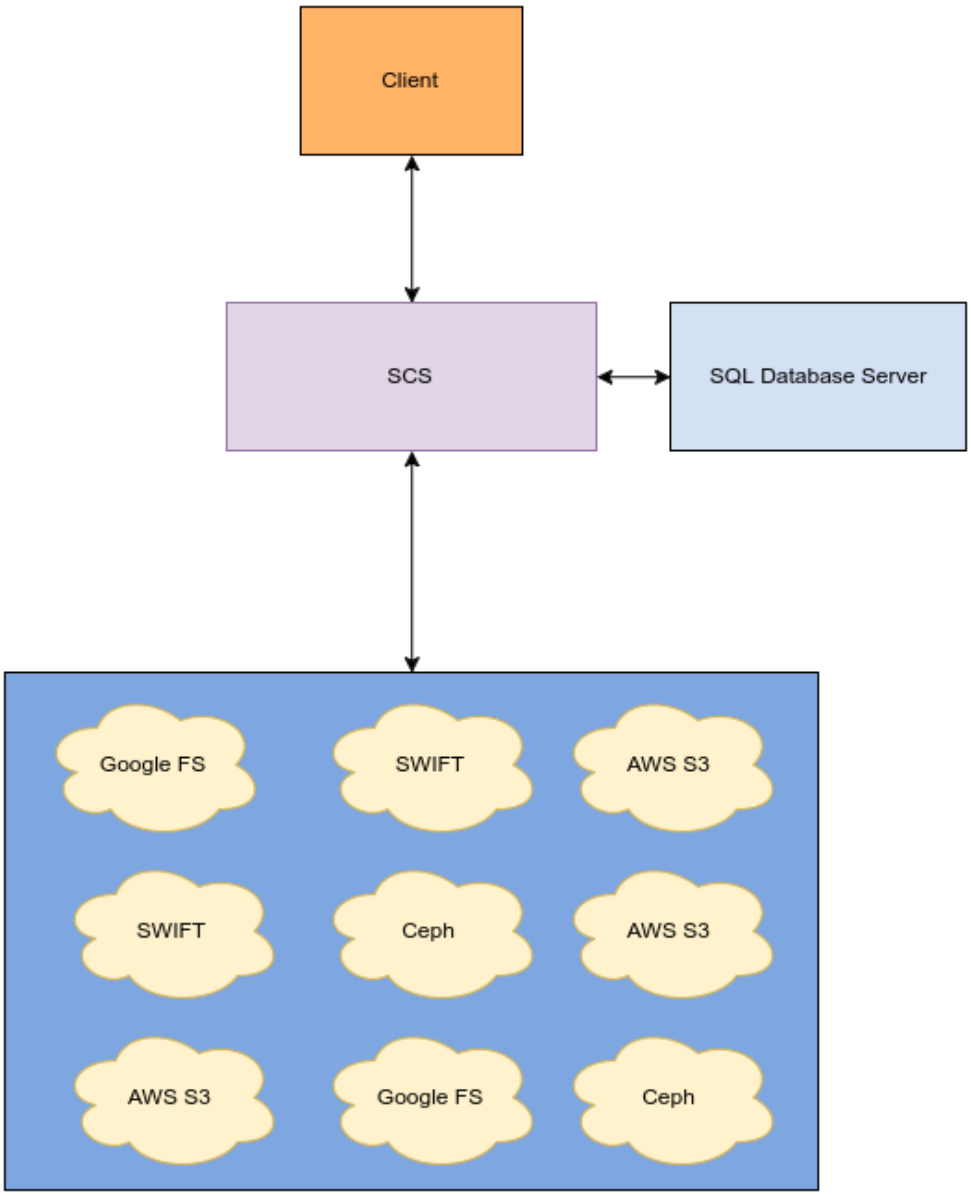

Figure 2: System Architect

- Khi người dùng tài khoản muốn lấy từ hệ thống về nội dung của một Data Object, làm sao để chúng ta biết chúng ta có thể lấy nội dung Data Object này từ Cloud nào trong số các Cloud của tài khoản? (Lưu ý là một Data Object có nhiều bản sao lưu trên nhiều Cloud Server khác nhau)
- Khi người dùng cập nhật nội dung của một Data Object, làm sao để hệ thống đồng bộ hóa giữa các bản sao của Object đó?
- Khi một Cloud mới được người dùng thêm vào hệ thống, hoặc khi người dùng quyết định loại bỏ một Cloud khỏi hệ thống, chúng ta sẽ thực hiện việc di chuyển dữ liệu giữa các Cloud của người dùng như thế nào?

Xuất phát từ việc giải quyết các vấn đề nêu trên, chúng ta sẽ xây dựng các cơ chế lưu trữ của hệ thống dựa trên nền tảng là một protocol cho phép các hệ thống phân tán lưu trữ, quản lý và truy vấn dữ liệu hiệu quả, đó là **Chord protocol**.

Chord protocol được xây dựng xung xoay quanh 2 đối tượng **Node** và **Value**, và bài toán nền tảng mà Chord Protocol giải quyết là: Cho một đối tượng value **x** và một hệ thống có **n** node, value **x** sẽ được lưu vào node nào trong **n** node trên.

Tại sao chúng ta lại lựa chọn Chord Protocol làm nền tảng để xây dựng các cơ chế lưu trữ của hệ thống ?

Lý do lựa chọn Chord protocol

Xây dựng các cơ chế lưu trữ cho SCS trên nền tảng Chord Protocol

3.3 Cloud Ring, Cloud Node và Data Object

Áp dụng Chord protocol vào hệ thống của chúng ta:

- Các **Node** trên Ring sẽ là các **Cloud** của User.
- Các **Value** được lưu trữ trên các Node là các **Data Object** mà User cần lưu trữ trên hệ thống.

Để sử dụng Chord Protocol, chúng ta sẽ chọn một **Consistent Hashing** để sinh ID cho các đối tượng trong hệ thống. Consistent Hashing được lựa chọn

ở đây là SHA-1, và chúng ta sẽ ký hiệu phương thức Hashing sử dụng SHA-1 là `SHA1_Hash()`.

Sử dụng Chord Protocol, chúng ta bắt đầu xây dựng các cơ chế xử lý lưu trữ cho hệ thống. Đầu tiên, chúng ta sẽ sử dụng Chord Protocol để xây dựng Chord logic ring - **Cloud Ring**.

3.4 Init Cloud Ring Process

Để có thể xây dựng Cloud Ring cho một user trong hệ thống, chúng ta cần có thông tin **Cloud_List**, là một danh sách các **Cloud Object**, để đại diện cho tập các Cloud mà user đó sở hữu. CloudObject chứa thuộc tính **Cloud_Config** - là thông tin định danh của một Cloud. Thông tin định danh của một Cloud bao gồm: Loại Cloud (S3, SWift, Google Cloud, Ceph,…), thông tin xác thực (account, password, token,…), địa chỉ truy cập của Cloud (Ip Address, Port,…), … Thông tin này sẽ được sử dụng để tạo ra ID cho Cloud đó.

Quá trình xây dựng Cloud Ring từ tập các Cloud của user diễn ra như sau:

- Đầu tiên chúng ta cần gán cho mỗi một Cloud một định danh **CloudID**. CloudID của một Cloud sinh ra dựa bằng cách sử dụng hàm băm SHA-1 hash thông tin định danh của Cloud đó (IP Address + Account name + Password + Container name…).
- Sau khi tạo ra CloudID cho các Cloud, chúng ta xếp các Cloud lên Cloud Ring, lúc này các Cloud đóng vai trò là các **CloudNode** trong Chord Logic Ring rồi xác định các thông tin định tuyến: Successor Node, Previous Node, Finger Table… cho cho từng Cloud Node trên Cloud Ring theo quy tắc của Chord Protocol. Quá trình sắp xếp các Cloud lên Cloud Ring được thực hiện bằng cách đưa lần lượt các Cloud trong Cloud_List join vào trong Cloud Ring theo thuận toán **Node_Join** của Chord Protocol.

Algorithm1: InitCloudRing(Cloud_List)

```
function InitCloudRing(Cloud_List){

    // Create CloudID for each Cloud in Cloud List
    for Cloud in Cloud_List:
```

```
    {
        Cloud_Identifier_Info = get_identifier_info(Cloud.Cloud_Config);
        Cloud.CloudID = SHA1_hash(Cloud_Indentifier_Info);
    }

    // Init Cloud Ring by first Cloud in Cloud List
    First_Cloud = Cloud_List.get_first_cloud();
    Cloud_Ring = init_cloud_ring(First_Cloud);

    // Put Other Clouds to Cloud Ring by Node Join
    for Cloud in Cloud_List except First_Cloud:
    {
        Cloud.join_node_to_cloud_ring(Cloud_Ring);
    }

    return Cloud_Ring;
}
```

Sau khi xây dựng xong Cloud Ring, chúng ta sẽ lưu thông tin Cloud Ring vào tài khoản User. Bước tiếp theo, chúng ta sẽ xây dựng cơ chế lưu trữ một Data Object lên hệ thống.

3.5 Process Data Object x in SCS System

Trong quá trình thiết kế cơ chế lưu trữ Data Object cho hệ thống SCS, chúng ta sẽ gặp và phải giải quyết hàng loạt vấn đề liên quan tới các tác vụ xung quanh Data Object, như các tác vụ lưu trữ, cập nhật, truy cập, xóa bỏ (CRUD process), cân bằng tải,vv..., cũng như hàng loạt các các yêu cầu đặt ra cho hệ thống về tốc độ - hiệu năng, tính high-available, tính replication - consistency của data,... khi hệ thống thực hiện các tác vụ nói trên. Chúng ta sẽ phân tích các vấn đề trên và tìm giải pháp để thiết kế một cơ chế lưu trữ phù hợp với các yêu cầu đã được đặt ra.

3.5.1 Create Data Object Process

Vấn đề đầu tiên chúng ta giải quyết, đó là lưu trữ một Data Object **x** mới lên hệ thống, với thông tin đầu vào là **x.Object_Name** và **x.Data**. Quá trình lưu trữ **x** lên hệ thống của chúng ta phải đảm bảo yêu cầu sau: **x** phải

được sao lưu thành **k** bản sao (giá trị của **k** sẽ do User thiết lập), và **k** bản sao này và lưu tại **k** Cloud trong số các Cloud mà người dùng có.(*)

Chúng ta sẽ đáp ứng yêu cầu (*) bằng cách tạo ra **k** bản sao, mỗi bản sao sẽ có một tên và ID riêng biết. Chúng ta sẽ đặt tên cho các replica của **x** bằng cách gán thêm các hậu tố **__replica(i)** vào tên của x. Ví dụ với Data Object có tên là **data.png** thì các bản sao có thể có tên là **data.png__replica1**, **data.png__replica2**, **data.png__replica3**,... (với k =3)

Lúc này, các replica của **x** sẽ có vai trò như các giá trị **Value** trong hệ thống sử dụng Chord Protocol. Để lưu trữ các replica của **x**, hệ thống sẽ hash tên của các replica (vừa được tạo ra ở bước trước) để tạo thành **replicaID** cho các replica này. Saud đó, cặp <**replicaID**,**x.Data**> sẽ tạo thành các **Key-Value** trong hệ thống Chord Protocol. Sau đó, hệ thống SCS sẽ sử dụng Chord Protocol để tìm ra Successor Node tương ứng với **replicaID** của từng cặp <**replicaID**,**x.Data**>. Cloud tương ứng với Sucessor Nodeđó sẽ được chọn để lưu trữ cặp <**replicaID**,**x.Data**> này.

Một vấn đề xảy ra ở đây, đó là có thể xảy ra trường hợp Cloud Node của một replicaID nào đó đã bị đầy - không thể chứa thêm Object nữa, hoặc không đủ khả năng để chứa Object này. Giải pháp của chúng ta trong trường hợp này, đó là trước khi lưu một replica của x vào một Cloud Node là Successor Node của replicaID, chúng ta cần kiểm tra xem Cloud Node đó có đủ khả năng lưu trữ replica đó không. Nếu trong trường hợp Cloud Node không có đủ khả năng lưu trữ replica của x, chúng ta sẽ sinh ra một tên khác cho replica này và tạo ra một ReplicaID mới, sao cho replica này sẽ được lưu vào một Cloud Node khác có đủ khả năng chứa nó. Trong một số trường hợp khi hệ thống quá tải (Ví dụ khi có quá nhiều Cloud Node trong hệ thống không còn đủ khả năng lưu trữ Data Object mới), chúng ta có thể cảnh báo User về tình trạng hệ thống.

Cần thảo luận thêm với thầy Tuy nhiên, có một vấn đề phát sinh ở đây, đó là chúng ta không hoàn toàn đảm bảo rằng, **k** key được sinh ra sẽ luôn luôn nằm trên **k** Cloud khác nhau, do chúng ta không thể nào điều khiển được replicaID nhận được sau khi hashing replica_name sẽ rơi vào node nào trên ring ? Đặt ID cho Node/Replica, sau đó lưu lại lastID used trong Node ?

Thứ hai, là có luôn cần đảm bảo **k** bản sao phải nằm trên **k** node khác nhau (một cách tuyệt đối ?) Nếu không cần thì ta tiếp tục sử dụng cách cũ. **Cần thảo luận thêm với thầy**

3.5.2 Lookup Data Object Process

Vấn đề tiếp theo mà chúng ta cần giải quyết, đó là sau khi Data Object **x** đã được lưu trên hệ thống, làm sao để User có thể truy cập tới nội dung của **x** thông qua hệ thống của chúng ta, với tham số truyền vào là tên của Data Object **x** - **x.Object_Name**?

Sau phần giải quyết vấn đề lưu trữ một Data Object mới lên hệ thống, chúng ta hiểu rằng một Data Object **x** bất kỳ sẽ có **k** replica lưu trên **k** Cloud Server, mỗi một replica có một **replicaID** riêng, và chỉ cần có được một trong số các replicaID là chúng ta có thể sử dụng Cloud Ring để tìm và lấy được nội dung của Data Object **x**. Tuy nhiên, chúng ta thấy rằng, không có cách nào để sinh ra trực tiếp replicaID từ tên của Data Object **x**. Do vậy, chúng ta cần phải có cách để lưu trữ các thông tin về các replica của **x**, hay nói cách khác chính là các **replicaID**.

Thông tin về các Replica của **x** cũng chính là các thông tin liên quan tới **x**, chúng được gọi là **Object metadata** của **x**. Vì vậy, giải pháp được sử dụng trong hệ thống SCS để thực hiện tác vụ Lookup và Get Data Object **x**, đó là tạo ra và lưu trữ đối tượng **Object metadata** của x. **Object metadata** của x sẽ lưu trữ các thông tin liên quan tới **x**, với vấn đề Lookup Data Object của chúng ta, thông tin về các bản sao của **x** và **x.Object_Name** sẽ được lưu vào Object metadata.

Quá trình lookup **cơ bản** sẽ diễn ra như sau:

Khi nhận được lookup request, SCS sẽ lấy ra thông tin **Object_Name** từ request, và tìm trong cơ sở dữ liệu **Object Metadata** nào tương ứng với **Object_Name** này. Sau đó SCS sẽ lấy ra một **replicaID** trong số các **replicaID** của Object đó, và dựa vào thuật toán Lookup của Chord Protocol để tìm xem Cloud Node nào đang chứa replica tương ứng với replicaID này (replicaID's successor Node). Bước cuối cùng, SCS Server trả về cho User các thông tin cần thiết như: replicaID và thông tin định danh của Cloud để User có thể kết nối trực tiếp tới Cloud Server để lấy nội dung của Data Object **x** về. Cơ chế tương tác trực tiếp giữa User và Cloud Server cho phép dữ liệu không cần phải đi qua hệ thống trung gian là SCS, qua đó giảm tải cho hệ thống SCS cũng như tăng hiệu năng truy cập, vì cách User truy cập trực tiếp tới Cloud Server sẽ nhanh hơn việc chúng ta phải lấy nội dung Object từ Cloud Server về SCS, sau đó lại từ SCS trả nội dung Object về User.

(Vấn đề- Hybrid Cloud ? - Cloud có thêm thuộc tính là private

hay public, nếu public thì cho phép người dùng connect trực tiếp, còn nếu private thì cho đi qua SCS rồi SCS trả về ?)

Như vậy, chúng ta đã xây dựng quy trình xử lý cơ bản cho thao tác Lookup Data Object. Tuy nhiên, như chúng ta đã nói ở phần đầu, các thao tác trên Data Object phải đảm bảo về các tính chất của hệ thống phân tán như tính High-available, cân bằng tải và tính nhất quán của dữ liệu - data consistency. Trong thao tác Lookup Data Object, các tính chất trên biểu hiện cụ thể thông qua các kịch bản sau:

Thứ nhất: Chúng ta xử lý ra sao khi một Replica của Data Object mà chúng ta muốn truy cập bị hỏng, do Cloud Node chứa Replica đó gặp sự cố?

Ghi chú: Vấn đề kiểm tra trong các Cloud Node, có Cloud Node nào gặp sự cố hay không được SCS lập lịch để thực hiện (đánh dấu replica đó đang bị hỏng/ lập lịch để tạo ra 1 replica khác trên 1 cloud Node khác). Ví dụ cứ 1 phút kiểm tra lại toàn bộ các Cloud của User A, xem có cloud nào có vấn đề gì không,nếu có vấn đề cập nhật vào thông tin của Cloud đó. chứ không để tới khi Truy cập vào một Data Object nào đó mới thực hiện việc kiểm tra, vì cách này sẽ tạo ra quá nhiều request kiểm tra.

Giải pháp:

- Để kiểm tra tình trạng các Cloud Node của một User, trên hệ thống SCS chúng ta cần tạo ra các tiến trình chạy ngầm, định kỳ kiểm tra tình trạng của các Cloud Node của các User. Tình trạng của các Cloud Node của một User sẽ được định kỳ cập nhật vào thông tin của User đó.
- Khi lấy ra thông tin một replica của **x** để trả về cho người dùng, chúng ta sẽ truy cập vào thông tin của User để lấy ra tình trạng hiện tại của các Cloud nó. Nếu Cloud Node chứa replica đó đang có trình trạng xấu (bị hỏng/ ngắt kết nối,...), SCS cần trả về một Replica khác của **x** nằm ở Cloud Node có tình trạng tốt.

Thứ hai: Chúng ta xử lý ra sao khi có quá nhiều truy cập vào một Data Object trong một khoảng thời gian ngắn ? (cân bằng tải giữa các replica)?

Giải pháp:

- SCS theo dõi xem trong **k** phút gần đây nhất, một tài khoản người dùng - User đang có những Data Object nào đang được client truy cập vào. Thông tin về lưu lượng truy cập gần đây

tới Data Object **x** của một User trong hệ thống được gọi là **Data_Object_Connection_Information** của **x**. SCS lưu trữ lại tất cả **Data_Object_Connection_Information** gần đây vào trong một danh sách và lưu trữ vào thông tin của User đó.

- **Data_Object_Connection_Information** của **x** là thông in cho biết **x** được bao nhiêu Client truy cập tới trong khoảng thời gian **k** phút gần đây, và ghi lại mỗi một replica của **x** đang phục vụ cho bao nhiêu connection ?
- Dựa vào **Data_Object_Connection_Information** của **x**, SCS sẽ sử dụng một trong các chiến lược lập lịch (scheduler) để lần lượt trả về cho request các replica khác nhau của x. Các chiến lược lập lịch có thể sử dụng ở đây là Round-Robin, least connection, kết hợp với thông tin của request (Ví dụ như địa điểm gửi request đang gần với replica nào nhất ?)

Thứ ba: Không phải bất cứ lúc nào các bản sao của một Data Object trên các Cloud Server cũng đồng bộ với nhau:Khi người dùng cập nhật nội dung của Data Object **x**, sự không nhất quán dữ liệu giữa các bản sao của **x** sẽ xảy ra trong một khoảng thời gian. Lý do là vì theo cơ chế cập nhật Data Object mà SCS sử dụng mà chúng ta sẽ nói tới ở phần sau - **Read After Write**, thì tại thời điểm người dùng cập nhật nội dung Data Object, sẽ chỉ có một trong số các bản sao của **x** được cập nhật. Các bản sao khác của x sẽ được đồng bộ và cập nhật vào một thời điểm khác. Vậy quá trình Lookup **x** trong khoảng thời gian trước khi tất cả các bản sao của x được đồng bộ sẽ diễn ra như thế nào ?

Giải pháp:

- Vấn đề Lookup ở đây có liên quan chặt chẽ tới cơ chế xử lý cập nhật Data Object **x**. Theo đó, khi người dùng thực hiện thao tác cập nhật Data Object **x** chúng ta cần lưu lại **các replica nào trong số các replica của x đã được cập nhật**, đồng thời đánh dấu **x** chưa được đồng bộ hóa. Hai thông tin: **x.is_synchronized = False** và **is_synchronized** - thuộc tính của một replica xác định replica đó đã được cập nhật hay chưa đã được cập nhật sẽ được lưu vào **Object Metadata** của x.
- Khi một Client thực hiện Lookup **x**, chúng ta phải truy cập vào Object Metadata của **x** để kiểm tra xem **x** đã được đồng bộ hay chưa bằng cách kiểm tra tham số **x.is_synchronized**. Nếu **x** chưa được đồng bộ, thì

theo cơ chế của Read After Write, SCS sẽ trả về cho Client một trong số replica đã được cập nhật - replica tương ứng với **updated_replicaID**.

Như vậy, trong quá trình giải quyết các vấn đề gặp phải trong hệ thống, **Object metadata** của **x** đã mở rộng ra và chứa các thông tin sau:

- ID của **x**
- Tên của Data Object **x**
- Số lượng các bản sao của x và thông tin về các bản sao của **x**
- Trạng thái đồng bộ: Được đồng bộ hay chưa được đồng bộ.
- Trong danh sách các replica của Object: Thông tin về một replica không chỉ có **replicaID**, mà còn là trạng thái đồng bộ **replica.is_synchronized** của replica đó nữa.
- ...

Trong các phần tiếp theo, những đối tượng dữ liệu và các phương thức xử lý mà chúng ta đã trình bày có thể tiếp tục được mở rộng hoặc điều chỉnh để đáp ứng cho việc giải quyết các vấn đề xảy ra khi thiết kế hệ thống. Phần tiếp theo, chúng ta sẽ xây dựng cơ chế để thực hiện việc cập nhật một Data Object.

3.5.3 Update Data Object Process

Như đã trình bày ở phần Lookup, quá trình Update Data Object của một User tuân theo nguyên tắc Read and Write: Cơ chế cơ bản của việc cập nhật nội dung cho Data Object **x** diễn ra như theo quy tắc Read After Write như sau:

Tham số đầu vào của quá trình cập nhật Data Object **x** là tên của **x** - x.Object_Name và nội dung mới mà **x** sẽ lưu trữ - x.New_Content. Khi SCS nhận được yêu cầu cập nhật từ người dùng, Hệ thống sẽ sử dụng **x.Object_Name** để lấy ra Object Metadata của **x**, sau đó cập nhật **x.New_Content** vào một trong các replica của **x**. Sau khi cập nhật xong nội dung cho replica được chọn, chúng ta thay đổi trạng thái của x sang thành chưa được đồng bộ - **x.is_synchronized = False** và lưu lại ReplicaID của replica mà chúng ta đã cập nhật lên phiên bản mới nhất vào **updated_ReplicaID**. Đồng thời chúng ta cập nhật trạng thái cho các replica, replica đã được cập nhật sẽ được thiết lập **replica.is_synchronized = True**, các replica chưa được đồng bộ còn lại được thiết lập **replica.is_synchronized = False** .

Các vấn đề cần giải quyết trong quá trình cập nhật Data Object x là:

Thứ nhất: Như ta đã nói, chiến lược của chúng ta là tạo ra một **Deamon Process** định kỳ thực thi công việc đồng bộ các replica cho các Data Object bị cập nhật nội dung với chu kỳ **k** phút. Để giúp **Deamon Process** này hoạt động, chúng ta sẽ lưu lại thông tin về các Data Object bị cập nhật nội dung vào một danh sách lưu trong thông tin của User sở hữu Object đó. Do trong **k** phút, số lượng Object mà hệ thống có thể đồng bộ được là có giới hạn, do đó độ dài danh sách các Data Object bị cập nhật cũng cần phải có giới hạn. Điều này có nghiã là nếu số lượng yêu cầu cập nhật của người dùng đưa vào hệ thống là quá nhiều và vượt quá số lựng Data Object có thể đồng bộ trong khoảng thời gian trên, chúng ta sẽ từ chối yêu cầu cập nhật của người dùng, và thông báo cho người dùng tạm ngừng việc cập nhật nội dung các Data Object cho tới khi các Data Object nằm trong danh sách chờ đồng bộ được đồng bộ hóa xong.

Điều này có nghĩa là trước khi thực hiện việc cập nhật nội dung cho Data Object, SCS sẽ kiểm tra xem danh sách chờ đồng bộ của User đã đầy chưa. Nếu danh sách chờ đồng bộ đã đầy, chúng ta sẽ từ chối yêu cầu cập nhật của User và trả về lý do từ chối.

Thứ hai: Chúng ta cần xác định cơ chế đồng bộ hóa. Cứ sau mỗi **k** phút, Deamon Process thực hiện nhiệm vụ đồng bộ dữ liệu hoạt động. Qúa trình đồng bộ sẽ diễn ra như sau:

- **Deamon Process** sẽ lần lượt lấy ra từ danh sách chờ đồng bộ hóa thông tin về Data Object **x** chưa được đồng bộ. Thông tin về một Data Object chưa được đồng bộ bao gồm tên của Data Object, replicaID của Replica đã được đồng bộ.
- **Deamon Process** sử dụng tên của Data Object lấy ra Object Meata-data tương ứng với Data Object cần đồng bộ, từ đó lấy ra danh sách các replica chưa được đồng bộ của Data Object đó, sau đó **Deamon Proccess** thực hiện việc lấy nội dung mới nhất của Data Object từ replica đã được đồng bộ lên SCS Server, sau đó nội dung lấy về SCS được thực hiện để đồng bộ cho các replica chưa được cập nhật nội dung mới nhất, các replica đã được cập nhật xong được đặt lại thuộc tính **repica.is_synchronized = True**.
- Sau khi các replica còn lại đã được cập nhật nội dung mới nhất, **Deamon Process** thay đổi trạng thái của Data Object thành đã được đồng bộ: **x.is_synchronized = True**

- Sau khi đồng bộ xong cho một Data Object có trong danh sách chờ đồng bộ hóa, **Deamon Process** loại bỏ Data Object này khỏi danh sách chờ, và lấy ra Data Object tiếp theo để thực hiện việc đồng bộ.

Thứ ba: Trong trường hợp Data Object **x** vừa mới cập nhật và chưa được thực hiện việc đồng bộ thì đã có thêm một yêu cầu cập nhật nhật nội dung **Data Object x** đến hệ thống SCS. Trong trường hợp này, chúng ta có ba lựa chọn:

- Hoặc là chúng ta sẽ từ chối yêu cầu cập nhật thứ 2, do **x** vẫn chưa được đồng bộ
- Hoặc chúng ta sẽ cho phép cập nhật nội dung ở yêu cầu thứ 2 lên lên một replica khác,
- Hoặc chúng ta cũng có thể yêu cầu người gửi yêu cầu cập nhật thứ 2 đổi tên cho **Data Object x**

Như vậy, một trong các điểm quan trọng nhất để thực hiện quá trình đồng bộ, đó là tài khoản người dùng phải lưu lại danh sách các Data Object đang chờ đồng bộ hóa. Danh sách này được sử dụng bởi **Deamon Process** để tiến hành đồng bộ hóa các replica cho các Data Object chưa được đồng bộ dữ liệu.

3.5.4 Delete Data Object Process

Cơ chế xóa một Data Object trên hệ thống: Đưa thông tin của Data Object bị xóa vào hàng chờ **Wait_Delete_Object_List** chứa trong thông tin của User, sau đó thực thi các bước sau:

- Bước 1: Đánh dấu Data Object bị xóa bằng cách thiết lập **is_deleted = True** trong Object Metadata
- Bước 2: Thiết lập một Deamon Process định kỳ thực hiện công việc sau:
  - Lấy ra một Data Object từ **Wait_Delete_Replica_List**.
  - Xóa các bản sao của Data Object đó.
  - Xóa Object Metadata của Data Object đó sau khi đã xóa mọi Object Metadata.

Note: Trong qúa trình lookup, SCS cần kiểm tra xem Data Object đã bị xóa hay chưa bằng cách đọc giá trị của thuộc tính **is_deleted**. Nếu Data Object đã bị xóa, hệ thống thông báo lại cho người dùng.

3.5.5 Update Data Object Name Process

Một thao tác nữa mà chúng ta cần phải xử lý, đó là đổi tên của Data Object **x**. Thao tác này xảy ra khi người dùng muốn đổi tên Data Object **x** từ **name_1** sang **name_2**.

Để xử lý thao tác này, chúng ta cập nhật trong Object Metadata tên của Data Object **x** sang tên mới, đồng thời tạo lại Object Metadata ID theo tên mới của **x**.

Một vấn đề đặt ra ở đây, đó là khi chúng ta thay tên của **x** như vậy, liệu chúng ta có phải đặt lại **replicaID** cho các replica của **x** hay không ? Vì nếu như sau này người dùng lại tên của **x** là **name_1** cho một Data Object mới, thì sẽ xảy ra khả năng 2 Data Object có một replicaID trùng nhau, trong trường hợp chúng ta dùng tên cơ sở là tên của Data Object + hậu tố để hash tạo ra replicaID ?

Giải pháp đề xuất:

- Tên cơ sở để tạo ra replicaID là tên Data Object lúc khởi tạo + **time_stamp** là thời gian Data Object đó được tạo ra.
- replicaID được tạo ra bằng một cách khác - không sử dụng hàm hashing để tạo ra, có thể dùng giải pháp như **auto-increment** ?

3.6 Process Cloud Node Join and Leave Events in SCS System

SCS dựa vào cơ chế Node Join and Leave của Chord Protocol để tạo ra cơ chế xử lý các sự kiện người dùng thêm một Cloud mới vào hệ thống và sự kiện Người dùng loại một Cloud khỏi hệ thống.

3.6.1 Process Cloud Node Join Event

Quá trình xử lý sự kiện thêm một Cloud Node vào hệ thống được SCS thực hiện khi hệ thống nhận được yêu cầu của người dùng, với tham số đầu vào là thông tin định danh của Cloud Node. Các bước xử lý được thực hiện như sau:

- Kiểm tra thông tin định danh của Cloud Node
- Thêm Cloud Node vào Cloud Ring của User theo các nguyên tắc của Chrord: Cập nhật Succesor Node, Predecessor Node, Ring Table cho các Node

- Khởi chạy một Deamon Process thực hiện công việc di chuyển các Data Object nằm sai vị trí trong Cloud Ring mới.

thảo luận Trong khoảng thời gian di chuyển các Data Object, cần ngừng lại mọi truy cập từ người dùng tới Cloud Node mới cũng như Sucessor Node của Cloud Node mới. Ví dụ như chuyển hướng Replica,...bằng cách đánh Dấu Cloud Node mới và Successor của Cloud Node mới đang ở trong trạng thái đang Synchronize.

3.6.2 Process Cloud Node Leave Event

Quá trình xử lý sự kiện loại bỏ một Cloud Node vào hệ thống được SCS thực hiện khi hệ thống nhận được yêu cầu của người dùng, với tham số đầu vào là thông tin định danh của Cloud Node. Các bước xử lý được thực hiện như sau:

- Đánh dấu trạng thái của Cloud Node sắp bị loại bỏ là **Waiting_Leave Node**
- Khởi chạy một Deamon Process thực hiện công việc di chuyển các Data Object nằm sai vị trí trong Cloud Ring mới từ Cloud Node bị loại bỏ sang Successor Node của nó.
- Sau khi quá trình di chuyển dữ liệu hoàn tất, loại bỏ Cloud Node vào Cloud Ring của User theo các nguyên tắc của Chrord: Cập nhật Succesor Node, Predecessor Node, Ring Table cho các Node.

Trong quá trình thực hiện di chuyển dữ liệu giữa Cloud Node sắp bị loại bỏ sang Successor Node, mọi truy cập tới Cloud Node bị loại bỏ bị ngừng lại, thực hiện chuyển hướng sang các replica nằm ở các Cloud Node khác.

3.7 Manage and Process User Information in SCS System

Sau khi thiết kế các cơ chế để quản lý đối tượng Data Object và Các Cloud Node, bây giờ chúng ta cần xây dựng các cơ chế quản lý những người sử dụng hệ thống - User.

3.7.1 Create New User Process

Thao tác đầu tiên mà chúng ta cần xử lý là thao tác tạo một người dùng mới. Thông tin đầu vào cho quá trình tạo một người dùng mới là:

- User Name - Pasword hoặc thông tin định danh (GoogleAuth, FacebookAuth,...)
- Init Cloud List
- User Role (User hoặc Admin. Admin có quyền tạo User mới).

Các bước xử lý được tiến hành như sau:

- Kiểm tra thông tin xác thực của tài khoản User
- Tạo Cloud Ring cho User
- Tạo ra các dữ liệu quản lý tài khoản mà chúng ta đã tạo ra ở phần trên:
  - Data_Object_Connection_Information_List: Danh sách thông tin về các Data Object được truy cập trong thời gian gần đây.
  - Is_Updating_Data_Object_List: Danh sách các Data Object đang được cập nhật.
  - Wait_Delete_Object_List: Danh sách các Data Object đang được thực thi quá trình xóa.
- Cuối cùng, chúng ta lưu các thông tin trên vào cơ sở dữ liệu

Algorithm 1: Create Account

Ở đây ta thấy có sự xuất hiện của các khái niệm **Cloud Ring** và **Cloud Node**. Ý nghĩa của chúng là gì? Phần tiếp theo sẽ giải đáp ý nghĩa của các khái niệm này.

```
def CreateAccount(User_Authentication, Cloud_Information_List):

    # Validate User Authentication and Cloud Authentication
    Validate(User_Authentication)
    for Cloud_Information in Cloud_Information_List:
        Validate_Cloud_Authentication(Cloud_Information)

    Cloud_List = []
```

```
##
# Create Cloud Object for each Cloud in User's Cloud List
for Cloud_Information in Cloud_Information_List:
```

3.8 Handle Cloud Node Failure

Trong trường hợp có một Cloud Node bị lỗi, các dữ liệu thuộc Cloud đó có thể bị mất ? Có cần lưu trữ 1 backup lưu trữ Cloud Node đó đang chứa các Data Object nào không ?

3.9 Process Folder Object in SCS System

Kiểu lưu trữ được sử dụng trong hệ thống SCS là Object Storage. Tuy nhiên, do thói quen của người sử dụng, chúng ta cần cung cấp cho người dùng các Folder, để cho phép người dùng dễ dàng quản lý dữ liệu của họ theo cấu trúc cây thư mục quen thuộc.

Để tạo ra các Folder, chúng ta sẽ lưu các Folder có trong kho lưu trữ của người dùng dưới dạng một Data Object. Data Object này sẽ có nội dung là các thông tin về các File, Folder con mà Folder này chứa - File/Sub Folder Name và đường dẫn tới Object Metadata của các File/ Sub Folder đó.

Chúng ta sẽ định nghĩa các cơ chế xử lý các thao tác trên đối tượng Folder ở phần dưới đây.

3.9.1 Create Folder Object

Một đối tượng Folder Object được tạo ra khi người dùng gửi lên yêu cầu tạo ra một folder. Tên của một Folder Object sẽ phải kết thúc bằng dấu /. Nội dung bên trong một folder Object sẽ là tên + định danh/ địa chỉ truy cập của các object (Cả các Data Object thông thường và Folder Object) nằm trong Folder đó.

Một vấn đề xảy ra ở đây là chúng ta cần có thư mục gốc cho tài khoản User. Các thông tin về thư mục gốc (Định danh, địa chỉ truy cập) sẽ nằm trong thông tin về User, tức là thư mục gốc sẽ được tạo ra ngay trong quá trình tạo một User mới.

Từ vấn đề các Data Object có thể nằm trong một Folder nào đó, chúng ta nhận thấy sẽ một vấn đề sẽ xảy ra trong quá trình tạo mới một Data Object:

Lấy một trường hợp cụ thể: Ta có cấu trúc thư mục như sau:

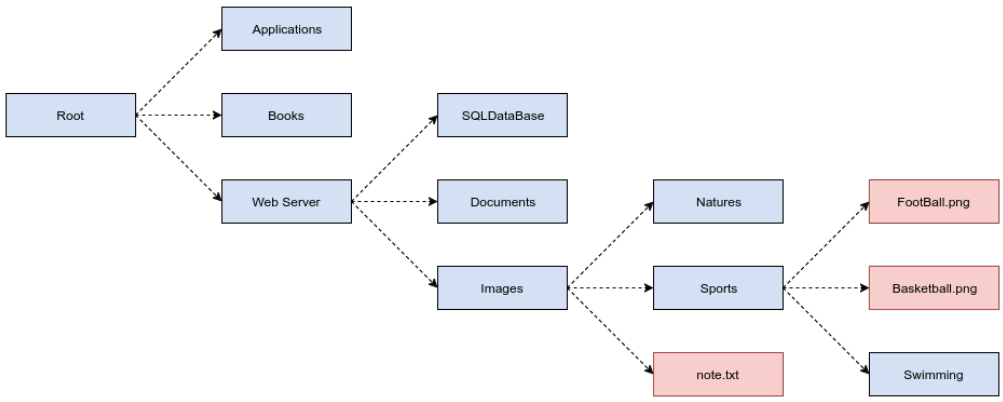

Figure 3: directory_example.png

Vấn đề thứ nhất: Khi đã có kiến trúc cây thư mục, làm sao để chúng ta phân biệt được 2 File cùng tên nằm trong 2 thư mục khác nhau?

Giải pháp đề xuất: tên của Data Object trên hệ thống đều phải là tên tuyệt đối. Ví dụ như trong cấu trúc cây thư mục phía trên, file `note.txt` thực chất sẽ có tên là `/Webserver/Images/note.txt`. Client sẽ truy cập tới Object thông qua tên tuyệt đối này.

Vấn đề thứ hai: Khi đã sử dụng một cấu trúc cây thư mục, thì khi một Data Object **x** được tạo ra trong một Folder, thì nội dung Folder chứa Data Object x phải được cập nhật bằng cách thêm 1 entry chứa thông tin về **x**

Nội dung của một **Folder Object** sẽ được thiết kế bao gồm:

- Danh sách thông tin về các Folder / Object là con của Folder Object đó

3.9.2 Update Folder Object Name

Tiếp theo, chúng ta cần xây dựng cơ chế để cập nhật tên một Folder

Giải pháp đề xuất: Khi một Folder đổi tên, tất cả mọi File/Folder nằm bên trong Folder và các Folder con của Folder đó đều bị cập nhật Object Metadata và đổi tên sang tên tương ứng với tên Folder mới.

Ví dụ, khi ta đổi tên Folder `Images` thành `Photos`, tất cả 6 File/Folder chứa trong `Images` và các thư mục con của `Images` đều phải cập nhật tên, ví dụ File `Football.png` phải cập nhật tên mới là `/Webserver/Photos/Sports/Football.png`

3.9.3 Delete Folder Object

Quá trình xử lý xóa một Folder được thực hiện bằng cách xóa tất cả mọi Data Object nằm trong Folder và các Folder con của Folder đó.

4. Thiết kế Biểu đồ lớp - Class Diagram của hệ thống