# MỤC LỤC

# LỜI MỞ ĐẦU

Cùng với sự phát triển của mạng máy tính, việc tính toán, quản lý ngày nay không chỉ đơn giản tập trung  trong máy tính đơn như trước nữa. Nó đòi hỏi các hệ thống tính toán phải được kết hợp từ một số lượng lớn các máy tính kết nối với nhau qua 1 mạng tốc độ cao. Chúng thường được gọi là các mạng máy tính hay còn có tên khác là các Hệ phân tán, nhằm ám chỉ tương phản với Hệ tập trung trước đây.

Ngày nay, hệ phân tán phát triển rất nhanh và được ứng dụng rộng khắp. Đó có thể là các dịch vụ thông tin phân tán, như các dịch vụ trên Internet chẳng hạn. Đó cũng có thể là các cơ sở dữ liệu phân tán như các hệ thống đặt vé máy bay, xe lửa…hoặc các hệ thống tính toán phân tán.

Mục đích của tiểu luận này nhằm nêu ra 1 cách khái quát nhất những khái niệm, những nguyên lý cơ bản của một hệ phân tán nói chung. Đồng thời phân tích sâu vào việc chia sẻ dữ liệu trong hệ phân tán, 1 trong những chức năng cơ bản nhất của hệ phân tán. Chúng ta thường gọi đó là hệ thống quản trị file phân tán. Ta cũng sẽ lần lượt nghiên cứu các mô hình khác nhau của hệ thống file phân tán như Sun NFS, Coda, Plan 9, XFS…

# I.Tổng quan về hệ phân tán

## I.1. Hệ phân tán là gì?

Có nhiều định nghĩa cho 1 hệ phân tán. Tuy nhiên, ta có thể định nghĩa hệ phân tán là một tập hợp bao gồm các máy tính tự trị được liên kết với nhau qua một mạng máy tính, và được cài đặt phần mềm hệ phân tán. Phần mềm hệ phân tán cho phép máy tính có thể phối hợp các hoạt động của nó và chia sẻ tài nguyên của hệ thống như phần cứng, phần mềm và dữ liệu.

*Một số tính chất quan trọng của một hệ phân tán:*

Thứ nhất chúng cho phép chúng ta chạy những ứng dụng khác nhau trên nhiều máy khác nhau thành một hệ thống duy nhất. Một ưu điểm khác của hệ phân tán đó là khi một hệ thống được thiết kế đúng cách, một hệ phân tán có thể có khả năng thay đổi tuỳ theo quy mô của hệ thống rất tốt. Tuy nhiên, tất cả mọi thứ đều có hai mặt của nó, một hệ phân tán cũng vậy bên cạnh những mặt ưu việt thì nó cũng có những nhược điểm đó là tính bảo mật kém

*Những ứng dụng của hệ phân tán:*

Cung cấp những thuận lợi cho việc tính toán đa mục đích đến những nhóm người dùng, tự động hoá công việc ngân hàng và hệ thống truyền thông đa phương tiện, ngoài ra chúng còn bao quát toàn bộ những ứng dụng thương mại và kĩ thuật. Hệ phân tán đã trở thành tiêu chuẩn để tổ chức về mặt tính toán. Nó có thể được sử dụng cho việc thực hiện tương tác hệ thống tính toán đa mục đích trong UNIX và hỗ trợ cho phạm vi rộng của thương mại và ứng dụng công nghiệp của những máy tính…

## I.2. Các đặc trưng cơ bản của hệ phân tán

### *a. Kết nối người sử dụng với tài nguyên*

Chia sẻ nguồn tài nguyên là một đặc tính cơ bản của hệ thống phân tán, nó là cơ sở cho những đặc tính khác và nó ảnh hưởng đến những kiến trúc phần mềm có sẵn trong các hệ phân tán. Các nguồn tài nguyên có thể là mục dữ liệu, phần cứng và các thành phần của phần cứng. Các nguồn tài nguyên được phân biệt từ một dữ liệu được quản lý với những quá trình xử lý đơn bởi nhu cầu của vài quá trình xử lý để chia sẻ chúng

### *b. Tính trong suốt (transparency)*

Một hệ phân tán được gọi là trong suốt nếu nó có khả năng che dấu tính rời rạc và những nhược điểm có thể của nó đối với người sử dụng cuối và người lập trình ứng dụng. Có 8 dạng trong suốt :

- Trong suốt truy cập : che dấu cách biểu diễn dữ liệu và cách thức truy cập tài nguyên.
- Trong suốt vị trí : che dấu vị trí thực của tài nguyên.
- Trong suốt di trú : che dấu khả năng di trú (di chuyển từ nơi này sang nơi khác) của tài nguyên.
- Trong suốt định vị lại : che dấu khả năng tài nguyên có thể di chuyển từ nơi này đến nơi khác ngay cả khi đang được sử dụng.
- Trong suốt bản sao : che dấu các bản sao được nhân ra.
- Trong suốt về tương tranh.

- Trong suốt về lỗi.
- Trong suốt truy cập nhanh.

***c. Tính mở (openess)***

Một hệ phân tán được gọi là có tính mở nếu nó có khả năng bổ sung thêm các dịch vụ mới mà không làm ảnh hưởng xấu đến các dịch vụ đã có.

***d. Tính co dãn (scalability)***

Một hệ phân tán được gọi là có tính co dãn nếu nó có thể thích nghi được với những sự thay đổi qui mô của hệ thống.. Tính co dãn thể hiện trên 3 khía cạnh.

- Dễ dàng bổ sung thêm tài nguyên và người sử dụng.
- Hệ thống thay đổi qui mô về mặt địa lý.
- Hệ thống thay đổi qui mô về quản trị.

***e. Tính chịu lỗi (Fault tolerance)***

Xử lý được những lỗi xảy ra trong quá trình làm việc. Bên cạnh tính chịu lỗi luôn đi kèm theo là khắc phục lỗi.

***f. Tính an toàn an ninh (security)***

## II. Các nguyên lý của hệ phân tán

Trong phần này, ta sẽ xem xét 1 cách tổng quan, tóm tắt các nguyên lý của hệ phân tán. Bởi nếu đi sâu thì bản thân trong mỗi nguyên lý lại còn có rất nhiều vấn đề cần phân tích. Có tổng cộng 7 nguyên lý cơ bản đối với 1 hệ phân tán, bao gồm:

1. Truyền thông (*Commmunication*).
2. Tiến trình (*Processes*).
3. Định danh (*Naming*).
4. Đồng bộ hóa (*Synchronization*).
5. Nhất quán và nhân bản (*Consistency & Replication*).
6. Chịu lỗi (*Fault tolerance*).
7. An toàn – an ninh (*Security*).

Sau đây ta đi vào phân tích sơ bộ từng nguyên lý của hệ phân tán.

### II.1. Truyền thông

Truyền thông giữa các tiến trình rất quan trọng trong một hệ phân tán. Truyền thông có thể chia thành 2 mức:

- Truyền thông ở mức mạng máy tính.
- Truyền thông ở mức midleware: bao gồm 4 mô hình được sử dụng rộng rãi: Gọi thủ tục từ xa (RPC), Triệu gọi đối tượng từ xa (RMI), Truyền thông hướng thông điệp (MOC) và Truyền thông hướng dòng (SOC).

#### II.1.1. Truyền thông ở mức mạng

Mô hình OSI được thiết kế cho phép các hệ thống mở truyền thông với nhau, phục vụ cho các ứng dụng phân tán.

Các tầng trong mô hình OSI:

1. Tầng vật lý (Physical layer)

2. Tầng liên kết dữ liệu (Data link)
3. Tầng mạng (Network)
4. Tầng vận chuyển (Transport)
5. Tầng phiên (Session)
6. Tầng trình diễn (Presentation)
7. Tầng ứng dụng (Application)

Mỗi tầng của mô hình OSI giải quyết một phần của việc giao tiếp. Và ở mỗi tầng lại có giao thức riêng của nó.

Mỗi hệ thống mở có các qui tắc về định dạng, nội dung, và ngữ nghĩa của thông điệp gửi và nhận – các qui tắc này được gọi là các giao thức (*protocol*). Để 1 nhóm các máy tính có thể truyền thông được với nhau, cần phải có các giao thức thống nhất giữa các máy tính. Có 2 loại giao thức khác nhau: giao thức hướng kết nối (*Connection-Oriented protocol*) phải thiết lập kết nối trước khi truyền.nhận dữ liệu, sau khi xong phải giải phóng kết nối. Và giao thức phi kết nối (*Connectionless-Oriented protocol*): không cần kết nối, thông tin được truyền ngay khi đã sẵn sàng.

### II.1.2. Truyền thông ở mức midleware

**a. Gọi thủ tục từ xa** (*Remote Procedure Call*):

RPC cho phép gọi các thủ tục nằm trên các máy khác. Khi 1 tiến trình trên máy A gọi 1 thủ tục trên máy B, thì tiến trình gọi trên máy A đó sẽ bị tạm dừng, thay vào đó sẽ thực thi thủ tục được gọi trên máy B. Phương pháp này được gọi là Gọi thủ tục từ xa (RPC). Đây là kĩ thuật được sử dụng rộng rãi nhất trong các hệ phân tán.

RPC xảy ra với các bước tóm tắt như sau:

1. Thủ tục client gọi client stub một cách bình thường.
2. Client stub xây dựng một thông điệp và gọi hệ điều hành cục bộ.
3. Hệ điều hành của client gửi thông điệp đến hệ điều hành từ xa.
4. Hệ điều hành từ xa gửi thông điệp cho server stub.
5. Server stub mở gói các tham số ra và gọi server.
6. Server thực thi và trả kết quả đến stub.
7. Server stub đóng gói nó vào thông điệp và gọi hệ điều hành cục bộ.
8. Hệ điều hành của server gửi thông điệp cho hệ điều hành của client.
9. Hệ điều hành của client trao thông điệp đến client stub.
10. Stub mở gói kết quả và trả về cho client.

**b. Triệu gọi đối tượng từ xa** (*Remote Object Invocation*):

Kỹ thuật hướng đối tượng được dùng rất phổ biến hiện nay trong việc phát triển các ứng dụng phân tán (distributed) và không phân tán (non-distributed). Một trong điều quan trọng của đối tượng đó là nó ẩn giấu đi những gì bên trong của nó với bên ngoài, mà nó sẽ chỉ cung cấp các giao diện (interface). Hướng tiếp cận này cho phép các đối tượng dễ dàng được thay thế và chỉnh sửa. RPC và ROI giúp ẩn dấu thông tin trong các hệ phân tán, tăng cường sự truy cập trong suốt.

**c. Truyền thông hướng thông điệp** (*Message Oriented Communication*)

Cơ chế truyền thông điệp có hai loại:

1. Truyền thông tạm thời hướng thông điệp.

2. Truyền thông hướng thông điệp dài lâu.

**d. Truyền thông hướng dòng** (*Stream Oriented Communication*)

Cũng có một số dạng truyền thông mà yếu tố đáp ứng thời gian đóng vai trò cốt yếu như dữ liệu âm thanh hoặc hình ảnh chẳng hạn. Do đó cần phải có 1 cơ chế truyền thông hướng dòng.

## II.2 Tiến trình

Tiến trình (*process*) là 1 chương trình đang trong quá trình thực thi nghĩa là một chương trình hiện đang được thực thi bởi một trong các bộ xử lý ảo của hệ điều hành. Đối với tiến trình thì vấn đề quản lý và lập lịch cho các tiến trình những vấn đề quan trọng cần giải quyết. Nhiều tiến trình có thể đồng thời chia sẻ cùng một CPU và các tài nguyên phần cứng khác.

### II.2.1 Luồng (threads) và mô hình đa luồng (multi-threading)

Luồng (*thread*) tương tự một tiến trình, tuy nhiên cũng có điểm khác biệt cơ bản giữa luồng với tiến trình. Một luồng là một đơn vị xử lý cơ bản trong hệ thống . Mỗi luồng xử lý tuần tự đoạn code của nó, sỡ hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng. Các luồng chia sẻ CPU với nhau giống như cách chia sẻ giữa các tiến trình: khi 1 luồng đang xử lý thì các luồng khác sẽ phải chờ cho đến lượt. Một luồng cũng có thể tạo lập các luồng con. Và 1 tiến trình có thể sỡ hữu nhiều luồng.

Một thuộc tính quan trọng của luồng là chúng cho phép khóa các lời gọi hệ thống mà không cần phải khóa toàn bộ tiến trình mà có luồng đang chạy.

Kỹ thuật đa luồng (*multi-threading*) cho phép khai thác tính song song khi thực thi một chương trình trên một hệ thống nhiều bộ xử lý. Khi đó, mỗi luồng được gán cho một CPU khác nhau trong khi dữ liệu dùng chung được lưu trữ trong bộ nhớ chính dùng chung. Phương pháp này thường được dùng trong môi trường UNIX.

### II.2.2 Di trú mã (code migration)

Di chuyển tiến trình từ máy này sang máy khác là một nhiệm vụ phức tạp và tốn kém nhưng nó sẽ cải thiện về mặt hiệu suất. Toàn bộ hiệu suất hệ thống có thể được nâng lên, nếu các tiến trình được di chuyển từ máy có mức độ xử lí quá nhiều đến máy có mức độ xử lí ít hơn. Di trú mã ũng có thể giúp tăng hiệu suất bằng cách tận dụng cơ chế song song mà không cần quan tâm đến vấn đề lập trình song song. Bên cạnh đó là tính mềm dẻo của mã di trú. Cách xây dựng ứng dụng phân tán truyền thống là tách rời ứng dụng thành các phần khác nhau, và quyết định phần nào được thực thi. Tuy nhiên nếu mã có thể di chuyển giữa các máy khác nhau ta có thể cấu hình động hệ phân tán.

Quy ước 1 tiến trình bao gồm 3 thành phần :

1. *Code segment* : chứa tập lệnh chương trình.

2. *Resource segment*: chứa các tham chiếu đến tài nguyên bên ngoài mà tiến trình cần.

3. *Execution segment* : chứa trạng thái thực thi hiện hành của tiến trình.

Có hai mô hình cơ bản cho việc di trú mã (di trú tiến trình) :

- Mô hình di động yếu (*weak mobility*): chỉ chuyển code segment kèm theo 1 số điều kiện ban đầu. Một đặc điểm của mô hình mã di trú yếu là một chương trình được chuyển luôn bắt đầu ở trạng thái khởi tạo của nó. Lợi thế của di trú yếu đó là tính đơn giản, nó chỉ cần máy đích có thể thực thi mã là được.

- Mô hình di động mạnh (*strong mobility*): chuyển luôn cả 3 thành phần. Đặc điểm này của mô hình này là tiến trình đang chạy có thể được dừng, sau đó di chuyển đến máy khác và rồi được thiết lập lại trạng thái đã bị dừng trước đó. Rõ ràng mô hình mã di trú di động mạnh tốt hơn nhiều so với mô hình mã di trú yếu, tuy nhiên sẽ khó thực hiện hơn.

### II.2.3 Tác tử mềm (software agents)

Agent có 2 đặc tính cơ bản đó là: tự trị và tương tác. Ngoài ra còn có các đặc tính riêng tuỳ theo từng tác tử, đó là:

1. Tính di động: tương ứng ta sẽ có tác tử di động (*mobile agent*).

Một tác tử di động là một tác tử có khả năng chuyển đổi giữa các máy khác nhau. Các tác tử di động đòi hỏi phải có tính di động mạnh. Các tác tử di động thường yêu cầu phải hỗ trợ mô hình mã di động mạnh (strong mobility), tuy không nhất thiết.

2. Tính thông minh: tương ứng ta sẽ có tác tử thông minh (*intelligent agent*).

Ngoài ra ta còn có các loại tác tử khác như:

- Tác tử giao diện (*interface agent*) giúp hỗ trợ cho người sử dụng trong việc chạy một hoặc nhiều ứng dụng.
- Tác tử thông tin (*information agent*) là tác tử liên quan mật thiết với tác tử giao diện. Chức năng chính của các tác tử này là quản lí thông tin từ nhiều tài nguyên khác nhau. Quản lí thông tin gồm sắp xếp, sàng lọc,…

| Thuộc tính | Chung cho tất cả các tác tử ? | Mô tả |
|---|---|---|
| Autonomous | Có | Có thể hoạt động trên chính nó |
| Reactive | Có | Đáp ứng đúng lúc để thay đổi môi trường. |
| Proactive | Có | Khởi tạo các hành động tác động đến môi trường. |
| Communicative | Có | Có thể trao đổi thông tin với người sử dụng và các tác tử khác. |
| Continuos | Không | Khoảng thời gian sống (*life*) tương đối dài |
| Mobile | Không | Có thể di trú từ nơi này đến nơi khác |
| Adaptive | Không | Có khả năng học |

*Một vài thuộc tính quan trọng của tác tử giữa các loại tác tử khác nhau.*

## II.3. Định danh

Các tên đóng vai trò quan trọng trong tất cả các hệ thống máy tính. Chúng được dùng để chia sẻ các tài nguyên, để định danh duy nhất các thực thể, để tham chiếu đến các nơi…Việc đặt tên tạo cơ sở cho phép các tiến trình có thể truy cập đến thực thể thông qua tên của chúng.

Trong một hệ thống phân tán, việc đinh danh thường được thực thi phân tán trên nhiều máy. Có ba vấn đề chính trong việc đinh danh trong hệ phân tán.

1. Đặt tên theo cách gần gũi với con người.
2. Các tên được sử dụng để định vị các thực thể di động.
3. Giải quyết cách tổ chức tên.

### II.3.1. Các khái niệm cơ bản

- Tên (*name*): là một chuỗi các bit hoặc các kí tự được dùng để tham chiếu đến 1 thực thể trong hệ phân tán.

- Để có thể thao tác trên một thực thể, ta cần phải truy cập (*access*) vào thực thể đó. Do đó chúng ta cần một điểm truy cập (*access point*). Tên của access point được gọi là địa chỉ (*address*). Một thực thể có thể có nhiều access point. Access point có thể thay đổi tại những thời điểm khác nhau. Ví dụ: khi bạn sử dụng một laptop và di chuyển từ vùng này đến vùng khác thì chắc chắn địa chỉ IP của máy sẽ bị thay đổi.
- Định danh (*identifier*): là một loại tên có những đặc tính sau:
  1. Một định danh tham chiếu nhiều nhất đến 1 thực thể.
  2. Mỗi thực thể được tham chiếu nhiều nhất bởi 1 định danh.
  3. Một định danh luôn tham chiếu tới cùng 1 thực thể.

Nhờ dùng định danh, chúng ta dễ dàng hơn khi đề cập đến một thực thể. Chúng ta cũng không thể sử dụng địa chỉ làm định danh được vì address có thể thay đổi.

- Không gian tên (*namespace*): là 1 cách tổ chức các tên trong hệ phân tán. Biểu diễn bằng 1 đồ thị có hướng - đồ thị tên (name graph).
- Phân giải tên (*name resolution*): duyệt đồ thị tên theo namepath tìm kiếm tên hoặc định danh của 1 thực thể.

## II.3.2. Định vị thực thể di động

Một phương pháp phổ biến để hỗ trợ các thực thể di động trong mạng có phạm vi lớn đó là *home-based*, bằng cách đưa ra 1 địa điểm chủ (*home location*), nơi sẽ giữ lại vết của địa điểm hiện tại của thực thể. Trong thực tế thì địa điểm chủ thường được chọn tại nơi mà thực thể được tạo ra. Một ví dụ về kỹ thuật home-based là trong *Mobile IP* (IP di động) Mỗi host di động sử dụng địa chỉ IP cố định. Tất cả các giao tiếp đến địa chỉ IP đó đầu tiên sẽ được đến *mobile host's home agent* (nơi quản lý các host di động). Nơi quản lý này được đặt trên một mạng LAN tương ứng với địa chỉ mạng chứa trong địa chỉ IP của host di động. Bất cứ khi nào host di động chuyển tới một mạng khác, nó đều yêu cầu một địa chỉ tạm thời để dùng cho các hoạt động giao tiếp. Địa chỉ chuyển tiếp (*care of address*) này được đăng ký tại *home agent*.

Khi home agent nhận được một gói tin gửi cho một host di động, nó sẽ tìm kiếm địa điểm hiện tại của host di động đó. Nếu host di động đó đang ở mạng cục bộ hiện hành thì gói tin sẽ được chuyển tiếp một cách dễ dàng. Ngược lại, nó sẽ tạo một đường ngang tới nơi mà host di động đang nằm bằng cách gói (*wrap*) dữ liệu trong một gói IP và gửi đến địa chỉ chuyển tiếp mà nó đang quản lý. Cùng lúc đó thì nơi gửi gói tin đi được thông báo của địa điểm hiện tại của host di động. Chú ý rằng địa chỉ IP được sử dụng một cách hiệu quả khi có 1 định danh cho host di động.

Ngoài hướng tiếp cận trên còn nhiều hướng khác nữa để giải quyết vấn đề thực thể di động.

## II.3.2. Xoá bỏ những thực thể không còn được tham chiếu

Để làm giảm bớt những vấn đề liên quan đến việc xóa những thực thể không còn được tham chiếu, hệ phân tán cung cấp 1 số các tiện ích để tự động xóa một thực thể khi nó không còn cần nữa. Những tiện ích đó được gọi chung là hệ thống thu gom rác phân tán (*distributed garbage collectors*). Trong phần này chúng ta sẽ tìm hiểu mối quan hệ giữa việc định danh (*naming*) và tham chiếu các thực thể (*referencing entities*), và việc tự động thu gom những thực thể không còn được tham chiếu nữa.

## II.4. Đồng bộ hóa

Trong hệ phân tán, việc tính thời gian của mỗi máy tính là khác nhau, vì vậy cần phải có một khái niệm gọi là thời gian vật lý để thống nhất về thời gian giữa các máy tính trên toàn cầu. Bên cạnh đó, khi các tiến trình cùng yêu cầu một đơn vị dữ liệu trong cùng một thời gian thì sẽ không tránh khỏi tương tranh bất khả kháng. Những điều trên chính là các vấn đề chính cần được giải quyết trong việc đồng bộ hoá.

### II.4.1. Đồng bộ hoá đồng hồ vật lý

- Mỗi máy tính đều cài đặt 1 đồng hồ vật lý, đó là các mạch đếm xung nhịp. Thật ra chúng không phải đồng hồ trong quan điểm thông thường. Nhiều khi ta gọi chúng là các bộ đếm định thời (*timer*). Bộ định thời trong máy tính thường là tinh thể thạch anh chạy rất chính xác. Khi được giữ ở một hiệu điện thế, tinh thể thạch anh dao động với tần số ổn định tùy thuộc vào loại tinh thể đó. Liên kết với tinh thể thạch anh là hai thanh ghi, 1 bộ đếm (*counter*) và thanh ghi giữ (*holding register*).

Mặc dù tần số của bộ dao động tinh thể luôn khá ổn định, nó không thể đảm bảo các tinh thể trong các máy tính khác nhau đều chạy chính xác cùng tần số. Thực tế khi một hệ thống có nhiều máy tính thì tất cả các tinh thể sẽ chạy với tần số khác nhau chút ít, dần gây ra sự mất đồng bộ và giá trị đọc ra sẽ khác nhau. Sự khác nhau về giá trị thời gian được gọi là sự sai lệch của đồng hồ. Và kết quả của sự sai khác này là chương trình có sự đòi hỏi thời gian liên kết với file, đối tượng, tiến trình, hay thông điệp sẽ không còn chính xác.

Trong một số hệ thống thời gian thực, thời gian đồng hồ là rất quan trọng. Đối với những hệ thống này đòi hỏi có các đồng hồ vật lí ngoài. Việc dùng nhiều đồng hồ vật lí như thế sẽ nảy sinh ra 2 vấn đề:

1. Làm thế nào để đồng bộ chúng với đồng hồ thế giới thực.
2. Làm thế nào để đồng bộ chúng với nhau.

- Việc đồng bộ giữa các đồng hồ vật lý cần phải dựa vào 1 thời gian chuẩn có giá trị toàn cầu – thời gian phối hợp toàn cầu UTC (*universal coordinated time*).

- Nếu các máy tính có các *wwv receiver* thì việc đồng bộ hóa sẽ được thực hiện theo UTC. Ngược lại, nếu các máy tính không có *wwv receiver* thì phải sử dụng các giải thuật đồng bộ hóa đồng hồ vật lý. Có 3 giải thuật phổ biến, đó là:

    1. Giải thuật Cristian.
    2. Giải thuật Berkeley.
    3. Giải thuật trung bình.

Tất cả các thuật toán đều có cùng mô hình hệ thống cơ bản. Mỗi máy xem như có một bộ đếm thời gian, nó tạo ra một ngắt H lần trong một giây. Gọi giá trị của đồng hồ này là C. Khi thời gian UTC là t, thì giá trị của đồng hồ trên máy p sẽ là Cp(t). Trong một thế giới lí tưởng chúng ta có Cp(t) = t cho tất cả p và t. Hay nói cách khác, lí tưởng là C(p).t = 1.

Bộ định thời thực không ngắt chính xác H lần trong một giây. Theo lí thuyết, bộ định thời với H = 60 cần phát ra 216000 *tick* trong một giờ. Thực tế những sai số tương đối đạt được với các chip đếm thời gian hiện đại đạt khoảng $10^{-5}$, có nghĩa là một máy nào đó có thể lấy giá trị từ 215998 đến 216002 *tick* trong một giờ. Một cách chính xác hơn, tồn tại một hằng số ρ thoả mãn:

$1 - \rho \leq dC.dt \leq 1 + \rho$

mà bộ định thời làm việc chính xác. Hằng số ρ được xác định bởi nhà sản xuất và được gọi là *Maximum Drift Rate*.

## II.4.2. Đồng bộ hoá đồng hồ logic

### a. Tem thời gian Lamport (1978)

Để đồng bộ đồng hồ logic, Lamport định nghĩa một mối quan hệ gọi được gọi là *happens-before* (xảy ra - trước khi). Sự kiện a xảy ra trước sự kiện b (Ký hiệu: a→b) được gọi là đúng nếu:

1. a, b là hai sự kiện xảy ra trong cùng 1 tiến trình, và a xảy ra trước b.
2. a, b không thuộc một tiến trình nhưng a gửi một thông điệp đi và b là sự kiện nhận thông điệp đó.

Happens – before là một quan hệ kéo theo, vì thế nếu a→b và b→c thì ta sẽ có a→c. Nếu hai sự kiện x và y xảy ra trong hai tiến trình khác nhau (thậm chí không gián tiếp qua đối tượng thứ ba) thì x→y là không đúng, và cả y→x cũng thế. Những sự kiện này được gọi là đồng thời.

Nếu ta có sự kiện x, thì ký hiệu C(x) là tem thời gian của x, thỏa mãn các điều kiện sau:

- Nếu a xảy ra trước b trong cùng 1 tiến trình thì C(a) < C(b).
- Nếu a và b biểu diễn tương ứng việc gửi nhận thông điệp thì C(a) < C(b).
- Mọi sự kiện phân biệt a và b thì C(a) ≠ C(b).

### b. Nhãn thời gian vector (Vector Timestamps)

Một nhãn thời gian vector VT(a) được gán cho một sự kiện a có thuộc tính. Nếu sự kiện a trước sự kiện b thì ta có VT(a) < VT(b). Vector nhãn thời gian được xây dựng bằng cách để mỗi tiến trình $P_i$ duy trì một vector $V_i$ với hai thuộc tính sau:

1. $V_i[i]$ là số sự kiện đã xảy ra cho đến bây giờ ở P*i*.
2. Nếu $V_i[j] = k$ thì $P_i$ hiểu rằng k sự kiện đã xảy ra ở Pi.

Thuộc tính đầu tiên được duy trì bởi việc tăng $V_i[i]$ đồng thời với mỗi sự kiện mới xảy ra ở $P_i$. Thuộc tính thứ hai được duy trì bằng các *piggy-backing* vector cùng với các thông điệp được gửi

## II.4.3. Trạng thái tổng thể (global state)

## II.4.4. Các giải thuật bầu chọn (election algorithm)

Nhiều thuật toán phân tán đòi hỏi 1 tiến trình đóng vai trò như điều phối viên (*coordinator*), người khởi xướng (*initiator*), hoặc không thì thực hiện 1 vai trò đặc biệt. Trong phần này ta sẽ xem xét các thuật toán để bầu chọn điều phối viên. Thuật ngữ điều phối viên được dùng như 1 tên tổng quát cho tiến trình đặc biệt.

Nếu tất cả các tiến trình đều giống hệt nhau, không có các đặc điểm phân biệt, thì không có cách nào để chọn ra một tiến trình đặc biệt. Vì thế chúng ta sẽ giả sử rằng mỗi tiến trình có một con số duy nhất, ví dụ như địa chỉ mạng của nó (để đơn giản ta cũng cho rằng mỗi tiến trình trên 1 máy). Nói chung, các thuật toán bầu chọn sẽ cố gắng xác định tiến trình với số tiến trình (*process number*) là cao nhất và chỉ định nó là điều phối viên. Các thuật toán khác nhau thì sẽ khác nhau trong cách xác định này.

### a. Giải thuật áp đảo (bully algorithm – Garcia Molina, 1982)

Khi một tiến trình bất kì chú ý rằng điều phối viên không còn đáp ứng các yêu cầu nữa, thì nó bắt đầu một cuộc bầu cử. Một tiến trình P sẽ tổ chức 1 cuộc bầu theo các bước sau:

1. P gửi một thông điệp bầu cử (ELECTION) cho tất cả các tiến trình với số tiến trình cao hơn.
2. Nếu không có ai phản hồi, P sẽ thắng cử và trở thành điều phối viên.
3. Nếu có ai đó với số tiến trình cao hơn trả lời lại, nó chuyển lại, và công việc của P đã xong.

**b. Giải thuật vòng (ring algorithm)**

Giả sử rằng các tiến trình đã được sắp theo trật tự vật lí và logic để mỗi tiến trình biết được tiến trình kế tiếp là ai. Khi một tiến trình thông báo không tìm thấy điều phối viên, nó xây dựng một thông điệp bầu cử gồm số hiệu riêng của nó và gửi thông điệp cho tiến trình kế tiếp nó. Nếu tiến trình kế tiếp đã *down*, bên gửi sẽ bỏ qua và nhảy đến tiến trình kế tiếp trên vòng, cho đến khi một tiến trình đang chạy được xác định. Tại mỗi bước, tiến trình gửi sẽ thêm số hiệu tiến trình (*process number*) của chính nó vào danh sách trong thông điệp để nó trở thành 1 ứng viên trong việc bầu điều phối viên.

### II.4.5. Loại trừ nhau (mutual exclusion)

- Giải thuật tập trung (centralized algorithm).
- Giải thuật phân tán (distributed algorithm).
- Giải thuật sử dụng token (token ring algorithm).

### II.4.6. Giao tác phân tán (distributed transaction)

**a. Các tính chất của giao tác - ACID**

1. "A" (nguyên tử - Atomic): đối với thế giới bên ngoài thì giao tác không thể phân chia được nữa.
2. "C" (nhất quán - consistent): giao tác không xâm phạm các bất biến của hệ thống.
3. "I" (cách ky - isolated): các giao tác đồng thời không gây trở ngại cho nhau.
4. "D" (lâu bền-durable):khi 1 giao tác đã cam kết thì các thay đổi là kéo dài lâu bền.

**b. Phân loại giao tác (Classifications of Transactions)**

- Giao tác phẳng (flat transaction)

Là giao tác đơn giản nhất, thỏa mãn 4 tính chất ACID trên. Hạn chế chính của giao tác phẳng là chúng không cho phép tách riêng các kết quả được cam kết (*commited*) hay hủy bỏ (*aborted*). Nói cách khác mức độ của tính nguyên tố của giao tác phẳng là yếu.

- Giao tác lồng nhau (nested transaction)

Khắc phục các hạn chế của giao tác phẳng ta sử dụng giao tác lồng nhau. Một giao tác lồng nhau có cấu trúc từ một số giao tác con, hay nói cách khác là trong giao tác lại bao gồm các giao tác khác. Mỗi giao tác con cũng có thể thực thi một hay nhiều giao tác con của chính nó.

- Giao tác phân tán (distributed transaction).

Để điều khiển tương tranh, có 2 tiếp cận: Điều khiển tương tranh "bi quan" (pessimistic concurrency control) và điều khiển tương tranh "lạc quan" (optimistic concurrency control).

## II.5. Nhất quán và nhân bản

Trong hệ phân tán, việc sử dụng các bản sao đóng vai trò khá quan trọng. Có những lý do sau để ta dùng các bản sao:

1. Tăng tính tin cậy. Nếu một hệ thống file được sao lưu nó có thể tiếp tục làm việc sau khi gặp sự cố bằng cách chuyển đến làm việc với các bản sao khác. Có nhiều bản sao giúp bảo vệ chống được việc dữ liệu bị hư hỏng.
2. Tăng hiệu năng, từ đó tăng tính sẵn sàng sử dụng tài nguyên.

Tuy nhiên, ta cũng phải trả giá cho việc sử dụng các bản sao. Vấn đề được đặt ra ở đây là làm thế nào để đảm bảo tính nhất quán.

Có 2 nhóm mô hình nhất quán:

- Các mô hình nhất quán lấy dữ liệu làm trung tâm (data centric consistency models)
- Các mô hình nhất quán lấy client làm trung tâm (client centric consistency models).

### II.5.1. Các mô hình nhất quán lấy dữ liệu làm trung tâm

#### a. Các mô hình nhất quán mạnh

Căn cứ vào bản thân các thao tác đơn lẻ đọc.ghi trên các dữ liệu dùng chung, ta có các mô hình nhất quán mạnh. Bao gồm:

- Mô hình nhất quán chặt (strict consistency): khi thỏa điều kiện sau: Bất kì thao tác đọc nào trên đơn vị dữ liệu x thì đếu sẽ trả về một giá trị tương ứng với thao tác ghi gần nhất trên x.
- Mô hình nhất quán tuần tự (sequential consistency): là 1 mô hình nhất quán yếu hơn 1 ít so với mô hình nhất quán chặt. Nó được đưa ra bởi Lamport (1979), theo ngữ cảnh bộ nhớ được chia sẻ cho các hệ thống đa vi xử lý Mô hình nhất quán tuyến tính.
- Mô hình nhất quán tuyến tính (linearizability consistency): mô hình nhất quán này yếu hơn mô hình nhất quán chặt nhưng lại mạnh hơn mô hình nhất quán tuần tự
- Mô hình nhất quán nhân quả (causal consistency): ở phần trước chúng ta cũng đã nói đến tính nhân quả khi đề cập đến vector tem thời gian. Nếu sự kiện B bị tác động và ảnh hưởng bởi sự kiện A trước đó, tính nhân quả đòi hỏi rằng mọi người phải thấy A trước khi thấy B.
- Mô hình nhất quán FIFO.

#### b. Các mô hình nhất quán yếu

- Mô hình nhất quán yếu (weak consistency): mô hình này có những đặc điểm sau:

1. Truy cập đến các biến đồng bộ hoá (synchronization variables) được kết hợp với một kho dữ liệu (data store), nhất quán một cách tuần tự.
2. Không có thao tác trên một biến đồng bộ được phép thực hiện cho đến khi tất cả các thao tác ghi trước đó đã hoàn thành ở mọi nơi.

3. Không có thao tác đọc ghi trên các đơn vị dữ liệu được phép thực hiện cho đến khi tất cả các thao tác trước đó đến các biến đồng bộ đã được thực hiện.

- Mô hình nhất quán đi ra (release consistency): nói chung một kho dữ liệu được gọi là nhất quấn nhẹ nếu nó tuân theo các qui tắc sau:

1. Trước khi một thao tác đọc ghi hoặc ghi trên đơn vị dữ liệu chia sẻ được thực hiện, tất cả yêu cầu đã thực hiện trước đó bởi các tiến trình phải được hoàn tất thành công.

2. Trước khi một sự giải phóng (đi ra - release) được phép thực thi, tất cả các thao tác đọc và ghi trước đó đã thực hiện bởi các tiến trình phải được hoàn tất.

3. Sự truy cập đến các biến đồng bộ hoá là nhất quán FIFO

- Mô hình nhất quán đi vào (entry consistency)

### II.5.2. Các mô hình nhất quán lấy client làm trung tâm

- Mô hình nhất cuối cùng (eventual consistency).
- Mô hình nhất quán đọc đều (monotonic reads).
- Mô hình nhất quán ghi đều (monotonic writes).
- Mô hình nhất quán đọc thao tác ghi (read your writes).
- Mô hình nhất quán ghi theo sau đọc (writes your reads).

### II.5.3. Các giao thức phân tán

#### a. Sắp đặt các bản sao (replica placement)

Vấn đề thiết kế chính đặt ra cho kho dữ liệu phân tán, là quyết định xem khi nào, ở đâu, và do ai sắp đặt các bản sao của kho dữ liệu. Có 3 loại bản sao như sau:

- Các bản sao thường trực (permanent replicas).
- Các bản sao máy chủ khởi tạo (server-initiated replicas)
- Các bản sao máy khách khởi tạo (client-initiated replicas)

#### b. Lan truyền cập nhật (update propagation)

Vấn đề được giải quyết ở đây là làm thế nào để lan truyền các cập nhật từ 1 bản sao đến các bản sao khác.

## II.6. Chịu lỗi

Một đặc tính riêng biệt của hệ phân tán giúp phân biệt với hệ thống máy đơn là khái niệm của lỗi riêng phần (*partial failure*). Một lỗi riêng phần có thể xảy ra khi một thành phần trong hệ thống bị sự cố, và lỗi này có thể ảnh hưởng đến hoạt động chung của các thành phần khác. Một mục tiêu quan trọng trong thiết kế hệ phân tán là xây dựng nên 1 hệ thống mà nó có thể tự động hồi phục lại các lỗi riêng phần mà không làm ảnh hưởng gì nghiêm trọng đến toàn bộ hiệu năng.

### II.6.1. Các khái niệm cơ bản

Sau đây là các đòi hỏi cho 1 hệ phân tán:

- Tính sẵn sàng (*availability*): hệ thống sẵn sàng sử dụng ngay bất kỳ lúc nào. Nói chung, thuộc tính này đảm bảo cho hệ thống luôn hoạt động một cách chính xác ở mọi thời điểm được yêu cầu và sẵn sàng hoạt động theo yêu cầu của người sử dụng.

- Tính tin cậy (*reliability*): hệ thống chạy liên tục mà không bị lỗi. Ngược với tính sẵn sàng, tính tin cậy được định nghĩa liên quan đến một khoảng thời gian thay vì một điểm thời gian. Một hệ thống có độ tin cậy cao có thể làm việc liên tục mà không bị gián đoạn trong khoảng thời gian khá dài

- Tính an toàn (*safety*): hệ thống có lỗi tạm thời thì vẫn không có thảm hoạ xảy ra. Ví dụ như các tiến trình hoạt động trong hệ thống điều khiển năng lượng nguyên tử hay đưa con ngườI lên vũ trụ cần một độ an toàn cao. Nếu các hệ thống điều khiển như thế tạm thời hỏng trong 1 khoảnh khắc, hậu quả thật kinh khủng.

- Tính bảo trì được (*maintainability*): khả năng này giúp hệ thống bị lỗi nhanh chóng khắc phục lỗi.

Nếu một hệ thống có được cả 4 tính trên thì được gọi là một hệ thống có độ tin cậy (*dependability*).

Khi phân loại các lỗi của hệ phân tán, ta có 3 loại lỗi sau: Lỗi nhất thời (*transient faults*) là những lỗi chỉ xảy ra 1 lần, mất đi, và không lặp lại nữa. Lỗi lặp (*intermittent faults*) là những lỗi xảy ra, mất đi, và sau đó lặp lại. Lỗi lâu dài hay còn gọi là lỗi thường trực (*permanent faults*).

Bên cạnh các loại lỗi trên, ta còn có các mô hình lỗi sau: lỗi sụp đổ (*crash failure*) khi server bị treo, lỗi bỏ sót (*omission failure*) khi server không đáp ứng được nhu cầu gửi hoặc nhận, lỗi thời gian (*timing failure*) khi thời gian có trả lời nhưng lại quá thời gian quy định, lỗi đáp ứng (*respond failure*) server có trả lời nhưng không đúng, và cuối cùng là lỗi tuỳ tiện (*arbitrary failure*) khi server trả lời 1 cách tuỳ tiện vào các thời điểm tùy tiện.

### II.6.2. Che dấu những hư hỏng bằng sự dư thừa.

Nếu một hệ thống phải chịu lỗi, cách tốt nhất là cố gắng ẩn đi sự xuất hiện của các lỗi từ những tiến trình khác. Kỹ thuật chính cho việc che dấu lỗi đó là sử dụng dư thừa.

Có 3 loại dư thừa:

- Dư thừa thông tin: bit thừa được thêm vào để cho phép hồi phục những bit đã bị sai khác, bị lỗi. Ví dụ, mã Hanmming có thể được thêm vào dữ liệu truyền để hồi phục khi có nhiễu trên đường truyền.

- Dư thừa thời gian: một hành đã được thực hiện, và sau đó, nếu cần, nó lại được thực hiện lần nữa.

- Dư thừa vật lý: là 1 kỹ thuật rất phổ biến cho việc chịu lỗi. Nó được dùng trong sinh vật học (động vật có vú thì có 2 mắt, 2 tai, 2 phổi…), trong kỹ thuật hàng không (Boeing 747 có đến 4 động cơ nhưng chỉ dùng 3 cái để bay), trong thể thao (nhiều trọng tài chỉ trong 1 tình huống phạm lỗi). Nó cũng được dùng để chịu lỗi trong các mạch điện tử…

### II.6.3. Khôi phục tiến trinh (process resilience)

Để khôi phục tiến trình, ta tổ chức thành các nhóm tiến trình giống nhau. Các nhóm tiến trình trên có thể động, nói cách khác chúng có thể thêm vào hoặc bớt đi các thành viên. Tuy nhiên, vấn đề cần giải quyết ở đây sẽ là quản lý các nhóm tiến trình đó như thế nào. Nếu đứng về phương diện nhóm, ta sẽ có giải pháp cho vấn đề trên là phân thành nhóm ngang hàng và nhóm phân cấp. Tương ứng, nếu đứng trên phương diện thành viên của nhóm, ta sẽ có: quản lý tập trung và quản lý phân tán.

Ngoài ra, ta còn phải tính đến nên có bao nhiêu bản sao tiến trình. Đó là vấn đề che dấu lỗi và cơ chế dùng bản sao.

### II.6.4. Truyền thông theo mô hình client.server tin cậy

Trong nhiều trường hợp, tính chịu lỗi trong hệ phân tán chỉ tập trung vào các tiến trình bị lỗi. Tuy nhiên, chúng ta cũng cần quan tâm đến các lỗi truyền thông. Trong thực tế, khi xây dựng các hệ thống truyền thông tin cậy, vấn đề cốt yếu là phải kiểm soát các lỗi bỏ sót và lỗi sụp đổ. Các lỗi tuỳ tiện có thể xảy ra dưới dạng sao chép các thông điệp, kết quả là trong các hệ thống mạng, các thông điệp này có thể được lưu giữ trong một thời gian khá dài và sau đó lại được đưa lên mạng sau khi người gửi đã gửi đi một thông điệp truyền lại.

Các phương pháp truyền thông tin cậy:

- Truyền thông điểm tới điểm (Point to point communication)
- Ngữ nghĩa RPC trong các lỗi hiện thời (RPC Semantics in the Presence of Failures)

### II.6.5. Truyền thông theo mô hình nhóm tin cậy

Mỗi tiến trình thiết lập 1 kết nối điểm - điểmm với tiến trình khác mà nó muốn truyền thông. Hiển nhiên, một tổ chức như vậy là không hiệu quả khi nó có thể hoang phí dải thông mạng. Tuy nhiên, nếu số lượng tiến trình nhỏ, thì như thế việc đạt được độ tin cậy xuyên suốt nhiều kênh điểm-điểm tin cậy là khá đơn giản, không hề phức tạp.

### II.6.6. Cam kết phân tán (distributed commit)

Giải thuật cam kết 2 pha (2PC) với 2 pha là: pha bầu cử (*voting phase*) và pha quyết định (*decision phase*). Tuy nhiên, để khắc phục trường hợp điều phối viên bị sụp đổ, người ta đã đề xuất ra giải thuật cam kết 3 pha – 3PC.

### II.6.5. Phục hồi lỗi (recovery)

Một khi có lỗi xảy ra, thì việc hồi phục lỗi là điều hết sức cần thiết.

- Phục hồi lỗi lùi (*backward recovery*): đưa hệ thống từ trạng thái lỗi hiện hành trở về trạng thái đúng trước đó.
- Phục hồi lỗi tiến (*forward recovery*): một khi hệ thống đã đi vào trạng thái lỗi, thay vì phải quay lui, ta cố gắng đem hệ thống đến trạng thái đúng mới ở trước mà tại đó ta lại có thể tiếp tục thực hiện bình thường.

## II.7. An toàn – an ninh

### II.7.1. Mở đầu

#### a. Mối đe doạ bảo mật, các chính sách và cơ chế an toàn – an ninh

Bảo mật trong hệ thống máy tính là chúng ta cố gắng bảo vệ các dịch vụ và dữ liệu của hệ thống, chống lại các sự đe doạ. Sau đây là 4 loại đe doạ vấn đề bảo mật mà chúng ta cần xem xét:

1. *Interception* (Chặn): nhằm nói đến tình huống một phần tử nào đó không được uỷ quyền mà lại giành được quyền truy cập vào các dịch vụ hoặc dữ liệu. Hay nói cách khác đó là các truy cập trái phép, nghe trộm.Ví dụ: Giao tiếp thông qua phương tiện truyền thông giữa hai đối tượng bị nghe trộm bởi một đối tượng nào đó. Sự chặn đứng cũng xảy ra khi dữ liệu bị sao chép không hợp pháp khi vào thư mục của người khác trong hệ thống file.

2. *Interruption* (Ngắt): nhằm nói đến tình huống trong đó các dịch vụ hoặc dữ liệu trở nên mất tác dụng, bị phá huỷ, không tìm thấy…ví dụ như khi một file bị hư hỏng hay bị mất. Trong trường hợp này các dịch vụ hoặc dữ liệu đã bị một kẻ nào đó có chủ tâm phá hoại nhằm làm cho người khác không thể truy cập được.

3. *Modification* (Biến đổi): sự biến đổi gây ra sự thay đổi dữ liệu làm cho dữ liệu không còn giữ nguyên được những đặc điểm ban đầu. Thường dữ liệu bị chặn đứng tức có kẻ truy nhập bất hợp pháp và sau đó thay đổi dữ liệu truyền đi, thay đổi chương trình để bí mật truy nhập vào các hoạt động của người được phép sử dụng hợp pháp chương trình.

4. *Fabrication* (Chế tạo): nhằm chỉ đến trạng thái trong đó việc phát sinh các dữ liệu thêm vào hay các hoạt động bất thường. Ví dụ: Một kẻ xâm nhập cố thêm cho bằng được 1 mục (entry) vào file mật khẩu hay cơ sở dữ liệu.

Các cơ chế bảo mật:

- *Encryption* (Mật mã): là nền tảng của bảo mật trong hệ thống máy tính. Mã hoá sẽ chuyển đổi dữ liệu thành một dạng nào đó khiến cho một kẻ xâm nhập trái phép không hiểu được. Mã hóa cũng cung cấp cơ chế cho phép kiểm tra tính toàn vẹn dữ liệu.

- *Authentication* (Xác thực): được sử dụng để nhận dạng chính xác các yêu cầu của user, client, server…Trong trường hợp của client, tiền đề cơ bản trước khi server đáp ứng nhu cầu của client là server phải nhận dạng được client. Thông thường, server nhận dạng client thông qua password, ngoài ra còn các phương pháp nhận dạng khác.

- *Authorization* (Uỷ quyền): sau khi client được xác thực, điều cần thiết là kiểm tra xem khi nào client được cấp quyền để bắt đầu thực hiện các hành động đáp ứng yêu cầu từ user. Hay nói cách khác, uỷ quyền giúp kiểm tra các quyền được thực hiện các hành động yêu cầu.

- *Auditing* (Kiểm toán): công cụ kiểm toán được sử dụng để phát hiện ra client nào truy nhập vào cái gì và bằng cách nào. Mặc dù kiểm định không thật sự cung cấp bất cứ tính năng bảo vệ nào chống lại sự đe doạ bảo mật nhưng nó thật sự hữu dụng trong việc phân tích tìm ra những lỗ hổng bảo mật để sau đó có cách chống lại xự xâm nhập. Cũng chính vì lý do này, các hacker không bao giờ để lại dấu vết có thể dẫn đến việc phát hiện ra họ.

## II.7.2. Kênh an toàn (secure channels)

Có 3 phương pháp xác thực chính:

- Xác thực dựa trên khoá bí mật: đây được xem là một phương pháp được là phổ biến trong việc chuyển các thông tin quan trọng ở thời điểm mà máy tính chưa phát triển. Khi máy tính được ra đời và phát triển thì việc mã hoá bằng khoá bí mật được thực hiện trên các hệ thống máy tính. Nó được mã hoá với số lượng khoá đa dạng hơn và các hàm dùng để mã hoá cũng phức tạp hơn. Việc cải tiến các khoá này được xem là vấn đề cần thiết. Cải tiến ở đây là cải tiến về kích thước của khoá phải được mở rộng và các hàm mã hoá khoá phải được phải được lựa chọn cẩn thận để sao cho thông tin bị lấy đi nhưng không sử dụng được do không giải mã được nó. Một trong số các thuật toán được xem như đáp ứng được yêu cầu cải tiến này đó là thuật toán DES.

- Xác thực sử dụng 1 trung tâm phân phối khoá: một trong những vấn đề khi sử dụng khóa bí mật để xác thực đó là tính co dãn. Nếu hệ phân tán có N máy chủ, thì hệ thống nói chung cần quản lí N(N-1).2 khóa, và mỗi máy chủ phải quản lí N-1 khóa (vì mỗi máy chủ đều đòi hỏi chia sẻ một khoá chia sẻ với mỗi N-1 máy chủ khác). Một sự lựa chọn khác là sử dụng một trung tâm phân phối khóa (KDC-*Key Distribution Center*). KDC chia sẻ khóa bí mật với các máy chủ, nhưng không có hai máy chủ nào có cùng một khóa chia sẻ. Như vậy nhờ sử dụng KDC chúng ta chỉ cần quản lí N khóa thay vì N(N-1).2, điều này rõ ràng là một sự cải tiến.

- Xác thực sử dụng khoá công khai.

### II.7.3. Kiểm soát truy cập (access control)

- Kỹ thuật mã hóa kết hợp với ma trận điều khiển truy cập (*access control matrix*) có thể được thi hành trong trường hợp hệ phân tán độc lập, cách ly với thế giới bên ngoài. Còn trong trường hợp bên ngoài cũng được phép truy cập vào hệ thống thì chúng ta phải sử dụng bức tường lửa (*firewall*). Có hai kiểu tường lửa :

1. Cổng lọc các gói tin (*packet-filtering gateway*).
2. Cổng mức ứng dụng (*application-level gateway*)

- Mã di động an toàn (*secure mobile code*): trong hệ phân tán thì là khả năng di trú mã giữa các host là hết sức cần thiết và quan trọng. Tuy nhiên, mã di động lại xuất hiện kèm theo các mối đe dọa an ninh nghiêm trọng. Ví dụ như, khi gửi 1 tác tử qua Internet, ta sẽ phải chống lại những host nguy hiểm, có ý định ăn cắp hoặc sửa đổi thông tin của tác tử. Một vấn đề khác là các host cần được bảo vệ chống lại các tác tử gây hại.

### II.7.4. Quản trị an toàn – an ninh

- Quản trị khoá *(key management*)
- Quản trị nhóm an toàn (*secure group management*)
- Quản trị uỷ quyền (*authorization management*)

### II.7.5. Kerberos

Một trong những hệ thống an toàn – an ninh được sử dụng rộng rãi đó là Kerberos. Kerberos được phát triển bởi MIT. Nó được dựa trên giao thức xác thực Needman-Schroeder mà ta đã nói ở phần trên (giao thức xác thực sử dụng trung tâm phân phối khóa - KDC). Mục đích của nó là tạo lập kênh an toàn giữa client và server.

### II.7.6. SESAME

SESAME là 1 hệ thống an toàn – an ninh khác, nó cũng khá giống với Kerberos. Tuy nhiên, nó lại dùng mã hoá công khai kết hợp với các khoá bí mật chia sẻ. Dự án SESAME được bắt đầu bởi sự nỗ lực tham gia của các công ty lớn ở Châu Âu, nhằm phát triển các chuẩn an toàn-an ninh cho hệ thống mở. SESAME được viết tắt từ *Secure European System for Application in a Multi-vendor Environment*.

### II.7.7. Thanh toán điện tử (electronic payment system)

Ta có 2 loại hệ thống thanh toán điện tử:

- Thanh toán trực tiếp giữa người bán và người mua. Các hình thức thanh toán có thể là tiền mặt, chuyển khoản, thẻ tín dụng.

- Thanh toán dựa trên việc chuyển tiền giữa các ngân hàng: lệnh chuyển tiền (*money order*), phiếu ghi nợ (*debit order*)

## III. Hệ thống quản trị file (tệp) phân tán

Chia sẻ dữ liệu là 1 trong những chức năng cơ bản của hệ phân tán. Hệ thống file phân tán cho phép nhiều tiến trình cùng chia sẻ dữ liệu trong khoảng thời gian dài 1 cách an toàn và tin cậy. Tất cả 7 nguyên lý mà ta đã nói ở trên đều được áp dụng cho các hệ thống file phân tán.

❖ **Tóm tắt nội dung:**

- Trong phần này ta xem xét hệ thống file phân tán đóng vai trò như 1 mô thức cho các hệ phân tán. Ta sẽ giới thiệu 2 hệ thống file phân tán là Sun NFS (*Network File System* - Hệ thống file mạng) và Coda. Trong 2 hệ thống đó, ta sẽ tập trung vào phân tích kỹ NFS.
- Ta cũng sẽ xem xét ngắn gọn thêm về 3 hệ thống file khác.
- Và cuối cùng là so sánh giữa các hệ thống file phân tán với nhau.

## III.1. Sun Network File System

Tên đầy đủ là Hệ thống file mạng của Sun Microsystem - thường được gọi tắt là NFS. NFS ban đầu được phát triển bởi Sun dành cho các máy trạm UNIX, tuy nhiên nó cũng có thể chạy tốt trên các hệ điều hành khác. Ý tưởng cơ bản của NFS là mỗi *file server* (máy chủ file) cung cấp 1 khung nhìn đã được chuẩn hóa về hệ thống file cục bộ của nó. Nói cách khác, bất kể hệ thống file cục bộ được thi hành như thế nào, thì mỗi *NFS server* (máy chủ hệ thống file mạng) hỗ trợ cùng mô hình. Mô hình này cùng với 1 giao thức truyền thông cho phép các client truy cập đến các file lưu trữ trên server. Hướng tiếp cận này cho phép 1 tập không thuần nhất các tiến trình, có khả năng chạy trên các máy và các hệ điều hành khác nhau, để cùng chia sẻ 1 hệ thống file chung.

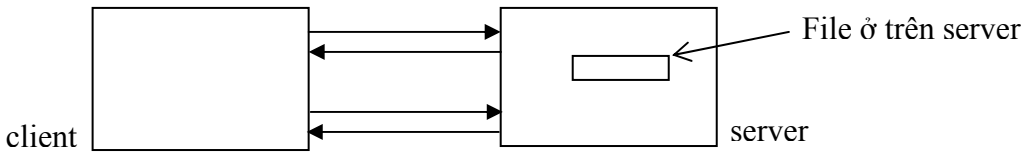
❖ **Sơ lược lịch sử của NFS:**

- Phiên bản đầu tiên của NFS đã không được phát hành mà chỉ được lưu hành nội bộ trong Sun.
- Phiên bản thứ 2 được kết hợp trong hệ điều hành SunOS 2.0 (1985).
- Vài năm sau, phiên bản thứ 3 của NFS được phát hành (1994) rồi đến phiên bản 4.

### III.1.1. Tổng quan về NFS

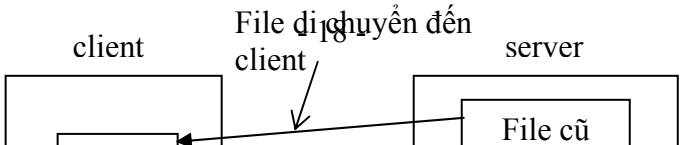
### a. Kiến trúc của NFS

Trong mô hình Dịch vụ file từ xa (*remote file service*), các client truy cập 1 cách trong suốt đến hệ thống file được quản lý bởi 1 server ở xa. Vì thế, thông thường các client không biết chỗ thực sự của các file. Thay vào đó, chúng được cung cấp 1 giao diện đến hệ thống file, tương tự như giao diện của các hệ thống file cục bộ thông thường. Trong trường hợp riêng, client chỉ được cung cấp 1 giao diện chứa nhiều các thao tác file khác nhau, server chỉ có nhiệm vụ thực thi các thao tác file đó. Chính vì vậy mô hình này còn được gọi là Mô hình truy cập từ xa (*remote access model*). (*H.1*)



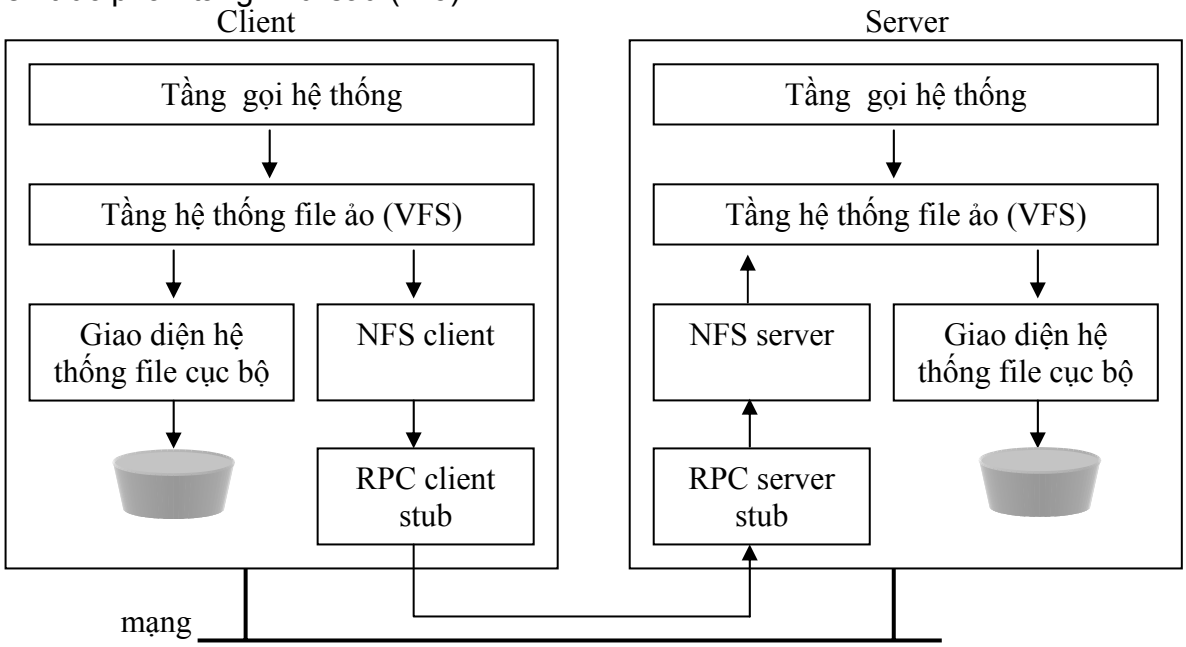

*H.1. Mô hình truy cập từ xa*

Trái lại, trong mô hình *Upload . Download* (tải lên . tải xuống) thì 1 client chỉ truy cập đến 1 file cục bộ sau khi đã tải nó xuống (download) từ server. Một khi client đã hoàn tất thao tác với file xong thì nó sẽ tải file lên (upload) lại server, để file đó có thể được các client khác sử dụng. Dịch vụ FTP của Internet dùng theo chính cách này. (*H.2*)



*H.2. Mô hình upload.download*

NFS dựa trên UNIX, tuy nhiên nó cũng có thể dùng cho nhiều hệ điều hành khác nhau. Với tất cả hệ thống UNIX hiện đại, nói chung NFS được thực thi theo kiến trúc phân tầng như sau (*H.3*)




*H.3. Kiến trúc NFS cơ bản dành cho hệ thống UNIX*

Một client truy cập đến hệ thống file sử dụng lời gọi hệ thống được cung cấp bởi hệ điều hành của nó. Tuy nhiên, giao diện hệ thống file UNIX cục bộ được thay bởi 1 giao diện đến *Hệ thống file ảo* (VFS). Các thao tác trên giao diện VFS hoặc được chuyển đến 1 hệ thống file cục bộ, hoặc được chuyển đến 1 thành phần riêng biệt gọi là *NFS client*, đảm trách việc điều khiển truy cập đến các file được lưu ở server từ xa. Trong NFS, tất cả client – server giao tiếp thông qua các RPC. NFS client thi hành các thao tác hệ thống file NFS khi các RPC đến server. Lưu ý rằng các thao tác được đưa ra bởi giao diện VFS có thể khác với các thao tác được đưa ra bởi NFS client. Như vậy ý tưởng chính của VFS là ẩn đi sự khác biệt giữa cáchệ thống file.

Ở bên server, chúng ta thấy tổ chức hoàn toàn tương tự. NFS server có nhiệm vụ xử lý các yêu cầu đến từ client.

Ta thấy, một lợi ích quan trọng của sơ đồ trên đó là sự độc lập của các hệ thống file cục bộ. Theo nguyên tắc thì nó sẽ không hề quan tâm dù hệ điều hành tại client hay server chạy 1 hệ thống file UNIX, 1 hệ thống file Windows 2000, hay thậm chí là 1 hệ thống file MS-DOS cũ. Chỉ có 1 điều quan trọng cần chú ý là các hệ thống file này phải tùy theo mô hình hệ thống file được đưa ra bởi NFS.

## b. Mô hình hệ thống file

Mô hình hệ thống file được đưa ra bởi NFS cũng giống như mô hình được đưa ra bởi hệ thống dựa trên UNIX. Các file sẽ được xem như dãy tuần tự các byte. Chúng được tổ chức phân cấp trong 1 đồ thị định danh, trong đó các nút biểu diễn các thư mục và các file. Để truy cập 1 file, client phải tìm tên của nó trong 1 dịch vụ định danh (*naming service*) và sẽ nhận được điều khiển file kết hợp (*associated file handle*). Ngoài ra mỗi file có 1 số các thuộc tính mà giá trị của chúng có thể được tìm và thay đổi. Ta xem xét chi tiết chúng ở phần sau.

❖ **Các thao tác với file của NFS phiên bản 3 và 4 :**

- Thao tác tạo mới (*creat*) được sử dụng để tạo ra 1 file, ở đây cũng sự khác biệt giữa NFS phiên bản 3 với phiên bản 4. Trong phiên bản 3, thao tác này được dùng để tạo ra các file chính quy (*regular file*), trong khi các file phi chính quy (*non-regular file*) lại được tạo ra bởi các thao tác riêng khác. Thao tác *liên kết* được dùng để tạo các liên kết cứng, còn *symlink* sẽ được dùng để tạo các liên kết biểu trưng (*symbolic link*). *Mkdir* để tạo các thư mục con. Đối với các file đặc biệt như các file thiết bị, các socket thì được tạo bởi thao tác *mknod*. Tuy nhiên, trong phiên bản 4 thì thao tác tạo mới lại được dùng để tạo ra 1 file phi chính quy, bao gồm cả các liên kết tượng trưng, các thư mục và các file đặc biệt. (*H.4*)

| Thao tác | Ph.bản 3 | Ph.bản 4 | Mô tả |
|---|---|---|---|
| Creat | Có | Không | Tạo 1 file chính quy |
| Creat | Không | Có | Tạo 1 file phi chính quy |
| Link | Có | Có | Tạo 1 liên kết cứng đến 1 file |
| Symlink | Có | Không | Tạo 1 liên kết tượng trưng đến 1 file |
| Mkdir | Có | Không | Tạo 1 thư mục con trong thư mục đã cho |
| Mknod | Có | Không | Tạo 1 file đặc biệt |
| Rename | Có | Có | Đổi tên 1 file |
| Remove | Có | Có | Xóa 1 file khỏi 1 hệ thống file |
| Rmdir | Có | Không | Xóa 1 thư mục con rỗng khỏi 1 thư mục |
| Open | Không | Có | Mở 1 file |
| Close | Không | Có | Đóng 1 file |
| Lookup | Có | Có | Tìm 1 file theo tên của nó |
| Readdir | Có | Có | Đọc các mục trong 1 thư mục |
| Readlink | Có | Có | Đọc tên đường dẫn lưu trong 1 liên kết biểu trưng |
| Getattr | Có | Có | Lấy các giá trị thuộc tính của 1 file |

| | | | |
|---|---|---|---|
| Setattr | Có | Có | Thiết lập1 hoặc nhiều giá trị thuộc tính của 1 file |
| Read | Có | Có | Đọc dữ liệu chứ trong 1 file |
| Write | Có | Có | Ghi dữ liệu vào 1 file |

*H.4. Danh sách các thao tác hệ thống file được hỗ trợ bởi NFS*

- Thao tác *rename* được dùng để thay đổi tên của 1 file đã có.
- File sẽ bị xóa bởi thao tác *remove*. Trong phiên bản 4 thì thao tác này sẽ xóa bất kỳ loại file nào. Tuy nhiên ở trong phiên bản 3 thì *rmdir* sẽ xóa thư mục con. Một file được xóa bởi tên của nó, và khi đó số các liên kết cứng đến nó sẽ bị giảm đi 1. Nếu số các đường liên kết giảm xuống 0 thì file đó sẽ bị hủy đi.

Phiên bản 4 cho phép các client mở và xóa các file (chính quy). Để mở 1 file, client sẽ cung cấp tên file , cùng với các giá trị khác nhau cho các thuộc tính. Sau khi 1 file đã được mở thành công, client có thể truy cập file đó bằng điều khiển file (*file handle*) của nó. Điều khiển đó cũng được dùng để đóng file, bằng cách đó client có thể nói với server rằng nó không cần truy cập đến file nữa. server khi này đã có thể giải phóng trạng thái mà nó đã duy trì để cung cấp cho client truy cập đến file.

- Thao tác *lookup* (tìm) được dùng để truy tìm 1 điều khiển file cho 1 tên đường dẫn cho trước.
- Thao tác *readlink* (đọc liên kết) được dùng để đọc dữ liệu liên kết với 1 symbolic link.
- Các file sẽ có các thuộc tính kèm theo. Các thuộc tính điển hình là: kiểu của file, độ dài file, định danh (identifier) của hệ thống file chứa file đó, và lần cuối cùng file được chỉnh sửa. Các thuộc tính của file có thể đọc hoặc thiết lập bằng cách sử dụng các thao tác *getattr* và *setattr*.
- Cuối cùng là các thao tác đọc (*read*) dữ liệu từ 1 file, và ghi (*write*) dữ liệu vào 1 file. Với thao tác đọc file (đọc thẳng tới), client chỉ ra số lượng các byte và khoảng cách giữa các byte để đọc. client sẽ được trả về số các byte đã đọc được. Với thao tác ghi dữ liệu vào file, client phải chỉ rõ ra vị trí bắt đầu ghi ở trong file, số lượng các byte được ghi, và dữ liệu ghi.

## III.1.2. Truyền thông

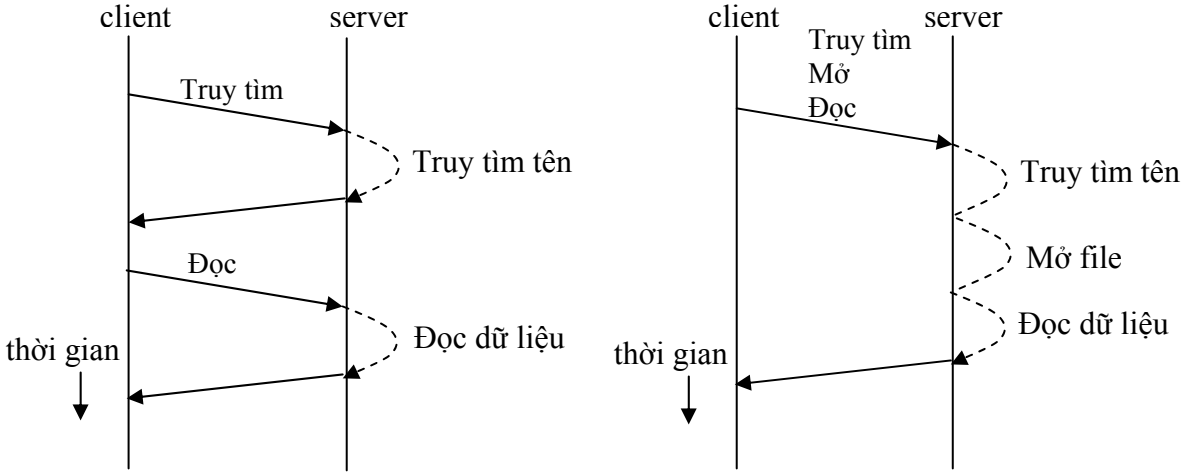
Một điều quan trọng trong NFS đó là sự độc lập với hệ điều hành, kiến trúc mạng, và các giao thức vận chuyển. Ví dụ như, các client chạy trên hệ thống Windows vẫn có thể giao tiếp với 1 máy chủ file (*file server*) UNIX.

Trong NFS, tất cả truyền thông giữa client và server đều theo giao thức *Open Network Computing RPC* (ONC RPC). Nói chung thì ONC RPC hoàn toàn tương tự với các hệ thống RPC khác.

Mọi thao tác NFS đều có thể thi hành khi 1 thủ tục đơn từ xa gọi đến 1 file server. Ví dụ như, để đọc dữ liệu từ 1 file lần đầu tiên, trước hết 1 client thông thường phải dùng thao tác *lookup* để truy tìm điều khiển file, sau đó nó mới có thể gửi 1 yêu cầu đọc (*H.5a*).

Trong ví dụ này của ta, client kết hợp cả yêu cầu tìm và yêu cầu đọc vào trong 1 RPC đơn (*H.5b*). Trong trường hợp phiên bản 4, ta cũng cần phải thao tác mở file trước khi hành động đọc diễn ra. Sau khi điều khiển file được tìm thấy, sẽ chuyển sang mở file, và sau đó server mới tiếp tục với thao tác đọc. Như vậy ta có

thể thấy, toàn bộ chỉ cần có 2 thông điệp phải trao đổi giữa client và server. Các thao tác sẽ được nhóm lại với nhau trong 1 thủ tục ghép (*compound procedure*). Nếu có lỗi thao tác dù bất kỳ lý do gì, thì không thao tác nào nữa trong thủ tục ghép được thực hiện, và kết quả cho đến lúc đó sẽ được trả về cho client. Ví dụ như, nếu thao tác truy tìm bị lỗi, thì việc mở file tiếp theo sẽ không được thực hiện nữa.




*H.5. (a) Đọc dữ liệu từ 1 file trong NFS phiên bản 3. (b). Đọc dữ liệu sử dụng thủ tục ghép trong phiên bản 4.*

### III.1.3. Stateless - Stateful

NFS là 1 hệ thống client – server truyền thống, mà trong đó các client gửi yêu cầu đến 1 *file server* (máy chủ file) để thực hiện các thao tác trên file. Một trong những điểm phân biệt khi só sánh với các hệ thống file phân tán khác, là việc các server có thể là *stateless* (tạm dịch là phi trạng thái). Nói cách khác, giao thức NFS không đòi hỏi các server duy trì bất kỳ trạng thái client nào. Ở NFS phiên bản 2 và 3 thì vẫn còn dùng cách tiếp cận này, tuy nhiên đến phiên bản 4 thì đã không dùng nữa. Lợi ích chính của cách tiếp cận này đó là đơn giản. Ví dụ, khi 1 *stateless server* bị sụp, thì về cơ bản ta không cần pha phục hồi (*recovery phase*) để đưa server trở lại trạng thái trước đó.

Ở phiên bản 4, hướng tiếp cận stateless đã bị bỏ đi dù rằng giao thức theo cách này cho phép server không cần phải duy trì nhiều thông tin trên các client của nó. Bên cạnh *stateless* ta còn có hướng tiếp cận *stateful* (tạm dịch là theo trạng thái). Một trong những lý do quan trọng để dùng *stateful* là do NFS phiên bản 4 có thể làm việc qua các mạng diện rộng (*wide-area networks*). Chính điều này đòi hỏi các client phải hiệu quả trong việc sử dụng các bộ nhớ đệm (*cach*). Từ đó cần phải có các giao thức nhất quán bộ đệm (*cach consistency protocol*). Ta còn có dịp bàn thêm về vấn đề này ở phần sau.

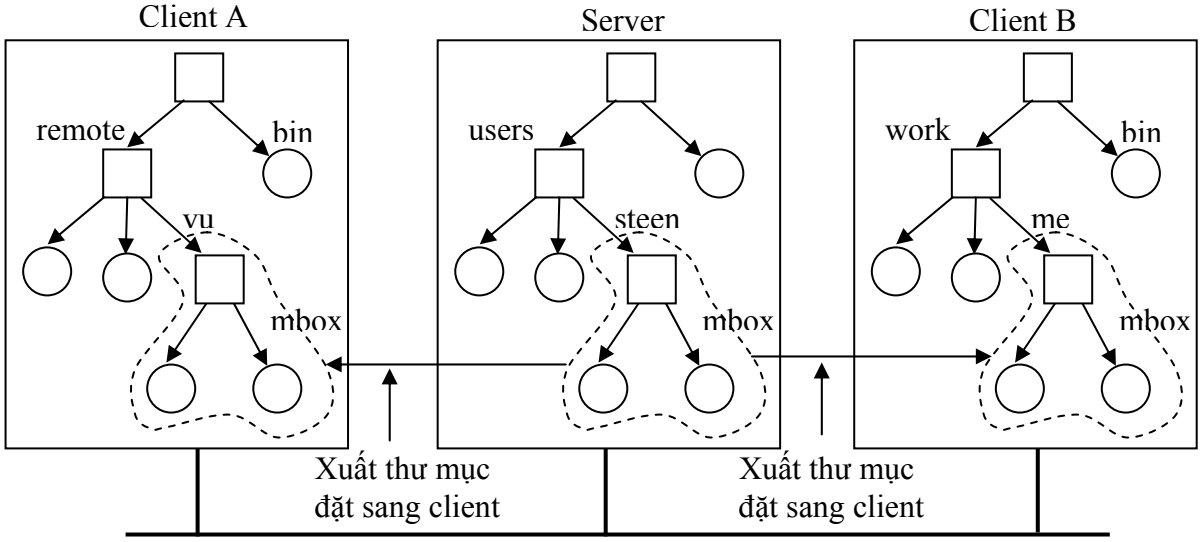
### III.1.4. Định danh

Cũng như bất kỳ hệ phân tán nào khác, việc định danh cũng đóng vai trò quan trọng trong NFS. Ý tưởng chính cho mô hình định danh NFS đó là cho các client truy cập trong suốt đầy đủ đến 1 hệ thống file từ xa được duy trì bởi 1 server. Sự trong suốt này có được bởi client có thể đặt (*mount*) 1 hệ thống file từ xa vào trong hệ thống file cục bộ của nó. (*H.6*).

Thay vì phải đặt (*mount*) toàn bộ cả hệ thống file sang, thì NFS cho phép các client chỉ cần đặt 1 phần của hệ thống file mà thôi (*H.6*). Một server được gọi là

xuất (*export*) 1 thư mục đi khi nó làm cho thư mục đó cũng có ở bên client. Thư mục xuất đi đó có thể được đặt ở trong 1 không gian tên cục bộ của client.

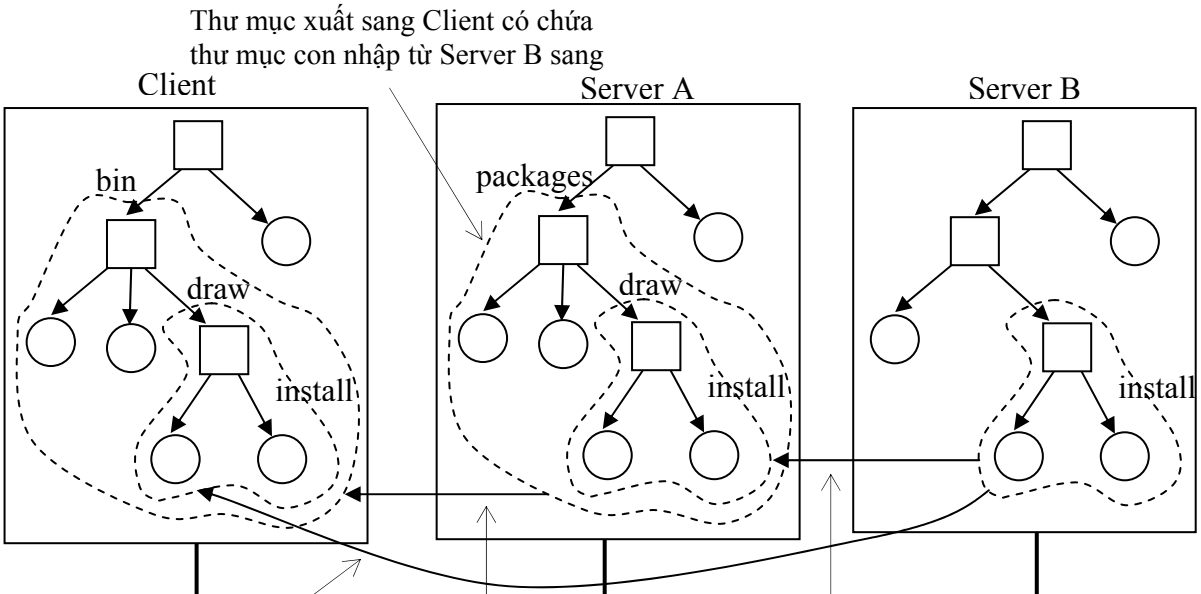
Như thế, theo nguyên tắc thì người sử dụng không chia sẻ các không gian tên. Ta hãy xem lại (*H.6*), file có tên là *.remote.vu.mbox* tại client A lại có tên là *.work.me.mbox* tại client B. Bởi vậy, tên của file phụ thuộc vào việc các client tổ chức không gian tên của chúng như thế nào, và nơi đặt các thư mục được xuất sang. Nhưng bù lại, theo cách này thì việc chia sẻ các file sẽ trở nên khó khăn hơn. Ví dụ như, A không thể nói cho B biết tên mà A đã dùng để gán cho file, bởi tên của đó sẽ hoàn toàn khác khi nó nằm trong không gian tên của B. Tuy nhiên, cũng có vài cách để khắc phục vấn đề này. Cách thông dụng nhất đó là cung cấp cho mỗi client 1 không gian tên đã được chuẩn hóa. Ví dụ như, mỗi client đều dùng thư mục cục bộ chuẩn là *.usr.bin* để đặt 1 hệ thống file vào (trên các client khác cũng có hệ thống file đó).




*H.6. Đặt (mounting) 1 phần hệ thống file từ xa trong NFS*

Ở đây ta chú ý rằng, 1 NFS server cũng có thể đặt (*mount*) vào bản thân nó các thư mục được xuất sang bởi các server khác. Tuy nhiên, nó lại không được phép xuất các thư mục này sang cho các client của nó. Để giải quyết được việc này, ta hãy xem xét ví dụ sau (*H.7*).

Giả sử rằng server A có hệ thống file $FS_A$ , mà từ đó xuất đi thư mục *.packages*. Thư mục này chứa 1 thư mục con là *.draw* đóng vai trò như 1 điểm đặt (*mount point*) cho hệ thống file $FS_B$, được xuất sang bởi server B và được đặt bởi A. Đến phiên server A cũng sẽ xuất *.packages.draw* sang cho các client của nó, và ta giả sử rằng client đặt *.packages* đó vào trong thư mục cục bộ *.bin* của nó (*H.7*).



*H.7 .Việc đặt (mounting) các thư mục từ nhiều server trong NFS*

Nếu việc phân giải tên bị lặp (như trong trường hợp NFS phiên bản 3), thì để phân giải tên *.bin.draw.install*, client sẽ liên hệ với server A khi nó đã phân giải cục bộ *.bin* và yêu cầu A trả về 1 điều khiển file cho thư mục *.draw*. Trong trường hợp đó, server A sẽ trả về 1 điều khiển file bao gồm 1 định danh cho server B, để chỉ có B có thể phân giải phần còn lại của tên đường dẫn, trường hợp này là *.install*. Như ta đã nói, loại phân giải tên này không được NFS hỗ trợ.

## a. Điều khiển file

Một điều khiển file là 1 tham chiếu đến 1 file trong hệ thống file. Nó không phụ thuộc vào tên của file mà nó tham chiếu đến. Một điều khiển file được tạo ra bởi server đang có hệ thống file trên đó, và là duy nhất đối với tất cả các hệ thống file được xuất đi bởi server. Điều khiển file được tạo ra khi file được tạo ra. Client không biết nội dung thực của điều khiển file. Điều khiển file dùng 32 byte trong NFS phiên bản 2, nhưng cũng có thể tùy biến độ dài lên đến 64 byte trong phiên bản 3 và 128 byte trong phiên bản 4.

Một điều khiển file được thực thi như 1 định danh thực sự cho 1 file trong hệ thống file. Điều này có nghĩa là chừng nào file còn tồn tại, thì nó sẽ chỉ có 1 điều khiển file. Một trong lợi ích của điều khiển file đó là làm tăng hiệu năng. Bởi một khi hầu hết các thao tác file chỉ đòi hỏi 1 điều khiển file thay vì tên của file, như vậy client có thể tránh phải lặp lại việc tìm tên file trước mỗi thao tác với file. Một lợi ích khác nữa là client có thể truy cập đến file ngay mà không phụ thuộc vào tên (tên hiện tại) của nó.

Vì 1 điều khiển file có thể được lưu trữ cục bộ tại 1 client, nên có 1 điểm quan trọng cần chú ý đó là 1 server không thể tái sử dụng lại 1 điều khiển file sau khi đã xóa file. Bởi nếu không, 1 client có thể bị lỗi khi truy cập đến file.

Để truy cập đến các file ở 1 hệ thống file ở xa, client sẽ cần phải cung cấp cho server 1 điều khiển file của thư mục, cùng với tên của file hoặc thư mục được phân giải. NFS phiên bản 3 giải quyết vấn đề này thông qua 1 giao thức đặt (*mount protocol*) riêng biệt. Sau khi đặt, client được đưa điều khiển file gốc (*root file handle*) của hệ thống file đã đặt, mà sau đó có thể dùng như 1 điểm bắt đầu cho việc truy tìm các tên file. Điều khiển file gốc có thể được sử dụng để tìm điều khiển file khác trong hệ thống file của server. Như vậy ta có thêm điểm lợi đó là không cần đến 1 giao thức đặt. Thay vào đó, việc đặt (*mounting*) này có thể được tích hợp vào trong giao thức chuẩn dành cho việc truy tìm file.

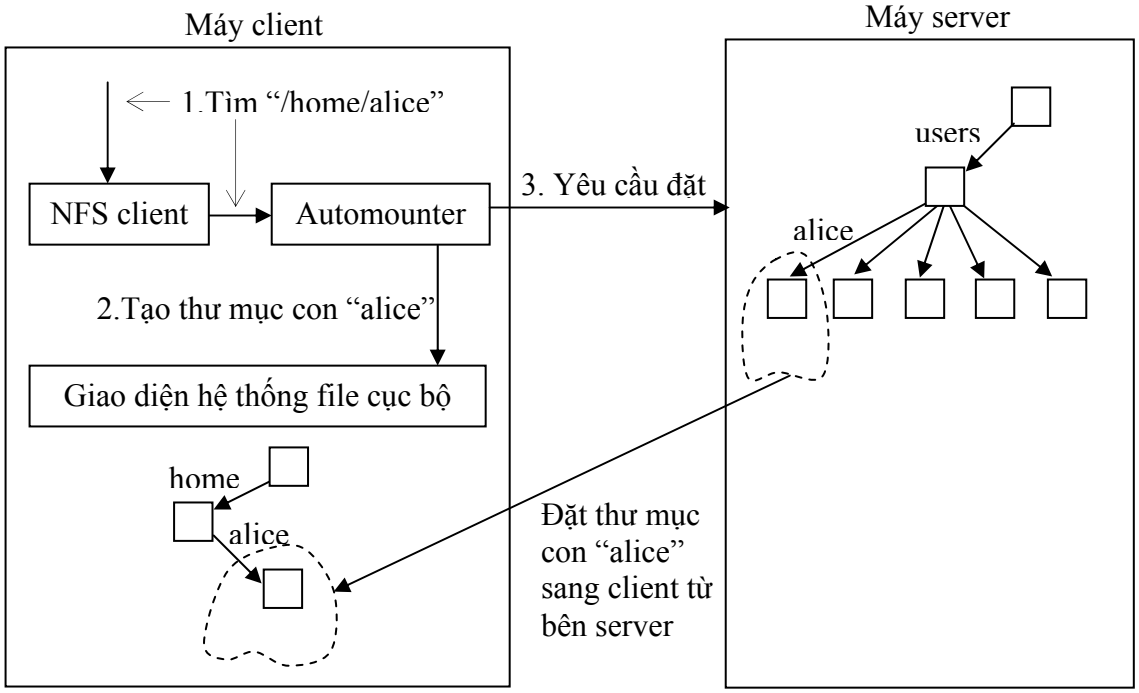
## b. Automounting

Như chúng ra đã nói ở trên, mô hình định danh NFS (*NFS naming model*) về cơ bản cung cấp cho người sử dụng không gian tên của họ. Việc chia sẻ trong mô hình này có thể sẽ khó khăn một khi người sử dụng đặt tên khác nhau cho cùng 1 file. Một trong những giải pháp cho vấn đề này đó là cung cấp cho mỗi người sử dụng 1 không gian tên cục bộ đã được chuẩn hóa, rồi sau đó mỗi client đều dùng thư mục cục bộ chuẩn đó để đặt hệ thống file vào.

Một vấn đề nữa với mô hình định danh NFS đó là phải quyết định xem khi nào thì 1 hệ thống file từ xa sẽ được đặt (*mount*) sang. Chúng ta hãy xem xét 1 hệ thống lớn với hàng ngàn người sử dụng. Giả sử rằng mỗi người sử dụng đều có 1 thư mục cục bộ *.home* được dùng để đặt các thư mục chủ (*home directories*) của người sử dụng khác sang. Ví dụ như thư mục chủ của Alice là *.home.alice*, mặc dù các file thực ra được lưu trữ trên 1 server ở xa. Thư mục này có thể được đặt (*mounted*) vào 1 cách tự động khi Alice vào máy tính của mình. Thêm vào đó, Alice cũng có thể truy cập đến các file công cộng (*public file*) của Bob bằng cách truy cập qua thư mục *.home.bob* của Bob. Như vậy, ta thấy một trong những lợi ích của hướng tiếp cận này đó là tòan bộ hệ thống sẽ trong suốt đối với Alice. Tuy nhiên cách tiếp cận này cũng có những nhược điểm của nó.

Việc đặt các hệ thống file từ xa trong NFS (thật ra là thư mục được xuất đi– *exported directories*), sẽ được điều khiển bởi 1 *automounter*, nó chạy như 1 tiến trình riêng biệt trên máy của client. Ta hãy xem xét 1 *automounter* đơn giản được thi hành như 1 server NFS cấp người sử dụng (*user-level NFS server*) trên hệ điều hành UNIX. Giả sử rằng, thư mục chủ của tất cả người sử dụng là đã có sẵn thông qua thư mục cục bộ *.home*, như ở trên ta đã mô tả. Khi 1 máy client khởi động, *automounter* sẽ bắt đầu với việc đặt thư mục này. Như vậy, hễ khi 1 chương trình cố gắng truy cập đến *.home*, nhân UNIX (*UNIX kernel*) sẽ thực hiện thao tác truy tìm (*lookup*) đến NFS client, ở trong trường hợp này, nó sẽ thực hiện việc yêu cầu đến *automounter* với vai trò như 1 NFS server (*H.8*).

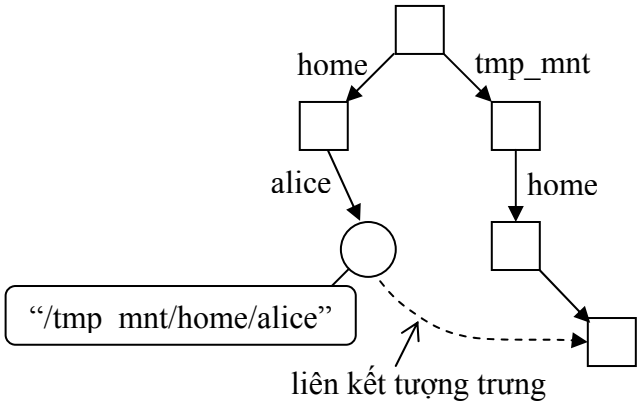
Ví dụ, giả sử rằng Alice đăng nhập. Chương trình đăng nhập sẽ thử đọc thư mục *.home.alice* để tìm kiếm thông tin, chẳng hạn như các kịch bản đăng nhập (*login script*). Automounter vì thế sẽ nhận yêu cầu truy tìm thư mục con *.home.alice*. Trước tiên nó tạo 1 thư mục con *.alice* trong *.home*. Sau đó nó tìm xem NFS server nào xuất đi thư mục chủ (*home directory*) của Alice để rồi đặt thư mục đó trong *.home.alice*. Vấn đề ở đây là *automounter* phải được bao gồm luôn trong các thao tác file để đảm bảo tính trong suốt. Nếu 1 file được tham chiếu không có sẵn bởi hệ thống file tương ứng chưa được đặt sang, thì *automounter* cũng phải biết.



*H.8. Một automounter đơn giản cho NFS*

Cách tiếp cận trên cũng có nhược điểm của nó. Và 1 trong những giải pháp đơn giản đó là để *automounter* đặt các thư mục vào trong 1 thư mục con đặc biệt, và thiết lập 1 liên kết tượng trưng (*symbolic link*) đến mỗi thư mục được đã đặt. (*H.9*)

Trong ví dụ này, thư mục chủ (*home directories*) người sử dụng được đặt sang thành thư mục con của *.tmp_mnt*. Khi Alice đăng nhập, automounter đặt thư mục chủ của Alice trong *.tmp_mnt.home.alice* và tạo 1 liên kết biểu trưng *.home.alice* tham chiếu đến thư mục con đó.




*H.9. Sử dụng các liên kết biểu trưng.*

## c. Các thuộc tính của file

Một NFS file có 1 số các thuộc tính kết hợp. Trong phiên bản 4, tập các thuộc tính file được chia ra thành:

- 1 tập các thuộc tính bắt buộc (*mandatory attributes*) - mọi sự thực thi đều phải hỗ trợ.
- 1 tập các thuộc tính được đề nghị (*recommended attributes*) - nên được hỗ trợ là tốt nhất.
- Và thêm 1 tập các thuộc tính được định danh (*named attributes*).

Các thuộc tính được định danh (*named attributes*) được mã hoá thành 1 mảng của các cặp (thuộc tính, giá trị), trong đó thuộc tính được biểu diễn là 1 chuỗi (*string*) và giá trị của nó là 1 dãy các byte. Chúng được lưu trữ cùng với file (hoặc thư mục) và NFS sẽ cung cấp các thao tác để đọc và ghi các giá trị thuộc tính.

Có tổng cộng 12 thuộc tính file bắt buộc (*H.10a*).

| Thuộc tính | Mô tả |
|---|---|
| TYPE | Kiểu của file (chính quy, thư mục, liên kết biểu trưng). |
| SIZE | Độ dài của file tính bằng byte |
| CHANGE | Cho client biết nếu có khi nào file bị thay đổi |
| FSID | Định danh duy nhất của hệ thống file. |

*(a)*

| Thuộc tính | Mô tả |
|---|---|
| ACL | Một danh sách điều khiển truy cập được kết hợp với file. |
| FILEHANDLE | Cung cấp điều khiển file. |
| FILEID | Định danh duy nhất cho file. |
| FS-LOCATIONS | Định vị trên mạng, nơi hệ thống file này có thể được tìm thấy |
| OWNER | Tên (chuỗi.ký tự) của chủ sở hữu file. |
| TIME-ACCESS | Thời điểm mà dữ liệu file được truy cập lần cuối. |
| TIME-MODIFY | Thời điểm mà dữ liệu file được chỉnh sửa lần cuối. |
| TIME-CREAT | Thời điểm mà dữ liệu file được tạo ra. |

*(b)*

*H.10. (a) Một số các thuộc tính file bắt buộc phổ biến trong NFS. (b) Một số các thuộc tính file được đề nghị phổ biến.*

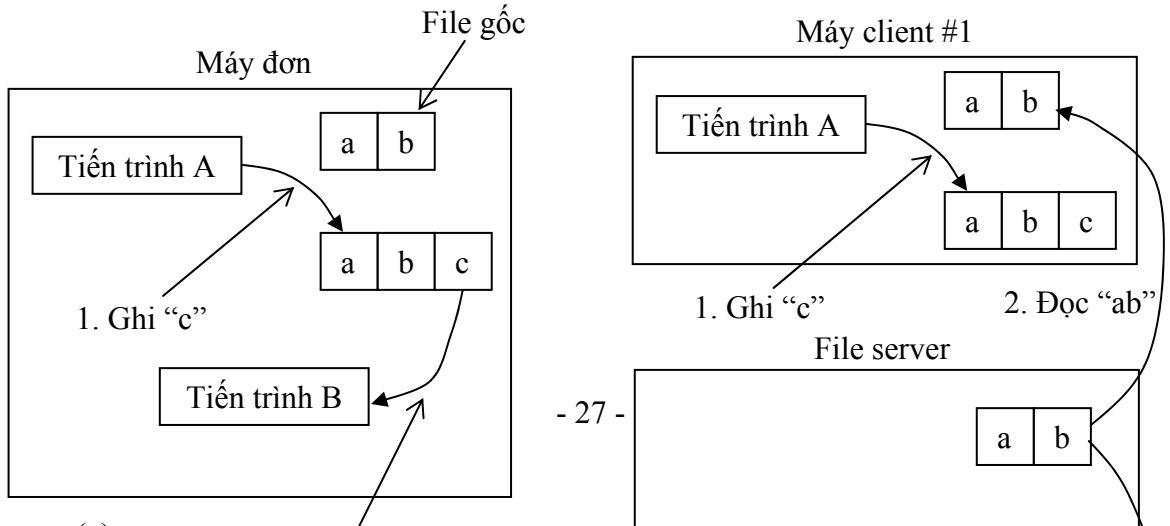
Các thuộc tính file được đề nghị phổ biến được liệt kê trong (*H.10b*). Phiên bản 4 NFS hiện tại có đến 43 thuộc tính được đề nghị.

## III.1.5. Đồng bộ hóa

Các file trong 1 hệ thống file phân tán được chia sẻ bởi nhiều client. Nếu việc chia sẻ không bao giờ xảy ra, thì quả thật như thế chẳng còn ý nghĩa của hệ thống file phân tán. Việc chia sẻ các file đòi hỏi cần phải có sự đồng bộ hóa. Sự đồng bộ hóa sẽ là khá đơn giản nếu các file được giữ trên 1 server trung tâm. Tuy nhiên khi ấy hiệu năng sẽ là 1 vấn đề. Bởi thế, các client thường được phép giữ 1 bản sao cục bộ của file trong khi chúng đang đọc và ghi nội dung file. Cách này tương tự với mô hình *upload.download* ở phần đầu ta đã nói (*H.2*).

### a. Ngữ nghĩa của việc chia sẻ file

Khi 2 hoặc nhiều hơn người sử dụng chia sẻ cùng 1 file, thì cần thiết phải định nghĩa  ngữ nghĩa (*semantics*) của việc đọc và ghi 1 cách chính xác để tránh phiền toái. Để giải thích các ngữ nghĩa của việc chia sẻ file trong NFS, trước hết chúng ta hãy xem xét 1 vài vấn đề có liên quan đến việc chia sẻ file.



*H.11. (a) Trên 1 vi xử lý đơn, khi 1 thao tác đọc theo sau 1 thao tác ghi, thì giá trị được trả về bởi đọc giá trị vừa được ghi. (b) Trong 1 hệ phân tán với bộ đệm, các giá trị cũ có thể được trả về.*

Trong các hệ thống vi xử lý đơn cho phép các tiến trình chia sẻ file, chẳng hạn như UNIX, ngữ nghĩa (*semantics*) thường là khi 1 thao tác đọc theo sau 1 thao tác ghi, thao tác đọc trả về giá trị vừa lúc được khi (*H.11*). Tương tự như thế, khi 2 thao tác ghi xảy ra kế tiếp nhau, theo sau là 1 thao tác đọc, giá trị đọc là những giá trị được lưu bởi lần ghi cuối cùng. Trong thực tế, hệ thống sẽ bắt ép 1 thời gian tuyệt đối với tất cả các thao tác và luôn luôn trả về giá trị gần nhất. Ta gọi mô hình này là ngữ nghĩa UNIX (*UNIX semantics*). Trong 1 hệ phân tán, ngữ nghĩa UNIX có thể dễ dàng có được với điều kiện chỉ có 1 file server và các client không giữ các file. Tất cả các thao tác đọc và ghi đều đến trực tiếp file server, và file server sẽ xử lý chúng 1 cách tuần tự.

Tuy nhiên trong thực tế, hiệu năng của 1 hệ phân tán thường rất thấp nếu trong đó tất cả các yêu cầu file đều phải đi đến 1 server. Vấn đề này sẽ được giải quyết bằng cách cho phép các client duy trì các bản sao cục bộ của các file thường được dùng nhiều (*heavily used files*) ở trong bộ nhớ đệm riêng (cục bộ) của nó.

### b. Khoá file trong NFS

Trong NFS, việc khoá file được điều khiển bởi 1 giao thức riêng biệt, và được thực thi bởi 1 bộ quản lý khóa (*lock manager*). Tuy nhiên, vì nhiều lý do khác nhau, việc khóa file sử dụng giao thức khóa NFS không được phổ biến. Ở trong phiên bản 4, việc khóa file đã được tích hợp vào trong giao thức truy cập file NFS. Cách tiếp cận này làm mọi việc đơn giản hơn cho các client khi khóa file.

Tuy nhiên, việc khóa file trong 1 hệ thống file phân tán cũng phức tạp bởi các client và server có thể lỗi trong khi đó các khóa vẫn được giữ lại. Khi đó, việc khôi phục lại 1 cách thích hợp sẽ trở nên quan trọng để đảm bảo tính nhất quán của các file được chia sẻ.

Bây giờ ta hãy xem xét cụ thể việc khóa file trong NFS phiên bản 4. Khóa file trong phiên bản 4 này khá đơn giản, chỉ cần 4 thao tác có liên quan đến việc khóa file này (*H.12*)

| Thao tác | Mô tả |
|---|---|
| Lock | Tạo ra 1 khóa cho 1 dãy các byte |
| Lockt | Kiểm tra xem khóa xung đột được cấp hay |

| | |
|---|---|
| | không |
| Locku | Xóa 1 khóa khỏi 1 dãy các byte |
| Renew | Hồi phục (làm mới lại) thời gian cấp cho khóa |

*H.12. Các thao tác trong NFS phiến bản 4 liên quan đến khóa file.*

- Thao tác *lock* (khóa) được dùng để yêu cầu 1 khóa đọc (*read lock*) hoặc khóa ghi (*write lock*) trên 1 dãy liên tiếp các byte trong file.
- Thao tác *lockt* được dùng để kiểm tra xem thử có khóa xung đột nào tồn tại hay không. Ví dụ như, 1 client có thể kiểm tra xem có khóa đọc nào đã được cấp trên 1 dãy các byte xác định ở trong file hay không, trước khi nó yêu cầu 1 khóa ghi cho các byte đó. Trong trường hợp xung đột, nó sẽ cho biết chính xác nguyên nhân của việc xung đột và cả dãy byte mà xung đột xảy ra trên đó nữa.
- Để xoá 1 khóa khỏi 1 file ta dùng thao tác *locku*.
- Các khóa đều được cấp cho 1 thời gian xác định (thời gian này được quyết định bởi server). Trừ phi 1 client hồi phục lại (làm mới lại - *renew*) khoảng thời gian đó trên các khóa đã được cấp, còn nếu không thì server sẽ tự động huỷ các khóa đó đi.

Bên cạnh các thao tác này, cũng còn có 1 cách nữa để khóa 1 file, đó là "chia sẻ chỗ đặt trước" (*share reservation*). Nó được dùng để thực thi NFS trên các hệ thống dựa trên Windows. Khi 1 client mở 1 file, nó sẽ xác định, chỉ ra kiểu truy cập mà nó yêu cầu (cụ thể là READ, WRITE hoặc BOTH), và kiểu truy cập mà server sẽ từ chối các client khác (NONE, READ, WRITE hoặc BOTH). Để biết chính xác điều gì sẽ xảy ra khi 1 client mới 1 file , mà file đó đã được mở thành công bởi 1 client khác, ta hãy xem (*H.13*) sau:

Trạng thái từ chối file hiệnhành

| Yêu cầu truy cập | NONE | READ | WRITE | BOTH |
|---|---|---|---|---|
| **READ** | Thành công | Lỗi | Thành công | Lỗi |
| **WRITE** | Thành công | Thành công | Lỗi | Lỗi |
| **BOTH** | Thành công | Lỗi | Lỗi | Lỗi |

(a)

Trạng thái từ chối file được yêu cầu

| Trạng thái truy cập hiện hành | NONE | READ | WRITE | BOTH |
|---|---|---|---|---|
| **READ** | Thành công | Lỗi | Thành công | Lỗi |
| **WRITE** | Thành công | Thành công | Lỗi | Lỗi |
| **BOTH** | Thành công | Lỗi | Lỗi | Lỗi |

(b)

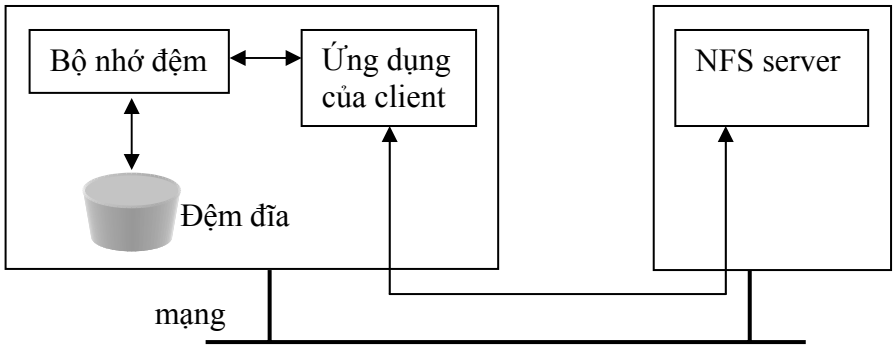
*H.13. Kết quả của thao tác đọc với share reservation trong NFS.*

## III.1.6. Lưu đệm (caching) và bản sao (replication)

Cũng như các hệ thống file phân tán khác, NFS sử dụng bộ đệm cho client để tăng hiệu năng. Thêm vào đó, nó cũng hỗ trợ việc tạo các bản sao của file.

### a. Lưu tạm cho client (client caching)

Trước đây, việc lưu tạm (*caching*) trong phiên bản 3 của NFS đã có nhiều nhược điểm, nó dẫn đến việc có nhiều cách thực hiện theo những chính sách khác nhau. Và hầu hết các cách đó đều không đảm bảo được tính nhất quán. Tuy nhiên, đến phiên bản 4, đã có 1 số thay đổi để đảm bảo tính nhất quán này. Mô hình lưu tạm (*caching model*) tổng quát của NFS như sau: (*H.14*)



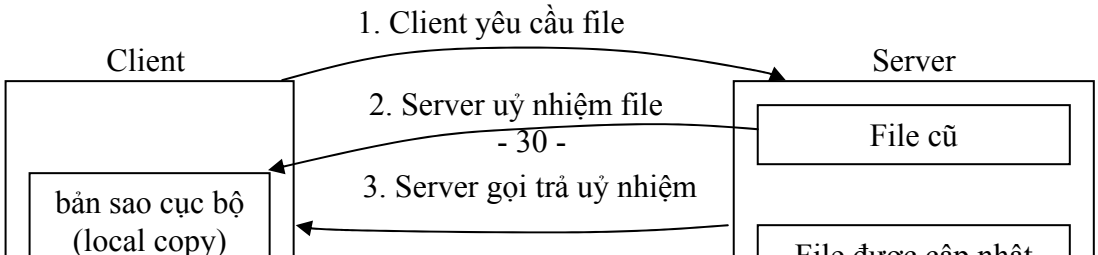

*H.14. Lưu tạm (đệm) ở bên phía client trong NFS*

Mỗi client có thể có 1 bộ nhớ đệm (*memory cach*) để chứa dữ liệu trước đó đọc từ server. Ngoài ra, còn có thể có đệm đĩa (*disk cach*) được thêm vào để mở rộng bộ nhớ đệm, sử dụng chung các tham số nhất quán.

NFS phiên bản 4 sử dụng 2 cách cho việc lưu tạm (đệm) dữ liệu file - *caching file data*. Cách đơn giản nhất là khi 1 client mở 1 file và lưu tạm dữ liệu mà nó nhận được từ server nhờ vào thao tác đọc. Thao tác ghi cũng có thể được thực hiện trong bộ nhớ đệm. Khi client đóng file, NFS yêu cầu rằng, nếu có sự thay đổi nào đã được diễn ra, thì dữ liệu được lưu tạm đó phải được đẩy về lại server. Một khi 1 file đã được lưu tạm, thì client có thể giữ dữ liệu của nó trong bộ đệm thậm chí sau khi đóng file. NFS đòi hỏi rằng, bất kỳ khi nào client mở 1 file đã đóng trước đó (mà đã được lưu tạm), client phải ngay lập tức *revalidate* (tái hiệu lực) dữ liệu đã lưu tạm.

Thêm 1 điểm ta cần chú ý nữa là server có thể uỷ nhiệm (*delegate*) 1 số quyền của nó đến cho client một khi file đã được mở. *Open delegation* (uỷ nhiệm mở file) diễn ra khi máy client được phép xử lý cụ bộ các thao tác đóng và mở (file) từ các client khác ở trên cùng 1 máy. Thông thường thì server phải đảm nhiệm việc kiểm soát dù cho việc mở 1 file có thành công hay không. Với *Open delegation*, máy client đôi khi cũng được cho phép tự đưa ra các quyết định, tránh việc phải cần liên lạc với server.

Một hệ quả của việc uỷ nhiệm 1 file đến client, đó là server khi cần có thể gọi trả lại (*recall*) uỷ nhiệm, ví dụ như, khi 1 client khác ở trên 1 máy khác cần nhận quyền truy cập đến file. Server có thể gọi client -đang được uỷ nhiệm - để trả lại uỷ nhiệm (*H.15*). Một khi gọi lại theo cơ chế này thì đòi hỏi server phải lưu lại vết của các client mà nó đã uỷ nhiệm file đến.



*H.15. Cơ chế gọi trả lại ủy nhiệm file trong NFS phiên bản 4.*

Việc lưu trữ tạm điều khiển file và các thư mục cũng dùng cách tiếp cận giống như trên.

### b. Bản sao các server

NFS phiên bản 4 hỗ trợ không nhiều cho việc nhân bản file. Chỉ toàn bộ hệ thống file mới có thể được nhân bản (bao gồm các file, các thuộc tính, các thư mục, và các khối dữ liệu).

## III.1.7. Chịu lỗi

### a. Lỗi RPC

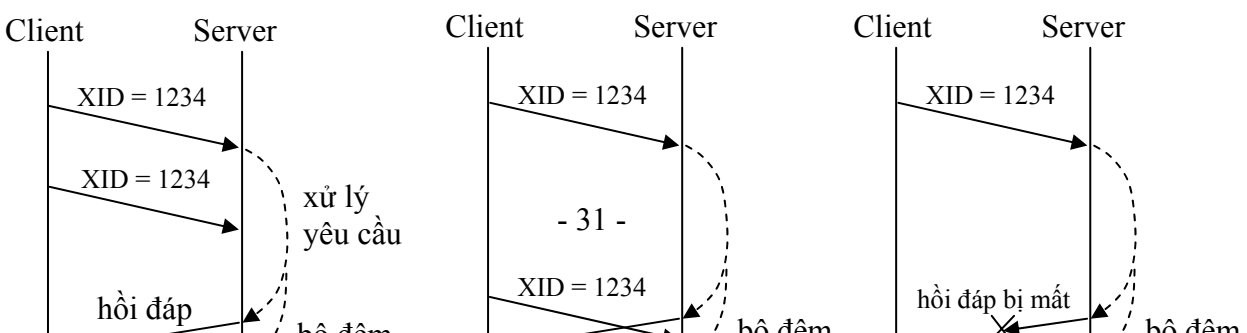
Vấn đề với cơ chế RPC khi được dùng bởi NFS đó là nó không đảm bảo tính tin cậy. Trên thực tế, các client và server RPC stub có thể được sinh ra dựa trên hoặc là giao thức vận chuyển hướng kết nối tin cậy như TCP, hoặc là giao thức vận chuyển phi kết nối không tin cậy như UDP.

Một trong những vấn đề chính nữa, đó là thiếu việc tìm ra các yêu cầu trùng lặp (*duplicate request*). Như vậy sẽ xảy ra trường hợp, khi 1 hồi đáp RPC bị mất và client truyền lại (*retransmit*) thành công yêu cầu gốc đến server, và kết quả là server sẽ phải thực hiện lại yêu cầu đó thêm lần nữa.

Những vấn đề trên được khắc phục bằng 1 bộ đệm các yêu cầu trùng lặp (*duplicate-request cache*) thực thi bởi server. Mỗi yêu cầu RPC từ client sẽ được mang theo 1 định danh giao tác (*transaction identifier* - XID) duy nhất ở phần header của nó, và nó sẽ được server lưu tạm khi nó đến server. Chỉ cần server không gửi hồi đáp, nó sẽ chỉ định yêu cầu RPC đang được thực hiện. Khi yêu cầu đã được xử lý xong, hồi đáp kết hợp (*associated reply*) của nó sẽ được lưu tạm lại, sau đó hồi đáp sẽ chính thức được gửi trả cho client.

Có 3 tình huống được đặt ra cần giải quyết ở đây:

- Trong trường hợp thứ nhất (*H.16a*), client gửi đi 1 yêu cầu, và khởi động 1 bộ định thời. Nếu hết thời gian trước khi có hồi đáp, client sẽ truyền lại yêu cầu gốc đó với XID giống như cũ. Tuy nhiên, bên phía server, thì do server chưa hoàn tất xong yêu cầu ban đầu, vì thế nó sẽ từ chối yêu cầu mà client truyền lại.
- Trường hợp 2, server có thể nhận 1 yêu cầu được truyền lại chỉ sau khi nó đã gửi đi hồi đáp cho client. Nếu thời điểm đến của yêu cầu được truyền lại đó gần sát với thời điểm mà serv7er gửi hồi đáp (*H.16b*), thì server sẽ kết luận rằng việc truyền lại với việc hồi đáp là đã bị chéo nhau, và vì thế server cũng lại từ chối yêu cầu truyền lại đó.



*H.16. Ba tình huống của việc xử lý truyền lại. (a) Yêu cầu vẫn đang được thực hiện. (b) Hồi đáp vừa mới được trả về. (c) Hồi đáp đã gửi trước đó, nhưng bị lỗi.*

- Trong trường hợp thứ 3, hồi đáp bị mất và yêu cầu truyền lại sẽ được đáp ứng bằng cách gửi đến client các kết quả mà trước đây ta đã lưu tạm trong bộ đệm (*H.16c*). Ở đây ta chú ý rằng các kết quả của thao tác file vẫn luôn được duy trì trong bộ đệm.

### b. Khóa file khi có lỗi

Việc khóa file trong NFS phiên bản 3 được xử lý thông qua 1 server riêng biệt. Theo cách này, thì các vấn đề có liên quan đến chịu lỗi được xử lý bằng cách khóa server. Trong phiên bản 4, vấn đề được giải quyết tương đối đơn giản. Để khóa 1 file, client đưa ra 1 yêu cầu khóa đến server. Giả sử rằng khóa được cấp cho client, thì ta cũng có thể gặp rắc rối một khi client hoặc server bị sụp đổ.

Để giải quyết vấn đề client bị sụp, server đưa ra 1 *lease* (dành riêng) trên mọi khóa mà nó cấp cho client. Một khi *lease* hết hạn, server sẽ xóa khóa đi, bằng cách đó nó giải phóng các tài nguyên liên kết (tức các file). Để cho server khỏi xóa 1 khóa, client sẽ làm mới lại (hồi phục lại - *renew*) lease của nó trước khi bị hết hạn bằng thao tác *renew*. Tuy nhiên cũng có tình huống các khóa bị xóa đi, ngay cả khi client không bị sụp đổ. Việc xóa không đúng này có thể xảy ra nếu không thể đưa thao tác *renew* đến cho server, ví dụ do khi đó mạng tạm thời bị đứt chẳng hạn.

Khi 1 server bị sụp và sau đó được phục hồi, có thể nó sẽ mất thông tin trên các khóa mà nó đã cấp cho client. Trong phiên bản 4, người ta đưa thêm vào 1 *grace period* (tạm dịch là giai đoạn tạm hoãn) mà trong giai đoạn này client có thể khôi phục, cải tạo lại các khóa đã được cấp trước đó cho nó. Việc hồi phục khóa này không hề phụ thuộc vào việc hồi phục các dữ liệu bị mất khác. Trong suốt *grace period*, chỉ các yêu cầu khôi phục khóa nói chung mới được chấp nhận, còn các yêu cầu khóa thông thường khác sẽ bị từ chối cho đến chừng nào *grace period* kết thúc.

Như ta thấy ở trên, việc dùng các *lease* trên các khóa sẽ làm nảy sinh ra nhiều vấn đề liên quan đến tính ổn định của việc gửi thông điệp làm mới *lease*. Ví dụ như, nếu thông điệp này bị lỗi hoặc đến server trễ, thì *lease* vô tình sẽ bị hết hạn dù cho nó đã yêu cầu được làm mới lại. Trong cả 2 trường hợp này, server sẽ khắc phục bằng cách phải đưa ra 1 lease mới cho client, trước khi client sau tiếp tục dùng file.

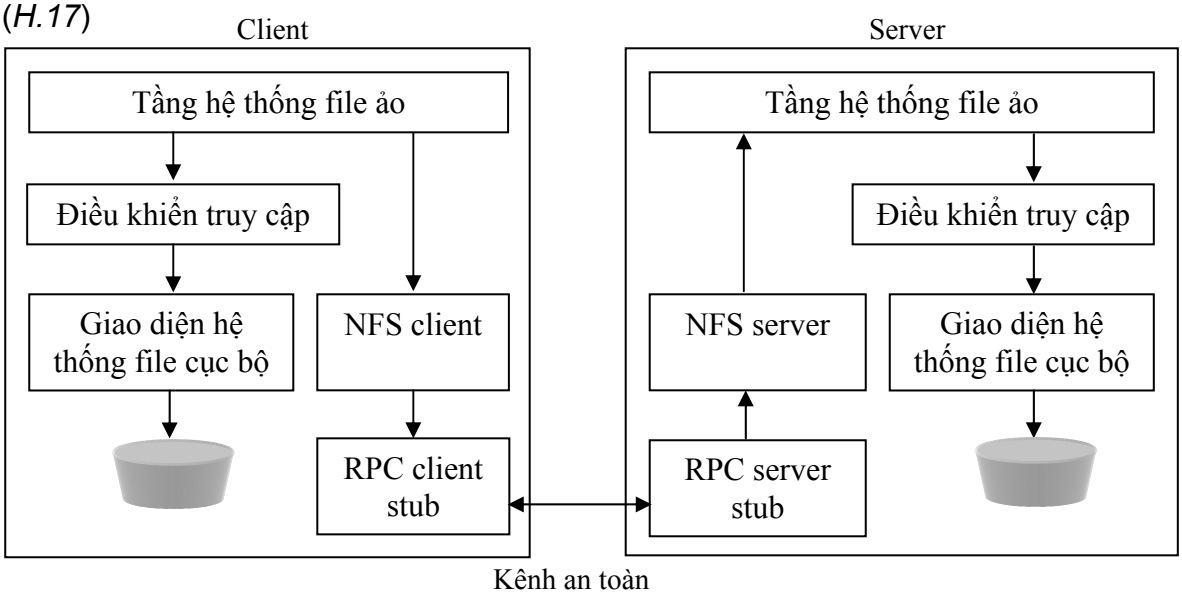
### c. Uỷ quyền mở khi có lỗi

Uỷ quyền mở được đưa ra để giải quyết vấn đề 1 client hoặc server bị sụp

## III.1.8. An toàn – an ninh

Như ta đã nói ở trước, ý tưởng chính của NFS đó là 1 hệ thống file từ xa sẽ được hiện diện tại client như thể nó là 1 hệ thống file cục bộ của client. Cũng chính vì vậy mà an toàn an ninh trong NFS luôn được tập trung chính vào truyền thông giữa client và server. Truyền thông an toàn có nghĩa là có 1 kênh an toàn được thiết lập ở giữa server và client, như trong phần *II.7.2* ta đã đề cập.

Thêm vào đó, để các RPC an toàn, thì cần thiết phải điều khiển các truy cập đến file, việc này được xử lý, điều khiển bởi các thuộc tính file điều khiển truy cập (access control file attributes) trong NFS. Một file server phụ trách việc xác minh các quyền truy cập của các client của nó, ta sẽ giải thích điều này ngay sau đây. Cùng với an toàn các RPC, kiến trúc an toàn an ninh NFS được giới thiệu trong (*H.17*)




*H.17. Kiến trúc an toàn an ninh NFS*

## a. RPC an toàn

Trong NFS phiên bản 4, thì chỉ có việc xác thực được quan tâm khi ta nói đến 1 RPC an toàn. Có 3 cách xác thực:

- S*ystem authentication* (xác thực hệ thống): là phương thức xác thực được sử dụng rộng rãi nhất. Xác thực hệ thống còn là phương thức xác thực dựa trên UNIX. Trong đó, các client đơn giản chỉ chuyền ID người dùng (user ID) và ID nhóm (group ID) của nó đến cho server, cùng với 1 danh sách các nhóm mà nó phải đòi hỏi là thành viên. Thông tin này được client gửi đi dưới hình thức có thể hiểu được của 1 văn bản đã được mã hoá (plaintext).

- *Secure NFS* (NFS an toàn): phương thức xác thực này sử dụng trao đổi khóa Diffie – Hellman để thiết lập 1 khóa phiên (sesion key), nó được dùng trong các phiên bản NFS cũ. Cách xác thực này tốt hơn so với xác thực hệ thống, nhưng bù lại nó phức tạp hơn, do đó nó cũng ít được dùng hơn.

- Và giao thức xác thực thứ 3 đó là *Kerberos* (phiên bản 4) mà ta đã từng đề cập đến trong phần *II.7.5* trước đây.

Trong NFS phiên bản 4, an toàn an ninh cũng đã được nâng cao với việc hỗ trợ RPCSEC - GSS. RPCSEC-GSS là 1 bộ khung (*framework*) an toàn an ninh tổng quát, có thể hỗ trợ rất nhiều cơ chế an toàn an ninh cho việc thiết lập các kênh truyền an toàn. Không những hỗ trợ cho các hệ thống xác thực khác nhau, mà nó còn hỗ trợ cả việc tích hợp các thông điệp (*message integrity*) và sự cẩn

mật (*confidentiality*), 2 đặc tính không được hỗ trợ trong các phiên bản NFS trước đây. RPCSEC-GSS được dựa trên 1 giao diện chuẩn cho các dịch vụ an toàn an ninh, có tên là GSS-API.

### b. Điều khiển truy cập

Uỷ quyền (authorization) trong NFS cũng tương tự với secure RPC (RPC an toàn): nó cung cấp các cơ chế, nhưng lại không chĩ rõ ra chính sách cụ thể nào. Việc điều khiển truy cập được hỗ trợ bởi thuộc tính file ACL (ACL file attribute). Thuộc tính này là 1 danh sách các mục (*entry*) điều khiển việc truy cập, trong đó mỗi mục chỉ ra các quyền truy cập cho 1 nhóm hoặc 1 người sử dụng xác định.

| Thao tác | Mô tả |
|---|---|
| Read_data | Cho phép đọc dữ liệu chứa trong 1 file. |
| Write_data | Cho phép chỉnh sửa dữ liệu của file. |
| Append_data | Cho phép thêm dữ liệu vào file. |
| Execute | Cho phép thực thi 1 file |
| List_directory | Cho phép liệt kê nội dung của 1 thư mục. |
| Add_file | Cho phép thêm 1 file mới vào thư mục |
| Add_subdirectory | Cho phép tạo 1 thư mục con trong 1 thư mục |
| Delete | Cho phép xóa 1 file |
| Delete_child | Cho phép xóa 1 file hoặc thư mục trong 1 thư mục |
| Read_acl | Cho phép đọc ACL (danh sách điều khiển truy cập) |
| Write_acl | Cho phép ghi ACL |
| Read_attributes | Khả năng để đọc các thuộc tính cơ bản khác của file. |
| Write_attributes | Cho phép thay đổi các thuộc tính cơ bản khác của file. |
| Read_named_attrs | Cho phép đọc các thuộc tính được định danh của file. |
| Write_owner | Cho phép thay đổi chủ. |
| Write_named_attrs | Cho phép ghi các thuộc tính được định danh của file. |
| Synchronize | Cho phép truy cập cục bộ 1 file tại server với các thao tác đọc, ghi đồng bộ. |

*H.18. Phân loại các thao tác được nhận diện bởi NFS đối với việc điều khiển truy cập.*

NFS phân biệt nhiều loại thao tác khác nhau. Ta chú ý đến thao tác Synchronize (đồng bộ hóa), thao tác này dùng để báo cho biết rằng, 1 tiến trình ở cũng chỗ với server có thể trực tiếp truy cập vào 1 file được hay không, bỏ qua giao thức NFS để từ đó có thể tăng hiệu năng. Mô hình NFS cho việc điều khiển truy cập có nhiều ngữ nghĩa hơn so với hầu hết các mô hình UNIX, bởi việc đòi hỏi NFS có thể liên tác với hệ thống Windows 2000.

Một điều khác nữa làm cho điều khiển truy cập file khác biệt so với các hệ thống file chẳng hạn như UNIX, đó là việc truy cập có thể được chỉ định cho những người sử dụng khác nhau và những nhóm khác nhau. Bởi theo truyền thống thì việc truy cập đến 1 file được chỉ định dành cho 1 người sử dụng (chủ sở hữu của file), 1 nhóm người sử dụng (chẳng hạn các thành viên của 1 nhóm dự án), và cho mọi người khác. NFS có nhiều loại người dùng và tiến trình khác nhau được trình bày trong bảng sau (*H.19*)

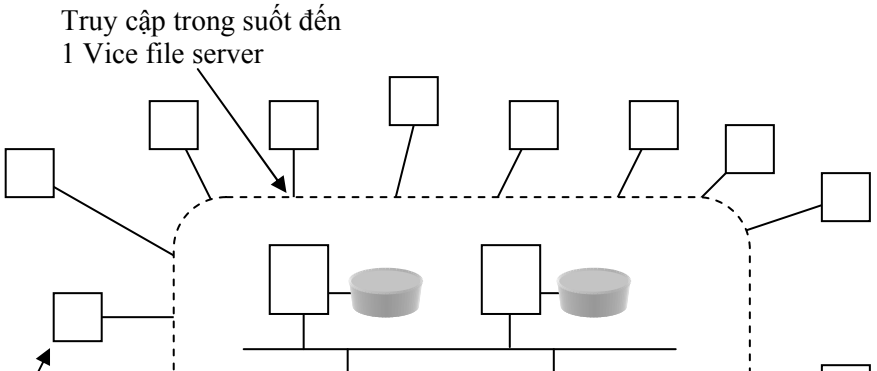
| Kiểu người dùng | Mô tả |
|---|---|
| Owner | Chủ sở hữu file |
| Group | Nhóm các người sử dụng có liên hệ với file. |
| Everyone | Bất kỳ người sử dụng hoặc tiến trình nào. |
| Interactive | Bất kỳ tiến trình nào đang truy cập file từ 1 trạm tương tác. |
| Network | Bất kỳ tiến trình nào đang truy cập file qua mạng. |
| Dialup | Bất kỳ tiến trình đang truy cập file thông qua kết nối dialup đến server. |
| Batch | Bất kỳ tiến trình đang truy cập file như 1 phần của công việc theo khối. |
| Anonymous | Những ai truy cập file mà không xác thực. |
| Authenticated | Bất kỳ người sử dụng hoặc tiến trình đã được xác thực. |

*H.19. Các loại người dùng và tiến trình khác nhau được phân biệt bởi NFS đối với việc điều khiển truy cập*.

## III.2. Hệ thống file Coda

Coda được phát triển tại trường Đại học Carnegie Mellon (CMU) vào những năm 1990, và bây giờ nó đã được tích hợp vào các hệ điều hành dựa trên UNIX phổ biến, chẳng hạn như Linux.

Coda là 1 hệ thống file phân tán có tính co dãn (về qui mô - *scalable*), có tính an toàn (*secure*), và có tính sẵn sàng cao (*high available*). Coda là hậu duệ của hệ thống file Andrew (AFS) phiên bản 2, cũng do CMU phát triển. Nó kết thừa nhiều điểm trong kiến trúc của AFS. Với AFS, nó có thể được thực thi với khoảng 10.000 máy trạm cần truy cập đến hệ thống. Để đáp ứng yêu cầu này, các nút AFS sẽ được chia thành 2 nhóm. Nhóm thứ nhất bao gồm 1 số tương đối ít các *Vice* file server, chúng sẽ được quản trị 1 cách tập trung. Nhóm còn lại bao gồm 1 lượng rất lớn các máy trạm *Virtue* , để đưa cho các người sử dụng và các tiến trình truy cập đến hệ thống file.



*H.20. Tổ chức tổng thể của AFS*

Coda cũng được tổ chức giống với AFS. Mọi máy trạm Virtue có 1 tiến trình cấp người dùng (user-level) thì được gọi là Venus, vai trò của nó tương tự như NFS client mà trước đây ta đã xét. Một tiến trình Venus có nhiệm vụ cung cấp truy cập đến các file được duy trì trên các Vice file server.

Không giống như NFS, Coda cung cấp 1 không gian tên chia sẻ tổng thể (globally shared name space) được duy trì bởi các Vice server. Các client truy cập đến không gian tên này bằng 1 thư mục con đặc biệt trong không gian tên cục bộ của chúng, chẳng hạn như *.afs*. Mỗi khi 1 client truy tìm 1 tên trong thư mục con này, Venus đảm bảo rằng phần thích hợp của không gian tên chia sẻ được đặt (mount) sang client.

- **Truyền thông**

Việc truyền thông giữa các tiến trình trong Coda được thực hiện bằng cách dùng các RPC. Tuy nhiên, hệ thống RPC2 dành cho Coda phức tạp hơn nhiều so với các hệ thống RPC truyền thống chẳng hạn như ONC RPC, được dùng bởi NFS.

RPC2 còn hỗ trợ các *side effect* (tạm gọi là hiệu ứng mặt). Một *side effect* là 1 cơ chế mà bằng cách đó client và server có thể truyền thông sử dụng 1 giao thức ứng dụng cụ thể (application-specific). Ví dụ, 1 client đang mở 1 file tại 1 video server. Điều cần thiết trong trường hợp này đó là thiết lập 1 luồng dữ liệu liên tục với chế độ truyền đẳng thời (như nhau về thời gian). Hay nói cách khác, dữ liệu truyền từ server đến client được đảm bảo nằm trong khoảng lớn nhất đến bé nhất thời gian trễ đầu cuối (end-to-end delay). RPC2 cho phép client và server thiết lập 1 kết nối riêng biệt để truyền dữ liệu video đến client kịp thời.

Một đặc điểm khác nữa mà RPC2 khác so với các hệ thống RPC còn lại, đó là nó hỗ trợ kỹ thuật multicasting.

- **Tiến trình**

Với Coda thì có 1 sự khác biệt rõ ràng giữa các tiến trình client và tiến trình server. Tương ứng với client và server đó là các tiến trình Venus và Vice. Cả 2 kiểu tiến trình được tổ chức bên trong như 1 tập các luồng đồng thời (concurrent threads). Các luồng trong Coda không có sự ưu tiên và nó hoạt động trong toàn không gian người sử dụng.

- **Định danh**

Coda duy trì 1 hệ thống tên tương tự như UNIX. Các file được nhóm lại vào trong những đơn vị gọi là *volume*. Một volume giống với 1 phân vùng đĩa UNIX (tức là 1 hệ thống file thực sự). *Volume* quan trọng bởi 2 lý do. Thứ nhất, chúng

tạo ra đơn vị cơ bản được dùng để cấu trúc nên toàn bộ không gian tên. Lý do thứ 2 đó là chúng tạo ra đơn vị cho sao việc lặp bên phía server.

Có một điều quan trọng mà ta cần chú ý đó là khi 1 *volume* từ không gian tên chia sẻ được đặt vào trong không gian tên của client, Venus sẽ theo cấu trúc của không gian tên chia sẻ (shared name space). Cũng như vậy, các client được đảm bảo rằng các file chia sẻ quả thực  có tên như nhau, dù việc phân giải tên lại dựa trên 1 thực thi cục bộ không gian tên. Như thế, ta thấy cách tiếp cận này khác cơ bản so với NFS.

- **Đồng bộ hóa.**
- **Lưu tạm (caching) và nhân bản (replication).**
- **Chịu lỗi.**

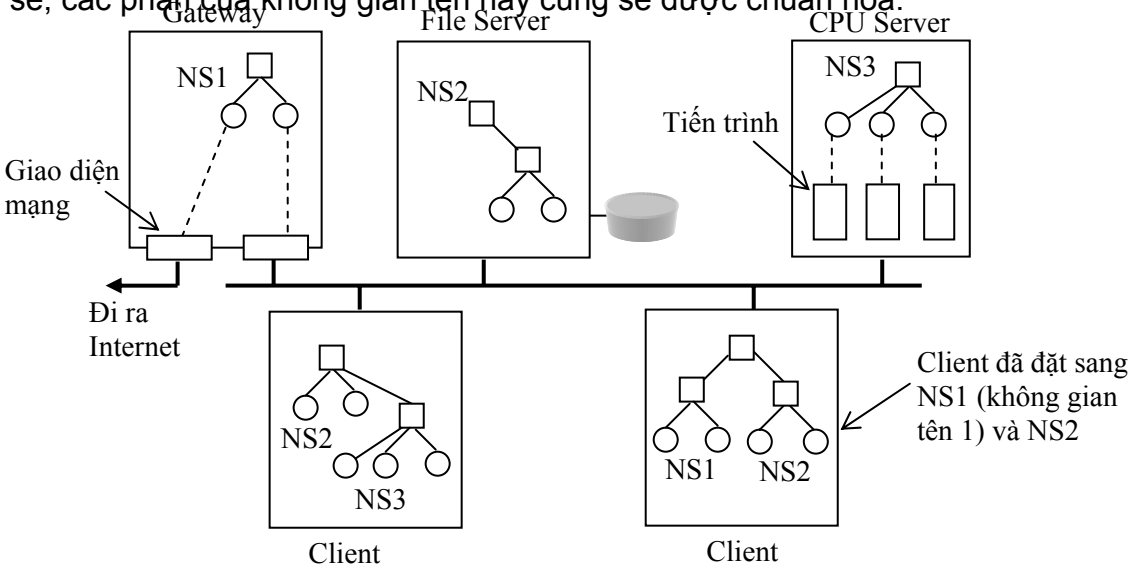
## III.3. Các hệ thống file phân tán khác

Ngoài 2 hệ thống file mà ta nói ở trên, thì còn có 1 số các hệ thống file khác nữa. Tuy nhiên, hầu hết đều giống với NFS hoặc Coda. Ở đây ta xét đến 3 hệ thống file khác nữa, đó là Plan 9, XFS, và SFS.

Trong Plan 9 thì mọi tài nguyên đều được đối xử như 1 file. Còn XFS là 1 ví dụ điển hình của 1 hệ thống file không có server (*serverless*). Và cuối cùng là SFS, đó là 1 hệ thống file mà trong đó các tên file cũng chứa thông tin an toàn an ninh.

### 1. Plan 9 - Tài nguyên thống nhất với file

Trong Plan 9, tất cả các tài nguyên đều được truy cập theo cùng 1 cách, cụ thể đó là các thao tác và cú pháp giống file. Ý tưởng này được kế thừa từ UNIX, tuy nhiên nó tốt và nhất quan hơn ở trong Plan 9. Mỗi server đưa ra 1 không gian tên phân cấp đến các tài nguyên mà nó điều khiển. Một client có thể đặt vào cục bộ 1 không gian tên được đưa ra bởi server, như vậy việc xây dựng không gian tên riêng của chính nó tương tự với cách tiếp cận trong NFS. Để cho phép việc chia sẻ, các phần của không gian tên này cũng sẽ được chuẩn hóa.




*H.21.Tổ chức tổng quát của Plan 9*

Một hệ thống Plan 9 bao gồm 1 tập các server để cung cấp tài nguyên cho các client dưới dạng không gian tên cục bộ. Để truy cập đến các tài nguyên của 1 server, client đặt không gian tên của server vào trong không gian tên của chính nó. Ở đây ta chú ý rằng, trong Plan 9 sự khác biệt giữa client và server không thực sự

quá rõ ràng. Ví dụ như, các server thường hành động như các client, trong khi các client lại cũng có thể xuất tài nguyên của chúng sang cho server. Tổ chức của Plan 9 được trình bày trong (*H.21*)

## 2. XFS - hệ thống file không có server (serverless)

xFS được phát triển như 1 phần của dự án Berkeley NOW. Ở đây ta cần chú ý là có 1 hệ thống file phân tán khác hoàn toàn khác được gọi là XFS ("X" viết hoa) được phát triển cùng thời điểm với xFS. Sau này, ta dùng "xFS" và "XFS" để chỉ nói đến hệ thống được phát triển là 1 phần hệ thống của dự án NOW.

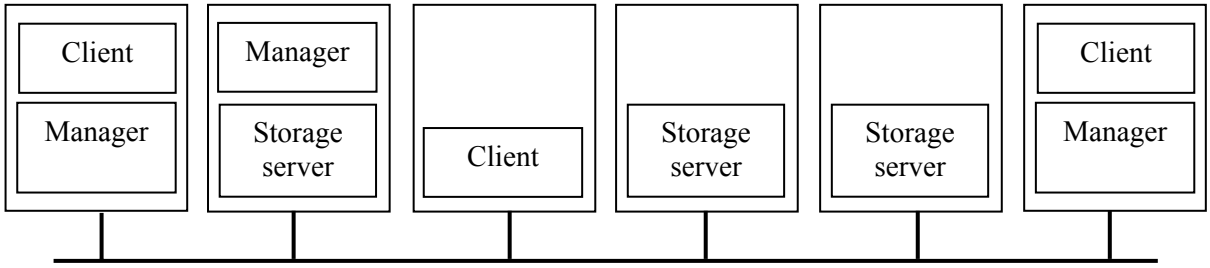
Việc thiết kế mà không có server của XFS là 1 điều khá lạ. Toàn bộ hệ thống file được phân tán qua nhiều máy, bao gồm các client. Hướng tiếp cận này trái ngược với các hệ thống file khác, thông thường được tổ chức theo kiểu tập trung, thậm chí còn có nhiều server dùng cho việc phân tán và tạo bản sao các file.

XFS được thiết kế để hoạt động trên mạng cục bộ, trong đó các máy được kết nối với nhau thông qua các đường liên kết tốc độ cao. XFS được thiết kế với mục đích đạt độ co dãn cao (tức thay đổi về quy mô - scalability) và cả tính chịu lỗi cao.

Trong kiến trúc của XFS, có 3 loại tiến trình khác nhau:

- *Storage server* (lưu trữ server): là tiến trình có nhiệm vụ lưu trữ các phần của 1 file. Chúng thực thi 1 dãy (array) các đĩa ảo tương tự như việc thực thi của các dãy đĩa dưới dạng RAID.
- *Metadata manager* (quản lý siêu dữ liệu): là tiến trình có nhiệm vụ lưu lại vết, nơi 1 khối dữ liệu file thật sự được lưu
- *Client*: là 1 tiến trình chấp nhận các yêu cầu của người sử dụng để thao tác trên file.

Môt nguyên lý thiết kế cơ bản của XFS đó là bất kỳ máy nào cũng có thể đóng vai trò của client, manager, server. Trong 1 hệ thống đối xứng hoàn toàn, mỗi máy sẽ có cả 3 tiến trình trên chạy. Tuy nhiên, cũng có thể sử dụng các máy dành để chạy các tiến trình *storage server*, trong khi máy khác lại chỉ chạy tiến trình *client* hoặc tiến trình *manager* (*H.22*)



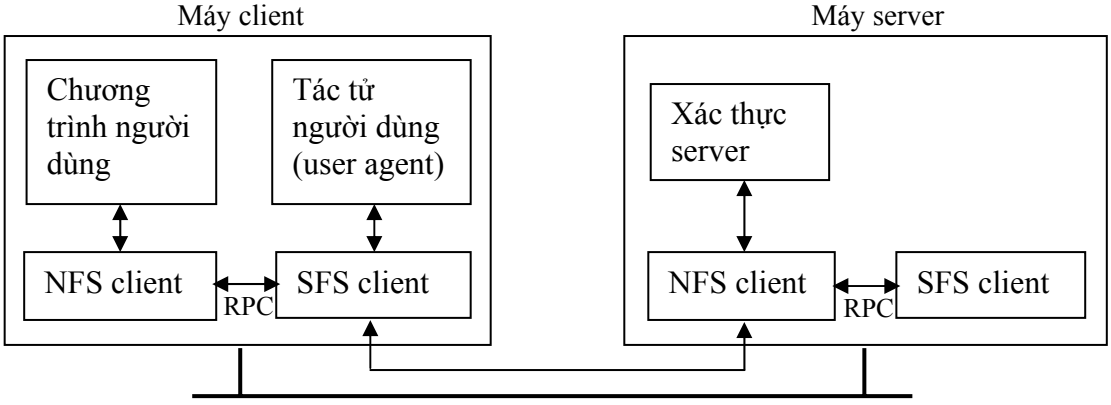

*H.22. Phân tán các tiên trình của XFS qua nhiều máy.*

## 3. SFS - an toàn an nình có thể thay đổi, co dãn (scalable security)

Trong Hệ thống file an toàn (*Secure File System* – SFS), nguyên lý thiết kế chính đó là tách riêng ra việc quản lý của các khoá với an toàn an ninh hệ thống file. Hay nói cách khác, SFS đảm bảo rằng các client không thể truy cập 1 file mà không sở hữu 1 khoá bí mật thích hợp.

Tổ chức tổng thể của SFS được trình bày trong (*H.23*). Để đảm bảo tính di chuyển được qua nhiều máy, SFS đã tích hợp nhiều thành phần NFS phiên bản 3 khác nhau. Trên máy client, có cả thảy 3 thành phần, không tính đến chương trình của người sử dụng. NFS client được sử dụng như 1 giao diện đến các chương trình người sử dụng, và trao đổi thông tin với SFS client. SFS client có nhiệm vụ

thiết lập 1 kênh an toàn với SFS server. Nó cũng có nhiệm vụ giao tiếp với tác tử người dùng SFS (SFS user agent), đó là 1 chương trình tự động xử lý xác thực người sử dụng.




*H.23. Tổ chức của SFS.*

Ở bên phía server cũng có 3 thành phần. Các NFS server giao tiếp với SFS server, thao tác như 1 NFS client đến NFS server. SFS server sẽ tạo tiến trình nhân (core process) của SFS. Tiến trình này có nhiệm vụ xử lý các yêu cầu file từ các SFS client.

## III.4. So sánh giữa các hệ thống file phân tán

### 1. Triết lý của hệ thống

Mục tiêu của các hệ thống file khác nhau thì thường cũng khác nhau.

- Như NFS, thì mục đích của nó là cung cấp 1 hệ thống cho phép client truy cập trong suốt đến 1 hệ thống file được lưu tại 1 server từ xa xác định.
- Với Coda thì khác, mục tiêu của nó là tính sẵn sàng cao (high available). Coda thừa kế mục đích thiết kế của AFS, trong đó một điều quan trọng đó là tính co dãn (scalability). Việc kết hợp tính sẵn sàng cao với tính co dãn giúp phân biệt Coda với hầu hết các hệ thống file phân tán khác.
- Mục đích chính của Plan 9 là cung cấp 1 hệ thống chia sẻ thời gian (timesharing) phân tán, trong đó tất cả các tài nguyên được truy cập theo cách như nhau, mà cụ thể là như 1 file.
- Với XFS thì mục đích của nó cũng khá giống với các hệ thống file phân tán khác, đó là tính sẵn sàng cao và tính co dãn. Tuy nhiên, điều quan trọng là đạt được mục tiêu này bằng 1 hệ thống mà không có server (serverlesss).
- Mục tiêu của SFS lại là an toàn an ninh có thay đổi, co dãn (scalable security). Nó thực hiện mục tiêu này bằng cách chia tách việc quản lý với an toàn an ninh hệ thống file và cho phép người dùng bắt đầu dịch vụ SFS (SFS service) của họ mà không có sự can thiệp từ 1 authority (quyền) ở trung tâm.

### 2. Truyền thông

Khi so sánh về truyền thông, thì hầu hết các hệ thống file phân tán đều dựa trên các dạng của RPCs.

- Với NFS, Coda và SFS thì đều sử dụng trực tiếp 1 hệ thống RPC cơ sở (underlying RPC system), đôi khi được tối ưu thêm để xử lý các trường hợp đặc biệt.

- Plan 9 cũng sử dụng 1 hệ thống RPC, nhưng trên thực tế giao thức đã được biến đổi đi để đáp ứng các thao tác file của nó, điều này làm cho nó khác đôi chút so với các giao thức còn lại.
- XFS cũng bắt đầu với 1 hệ thống RPC để xử lý, điều khiển tất cả các truyền thông. Tuy nhiên bởi lý do hiệu năng và một số lý do khác nữa trong XFS, mà hệ thống RPC đã được thay thế bằng các active messages.

### 3. Tiến trình

Các hệ thống khác ở nhau cách mà client đóng vai trò đối với toàn hệ thống file.

- Với việc dùng cách tổ chức client-server, trong NFS phiên bản 3, hầu hết các công việc thực sự chỉ được làm bởi file server, trong khi đó 1 NFS client chỉ đơn thuần yêu cầu các thao tác để được server tiến hành. Đối với NFS phiên bản 4 thì các client đã được cho phép lưu tạm các file và xử lý các thao tác 1 cách cục bộ.
- Với AFS và Coda ở phía client ta sẽ có tiến trình Venus, đảm nhiệm 1 lượng lớn công việc ở bên phía client.
- Ngược lại, các nhà phát triển Plan 9 lại cố gắng làm sao để giữ cho các client đơn giản đến mức có thể. Tuy nhiên 1 tiến trình client cũng được cho phép lưu tạm (caching) các file, nhưng hệ thống vẫn làm việc tốt nếu các bộ đệm không được dùng gì cả.

Các server cũng rất khác nhau khi so sánh giữa các hệ thống.

### 4. Định danh

Có 2 hướng tiếp cận căn bản trong việc tổ chức không gian tên. Hướng thứ nhất đó là mỗi người sử dụng lấy không gian tên riêng của chính họ. Cách này được dùng trong NFS và Plan 9. Nhược điểm của không gian tên trên mỗi người dùng như thế này, đó là khó chia sẻ các file dựa trên tên của nó. Và để giảm bớt nhược điểm này, các phần của không gian tên sẽ được chuẩn hóa.

Hướng tiếp cận thứ 2 đó là cung cấp 1 không gian tên chia sẻ tổng thể, được dùng trong Coda, xFS và SFS. Trong tất cả các hệ thống này, mỗi người sử dụng cũng có khả năng gia thêm vào không gian tên tổng thể 1 không gian tên cục bộ riêng. Ví dụ, SFS client sẽ cho phép 1 người sử dụng tạo cục bộ các liên kết biểu trưng (symbolic link). Các tên này là riêng tư với người dùng, và sẽ không thể thấy với người dùng ở các SFS client khác.

Ngoài ra, cũng có nhiều điểm khác nhau đối với tham chiếu file giữa các hệ thống file phân tán.

### 5. Đồng bộ hóa

- NFS cung cấp các ngữ nghĩa phiên (session semantics), điều này có nghĩa chỉ các cập nhật của tiến trình cuối đóng file là được nhớ bởi server.
- Trong Coda, các ngữ nghĩa giao tác (transactional semantics) được hỗ trợ theo hướng, chỉ các phiên được cho phép này mới có thể được tuần tự hóa. Tuy nhiên, trong thao tác bị ngắt kết nối, các ngữ nghĩa đó có thể không được đảm bảo, bởi thế dẫn đến cập nhật các xung đột cần được giải quyết sau này.
- Bởi các thao tác trên file trong Plan 9 cơ bản được xử lý bởi file server, nên Plan 9 cung cấp các ngữ nghĩa UNIX. Các ngữ nghĩa này cũng được cung cấp bởi xFS.

### 6. Lưu tạm (caching) và nhân bản (replication)

Tất cả các hệ thống mà ta đã xét đều hỗ trợ việc lưu tạm bên phía client. Chỉ có Plan 9 là mọi các thao tác ghi được chuyển tức thì đến server. Các hệ thống khác đều thực thi các đệm ghi lại (write-back caches), cho phép 1 client thực hiện 1 loạt các thao tác ghi trên dữ liệu được lưu tạm trước khi đẩy thẳng vào bộ đệm.

Các hệ thống file phân tán này cũng hỗ trợ việc sao lặp file bởi server.

## 7. Chịu lỗi

- Đối với Coda, nó dùng đệm để lưu tạm và các bản sao của nó để có thể đạt được tính sẵn sàng cao (high available).
- Còn trong xFS, kỹ thuật striping được dùng để bảo vệ server đơn khỏi bị sụp đổ.
- Việc hồi phục dựa trên client được dùng trong NFS. Theo cách này, 1 server đã bị mất thông tin khi nó bị sụp đổ, thì các client được phép khôi phục lại các tài nguyên như các khóa chẳng hạn.
- XFS thì sử dụng các điểm kiểm tra (checkpoint), và các client ghi lại nhật ký (log) để phục vụ việc phục hồi đến điểm mà dữ liệu của manager (bộ quản lý ) nhất quán với thông tin được lưu trong nhật ký.

## 8. An toàn an ninh

- NFS phiên bản 4 tách biệt giữa cơ chế an toàn an ninh với việc thực thi các cơ chế này. Để thực thi các kênh an toàn, NFS cung cấp 1 giao diện chuẩn ở dạng RPCSEC-GSS, bằng cách đó các hệ thống an toàn an ninh đang tồn tại có thể được truy cập. NFS phiên bản 4 còn cung cấp 1 danh sách mở rộng các thao tác được dùng cho điều khiển truy cập (ACL).
- Coda và Plan 9 cũng cung cấp các kênh truyền an toàn, nhưng cả 2 đều thực thi dựa trên giao thức xác thực Needham – Schroeder. Các khóa mật chia sẻ được dùng trong cách tiếp cận này. Coda cũng có cách khác với NFS trong việc điều khiển truy cập. Nó chỉ xử lý điều khiển truy cập đối với các thao tác trên thư mục. Với Plan 9 (và cả xFS) thì chủ yếu theo hướng tiếp cận UNIX chuẩn, bằng cách phân biệt các thao tác đọc, ghi, và thực thi.
- SFS có cách riêng của nó để cung cấp các kênh an toàn. Nó dựa vào cơ chế xác thực server với người sử dụng, có thể dùng thêm các tác tử đặc biệt. SFS kế thừa các cơ chế điều khiển truy cập từ NFS phiên bản 3.

Bảng (*H.24*) giúp tổng hợp lại sự so sánh giữa 5 hệ thống file phân tán.

| | **NFS** | **Coda** | **Plan 9** | **xFS** | **SFS** |
|---|---|---|---|---|---|
| Mục tiêu thiết kế | Truy cập trong suốt | Tính sẵn sàng cao | Tính như nhau | Hệ thống không server | An toàn an ninh thay đổi |
| Mô hình truy cập | Từ xa | Up.Download | Từ xa | Log-based | Từ xa |
| Truyền thông | RPC | RPC | Đặc biệt | Thông điệp linh động | RPC |
| Tiến trình client | Thin.Fat | Fat | Thin | Fat | Vừa |
| Các nhóm | Không | Có | Không | Có | Không |

| | | | | | |
|---|---|---|---|---|---|
| server | | | | | |
| Không gian tên | Mỗi client | Toàn cục | Mỗi tiến trình | Toàn cục | Toàn cục |
| Phạm vi ID file | File server | Toàn cục | Server | Toàn cục | Hệ thống file |
| Ngữ nghĩa chia sẻ | Phiên | Giao tác | UNIX | UNIX | Không được chỉ ra |
| Đơn vị lưu tạm (đệm) | File (v4) | File | File | Khối | Không được chỉ ra |
| Nhân bản | Tối thiểu | ROWA | Không | Striping | Không |
| Chịu lỗi | Truyền thông tin cậy | Nhân bản và lưu tạm | Truyền thông tin cậy | Striping | Truyền thông tin cậy |
| Hồi phục | Dựa vào client | Khôi phục | Không được chỉ ra | Điểm kiểm tra và ghi nhật ký | Không được chỉ ra |
| Các kênh an toàn | Các cơ chế đang tồn tại | Needham – Schroeder | Needham – Schroeder | Không đường dẫn tên | Tự chứng nhận |
| Điều khiển truy cập | Nhiều thao tác | Các thao tác thư mục | Dựa trên UNIX | Dựa trên UNIX | Dựa trên NFS |

## IV. Kết luận

Hệ phân tán là 1 hệ thống có chức năng và dữ liệu phân tán trên các trạm (máy tính) được kết nối với nhau qua 1 mạng máy tính. Việc thiết kế 1 hệ phân tán phải tuân theo 7 nguyên lý mà ta đã đề cập đến ở phần đầu.

Trong việc xây dựng các hệ phân tán, thì 1 trong những mô thức quan trọng của nó đó là các hệ thống file phân tán. Như ta biết, chia sẻ dữ liệu là 1 trong những chức năng cơ bản của hệ phân tán. Hệ thống file phân tán cho phép nhiều tiến trình cùng chia sẻ dữ liệu trong khoảng thời gian dài 1 cách an toàn và tin cậy. Ở trên ta cũng đã xem xét 1 số các hệ thống file khá phổ biến như: NFS, Coda, Plan 9, xFS, SFS. Khi phân tích các hệ thống này, giúp ta hiểu sâu hơn về các nguyên lý của 1 hệ phân tán nói chung.

Tuy đã có nhiều cố gắng của bản thân, nhưng bởi sự hạn chế về kiến thức nên trong tiểu luận vẫn còn nhiều sai sót. Cuối cùng, em xin cám ơn thầy GS-TS. Nguyễn Thúc Hải đã giúp đỡ em trong suốt quá trình hoàn tất tiểu luận này.

❖ **Các tài liệu tham khảo được dùng trong tiểu luận:**

[1]. *Distributed system: principles and paradigms – A.S.Tanenbaum, M.V.Steen.*

[2]. *Distributed system: concepts and design – G.Couloms, J.Dollimore, T.Kinberg.*

[3]. Các địa chỉ website:

- http:..www.cs.sfu.ca.CC.401.tiko.lecnotes.ch4.3.pdf
- http:..en.wikipedia.org.wiki.Network_File_System_(Sun)
- http:..courses.washington.edu.css434.students.NFS.ppt